BUSINESS FIRST
Application Composer

Application Composer

USER GUIDE

Reference: APPCOMPOSER_USER_046_EN

For future updates of this document, visit our Product Documentation section on
www.myw4.com

Application Composer

USER GUIDE

Reference: APPCOMPOSER_USER_046_EN

First steps

Application Composer is the graphical workshop for the creation of Application Engine applications.

Application Composer reduces technical complexity and makes it easier and faster to design Application Engine applications:

- Application Composer makes it possible to configure the applications with no previous knowledge of XML.
- A keen knowledge of data description syntax is no longer required when starting to use Application Engine.
- Application Composer provides multiple services, making development faster, such as discovery, execution for different types of viewers, definition of resources, etc.

In this section, we will review the first steps when starting with Application Composer:

> 1.1 Installing Application Composer, page 11
> 1.2 Starting Application Composer, page 11
> 1.3 Opening a preexisting application, page 11
> 1.4 Creating an application, page 12

We will also introduce the user interface main components and review the preference options:

> 1.5 Introducing the UI, page 13
> 1.6 Preferences, page 21

1.1 Installing Application Composer

Windows

Start the installation wizard (file *ApplicationComposer_X.X_Setup.exe* in the *ApplicationComposer* folder of the installation CD).

1.2 Starting Application Composer

Windows

Double-click *ApplicationComposer.exe* in the *Studio* folder of your Application Composer setup directory (default is *C:\Program Files\W4 BUSINESS FIRST\ApplicationComposer\Studio*).

[ALTERNATIVELY] You can use any Windows Start menu or desktop shortcuts, which you may have created during setup.

[ALTERNATIVELY] From a DOS command prompt, navigate to the `Studio` folder of your Application Composer setup directory and run *application_composer.bat*.

1.3 Opening a preexisting application

TO OPEN A PREEXISTING APPLICATION

1. Select **File** ▸ **Open**.

   The sub-menu that is displayed contains two sections: the top section lists the recent applications, i.e. the applications you have already worked with using your copy of Application Composer, while the **Application...** option can be used to open an application that has never been edited with your copy of Application Composer.
2. To open a recent application, just select it in the sub-menu.
3. To open an application for the first time, select **Application...**

The following window is displayed:

Fig 1.1 *Opening an application*

4 Browse to the appropriate application directory.

5 Click **Validate**.

1.4 Creating an application

TO CREATE AN APPLICATION

1 Select **File** ▸ **New**.

The *New : Application* window is displayed:

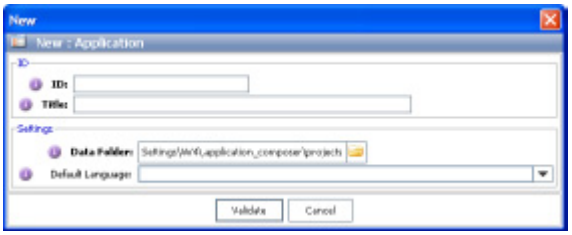

Fig 1.2 *Creating an application*

2 Set the fields:

ID - The application's Java identifier. This value will be the default package where the application's Java classes will be stored.

Title - The application's name in the default language. This value will be displayed in the generated application's title bar.

Data folder - The target directory where your application files will be stored. The default value for this field can be configured via the `DEFAULT_APPLICATION_PATH` resource in the application_composer.ini file.

Default language - The application's default language. Applications can handle multiple languages.

For further information regarding language management:

› 2.11 Managing application resources, page 35

3 Click **Validate**.

NOTE The icon flags context-sensitive help. To display the corresponding tip, hover over the icon for a while. This icon is available in many of the Application Composer windows.

1.5 Introducing the UI

The user interface has permanent components such as the main tool bar and the *class hierarchy*, and also a number of tabs, which can be displayed or hidden.

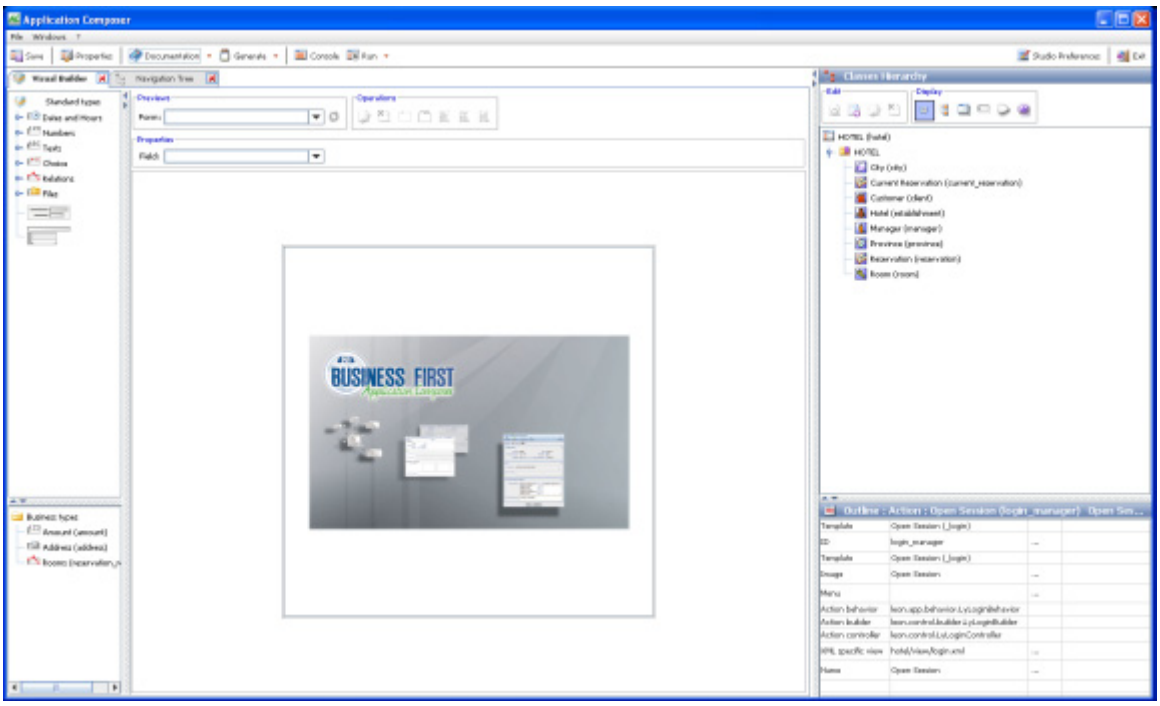

Fig 1.3 *The user interface*

For further information regarding the various user interface components:

- 1.5.1 Main tool bar, page 14
- 1.5.2 Class hierarchy, page 14
- 1.5.3 Navigation tree, page 15
- 1.5.4 Visual builder, page 16
- 1.5.5 Discovery, page 18
- 1.5.6 Java classes, page 18
- 1.5.7 Diagram, page 19
- 1.5.8 Other tabs, page 20

1.5.1 Main tool bar

The main tool bar is always available at the top of Application Composer's main window.

It includes the following items:

- **Save**
  - 3.8 Saving a project, page 51
- **Properties**
  - 2.1 Editing an application, page 25
- **Documentation**
  - 2.9 Generating application documentation, page 32
- **Generate**
  - 2.12 Generating the final application, page 37
- **Console** - To display the execution console
- **Run**
  - 2.13 Running the final application, page 40
- **Preferences**
  - 1.6 Preferences, page 21

1.5.2 Class hierarchy

The *Class hierarchy* displays the current application as a tree view.

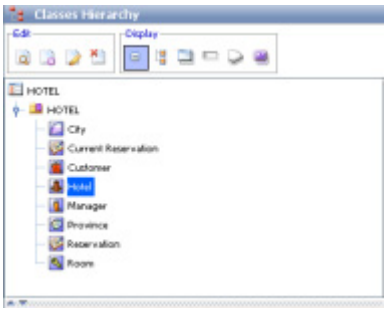

Fig 1.4 *The class hierarchy*

The root item is the application. The second and third levels are the project and the application classes respectively.

In addition to application classes, a project may contain sub-projects, actions, fields, filters, and sort definitions.

The class hierarchy tool bar has the following icons:

- In the *Edit* area:
  - **Details** - Opens the read-only form for the currently selected object.
    - Reviewing application details, page 25
    - 3.2 Viewing details of a project or sub-project, page 43
    - 4.2 Viewing details of a class, page 54
  - **Create class** - Creates an application class in the currently selected package or in the parent package of the currently selected class.
    - 4.1 Creating a class, page 53
  - **Modify** - Opens the edit form for the currently selected object.
    - 2.1 Editing an application, page 25
    - 3.3 Editing a project, page 44
    - 4.4 Editing a class, page 56
  - **Delete** - Deletes the currently selected object.
    - Removing an application, page 25
    - 4.5 Removing a class, page 57
- In the *Display* area:
  - **Expand / Shrink tree** : Displays or hides project items (sub-projects, classes, etc).
  - **Show / Hide inheritance hierarchy** : If disabled, all classes are displayed at the same level even though some classes extend some others.
  - **Show / Hide actions**
  - **Show / Hide fields**
  - **Show / Hide filters**
  - **Show / Hide sorts**

1.5.3 Navigation tree

The *Navigation tree* displays a tree view of the actions that make up the application.

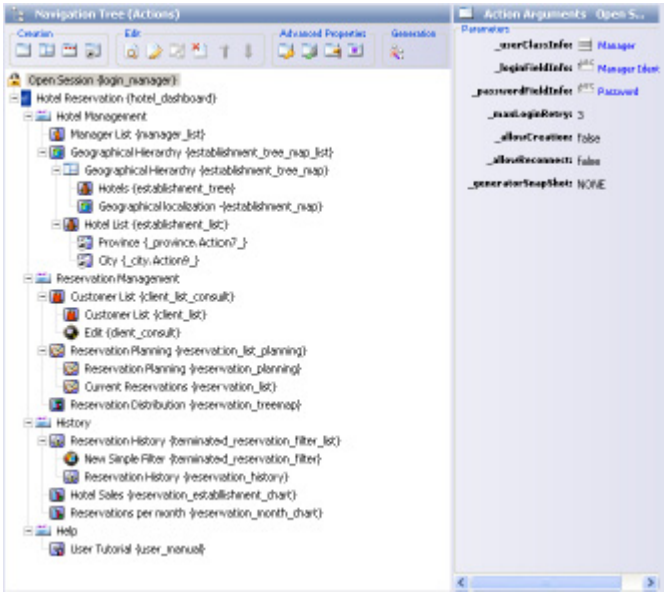

Fig 1.5 *The navigation tree*

To display the navigation tree, select **Windows** ▸ **Navigation tree**.

For further information regarding actions:

> 7 Working with actions, page 140

1.5.4 Visual builder

The *Visual builder* is used to build and preview your application forms.

Using the visual builder you can drag and drop fields to the graphical representation of the currently selected class.

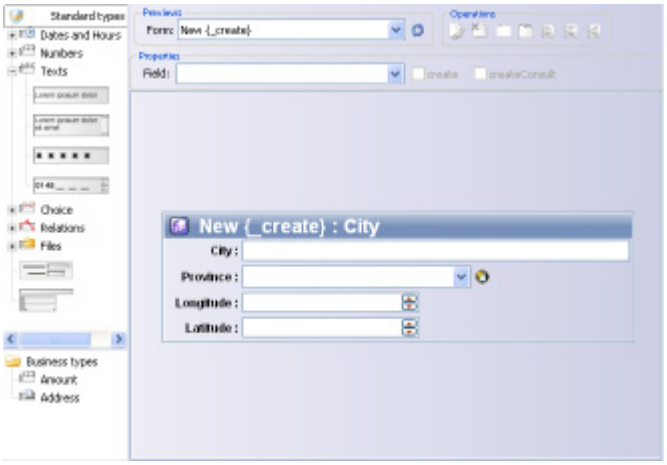

Fig 1.6 *The visual builder*

To display the visual builder, select **Windows** ▸ **Visual builder**.

Form - To select the action for which you want to display a form preview.

Field - To select either a specific field in the form, or all the fields in the form.

To select more than one field, hold down the CTRL key then click in turn the appropriate fields in the form preview.

create / createConsult / hidden / optional - To apply these marks to the selected field.

For futher information regarding marks:
› 5.30 Setting attribute marks, page 120

Field settings - To display the edit form(s) for the selected field(s).

Delete the field - To delete the selected field(s).

Group - To organize the selected fields into a group.

Tab - To organize the selected fields into a tab.

Align left - To align the selected fields to the left.

Center - To align the selected fields to the center.

Align right - To align the selected fields to the right.

1.5.5 Discovery

Discovery is used to explore relational databases, Java files (Bean, EJB), CSV files, W4 procedure models, or ECM systems, in order to extract their structure and implement their data as part of an application.

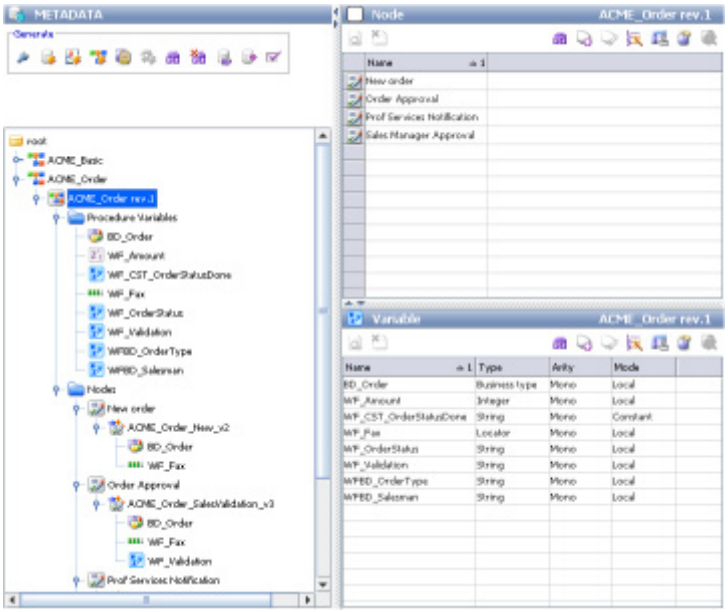

Fig 1.7 *Discovery*

To display Discovery, select **Windows** ▸ **Discovery**.

> 10 Working with Discovery, page 176

1.5.6 Java classes

The *Java classes* tab displays the Java classes (behaviors, builders) that have been defined for an application class, for the currently selected project or application in the class hierarchy.

