

Installing wasora

This file contains brief instructions to download, and/or compile and/or install [wasora](#). The detailed discussed is deferred to the full documentation (see directory `doc`).

Getting wasora

The wasora code can be obtained essentially in either source or binary form. The v0.3.x series uses [Mercurial](#) as the version control system and the repository is hosted at [Bitbucket](#) (v0.2.x series used [Bazaar](#) and [Launchpad](#)). Tarballs containing either sources or binaries are periodically prepared from the repository sources and may be downloaded from <http://www.talador.com.ar/jeremy/wasora/>.

In order of decreasing level of user expertise, to get wasora either

- Clone the [Mercurial Bitbucket](#) repository:

```
$ hg clone https://bitbucket.org/gtheler/wasora
```

and proceed to the [Bootstrapping](#) section. You may keep your tree up-to-date by pulling incremental changes:

```
$ cd wasora
$ hg pull
pulling from https://bitbucket.org/gtheler/wasora
searching for changes
no changes found
$
```

- Download the latest source tarball listed in wasora's [home page](#) (a list of files and versions available to be downloaded can be found by browsing <http://www.talador.com.ar/jeremy/wasora/download>)
Then proceed to the [Compiling](#) section.
- Download the latest binary tarball for your architecture. Currently, the options are
 - linux-amd64: GNU/Linux 64-bit statically linked binary
 - linux-i386: GNU/Linux 32-bit statically linked binary
 - win32-mingw: [MinGW](#)-based 32-bit Windows binary
 - win32-cygwin: [Cygwin](#)-based 32-bit Windows binary (includes `cygwin1.dll`)

and proceed to the [Installing](#) section.

The provided binaries are statically linked to the required libraries to avoid having a user that expects to run wasora out of the box dealing with unresolved library dependences. However, there may be some libraries that are not available in a certain configuration. A full source compilation is recommended.

Note that using wasora in non-free operating systems is highly discouraged. Please try switching to [GNU/Linux](#).

Bootstrapping

Skip this section if you did not clone the [repository](#).

The repository development tree has to be bootstrapped by [autoconf & friends](#) to be able to configure and make the code. The script `autogen.sh` generates the files that `autoconf` needs to produce a working configure script:

```
$ ./autogen.sh
cleaning... done
getting hg revision id... done
building changelog... done
formatting readme & install... done
building configure.ac... done
building src/Makefile.am... done
calling autoreconf...
configure.ac:21: installing './compile'
configure.ac:16: installing './config.guess'
configure.ac:16: installing './config.sub'
configure.ac:18: installing './install-sh'
configure.ac:18: installing './missing'
parallel-tests: installing './test-driver'
src/Makefile.am: installing './depcomp'
done
$
```

At this point, a tree similar to the source distribution tarball is obtained, which can be configured and compiled as described in the section [Compiling](#) below. The bootstrapped files are listed in `.hgignore` so Mercurial should not report any changes in the status of the working tree after executing `autogen.sh`:

```
$ hg status
$
```

To clean the working tree and revert it to a fresh-clone status, the `autoclean.sh` script should be executed:

```
$ ./autoclean.sh
```

Note that `autogen.sh` calls `autoclean.sh` first.

Compiling

Skip this section if you downloaded a binary tarball.

wasora follows the standard GNU `./configure && make` procedure. So uncompress the downloaded tarball into a proper location within your home directory:

```
$ tar xvzf wasora-0.3.4.tar.gz
$ cd wasora
```

Then execute the `configure` script so it can check the environment is able to build wasora:

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
[...]
config.status: creating src/Makefile
config.status: executing depfiles commands
```

```
## ----- ##
## Configuration summary ##
## ----- ##
GSL library (required): yes, version 1.16

IDA library (optional): yes, version unknown
    differential-algebraic systems will be solved

Readline library (opt): yes, version 6.3
    run-time debugging-like capabilities will be provided
```

Now proceed to compile with 'make'

```
$
```

If any error is reported, there may be likely problems with finding one or more libraries. See the section [Required libraries](#) below to see how the error can be fixed.

Once the `configure` step successfully tells us to proceed to compile with `make`, that is what we do:

```
$ make
```

The executable binary will be located in the current directory and called `wasora`. At this point you may want to check if wasora actually works by executing the test suite:

```
$ make check
```

If `gnuplot` is installed, some graphical windows should pop up. See the `README` for a full list of the test involved.

The usual way to finish the compilation of a program that follows the GNU standard is to perform a system-wide installation by executing (as root):

```
# make install
```

This would leave the executable `wasora` available to be executed by any user of the system. However, other workflows may be used to run wasora. See the section [Installing](#) for details.

By default, `configure` sets `CFLAGS=-O2`. To obtain a binary with debugging symbols, call either `./configure` or `make` with `CFLAGS=-g`, i.e.

```
$ ./configure CFLAGS=-g
$ make
```

or

```
$ ./configure
$ make CFLAGS=-g
```

Required libraries

The code depends on a few libraries (and its development headers). Some of them are mandatory and some of the are optional. The name in parenthesis refers to the Debian-based package name

- Mandatory for compilation:
 - [GNU Scientific Library](#) (`libgsl0-dev`)
- Needed if DAE systems are to be solved:
 - [IDA SUNDIALS Library](#) (`libsundials-serial-dev`)
- Needed if debug-mode is desired:
 - [GNU Readline](#) (`libreadline-dev`)

Therefore, in Debian-based GNU/Linux boxes, one would do

```
# apt-get install libgsl0-dev libsundials-serial-dev libreadline-dev
```

and all the required libraries (and development headers) should be detected by `configure`. Note that some wasora plugins (such as [milonga](#)) may need further additional libraries (for instance [PETSc](#) and [SLEPc](#)).

If `configure` is still unable to detect the GSL, it can be instructed to download and compile it in a local subdirectory using the `--enable-download-gsl` option:

```
$ ./configure --enable-download-gsl
```

If no Internet connection is available, the file `gsl-1.16.tar.gz` may be separately obtained (for example from <http://ftpmirror.gnu.org/gsl/>) and copied into the wasora directory.

When giving the `--enable-download-gsl` option, the generated binary will be statically linked against the downloaded library.

By default, `configure` checks for the optional libraries. However, they can be explicitly disabled by using the `--without-ida` and `--without-readline` options in `configure`:

```
$ ./configure --without-ida --without-readline
[...]
config.status: creating src/Makefile
config.status: executing depfiles commands
```

```
## ----- ##
```

```
## Configuration summary ##
## ----- ##
GSL library (required): yes, version 1.16

IDA library (optional): no
differential-algebraic systems will NOT be solved

Readline library (opt): no
run-time debugging-like capabilities will NOT be provided

WARNING: there is at least one optional library missing.
If this was not the desired result, check config.log for clues.
```

Now proceed to compile with 'make'

\$

Call `./configure --help` to see all the available options. Which libraries wasora was finally linked against to can be checked by executing it with the `-v` option:

```
$ ./wasora -v
wasora 0.3.5 default (2014-08-17 03:11 -0300)
wasora's an advanced suite for optimization & reactor analysis

revision id d033f6fa994d319b4d60978c073dcb8aaa6a221a
last commit on 2014-08-17 03:11 -0300 (rev 5)

compiled on 2014-08-17 12:20:38 by jeremy@tom (linux-gnu x86_64)
with gcc (Debian 4.9.1-4) 4.9.1 using -O2 and linked against
GNU Scientific Library version 1.16
GNU Readline version 6.3
SUNDIALS Library version 2.5.0
```

```
wasora is copyright (C) 2009-2014 jeremy theler
licensed under GNU GPL version 3 or later.
wasora is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

\$

If something goes wrong and the compilation fails, please feel free to ask for help at the wasora mailing list at wasora@talador.com.ar.