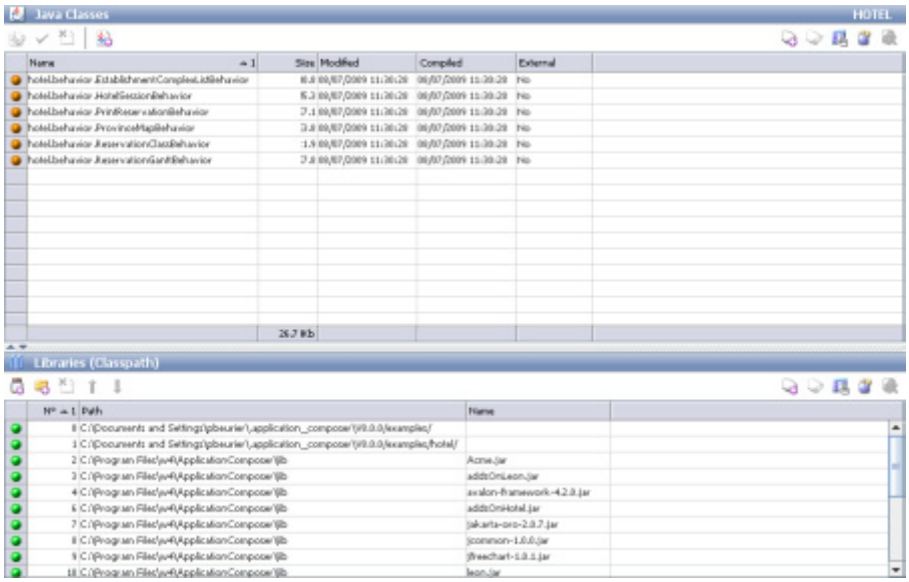

Fig 1.8 *The Java classes tab*

To display this tab, select **Windows** ▸ **Java Classes**.

> 8 Managing Java classes and libraries, page 166

1.5.7 Diagram

The *Diagram* tab displays a UML representation of the currently selected logical class in the class hierarchy.

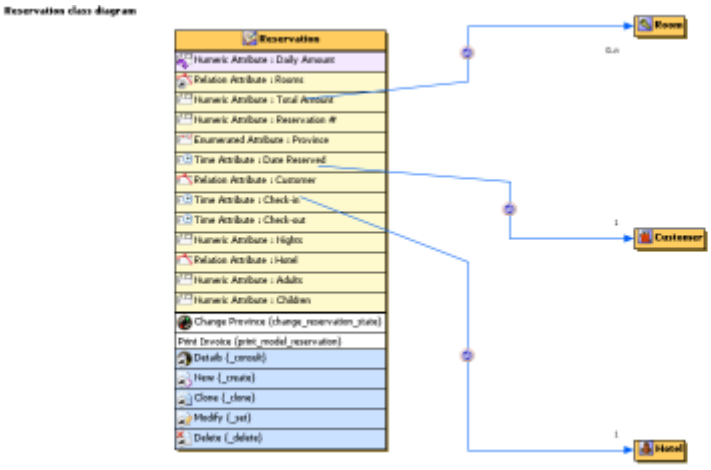

Fig 1.9 *The Diagram tab*

To display this tab, select **Windows** ▸ **Diagram**.

1.5.8 Other tabs

The following tabs can be displayed via the **Windows** menu:

- **Classes** - Displays all the Java classes in the application and allows you to manage the CLASSPATH used for compilation.

> 8 Managing Java classes and libraries, page 166

- **Attributes** - Displays the available fields for the currently selected logical class, project, or application.

> 5 Working with attributes, page 76

- **Actions** - Displays the available actions for the currently selected logical class, project, or application.

> 7 Working with actions, page 140

- **Routes** - Displays the routes required to define the path used by the Application Engine environment for cross-reference calculation.

> 6 Working with routes, page 136

The following resources tabs can also be displayed via the **Windows** menu:

- **Strings**
- **Messages**
- **Files**
- **Images**
- **Colors**
- **Fonts**
- **Parameters**

Resources are used by the generic code and potentially also by the application code when generating the views. For example, resources define the strings such as action names or menu names. They also define the fonts and the colors to be used when the Application Engine core builds the views.

> 2.11 Managing application resources, page 35

1.6 Preferences

Preferences manage the general properties of both Application Composer, and the external applications it uses.

Preference options fall into the following tabs:

General

Installation dir - The directory where Application Engine sources are located.

Internet browser - The access path to the web browser's executable file. This path is used when running an application in web mode.

Level - To select your preferred profile:

- **Beginner** - To build a basic application using flat files.
- **Standard** - To implement routes, Java behaviors, and specific data sources.
- **Advanced** - To implement specific views, structure attributes, user actions, advanced class-related features (inheritance, rules, etc), advanced action-related features (specific marks, etc), and advanced field-related features (units, etc).
- **Expert** - To implement advanced configuration features (XML import, dynamic labels, etc).

Use spaces instead of tabs

Number of spaces for one tab

Automatic calculation of IDs - To have Application Composer automatically specify the identifiers for you when creating objects such as classes, attributes, options, actions, etc. Default IDs are based upon the values specified in the *Suffixes* and *Actions* areas.

For instance, an attribute's default ID is as follows:

<*class*><*FIELD*> (example: *cityField*)

where

- *class* stands for the name of the corresponding class
- *FIELD* is the value specified in the preferences' *General* tab (*Attributes* field in the *Suffixes* area)

Subsequent attributes in the class will be suffixed by an increment number:

<class><FIELD>_<n> (example: *cityField_1*)

Java

Use an IDE - Specifies whether an IDE should be used. If so, specify its access path in the **Executable of the IDE** field, and also its communication port.

JDK folder - The path to the JDK used by Application Engine to compile the Java classes.

Compiler - The compiler program's command, when no IDE is used.

Editor - The access path to the editor's executable file, when no IDE is used.

Copyright - The copyright to be specified when generating a Java class.

Tomcat

Ignore Xerces at deployment time - To specify whether Xerces will be ignored at deployment time. To be used when there is a conflict between parsers.

URL - Full URL for running an application in web mode.

Override Tomcat configuration - Select this option if you want to make changes to either of the **Application folder**, **Executable folder**, **Start command** or **Stop command** fields.

Tomcat folder - By default Application Composer uses the Tomcat instance embedded in the Application Engine installation. Change as appropriate if you want to use another Tomcat instance.

Application folder - The access path to the application server's application directory.

Executable folder - The access path to the application server's executable directory. This directory is used to determine the start and stop commands.

Start command - Change as appropriate if you want to use a custom start command.

Stop command - Change as appropriate if you want to use a custom stop command.

Model discovery

Use a specific ECM model

Jar containing the ECM model

Ant

Target for JAR generation

Target for WAR generation

Target for WAR update

Target for WebService WAR generation

Target for B1A directory tree generation

Target for B1A generation

String generation

Actions

Class

Attribute

Filter

Sort

Business First

Extension Bus application directory

Working with applications

Application creation has been covered in a previous section:

> 1.4 Creating an application, page 12

In this section we will cover the following topics:

- How to perform basic application-related actions:

> Reviewing application details, page 25
> Removing an application, page 25
> 2.1 Editing an application, page 25

- How to add a specific Java class for an application's or a user session's behavior:

> 2.2 Specifying an application behavior, page 27
> 2.3 Specifying a session behavior, page 28

- How to add a specific Main class to be used either at application startup (fat client), or by the servlet container:

> 2.4 Adding a specific main class, page 29
> 2.5 Adding a specific servlet class, page 30

- How to use import and export features:

> 2.6 Exporting an application, page 31
> 2.7 Importing an application, page 31
> 2.8 XMI import, page 32

- How to generate application documentation:

> 2.9 Generating application documentation, page 32

- How to manage the applications and the resources they are using:

> 2.10 Managing applications, page 34
> 2.11 Managing application resources, page 35

- How to generate and run the final application:

> 2.12 Generating the final application, page 37
> 2.13 Running the final application, page 40

Reviewing application details

The **Details** option in the class hierarchy displays the current application's read-only form.

For further information regarding the fields in this application read-only form:
> 2.1 Editing an application, page 25

Removing an application

The **Delete** option in the class hierarchy removes the current application.

2.1 Editing an application

The **Modify** option in the class hierarchy displays the current application's edit form:

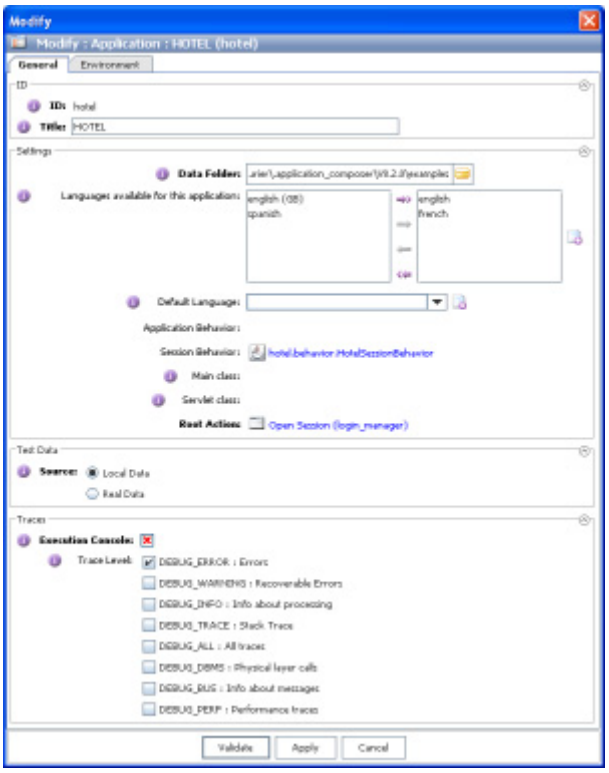

Fig 2.1 *Editing an application, General tab*

Data folder

Languages available for this application - The languages supported by the application, in addition to the default language. Apcation Composer manages one property file per language.

Default language

Application behavior - [optional] The Java class that overloads the application's global behavior (`LyApplicationBehavior`).

For further information regarding application behaviors:

> 2.2 Specifying an application behavior, page 27

Session behavior - [optional] The Java class that overloads the user session's behavior (`LySessionBehavior`).

For further information regarding session behaviors:

> 2.3 Specifying a session behavior, page 28

Main Class - [fat client, optional] The Java class to be executed at application startup. Since Application Composer has a default main class (`leon.app.LyMain`), this parameter is optional.

Servlet Class - [thin client, optional] The Java class to be used as the main Servlet. Since Application Composer has a default servlet class (`leon.view.web.LyServlet`), this parameter is optional.

Root action - The action to be executed at application startup, as specified in the *Navigation tree*.

Sources - Specifies whether the application is going to be linked to the **Real Data** source of the application (i.e. the data source specified at application level), or to **Local Data** (i.e. flat files managed by Application Composer).

NOTE When it is imported, an Application Engine application may use its own data source (for example a RDBMS).

Execution console - Specifies whether the execution console should be displayed when running an Application Composer application.

Trace level - [optional] Specifies what types of warning and error messages should be sent to the standard output, or to a log file, if specified, when running an application.

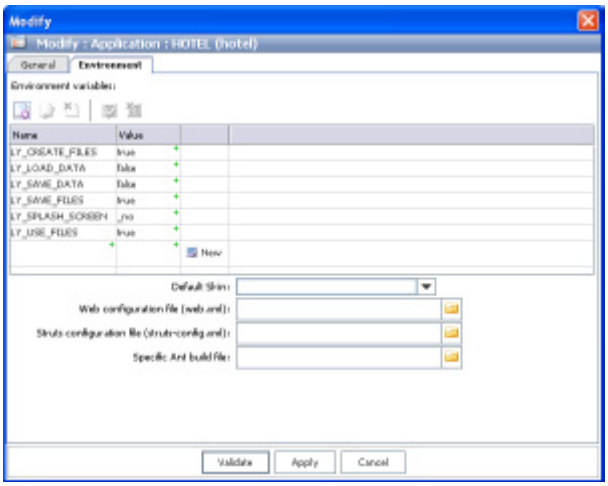

Fig 2.2 *Editing an application, Environment tab*

Environment variables - You can modify the environment in which the application will be executed (initialization parameter).

Default skin

Web configuration file

Struts configuration file

Specific Ant build file

2.2 Specifying an application behavior

Right-click the application in the class hierarchy then select **Application behavior...** from the context menu to add a specific Java class for the application's behavior (`LyApplicationBehavior`).

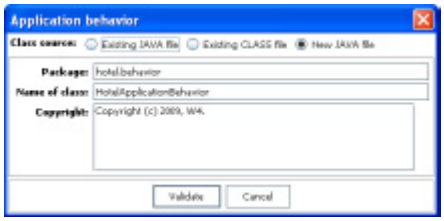

Fig 2.3 *Specifying an application behavior*

Existing JAVA file - To specify a preexisting Java file for the application behavior. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for the application behavior. The appropriate class should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new Java file for the behavior.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class package. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: *<application id>.behavior*. The behavior directory is created at the application's root level.

Name of class - The class name, by default: *IdentifierApplicationBehavior* with the application identifier's first letter in uppercase.

Copyright - The class comment. By default, the value of the LY_DEFAULT_COPYRIGHT resource set in the application_composer.ini file.

2.3 Specifying a session behavior

Right-click the application in the class hierarchy then select **Session behavior...** from the context menu to add a specific Java class for the user session's behavior (`LySessionBehavior`).

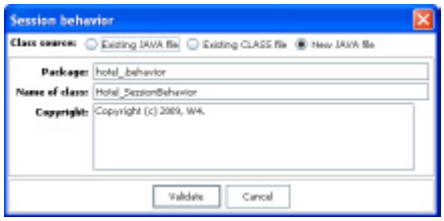

Fig 2.4 *Specifying a session behavior*

Existing JAVA file - To specify a preexisting Java file for the session behavior. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for the session behavior. The appropriate file should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new Java file for the behavior.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <*application id*>.behavior. The behavior directory is created at the application's root level.

Name of class - The class name, by default: *IdentifierSessionBehavior* with the session identifier's firt letter in uppercase.

Copyright - The class comment. By default, the value of the LY_DEFAULT_COPYRIGHT resource set in the application_composer.ini file.

2.4 Adding a specific main class

You use this feature when you want to use a specific main class at application startup (fat client).

The default class is `leon.app.LyMain`.

Right-click the application in the class hierarchy then select **Main specific class...** from the context menu.

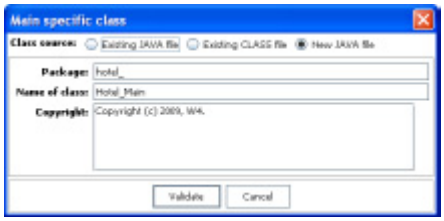

Fig 2.5 *Adding a specific main class*

Existing JAVA file - To specify a preexisting Java file for the specific main class. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for the specific main class. The appropriate class should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new file for the specific main class.

Application Composer is going to generate the Java class in the application directory and create the appropriate directories for the class package. When the file has been generated, the class is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <application id>.

Name of class - The class name, by default: *IdentifierMain* with the application identifier's first letter in uppercase.

Copyright - The class comment. By default: The value of the LY_DEFAULT_COPYRIGHT resource set in the application_composer.ini file.

2.5 Adding a specific servlet class

You use this feature when you want to add a specific servlet class to be used by the servlet container.

This class extends `leon.view.web.LyServlet`, which is the default value.

Right-click the application in the class hierarchy then select **Servlet specific class...** from the context menu.

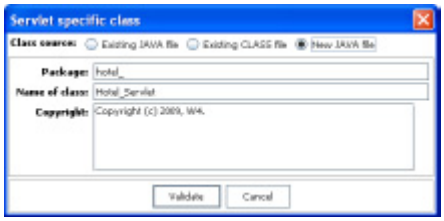

Fig 2.6 *Adding a specific servlet class*

Existing JAVA file - To specify a preexisting Java file for the specific servlet class. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for the specific servlet class. The appropriate file should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new file for the servlet class.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class package. When the file has been generated, it is opened with the text editor specified in the preferences.

Package - The class package, by default: <*application id*>.

Name of class - The class name, by default: *IdentifierServelt* with the application identifier's first letter in uppercase.

Copyright - The class comment. By default: The value of the LY_DEFAULT_COPYRIGHT resource set in the application_composer.ini file.

2.6 Exporting an application

This feature allows you to generate a standalone application, i.e. an application that does not need Application Composer to be run.

Select **File** ▸ **Export application** to perform this action.

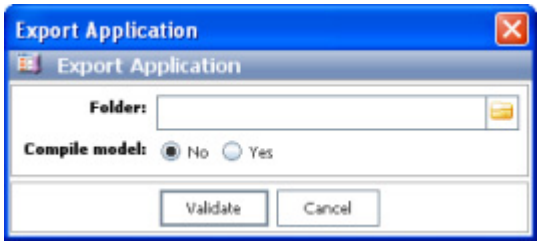

Fig 2.7 *Exporting an application*

Folder - Select the target folder.

Compile model - Specifies whether the application configuration is stored as a non-compiled XML format (by default: No) or as a compiled Java format.

An application that uses a compiled meta-model starts faster than an application that does not. Compilation dispenses with the meta-model parsing and uses the memory structure of this meta-model directly.

However, a compiled meta-model cannot be edited directly in the application, although the XML parameters can be edited.

Configuration management will be used for applications whose meta-model is not compiled.

2.7 Importing an application

This is the reverse operation of the export action: Application Composer loads an application in its memory structure so that it can be edited via the graphical interface.

Select **File** ▸ **Import application** to perform this action.

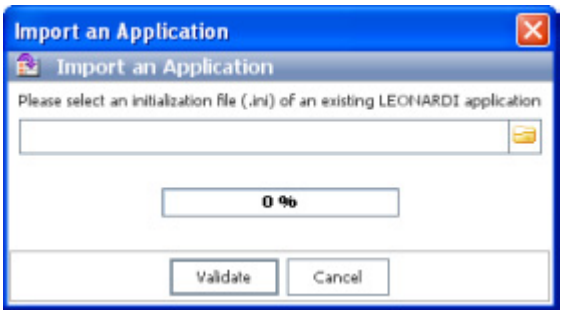

Fig 2.8 *Importing an application*

Please select... - To specify the access path to the application's .ini file.

A progress bar is displayed during the import process.

PLEASE NOTE To avoid losing string translations, images and Java classes, the project identifier should be the same as that of the application.
If not, follow these steps to restore any missing items:
- In the imported application's target directory, rename the property files by changing the project identifier for that of the application.
- Proceed in a similar way to rename the class packages.

2.8 XMI import

This feature allows you to import an UML model previously exported in the XMI format via an UML modeling tool.

Select **File** ▸ **XML import** to perform this action.

For further information:

› 11 Importing a UML model, page 188

2.9 Generating application documentation

Via the **Documentation** menu in the main tool bar you can have Application Composer automatically generate the application's documentation.

Documentation ▸ **Generate** - To generate an automated documentation based on the current application.

PLEASE NOTE You should have specified a JDK in the preferences before you start generating documentation.

Documentation ▸ **See model** - To display the reference documentation for the classes, fields, and actions that make up your application.

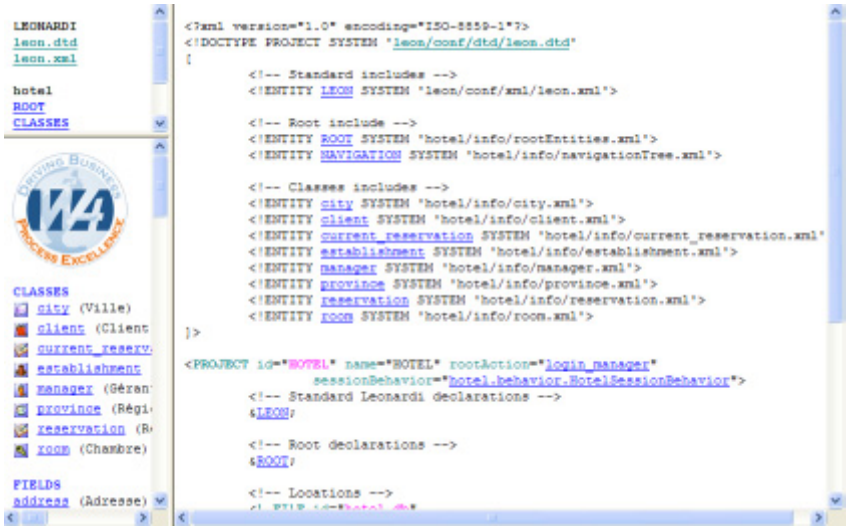

Fig 2.9 *Viewing the generated documentation (1/2)*

Documentation ▸ **See specific views** - To display the documentation for the application's specific views.

Fig 2.10 *Viewing the generated documentation (2/2)*

Documentation ▸ **See specific Java classes** - To display the application's Javadoc.

Documentation ▸ **Generation form** - To specify:

- A documentation type: Combines a format (PDF, HTML) and a target audience (developer or end user). There are three default documentation types:
  - Developer, HTML
  - End user, HTML
  - End user, PDF

You can create custom documentation types.

- The language for the generated documentation
- Whether the documentation should contain screenshots of the application views.

Documentation ▸ **See existing documentation** - To display the list of the previously generated documentation sets.

Documentation ▸ **Documentation components** - To specify which documentation components should be added relative to the documentation template.

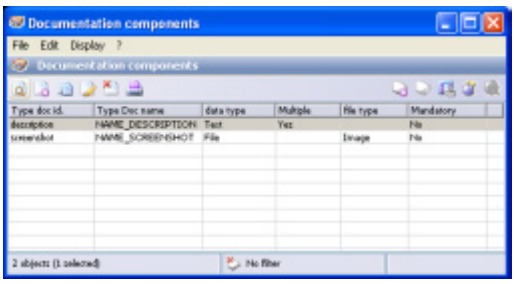

Fig 2.11 *Documentation components*

For instance you can specify a link to a specific page, a placeholder for a screenshot, etc.

Documentation ▸ **Generate WebService Javadoc**

2.10 Managing applications

You can display the list of all the available applications.

To display the **Application management** window, select **File** ▸ **Application management**.

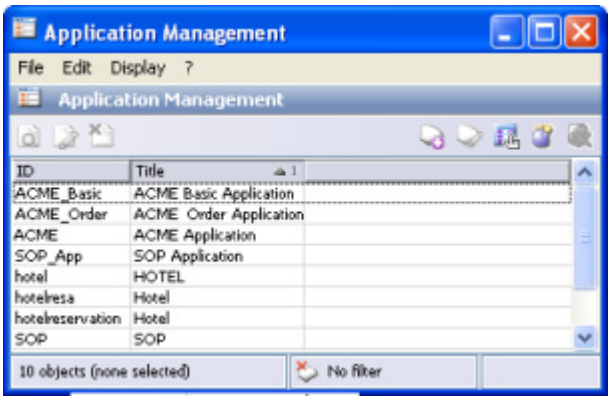

Fig 2.12 *Managing the applications*

In the *Application management* window, you can:

- **Details** - View the details of an application
- **Modify** - Edit an application
- **Delete** - Remove an application

PLEASE NOTE The *Delete* action removes the application from the disk. Make sure you make a backup of the application, e.g. via the *Export application* feature, prior to using this action.

2.11 Managing application resources

A resource is an application's execution parameter. It is used by the generic code and where appropriate by the application code when generating the views.

Resources define the strings such as action names, menu names, etc. They also define the fonts and the colors to be used when the Application Engine core builds the views.

The resources that can be managed are as follows:

- Strings
- User messages (warnings, errors, etc)
- Files used by the application
- Images
- Colors
- Fonts
- Parameters

Select **Windows** ▸ **Strings** / **Messages** / **Files** / **Images** / **Colors** / **Fonts** / **Properties** to display the resources.

Certain resources are parameters that can lead to specific behaviors (for example, the fact that you can select several lines in a list view).

Resources are a key / value pairs that can be modified centrally.

Application resources depend on the language used when the application starts. There are as many resource definitions as supported languages. Application Engine resources are defined by default in English, Spanish and French.

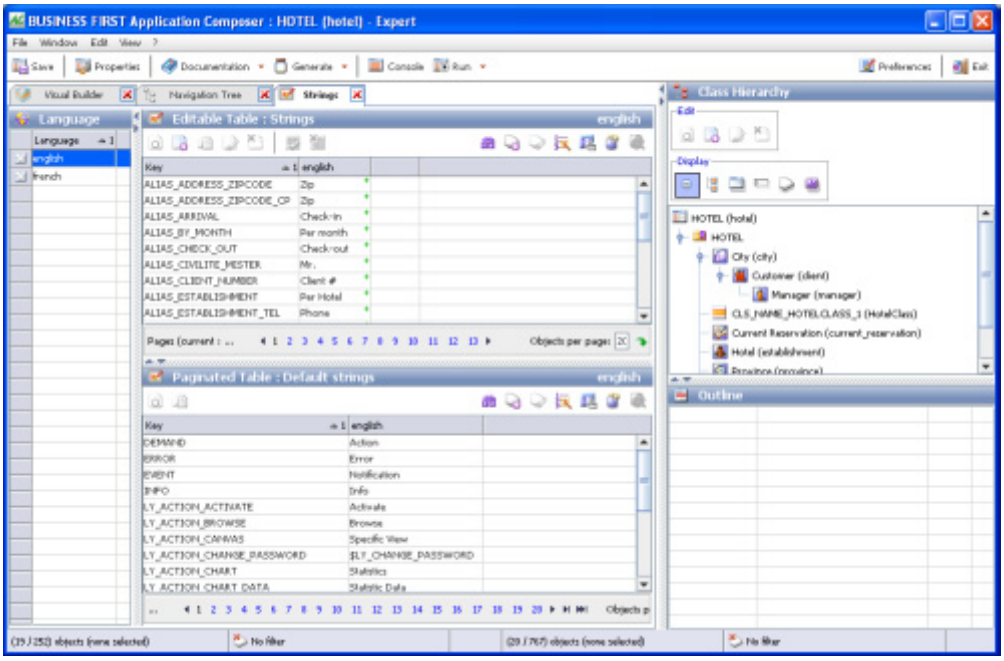

Fig 2.13 *Mnaging application resources*

For each supported language, the application-specific resources are listed in the upper area, whereas the Application Engine *generic* resources are listed in the lower area.

NOTE The names of the default Application Engine resource keys start with LY_

To edit the value of an Application Engine generic resource, e.g. to change a message, a translated string, a color, etc, you just need to create a specific resource with the same key identifier.

2.12 Generating the final application

Via the **Generate** menu you can specify a target format for the application to be generated.

Generate ▸ **JAR file** - Generates the application's Java archive.

Generate ▸ **WAR - Application** - Generates the war file for the application's web deployment.

Generate ▸ **WAR - Web Services** - Generates the application as a web service.

Generate ▸ **Deploy application to Extension Bus** - Generates the application's Java archive to the *<ExtensionBus_Home>/<InstanceName>/applications* folder.

NOTE **When is deployment to W4 Extension Bus required?**
Prior to generating an application in Application Composer you may need to deploy it to W4 Extension Bus. This is required when your process includes business data that need to be evaluated, e.g. when business data is used as part of link conditions or in variable mapping.

Generate ▸ **Generate database script** - Generates a file with all the SQL queries required to create the database tables and columns required by the application.

> 2.12.1 Generate database script, page 37

Generate ▸ **Generate / alter database** - Creates the application's database tables.

> 2.12.2 Generate / alter database, page 38

Generate ▸ **Generate Eclipse plugin** - To generate an Application Engine application as an Eclipse plugin.

> 2.12.3 Generate Eclipse plugin, page 40

2.12.1 Generate database script

This feature generates a file with all the SQL queries required to create the database tables and columns for the application.

Select **Generate** ▸ **Generate database script** to perform this action.

The SQL generator uses the configuration of the physical link that is defined for the classes and attributes.

> 4.14 Setting physical binding for a class, page 64

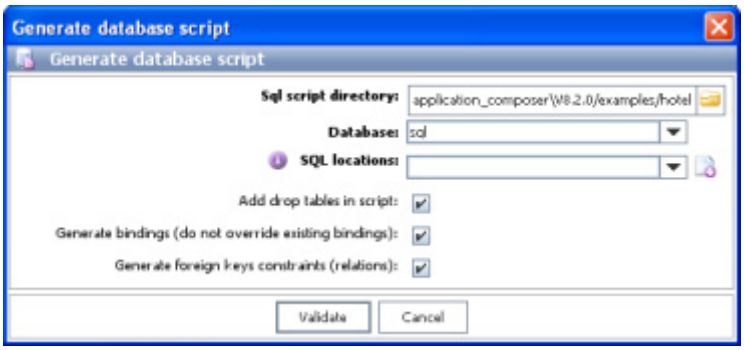

Fig 2.14 *Generating the database script*

SQL script directory - The target directory for the script.