Installing

Further information

See the file `README` for a description of wasora and the test suite.
See the contents of directory `doc` for full documentation.

Home page: <http://www.talador.com.ar/jeremy/wasora>

Mailing list and bug reports: wasora@talador.com.ar

wasora is copyright (C) 2009–2014 jeremy theler

wasora is licensed under [GNU GPL version 3](#) or (at your option) any later version.

wasora is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

See the file `COPYING` for copying conditions.

The text below the cutting line corresponds to the original FSF instructions for installing software (as wasora) that follows the GNU `configure & make` convention.

Installation Instructions

Copyright (C) 1994-1996, 1999-2002, 2004-2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved. This file is offered as-is, without warranty of any kind.

Basic Installation

Briefly, the shell commands `./configure; make; make install` should configure, build, and install this package. The following more-detailed instructions are generic; see the `README` file for instructions specific to this package. Some packages provide this `INSTALL` file but do not implement all of the features documented below. The lack of an optional feature in a given package is not necessarily a bug. More recommendations for GNU packages can be found in `*note Makefile Conventions: (standards)Makefile Conventions`.

The `configure` shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile` in each directory of the package. It may also create one or more `.h` files containing system-dependent definitions. Finally, it creates a shell script `config.status` that you can run in the future to recreate the current configuration, and a file `config.log` containing compiler output (useful mainly for debugging `configure`).

It can also use an optional file (typically called `config.cache` and enabled with `--cache-file=config.cache` or simply `-C`) that saves the results of its tests to speed up reconfiguring. Caching is disabled by default to prevent problems with accidental use of stale cache files.

If you need to do unusual things to compile the package, please try to figure out how `configure` could check whether to do them, and mail diffs or instructions to the address given in the `README` so they can be considered for the next release. If you are using the cache, and at some point `config.cache` contains results you don't want to keep, you may remove or edit it.

The file `configure.ac` (or `configure.in`) is used to create `configure` by a program called `autoconf`. You need `configure.ac` if you want to change it or regenerate `configure` using a newer version of `autoconf`.

The simplest way to compile this package is:

1. `cd` to the directory containing the package's source code and type `./configure` to configure the package for your system.
Running `configure` might take a while. While running, it prints some messages telling which features it is checking for.
2. Type `make` to compile the package.
3. Optionally, type `make check` to run any self-tests that come with the package, generally using the just-built uninstalled binaries.
4. Type `make install` to install the programs and any data files and documentation. When installing into a prefix owned by root, it is recommended that the package be configured and built as a regular user, and only the `make install` phase executed with root privileges.
5. Optionally, type `make installcheck` to repeat any self-tests, but this time using the binaries in their final installed location. This target does not install anything. Running this target as a regular user, particularly if the prior `make install` required root privileges, verifies that the installation completed correctly.
6. You can remove the program binaries and object files from the source code directory by typing `make clean`. To also remove the files that `configure` created (so you can compile the package for a different kind of computer), type `make distclean`. There is also a `make maintainer-clean` target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.
7. Often, you can also type `make uninstall` to remove the installed files again. In practice, not all packages have tested that uninstallation works correctly, even though it is required by the GNU Coding Standards.
8. Some packages, particularly those that use Automake, provide `make distcheck`, which can be used by developers to test that all other targets like `make install` and `make uninstall` work correctly. This target is generally not run by end users.

Compilers and Options

Some systems require unusual options for compilation or linking that the `configure` script does not know about. Run `./configure --help` for details on some of the pertinent environment variables.

You can give `configure` initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

```
./configure CC=c99 CFLAGS=-g LIBS=-lposix
```

*Note Defining Variables::, for more details.

Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you can use GNU `make`. `cd` to the directory where you want the object files and executables to go and run the `configure` script. `configure` automatically checks for the source code in the directory that `configure` is in and in ... This is known as a "VPATH" build.