Database - Select the appropriate database type. Selecting the right value ensures the generated SQL queries are compatible with the database in use.

SQL location - Select the appropriate data provider.

To create a data provider, click New RDBMS data.

For further information regarding data provider creation:
> 9.1 Adding a data source, page 170

Add drop tables in script

Generate bindings

Generate foreign keys constraints

2.12.2 Generate / alter database

This feature not only generates the database creation script but also creates the application's database tables.

Select **Generate** ▸ **Generate / alter database** to perform this action.

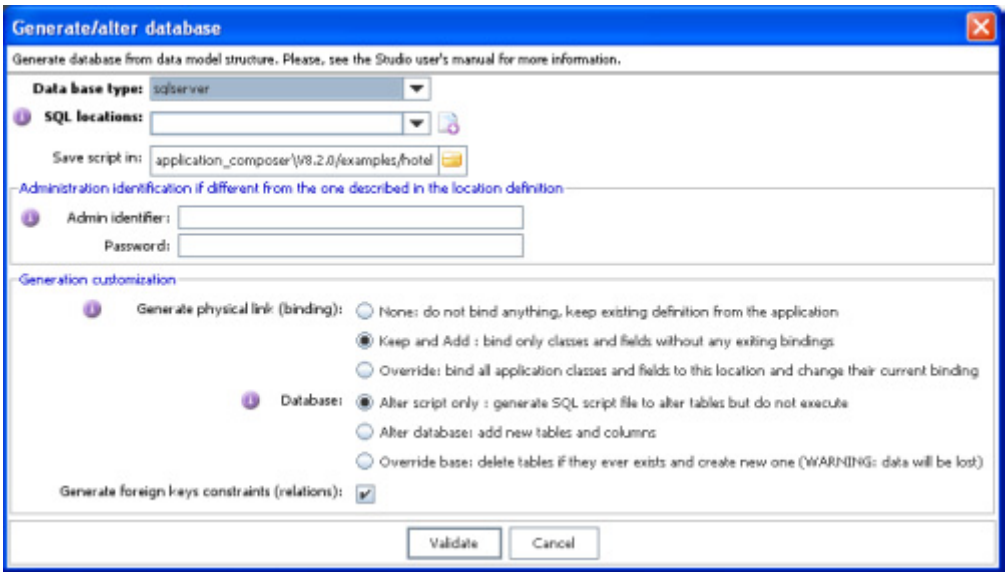

Fig 2.15 *Generating the database*

Database type - Select the appropriate database type. Selecting the right value ensures the generated SQL queries are compatible with the database in use.

SQL location - Select the appropriate *location* for the database.

A *location* specifies the required data for the connection such as driver, connection URL, login and password, etc.

To create a *location*, click New RDBMS data.

For further information regarding *location* creation:

> 9.1 Adding a data source, page 170

Save script in - The target directory for the generated script. If empty, the script is not saved.

Admin identifier - **Password** - To override the administration login specified for the currently selected location.

Binding generation to the location

- **None** - No binding is performed. Select this option if bindings have already been defined. Please note: Unbound items are not added to the database.
- **Keep and add** - Binds the application classes and fields with no binding to the current location. Preexisting bindings are preserved.
- **Override** - All application classes and fields will be bound to the current location. Preexisting bindings are overwriten.

Database

- **Alter script only**
- **Alter database** - Creates only the new tables and preserves any preexisting tables.
- **Override base** - Removes preexisting database and recreates the database from scratch.

PLEASE NOTE Should you select the *Override base* option, all your data, including any manual changes (triggers, etc) will be overriden.

Generate foreign keys constaints

2.12.3 Generate Eclipse plugin

Application Engine applications can be executed as Eclipse plugins.

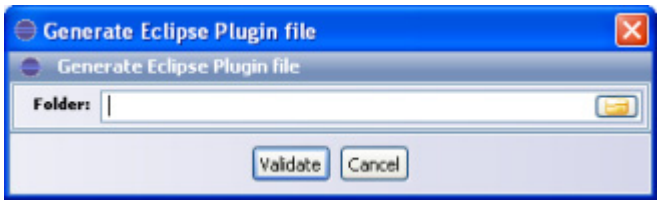

Fig 2.16 *Generating an application as an Eclipse plugin*

Folder - Select the target location to which the plugin zip file will be generated.

2.13 Running the final application

The **Run** menu in the main tool bar allows you to run the application using one of the available viewers (depending on your Application Engine license, some viewers may be not available).

Run ▸ **Display SWING** - To run the application in fat client mode, based on the SWING library.

Run ▸ **Display Web** - Runs the application in thin client mode.

Run ▸ **Display Web (JAR update)**

Run ▸ **Choose a skin** - To select a skin from the standard skins shipped with Application Composer. The available skins in the list are those included in the application's classpath.

Run ▸ **Suspend application** - To stop the currently running application. This option is disabled when no application is running. Please note: Any data that may have been created while the application was running will not be lost.

Run ▸ **Run Ant Build**

NOTE The available options depend on which viewers are installed with Application Engine (See your License terms).

Working with projects

The project is the second-level node in the class hierarchy, just after the application. It is automatically created at the same time as the application and its name is the same as the application ID.

Projects store application classes and/or sub-projects. An application includes one project and may have multiple sub-projects.

Projects allow you to better organize data and have no impact on the application execution.

In this section we will cover the following topics:

- How to perform basic project-related actions:
  - 3.1 Creating a sub-project, page 42
  - 3.2 Viewing details of a project or sub-project, page 43
  - 3.3 Editing a project, page 44
  - 3.8 Saving a project, page 51
- How to create filters, sort criteria and comments for a project:
  - 3.5 Creating filters for a project, page 46
  - 3.6 Creating sort modes for a project, page 49
  - 3.7 Setting comments for a project or sub-project, page 50

3.1 Creating a sub-project

TO PERFORM THIS STEP

1. Right-click the project or the relevant sub-project in the class hierarchy.
2. Select **Create Sub-project...** from the context menu.

The *Create Sub-Project* window is displayed:

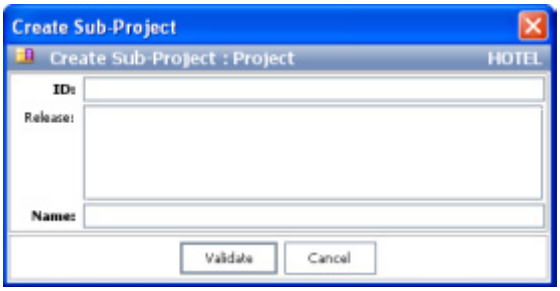

Fig 3.1 *Creating a sub-project*

3 Set the fields:

ID - The sub-project identifier.

Release - [Optional] This field can be used to specify the sub-project version.

Name - The sub-project name.

4 Click **Validate**.

3.2 Viewing details of a project or sub-project

TO PERFORM THIS STEP

1 Right-click the project or the relevant sub-project in the class hierarchy.

2 Select **Details** from the context menu.

3 The *Details : Project* window is displayed:

Fig 3.2 *Viewing the details of a project*

Root action - The project's root action is the first action that is invoked when the application starts.

For further information regarding root actions:

> Specifying the root action, page 145

Project models

3.3 Editing a project

TO PERFORM THIS STEP

1 Right-click the relevant project in the class hierarchy.

2 Select **Modify...** from the context menu.

The *Modify : Project* window is displayed:

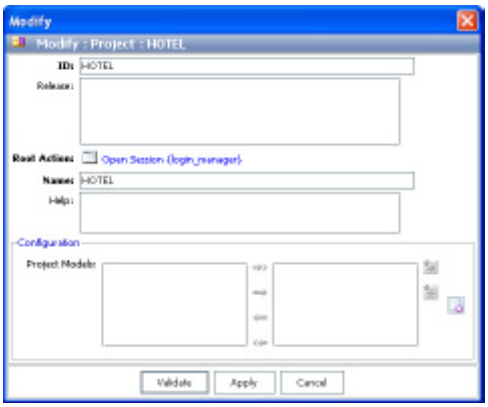

Fig 3.3 *Editing a project*

3 The following fields can be edited:

- ID
- Release
- Root action
- Name
- Help

For further information regarding root actions:

> Specifying the root action, page 145

4 Add one or more data models to the project:

4.1 In the *Modify : Project* window, click **New** to the right of the *Configuration* area.

The *New : Data model* window is displayed:

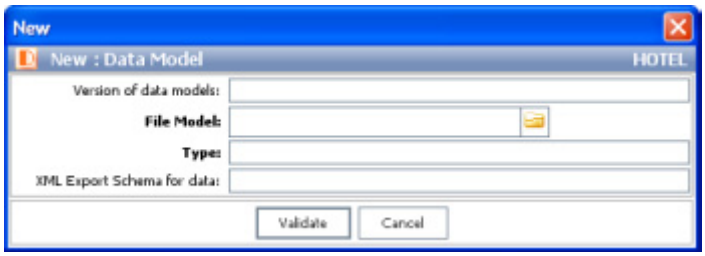

Fig 3.4 *Adding a data model to the project*

4.2 Set the following fields:

Version of the data models - [Optional] This field is used to manage compatibility models.

File model - The XML file with the model declarations.

Type - Either a data model or another model type.

XML export schema for data

4.3 Repeat these steps to add as many data models as needed.

4.4 Click **Validate**.

5 Back in the *Modify : Project* window, click **Validate**.

3.4 Editing a sub-project

TO PERFORM THIS STEP

1 Right-click the relevant sub-project in the class hierarchy.

2 Select **Modify...** from the context menu.

3 The following fields can be edited:

- ID
- Release
- Name
- Help

4 Click **Validate**.

3.5 Creating filters for a project

You can create simple and extended filters for a project.

A simple filter is an expression made up of an attribute, a filter condition, a value, and sometimes a *modifier*.

An extended filter is a set of expressions and/or *relation filters* and/or *predefined criteria* linked together by AND/ OR operators, e.g. *Reservation state = 'pending' AND Departure date > 03/04/2010*.

For further information regarding modifiers:
> Modifiers, page 46

For further information regarding relation filters:
> Relation filters, page 46

For further information regarding predefined criteria:
> Predefined criteria, page 46

In this section:

Modifiers

Not - Specifies the opposite of the selected value.

For example, specifying:
[State] [Equals to] [Cancelled]
would match all the workcases with a state other than *Cancelled*.

Case sensitive - Specifies an exact case-sensitive match.

Relation filters

Predefined criteria

3.5.1 Creating a simple project filter

TO PERFORM THIS STEP

1. Right-click the relevant project in the class hierarchy.
2. Select **Project filters...** from the context menu.

   The *Project : Filters* window is displayed:

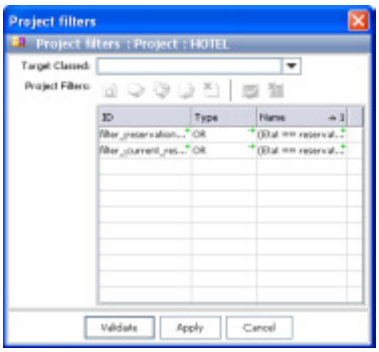

Fig 3.5 *Creating filters for a project*

3. Select the target class in the drop-down list.
4. Click **Create a simple filter...** in the *Project filters* area.

   The *Create a simple filter* window is displayed:

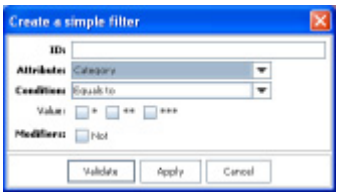

Fig 3.6 *Creating a simple filter*

5. Set the fields:

   ID

   Attribute - Select the appropriate attribute.

   Condition - Select the appropriate condition.
   The options available in this drop-down list depend on the type of the selected attribute.

   Value - Type / Select the appropriate value.
   The *Value* field depends on the type of the selected attribute.

   Modifiers - Where appropriate, select a modifier.
6. Click **Validate**.
7. Back in the *Project Filters* window, click **Validate**.

3.5.2 Creating an extended project filter

TO PERFORM THIS STEP

1. Right-click the relevant project in the class hierarchy.
2. Select **Project filters...** from the context menu.

   The *Project : Filters* window is displayed.
3. Select the target class in the drop-down list.
4. Click **Create an extended filter...** in the *Project filters* area.

   The *Create an extended filter* window is displayed:

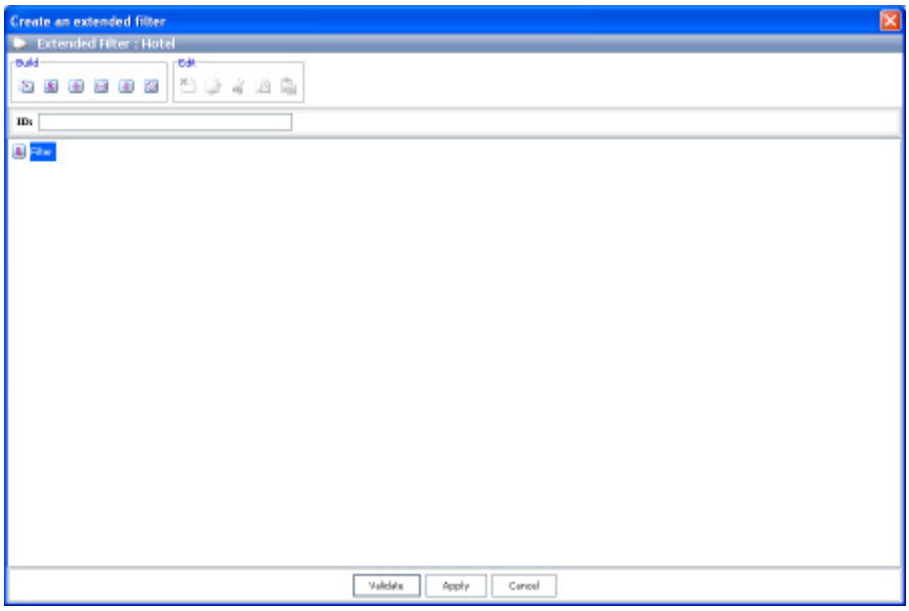

Fig 3.7 *Creating an extended project filter*

5. Set up the operator.

   By default the AND filter is selected. Where appropriate, click **Change type** to switch to the OR operator.
6. If your extended filter has more than one level, add as many AND or OR operators as required by clicking **Add AND operator...** or **Add OR operator...** in the *Build* toolbar.
7. Below the operator(s), either:
   - Add an elementary filtering expression
   - Create a new filter on the objects referenced by the relationship
   - Create a filter criterion implemented by an external Java class. This class will manage the filter and will need to implement the `leon.info.LyFilterElement` interface. The id attribute allows you to identify the filter criterion in order to reference it.
8. Click **Validate**.
9. Back in the *Project Filters* window, click **Validate**.

3.6 Creating sort modes for a project

TO PERFORM THIS STEP

1 Right-click the relevant project in the class hierarchy.

2 Select **Project sorts...** from the context menu.

The *Project : Sorts* window is displayed:

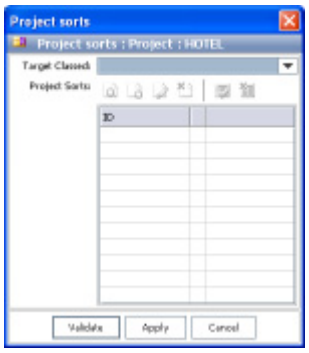

Fig 3.8 *Creating sort criteria for a project (1/3)*

3 Select the target class in the drop-down list.

4 Click **New** in the *Project Sorts* area.

The *New : Sort* window is displayed:

Fig 3.9 *Creating sort criteria for a project (2/3)*

5 In the *Criteria* area, click **New** to define a sort criterion.

The *New : Sort criterion* window is displayed:

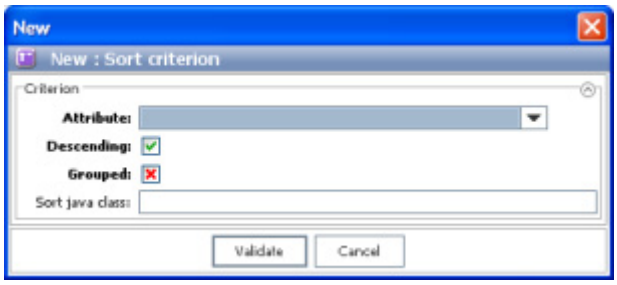

Fig 3.10 *Creating sort criteria for a project (3/3)*

6 Set the fields:

Attribute - Select the target attribute.

Descending

Grouped

Sort Java class - [Optional] Specify the Java class used to perform the sort mode.

7 Click **Validate**.

8 Back in the *New : Sort* window click **Validate**.

3.7 Setting comments for a project or sub-project

TO PERFORM THIS STEP

1 Right-click the relevant project or sub-project in the class hierarchy.

2 Select **Set comments...** from the context menu.

The *Set comments : Project* window is displayed:

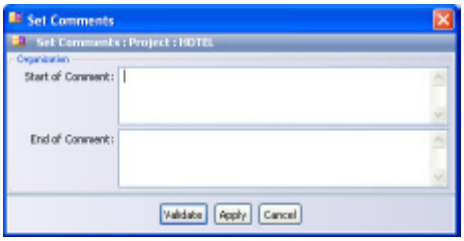

Fig 3.11 *Setting comments for a project*

3 Set the fields:

Start of comment

End of comment

NOTE Comments are added to the application's description XML file and also to the generated documentation.

4 Click **Validate**.

3.8 Saving a project

The **Save** button in the main tool bar saves the current project.

It is recommended that you save your work at regular intervals in order to avoid losing information should Application Composer shut down unexpectedly, or should there be a system failure.

When opening an application in Application Composer, the latest data is loaded. The application's previous version is still available in the save directory (.bak files).

When closing Application Composer, users are prompted to save the data.

Working with application classes

In this chapter we will cover the following class-related topics:

- How to perform basic class-related actions:

- How to implement advanced class-related features:

4.1 Creating a class

TO PERFORM THIS STEP

1 Right-click the relevant project in the class hierarchy then select **Create class** from the context menu.

[Alternatively] Right-click any class in the class hierarchy then select **Create class** from the context menu.

The *New : Class* window is displayed:

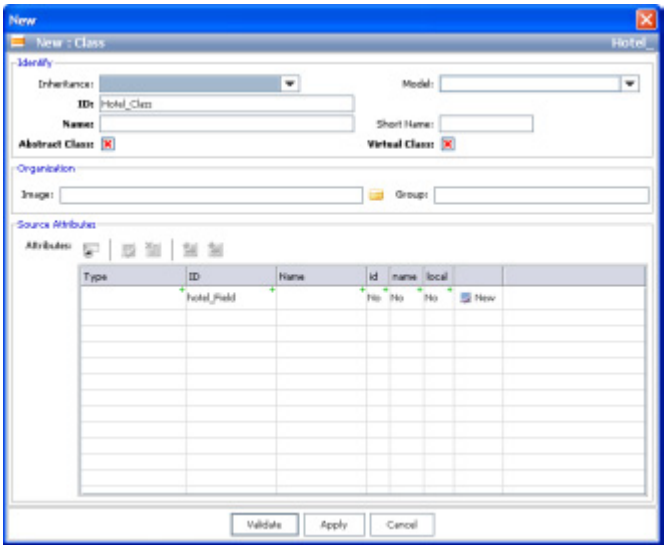

Fig 4.1 *Creating a new class*

2 Set the fields:

Inheritance - The reference of the class the current class extends, i.e. its super-class. A class inherits the fields and actions in its super-class.

Model - The reference of the class used as model to build the new class. This attribute allows you to describe a new class by extending an existing class.

ID - The class identifier. It must be unique.

Name - The class name.

Create a new string

Choose a string

Short name - The class short name, used when limited space is available for displaying the full name.

Abstract class - To specify whether the class is abstract. An abstract class cannot be instantiated.

Virtual class - To specify whether the class is virtual. A virtual class is used to modify the presentation of an existing class that is specified by the extends attribute, so that it can be used in a particular action.

Image - The icon that represents the class.

Group - The group to which the class belongs.

3 To create an attribute for the class:

3.1 Set the fields in the *Attributes* tab:

Type - Select the appropriate attribute type.

ID - Review the value and change if appropriate.

Name - Specify the attribute name.

Create a new string

Choose a string

id - **name** - **optional** - **local** - Set the marks as appropriate.

Attribute shortcut - To create a reference to an existing attribute.

Validate current changes - To validate the changes made to an attribute.

Cancel current changes - To discard the changes made to an attribute.

Move up - To move an attribute up in the list.

Move down - To move an attribute down in the list.

3.2 Click **New** at the end of the row.

4 Click **Validate**.

For further information relating to marks:

> 5.30 Setting attribute marks, page 120

For further information relating to attributes:

> 5 Working with attributes, page 76

4.2 Viewing details of a class

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy.

2 Select the **Details** option from the context menu.

The *Details : Class* window is displayed:

Fig 4.2 *Viewing the details of a class*

In this form you can:

- Set the default sort mode:
  > 4.7 Sorting objects in a class, page 58
- Specify a cache policy:
  > 4.8 Specifying a cache policy for a class, page 58
- Specify help files:
  > 4.9 Specifying help files for the class, page 59
- Generate the class interface:
  > 4.12 Generating the interface class, page 62
- Specify the class behavior:
  > 4.13 Specifying the class behavior, page 63
- Set physical binding:
  > 4.14 Setting physical binding for a class, page 64
- Set rules:
  > 4.16 Setting class rules, page 70

For further information regarding the fields in this form:

> 4.1 Creating a class, page 53

4.3 Creating a new view for a class

You can create an application class that is a sub-set of a class. There is a logical link between both classes and data is not duplicated.

The target class displays the same data as the original application class but within a different view (form, tables, etc).

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Create a new view for this class** from the context menu.

The *Create a new view for this class* window is displayed.

2 Set the fields.

For further information regarding the fields in this form:

> 4.1 Creating a class, page 53

3 Click **Validate**.

The new view mays contain only a part of the fields, or it my have a different layout, filter, etc.

4.4 Editing a class

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Modify** from the context menu.

The *Modify : Class* window is displayed:

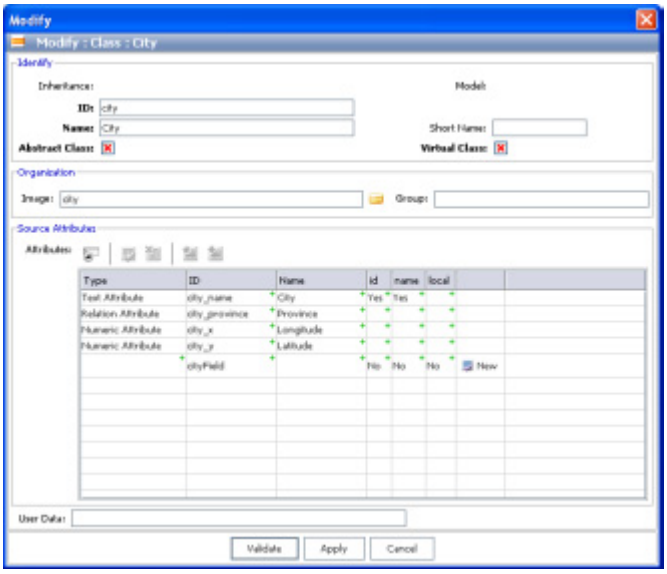

Fig 4.3 *Modifying a class*

2 Set the fields.

User data - [Optional] Specifies data that is purely applicative.

For further information on the fields in this form:

> 4.1 Creating a class, page 53

3 Click **Validate**.

4.5 Removing a class

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Delete** from the context menu.

NOTE You cannot delete a class that has dependencies.

4.6 Specifying class extends

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Specify extends** from the context menu.

The *Specify extends* window is displayed:

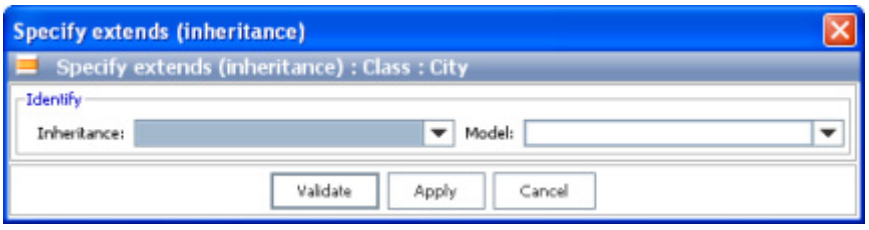

Fig 4.4 *Specifying class extends*

2 Set the fields:

Inheritance - Specify the class that is the parent of the current class.

Model

3 Click **Validate**.

4.7 Sorting objects in a class

You can specify a default sort mode that is applied if no sort mode has been selected by the user in the list views (tables).

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Default sort...** from the context menu.

The *Default sort* window is displayed:

Fig 4.5 *Sorting objects in a class*

2 Set the fields:

Attribute - Select the appropriate field for the sort criterion.

Descending - When the option is selected, the sort is performed in decreasing order, otherwise in increasing order.

Grouped

Sort Java class - The full name of the Java class used to perform the sort (comparison between two values). This class should implement the `leon.info.infointerface.LyComparatorInterface` interface.

3 Click **Validate**.

4.8 Specifying a cache policy for a class

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Cache management...** from the context menu.

The *Cache management* window is displayed:

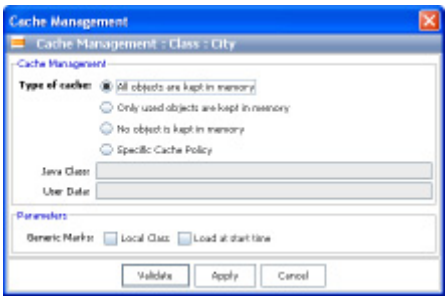

Fig 4.6 *Specifying a cache policy*

2 Set the fields:

Type of cache

- **All the objects are kept in memory** (FullCache) - An object is loaded only if it is unknown, and it is never unloaded. This cache is efficient provided data volume is not too large.
- **Only used objects are kept in memory** (Auto cache) - Only the objects used by the application code or displayed in the views are stored in the memory. When closing the view or when releasing the object list, the data is released from memory.
- **No object is kept in memory** (No Cache) - The data provider is accessed whenever necessary.
- **Specific cache policy** - To specify a preexisting class for the cache. The appropriate class should be selected via the **Java class** field. The class should extend the `leon.data.LyCache` class.

Java class - The class name. Change as appropriate.

User data - This field can be used to configure a specific cache (number of managed objects, refresh delay, name of local cache, etc).

Generic marks

- **Local class** - The class is not linked to the data provider (*local* means that this class does not exist in the database). Therefore, the cache is always a full Cache. There is no call to the physical link.
- **Load at start time** - Specifies whether data is loaded as soon as the application starts, or the first time the user invokes it.

3 Click **Validate**.

4.9 Specifying help files for the class

An icon such as is displayed next to the fields for which help files have been specified. Clicking this icon displays the specified HTML document in the default Web browser.

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Define help files...** from the context menu.

The *Define help files* window is displayed:

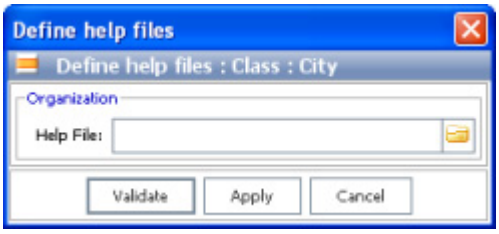

Fig 4.7 *Specifying help files for the class*

2 Set the fields:

Help file

3 Click **Validate**.

4.10 Adding specific marks for a class

Specific marks are not used by the Application Engine code but can be used in the specific Java code of the application.

TO CREATE A SPECIFIC MARK

1 Right-click the relevant class in the class hierarchy then select **Specific marks...** from the context menu.

The *Specific marks* window is displayed:

Fig 4.8 *Specifying specific marks for the class*

2 Type the name of the mark.

3 Click **Add**.

4 Click **Validate**.

TO APPLY A SPECIFIC MARK

1 Right-click the relevant class in the class hierarchy then select **Specific marks...** from the context menu.

The *Specific marks* window is displayed.