With a non-GNU `make`, it is safer to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use `make distclean` before reconfiguring for another architecture.

On MacOS X 10.5 and later systems, you can create libraries and executables that work on multiple system types—known as “fat” or “universal” binaries—by specifying multiple `-arch` options to the compiler but only a single `-arch` option to the preprocessor. Like this:

```
./configure CC="gcc -arch i386 -arch x86_64 -arch ppc -arch ppc64" \  
           CXX="g++ -arch i386 -arch x86_64 -arch ppc -arch ppc64" \  
           CPP="gcc -E" CXXCPP="g++ -E"
```

This is not guaranteed to produce working output in all cases, you may have to build one architecture at a time and combine the results using the `lipo` tool if you have problems.

Installation Names

By default, `make install` installs the packages commands under `/usr/local/bin`, include files under `/usr/local/include`, etc. You can specify an installation prefix other than `/usr/local` by giving `configure` the option `--prefix=PREFIX`, where `PREFIX` must be an absolute file name.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you pass the option `--exec-prefix=PREFIX` to `configure`, the package uses `PREFIX` as the prefix for installing programs and libraries. Documentation and other data files still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like `--bindir=DIR` to specify different values for particular kinds of files. Run `configure --help` for a list of the directories you can set and what kinds of files go in them. In general, the default for these options is expressed in terms of `${prefix}`, so that specifying just `--prefix` will affect all of the other directory specifications that were not explicitly provided.

The most portable way to affect installation locations is to pass the correct locations to `configure`; however, many packages provide one or both of the following shortcuts of passing variable assignments to the `make install` command line to change installation locations without having to reconfigure or recompile.

The first method involves providing an override variable for each affected directory. For example, `make install prefix=/alternate/directory` will choose an alternate location for all directory configuration variables that were expressed in terms of `${prefix}`. Any directories that were specified during `configure`, but not in terms of `${prefix}`, must each be overridden at install time for the entire installation to be relocated. The approach of makefile variable overrides for each directory variable is required by the GNU Coding Standards, and ideally causes no recompilation. However, some platforms have known limitations with the semantics of shared libraries that end up requiring recompilation when using this method, particularly noticeable in packages that use GNU Libtool.

The second method involves providing the `DESTDIR` variable. For example, `make install DESTDIR=/alternate/directory` will prepend `/alternate/directory` before all installation names. The approach of `DESTDIR` overrides is not required by the GNU Coding Standards, and does not work on platforms that have drive letters. On the other hand, it does better at avoiding recompilation issues, and works well even when some directory options were not specified in terms of `${prefix}` at `configure` time.

Optional Features

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving `configure` the option `--program-prefix=PREFIX` or `--program-suffix=SUFFIX`.

Some packages pay attention to `--enable-FEATURE` options to `configure`, where `FEATURE` indicates an optional part of the package. They may also pay attention to `--with-PACKAGE` options, where `PACKAGE` is something like `gnu-as` or `x` (for the X Window System). The `README` should mention any `--enable-` and `--with-` options that the package recognizes.

For packages that use the X Window System, `configure` can usually find the X include and library files automatically, but if it doesn't, you can use `theconfigureoptions-x-includes=DIRand-x-libraries=DIR'` to specify their locations.

Some packages offer the ability to configure how verbose the execution of `make` will be. For these packages, running `./configure --enable-silent-rules` sets the default to minimal output, which can be overridden with `make V=1`; while running `./configure --disable-silent-rules` sets the default to verbose, which can be overridden with `make V=0`.

Particular systems

On HP-UX, the default C compiler is not ANSI C compatible. If GNU CC is not installed, it is recommended to use the following options in order to use an ANSI C compiler:

```
./configure CC="cc -Ae -D_XOPEN_SOURCE=500"
```

and if that doesn't work, install pre-built binaries of GCC for HP-UX.

HP-UX `make` updates targets which have the same time stamps as their prerequisites, which makes it generally unusable when shipped generated files such as `configure` are involved. Use GNU `make` instead.