2 Select the check box of the appropriate specific mark.

3 Click **Validate**.

4.11 Specifying application data

You can have the class manage data intended for the specific application code.

Specific data is specified as key/value pairs and, just as with specific marks, using specific data is up to application developers.

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Application data...** from the context menu.

The *Application data* window is displayed:

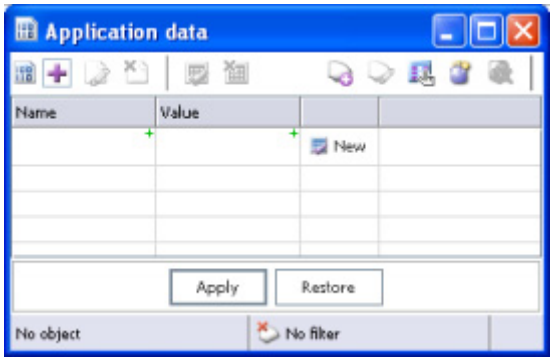

Fig 4.9 *Specifying application data*

2 Click **New...**

The *New : Data* window is displayed:

3 Set the fields:

Name - Name of the data item.

Value - Value of the data item.

4 Click **Validate**.

5 Back in the *Application data* window, click **Apply**.

4.12 Generating the interface class

You can generate an optional Java class to have fewer abstract accessors to the application data.

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Interface class...** from the context menu.

The *Interface class* window is displayed:

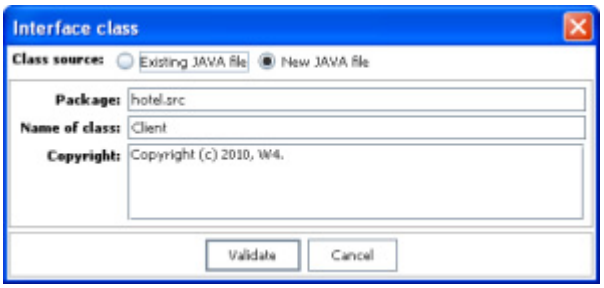

Fig 4.10 *Generating the class interface*

2 Set the fields:

Existing Java file - To specify a preexisting existing Java class. The appropriate file should be selected via the **Existing implementation** field.

New Java file - To have Application Composer create a new Java class.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <application id>.src. The src directory will be created to the root of the application.

Name of class - The class name, by default: *Identifier* with the class identifier's first letter in uppercase.

Copyright - The class comment, with the value of the LY_DEFAULT_COPYRIGHT resource set in the application_composer.ini file as the default value.

3 Click **Validate**.

4.13 Specifying the class behavior

The class behavior is a Java class that extends `leon.app.behavior.LyClassBehavior`. In this class you can specify specific behaviors for the various forms (create, edit, read-only, etc).

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Class behavior...** from the context menu.

The *Class behavior* window is displayed:

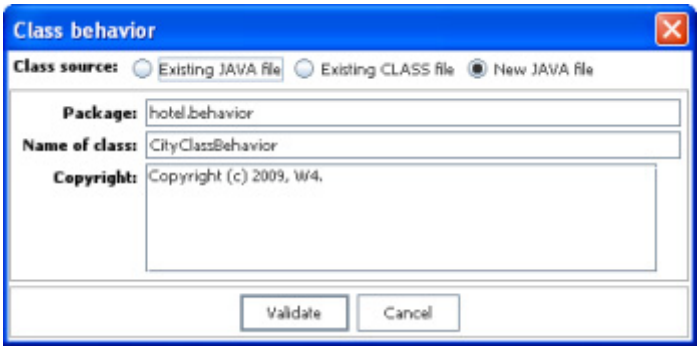

Fig 4.11 *Specifying the class behavior*

2 Set the fields:

Existing Java file - To specify a preexisting Java file. The appropriate file should be selected via the **Existing implementation** field.

Existing Class file - To specify a preexisting class. The appropriate file should be selected via the **Class** field.

New Java file - To have Application Composer create a new Java file.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. However you can change some parameters before generation via the next fields. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: *<application id>.behavior*. The behavior directory will be created to the root of the application.

Name of class - The class name, by default: *IdentifierClassBehavior*, with the class identifier's first letter in uppercase.

Copyright - The class comment, with the value of the LY_DEFAULT_COPYRIGHT resource set in the application_composer.ini file as the default value.

3 Click **Validate**.

4.14 Setting physical binding for a class

This section describes how to configure the link between the application class and the data provider (RDBMS, LDAP active directory, etc).

NOTE The data provider must be previously configured for the application.

To configure the data provider:

> 9 Managing data sources, page 170

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Set physical binding...** from the context menu.

The *Set physical binding* window is displayed:

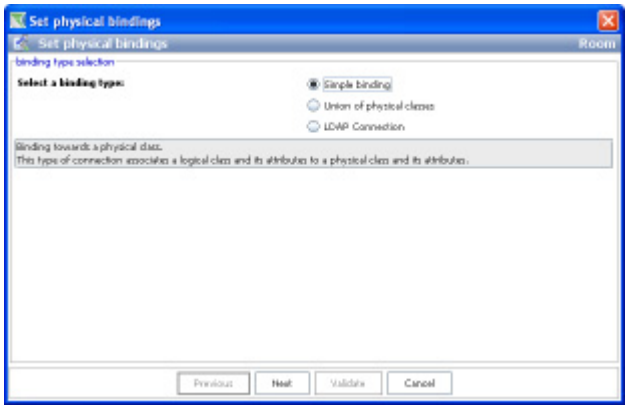

Fig 4.12 *Modifying the link to the physical layer of the class*

2 Select the appropriate binding type :

- **Simple binding** - The most common binding, to be used e.g. when the application class is a database table.
- **Union of physical classes** - To be used e.g. when the application class is associated with several database tables.
- **LDAP connection** - Link to an LDAP active directory.

3 The remaining steps are dependent upon the previously selected binding type.

- **Simple binding**

> Creating simple bindings, page 65

- **Union of physical classes**

> Creating union bindings, page 66

- **LDAP connection**

> Creating LDAP bindings, page 67

4 Click **Validate**.

Creating simple bindings

TO PERFORM THIS STEP

1 Set the fields in the first screen of the simple binding wizard:

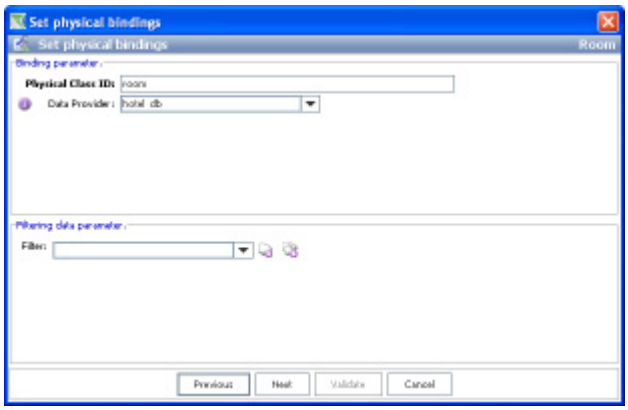

Fig 4.13 *Modifying the link to the physical layer of the class, simple binding (1/2)*

Physical class ID - Enter the identifier of the class in the physical layer. This value depends on the data provider type (e.g. table in a relational database, etc).

Data provider - Select the previously created data source.

For a RDBMS, this is the name of the database table.

Filter - Select a filter to limit how much data is loaded.

Alternatively, you can create either a simple, or an extended filter.

2 Click **Next**.

The second screen of the simple binding wizard is displayed:

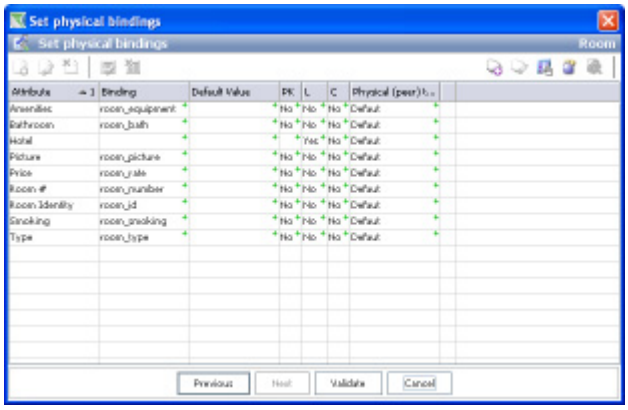

Fig 4.14 *Modifying the link to the physical layer of the class, simple binding (2/2)*

This screen lists the application class fields as a table, in which each row represents a class attribute. The table columns are as follows:

3 Set the fields:

Attribute - The attribute identifier.

Binding - The modifiable values correspond to the physical value of the field.

Default value - The default value, which can be changed, specified for the field.

PK - Specifies whether the attribute is the primary key or the unique identifier for its physical representation, in the data provider.

L - To specify whether the attribute is local (i.e. if it does not have any physical representation in a data provider).

C - To specify whether the field data should be loaded automatically (field is loaded upon load request if the fields to be loaded are not specified).

Physical type - There are three options:

- **Default**
- **BLOB**
- **CLOB**

4 Click **Validate**.

Creating union bindings

Using bindings of physical classes allows you to join two application classes together. Each class has its own physical link (simple or binding), which allows you to manage several data providers.

TO PERFORM THIS STEP

1 Set the fields in the first screen of the union binding wizard:

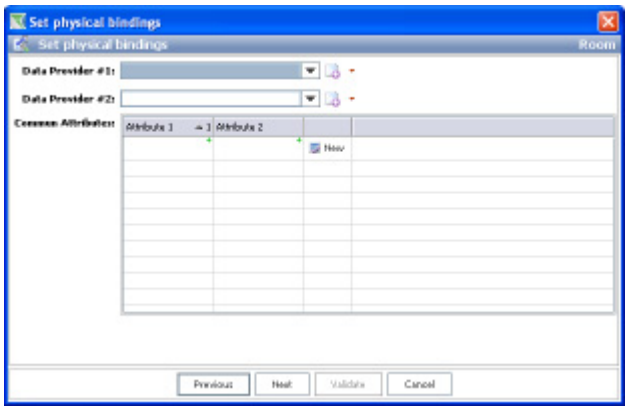

Fig 4.15 *Modifying the link to the physical layer of the class, union of physical classes(1/2)*

Data provider # 1, **Data provider # 2** - To select the data providers. They are either a simple binding, or a binding of physical classes.

Alternatively you can create either a simple binding, or a union binding.

Common attributes - Binding between classes is performed by specifying one or more equivalent fields for both classes.

2 Click **Next**.

The second screen of the union binding wizard is displayed:

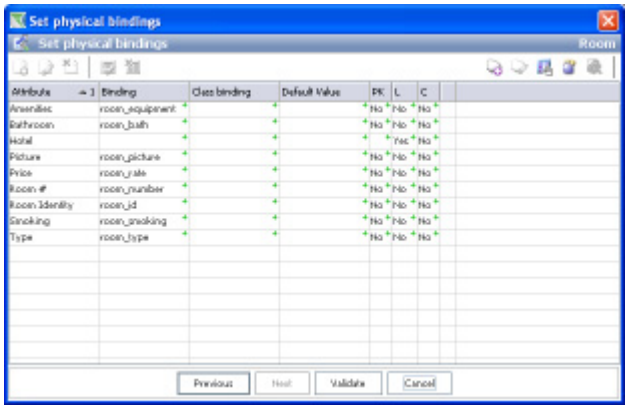

Fig 4.16 *Modifying the link to the physical layer of the class, union of physical classes(2/2)*

This screen lists the application class fields as a table, in which each row represents a class attribute. The table columns are as follows:

3 Set the fields:

Attribute - The attribute identifier.

Binding - The modifiable values correspond to the physical value of the field.

Class binding - Name of the physical class to which the physical field belongs.

Default value - The default value, which can be changed, specified for the field.

PK - Specifies whether the attribute is the primary key or the unique identifier for its physical representation, in the data provider.

L - To specify whether the attribute is local (i.e. if it does not have any physical representation in a data provider).

C - To specify whether the field data should be loaded automatically (field is loaded upon load request if the fields to be loaded are not specified).

4 Click **Validate**.

Creating LDAP bindings

This is a soecial case of simple binding to be used when the data source is an LDAP active directory.

TO PERFORM THIS STEP

1 Set the fields in the first screen of the LDAP binding wizard:

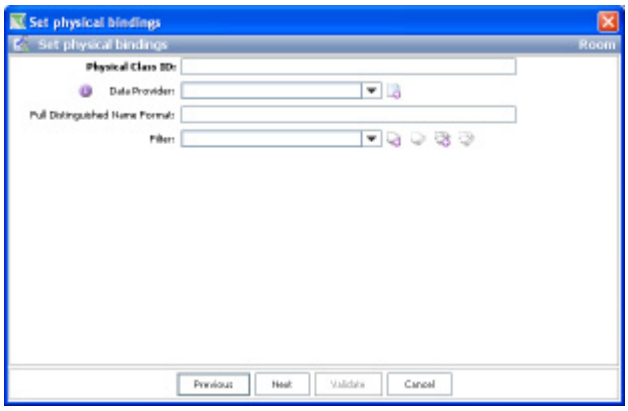

Fig 4.17 *Modifying the link to the physical layer of the class, LDAP connection*

Physical class ID - The identifier of the class in the LDAP active directory.

Data provider - Select the appropriate LDAP active directory.

Alternatively you can specify a new LDAP directory.

Full distinguished name format - The *full distinguished name* for the class (without the FDN of the base object declared in the connector).

Filter - [Optional] Select a filter.

Alternatively you can create either a simple filter, or an extended filter.

2 Click **Next**.

The second screen of the LDAP binding wizard is displayed.

3 Set the fields:

Attribute - The attribute identifier.

Binding - The modifiable values correspond to the physical value of the field.

PK - Specifies whether the attribute is the primary key or the unique identifier for its physical representation, in the data provider.

L - To specify whether the attribute is local (i.e. if it does not have any physical representation in a data provider).

C - To specify whether the field data should be loaded automatically (field is loaded upon load request if the fields to be loaded are not specified).

4 Click **Validate**.

4.15 Setting class controls

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Set controls...** from the context menu.

The *Set controls* window is displayed:

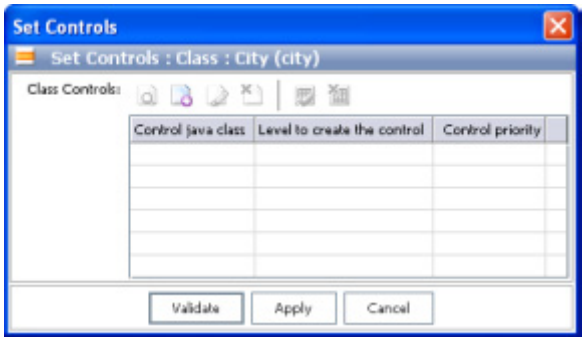

Fig 4.18 *Modifying the controls for the class (1/4)*

2 Click **New**.

The *New : Control* window is displayed:

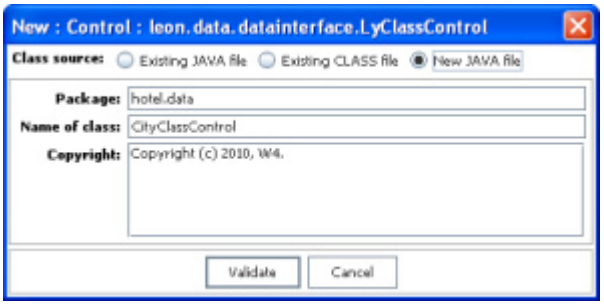

Fig 4.19 *Modifying the controls for the class (3/4)*

3 Set the fields:

Existing JAVA file - To specify a preexisting Java file for the class control. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for the control. The appropriate class should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new file for the control.

Application Composer is going to generate the Java class in the application directory and create the appropriate directories for the class package. When the file has been generated, the class is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <*application id*>.*data*. The data directory will be created to the root of the application.

Name of the class - The class name, by default: *IdentifierFormControl* with the class identifier's first letter in uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the `studio_composer.ini` file as the default value.

4 Click **Validate**.

5 Back in the *Set controls* window, click **Validate**.

4.16 Setting class rules

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Set rules...** from the context menu.

The *Set rules* window is displayed:

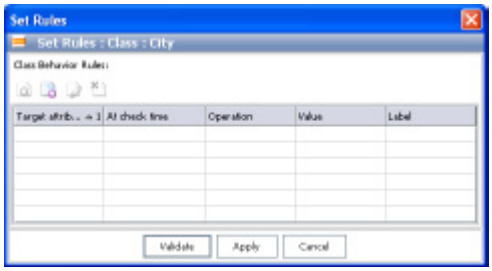

Fig 4.20 *Setting class rules (1/2)*

2 Click **New**.

The *New : Rules* window is displayed:

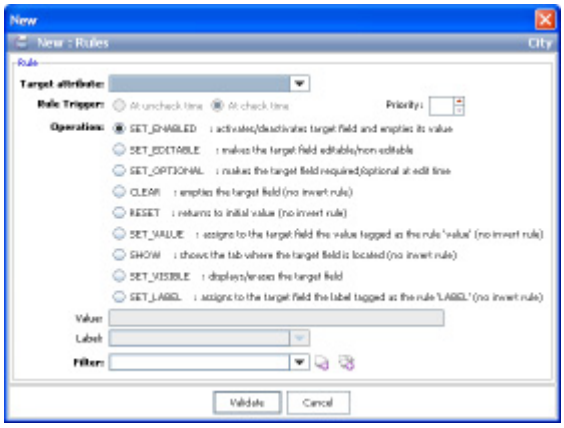

Fig 4.21 *Setting class rules (2/2)*

3 Set the fields:

Target attribute - Specifies the field to which the rule applies.

Role trigger

Priority - Priority level of the rules (relative to other rules).

Operation

- **SET_ENABLED** - Enables/disables the field and clears its value.
- **SET_EDITABLE** - Makes the field editable/not editable.
- **SET_OPTIONAL** - Makes the field optional/mandatory.
- **CLEAR** - Empties the field (no reverse rule).
- **RESET** - Returns the field to its original value (no reverse rule).
- **SET_VALUE** - Assigns to the field the value specified for the **Value** field (no reverse rule).
- **SHOW** - Displays the field to the user, i.e. displays the tab in which the field is located (no reverse rule).
- **SET_VISIBLE** - Displays/Hides the field.
- **SET_LABEL** - Assigns to the field the label that is specified for the **Label** field (no reverse rule).

Value - The value to be specified with the SET_VALUE operation.

Label - The value to be specified with the SET_LABEL operation.

Filter - To set a filter for the control data.

Alternatively, you can create either a simple, or an extended filter.

4 Click **Validate**.

4.17 Setting class labels

A dynamic label is a label calculated at runtime. It contains a value that is a String with both fixed and variable parts, as well as parameters allowing, at runtime, to replace these variable parts by data of dynamic nature that depends on the use context. This label can change when the data to which it applies, is modified.

TO PERFORM THIS STEP

1 Right-click the relevant class in the class hierarchy then select **Set labels...** from the context menu.

The *Set labels* window is displayed:

Fig 4.22 *Setting class labels (1/2)*

2 Click **New**.

The *New : Dynamic label* window is displayed:

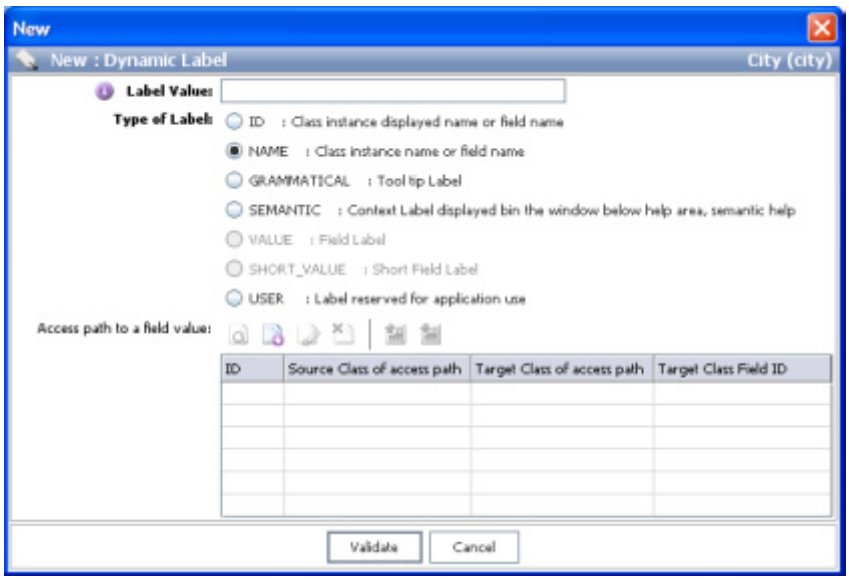

Fig 4.23 *Setting class labels (2/2)*

3 Set the fields:

Label value - The value of the label (or the corresponding entry in the dictionary). It should contain variable parts such as {i}, as with standard management of Java messages. The number of variable parts should match the number of access paths to be specified in the lower part of the form.

ID - The ID's visible name of the class instance. This label takes precedence over the combination of fields with the *id* mark. This label should be defined only with fields with the *id* mark.

NAME - The instance name of the class or the name of the fields. This label takes precedence over the combination of fields with the *name* mark and also over the *NAME* item.

GRAMMATICAL - The label displayed in the tooltips. This label takes precedence over the tooltip item.

SEMANTIC - Roll-over label displayed over the help line to the bottom of the forms.

VALUE - Presentation label for the field value.

SHORT_VALUE - Short presentation label for the field value.

USER - Application label.

4 To create an access path to a field value:
> Access path to the field value, page 73

5 Click **Validate**.

Access path to the field value

TO PERFORM THIS STEP

1 In the *New : Dynamic label* window, click **New**.

The *New : Access path* window is displayed:

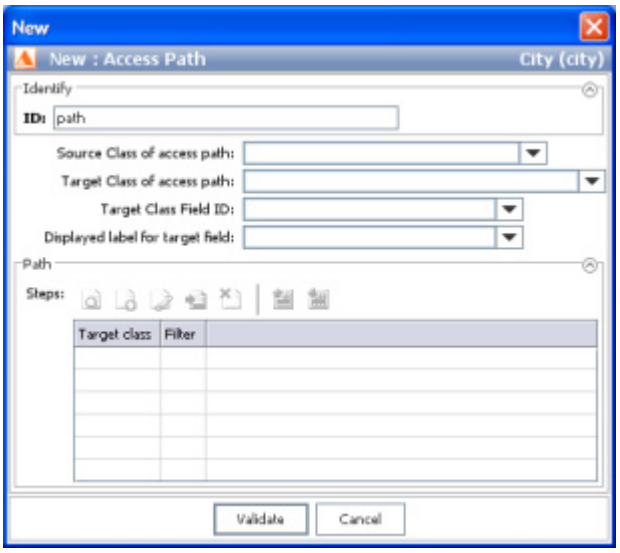

Fig 4.24 *Creating an access path*

2 Set the fields:

ID - The access path identifier.

Source class of access path - Source class of the access path, for an absolute access path.

Target class of access path - Target class of the access path. If not specified the current class will be used.

Target class of field ID - Final item of the access path. If not specified, the access path is equivalent to a route.

Displayed label for target field - Dynamic label to use for the target field.

3 To create steps for the access path:

3.1 Click **New** in the *Path* area.

The *Add step* window is displayed:

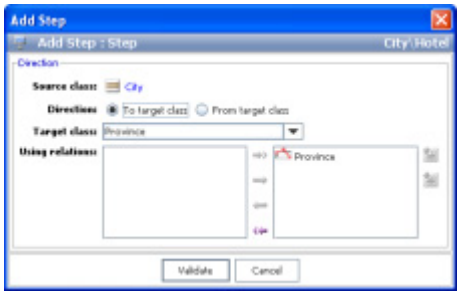

Fig 4.25 *Creating a step*

3.2 Set the fields:

To the target class - The target class of the step is the destination.

From the target class - The target class of the step is the origin.

Target class - Target class of the step.

Using relation - Relation field to link the source class to the target class.

3.3 Click **Validate**.

4 Back in the *New : Access path* window, click **Validate**.

4.18 Setting class comments

TO PERFORM THIS STEP

1 Right-click the relevant project or sub-project in the class hierarchy then select **Set comments...** from the context menu.

The *Set comments : Project* window is displayed.

2 Set the fields:

Start of comment

End of comment

NOTE Comments are added to the application's description XML file and also to the generated documentation.

3 Click **Validate**.

Working with attributes

The *Attributes* area displays the attributes of the currently selected class or project in the *Class hierarchy*.

In this section we will cover the following topics:

- General purpose topics:
  - The Attributes tab, page 77
  - Creating an attribute: Overview, page 78
  - Editing an attribute: Overview, page 79
  - Viewing attribute details, page 79
- How to create an attribute:
  - 5.1 Creating a numeric attribute, page 80
  - 5.2 Creating a text attribute, page 81
  - 5.3 Creating a multiple choice attribute, page 82
  - 5.4 Creating a time attribute, page 84
  - 5.5 Creating a relation attribute, page 85
  - 5.6 Creating a file attribute, page 87
  - 5.7 Creating a table attribute, page 88
  - 5.8 Creating a structure attribute, page 89
  - 5.9 Creating a typed field attribute, page 90
  - 5.10 Creating an attribute reference, page 91
  - Creating an import attribute, page 92
- How to edit an attribute:
  - 5.11 Editing a numeric attribute, page 93
  - 5.12 Editing a text attribute, page 93
  - 5.13 Editing a multiple choice attribute, page 94
  - 5.14 Editing a time attribute, page 95
  - 5.15 Editing a relation attribute, page 96
  - 5.16 Editing a file attribute, page 97
  - 5.17 Editing a table attribute, page 98
  - 5.18 Editing a structure attribute, page 99
  - 5.19 Editing a typed field attribute, page 100
  - 5.20 Editing an attribute reference, page 101

- How to implement attribute-related advanced features:
  - 5.21 Setting the read-only control, page 101
  - 5.22 Specifying allowed character sets, page 102
  - 5.23 Encrypting a value, page 103
  - 5.24 Adding a tooltip to a field, page 104
  - 5.25 Setting physical binding for a relation attribute, page 104
  - 5.26 Setting physical binding for a table attribute, page 111
  - 5.27 Setting physical binding for other attribute types, page 113
  - 5.28 Setting units, page 114
  - 5.29 Converting an attribute type, page 119
  - 5.30 Setting attribute marks, page 120
  - 5.31 Adding specific marks to an attribute, page 124
  - 5.32 Associating specific graphical components to an attribute, page 124
  - 5.33 Adding specific data, page 126
  - 5.34 Editing formatting constraints, page 127
  - 5.35 Setting attribute controls, page 128
  - 5.36 Setting attribute labels, page 129
  - 5.37 Specifying a cache policy for an attribute, page 132
  - 5.38 Setting attribute rules, page 133

The Attributes tab

To display the *Attributes* tab, select **Window** ▸ **Attributes**.

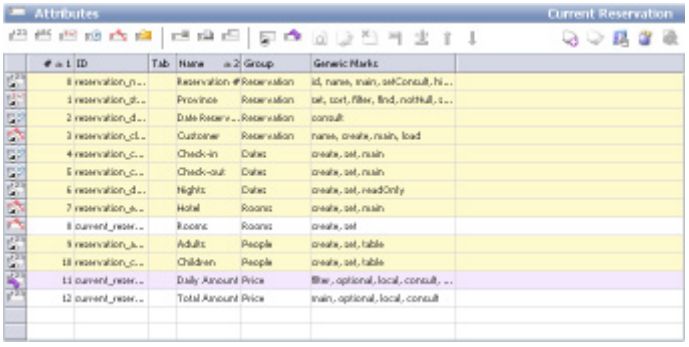

Fig 5.1 *The Attributes tab*

NOTE Attribute background colors:
- Yellow: Attribute reference
- Grey: The attribute extends the main class
- Purple: Imported attribute

Attributes can be specified at package level, typically so that they can be shared by several classes. Since they are organized into packages, no class is given priority over other classes.

The **#** column is the field index in the application class. The attributes are sorted by default according to this fields. However you can click the column headers to apply another sort

mode. If the attributes are managed by a package, and not by an application class, this order doesn't make sense and the column is no longer displayed.

Creating an attribute: Overview

Attributes are created either from within the class create form, or via the *Attributes* tab.