On OSF/1 a.k.a. Tru64, some versions of the default C compiler cannot parse its `<wchar.h>` header file. The option `-nodtk` can be used as a workaround. If GNU CC is not installed, it is therefore recommended to try

```
./configure CC="cc"
```

and if that doesn't work, try

```
./configure CC="cc -nodtk"
```

On Solaris, don't put `/usr/ucb` early in your `PATH`. This directory contains several dysfunctional programs; working variants of these programs are available in `/usr/bin`. So, if you need `/usr/ucbin` your `PATH`, put it `_after_ /usr/bin`.

On Haiku, software installed for all users goes in `/boot/common`, not `/usr/local`. It is recommended to use the following options:

```
./configure --prefix=/boot/common
```

Specifying the System Type

There may be some features `configure` cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the *same* architectures, `configure` can figure that out, but if it prints a message saying it cannot guess the machine type, give it the `--build=TYPE` option. `TYPE` can either be a short name for the system type, such as `sun4`, or a canonical name which has the form:

```
CPU-COMPANY-SYSTEM
```

where `SYSTEM` can have one of these forms:

```
OS  
KERNEL-OS
```

See the file `config.sub` for the possible values of each field. If `config.sub` isn't included in this package, then this package doesn't need to know the machine type.

If you are *building* compiler tools for cross-compiling, you should use the option `--target=TYPE` to select the type of system they will produce code for.

If you want to *use* a cross compiler, that generates code for a platform different from the build platform, you should specify the "host" platform (i.e., that on which the generated programs will eventually be run) with `--host=TYPE`.

Sharing Defaults

If you want to set default values for `configure` scripts to share, you can create a site shell script called `config.site` that gives default values for variables like `CC`, `cache_file`, and `prefix`. `configure` looks for `PREFIX/share/config.site` if it exists, then `PREFIX/etc/config.site` if it exists. Or, you can set the `CONFIG_SITE` environment variable to the location of the site script. A warning: not all `configure` scripts look for a site script.

Defining Variables

Variables not defined in a site shell script can be set in the environment passed to `configure`. However, some packages may run `configure` again during the build, and the customized values of these variables may be lost. In order to avoid this problem, you should set them in the `configure` command line, using `VAR=value`. For example:

```
./configure CC=/usr/local2/bin/gcc
```

causes the specified `gcc` to be used as the C compiler (unless it is overridden in the site shell script).

Unfortunately, this technique does not work for `CONFIG_SHELL` due to an Autoconf limitation. Until the limitation is lifted, you can use this workaround:

```
CONFIG_SHELL=/bin/bash ./configure CONFIG_SHELL=/bin/bash
```

configure Invocation

`configure` recognizes the following options to control how it operates.

`--help -h` Print a summary of all of the options to `configure`, and exit.

`--help=short` `--help=recursive` Print a summary of the options unique to this packages, `configure`, and `exit`. The `short` variant lists options used only in the top level, while the `recursive` variant lists options also present in any nested packages.

`--version -V` Print the version of Autoconf used to generate the `configure` script, and exit.

`--cache-file=FILE` Enable the cache: use and save the results of the tests in `FILE`, traditionally `config.cache`. `FILE` defaults to `/dev/null` to disable caching.

`--config-cache -C` Alias for `--cache-file=config.cache`.

`--quiet` `--silent -q` Do not print messages saying which checks are being made. To suppress all normal output, redirect it to `/dev/null` (any error messages will still be shown).

`--srcdir=DIR` Look for the packages source code in directory `DIR`. Usually `configure` can determine that directory automatically.

`--prefix=DIR` Use `DIR` as the installation prefix. *note Installation Names:: for more details, including other options available for fine-tuning the installation locations.

`--no-create -n` Run the configure checks, but stop before creating any output files.

`configure` also accepts some other, not widely useful, options. Run `configure --help` for more details.