TO CREATE AN ATTRIBUTE VIA THE CLASS CREATE FORM

1. In the *Source attributes* area of the class create form, select the appropriate attribute type in the **Type** drop-down list.
2. Edit the ID, as appropriate.
3. Specify the name.
4. Click **New** at the end of the row.

TO CREATE AN ATTRIBUTE VIA THE ATTRIBUTES TAB

1. In the *Attributes* tab toolbar, click the relevant icon:
   - **Numeric attribute**
   - **Text attribute**
   - **Multiple choice attribute**
   - **Time attribute**
   - **Relation attribute**
   - **File attribute**
   - **Table attribute**
   - **Structure attribute**
   - **Typed field attribute**
   - **Attribute reference**
   - **Import attribute**
2. Set the fields in the attribute's create form.

PLEASE NOTE When you create an attribute via the *Attributes* tab, make sure the appropriate class or project is selected in the *Class hierarchy*.

The attribute's create form varies depending on the type of the attribute to be created. However, most of them share the following fields:

ID - The attribute's unique identifier, used by the application code. It is never displayed to the end user but is used to identify this attribute and its values in the application's Java code.

Name - The name that is displayed in the application to identify this attribute in the various application views (form, lists, etc).

Short name - [Optional] To use a shorter alternative to the name in the views where the available space is limited. It can be used for example in a column instead of the name, and the full name can be displayed as a tooltip when you hover over the column header for a while.

Tab - To include the attribute in a tab.

Group - To include the attribute in a group.

NOTE

Create a new string
Choose a string

Editing an attribute: Overview

Attributes are edited by making changes to their properties via their edit forms.

You can display an attribute's edit form in several ways:

- Double-click the attribute in the *Attributes* tab.
- Click the field icon in the first column of the *Attributes* tab.
- Right-click the attribute in the *Attributes* tab then select **Modify...** from the context menu.
- Select the attribute in the *Attributes* tab then click **Modify...** in the tool bar.
- Click **Modify...** in the attribute's read-only form.

Viewing attribute details

You can display the read-only form of the currently selected attribute by clicking the **Details** icon in the *Attributes* tab tool bar.

A number of actions are available in the attribute's read-only form, depending on the attribute type. The following actions however are shared by all the attributes:

Modify... - Displays the current attribute's edit form.

Previous - Displays the previous attribute's edit form.

Next - Displays the next attribute's edit form.

5.1 Creating a numeric attribute

The create form for a numeric attribute is as follows:

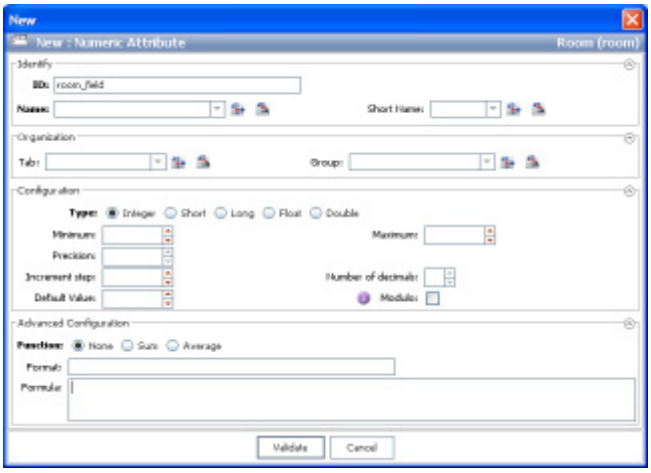

Fig 5.2 *Creating a numeric attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Type - The type of the numeric value:

- **Integer**
- **Short**
- **Long**
- **Float**
- **Double**

Minimum - **Maximum** - To specify the allowed range of values.

Precision - To specify the precision used to compare two values for this number. If the difference between the values is lower than the precision, the values are considered as being equal.

No increment - Specifies the increment step to be used when the end user clicks the spin buttons in the forms.

Number of decimals - To specify how many decimals are expected for real values (float / double).

Default value - To specify a defaut value for fields with the *not null* mark.

Modulo - When this option is selected, any value outside the range between the *Minimum* and *Maximum* values is increased or decreased to fall inside the range of values: the difference between the maximum and the minimum value is subtracted from the input value while the result does not fall within the range of values.

Formula - Specifies that the field value is obtained via a simple calculation formula based on other fields' values. The expected value is a mathematical formula with brackets containing basic calculation operators (+, -, *, /) and numeric field identifiers.

Example:

- For VAT amount: netInvoice * 0.196 (where netInvoice is the invoice tax-free amount)
- For Tax inclusive amount: netInvoice + invoiceVAT (where invoiceVAT is the tax amount)

Functions - When data is displayed in a table, specify whether the sum or average should be displayed at the bottom of the column.

Format - To specify the format for the displayed data. The syntax is specified in the `.text.NumberFormat` Java class.

5.2 Creating a text attribute

The create form for a text attribute is as follows:

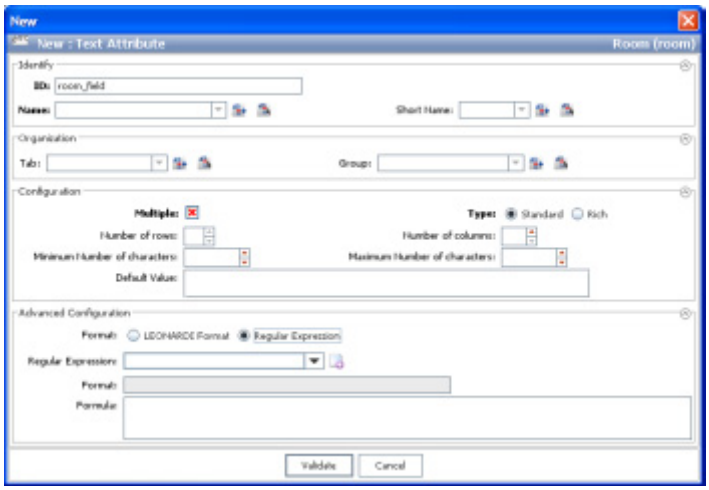

Fig 5.3 *Creating a text attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Multiple - To specify that the text field takes up several lines. When this option is not selected, the field is single line.

Type

Number of rows - To specify how many rows a multiple field should take up. This attribute is optional. This property does not set a maximum for the characters that may be entered.

Number of columns - To specify how many columns a multiple field should take up. This attribute is optional. This property does not set a maximum for the characters that may be entered.

Minimum number of characters - **Maximum number of characters** - To set a minimum / maximum for the characters that can be entered.

Default value - To set a default value for the fields in the form, or for optional attributes with the *notNull* mark.

For further information regarding optional, not null marks:

> 5.30 Setting attribute marks, page 120

Format - To ensure that the value matches a particular format. There are two alternatives:

- **Regular expressions** - More efficient (although more complex to define).
- **Application Engine format** - A simple alternative for text formatting. The expected format is a string made up of non editable areas and editable numeric parts. The numeric parts are defined by an interval of integer values. They can be used to authorize only inputs of numbers whose values fall within the interval corresponding to their position. In fat client, the format is managed by a graphical component to ensure the input is correct. The numeric values are included in square brackets [min - max] and the non modifiable text inside the brackets is kept as is.

Formula

5.3 Creating a multiple choice attribute

The create form for a multiple choice attribute is as follows:

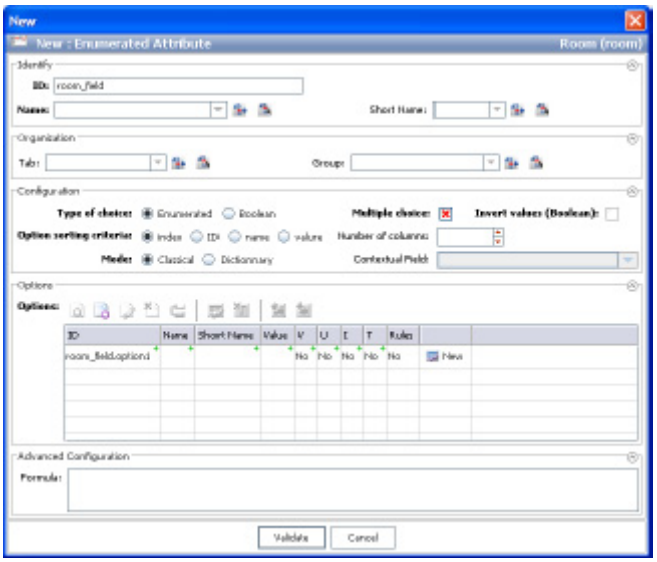

Fig 5.4 *Creating a multiple choice attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Type of choice - To change the representation mode and the type of the supported values. Types are either **Enumerate** (a number of option buttons) or **Boolean** (two check boxes).

Multiple choice - To specify whether several options can be specified for the enumerate value. If this option is selected, the field will be represented in the form of check boxes, otherwise in the form of radio buttons.

Invert values (Boolean) - To switch between the true/false values.

Option sorting criteria - To display the options according to the specified order.

Number of columns - To specify the number of options displayed for each line of the form. If this value is not specified, default formatting applies.

Mode

- **Classical** - To display the field as a series of check boxes.
- **Dictionary** - To display the field as a drop-down list in which the various options are organized into subsections.

Contextual field - To filter the options for a field according to another enumerated attribute.

Options - Possible values for the enumeration.

Value - Storage value for the current option in the data source. This value is used by the connector(s) but is never displayed to the end user.

V (Default value) - If selected, specifies that the current option is enabled by default. With a multiple field, several options can be enabled by default.

U (Option for the unknown values) - To specify that the current option is used to represent all the unknown values. This avoids errors with obsolete or incorrect values when data is being loaded.

I (Inactive) - If selected, specifies that the option cannot be selected by the end user. With this mode, data is obtained from the data provider but the option is not available in edit mode.

T (Column in editable table) - With multiple choices, specifies whether the option is displayed as a column or not in an editable table.

Rules can be added to the options via the **Rules** icon in the *Options* area tool bar.

Rules are triggered when selecting/deselecting the specified option or when a filter that did not previously function now does, or else when a filter does not function anymore but previously did.

For further information regarding rules:

> 4.16 Setting class rules, page 70

When several options are created, their order can be changed via the **Move up** and **Move down** icons in the *Options* area tool bar.

Formula

5.4 Creating a time attribute

The create form for a time attribute is as follows:

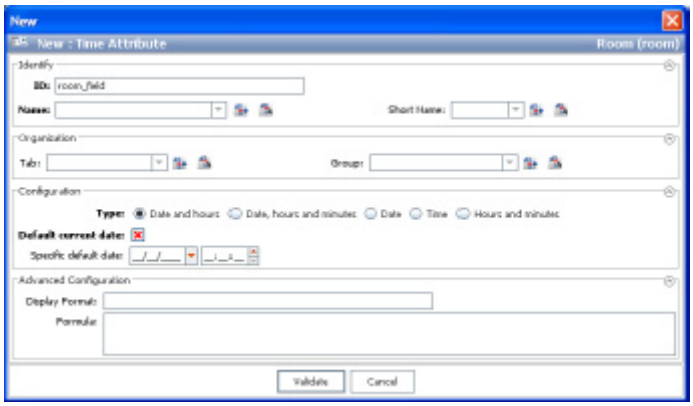

Fig 5.5 *Creating a time attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Type - To specify precision and content for the time attribute:

- **Date and hours**
- **Date, hours and minutes**
- **Date**
- **Time**
- **Hours and minutes**

Default current date - To specify that the field defaults to the current date.

Specific default date - To specify a default date other than the current date.

Display format - The syntax for the time attributes is the one specified by the `java.text.SimpleDateFormat` class. The default format for English language is: MM/dd/yyyy HH/mm/ss.

Formula

5.5 Creating a relation attribute

The create form for a relation attribute is as follows:

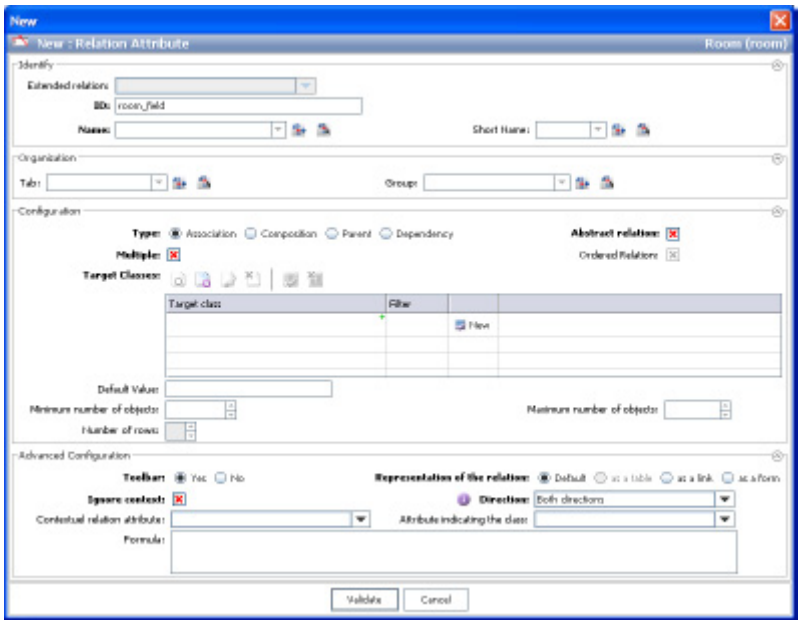

Fig 5.6 *Creating a relational attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Extended relation - Specifies the relation extended by the current relation. A relation can only extend another relation that is specified in a parent class.

Type - Specifies the type of relation and how it will be displayed in the form. The type determines on the one hand which graphical objects, and which values are suggested, and on the other hand the dependencies between the object containing the relation and the target object(s).

- **Association** (default type) - Specifies a simple relation between two classes. With an association, no dependency is managed.
- **Composition** - With a composition, the taget object is included in the form of the source object. Target objects are deleted if the object that points to them is deleted.
- **Parent** -
- **Dependency** -

Depending on the type of relation and whether or not the *Multiple* option is enabled, the available graphical representations in edit mode are as follows:

- Simple association
- Multiple association
- Simple composition relation
- Multiple composition relation

Multiple - To make it possible to select several objects in the relation.

Target classes - Indicates in which class(es) the objects to which the relationship points should be searched.

Default value - To specify an initial value in the forms or for the relations with the *notNull* mark. You can specify an object identifier or a variable of $USER type specified for the session.

Minimum number of objects - **Maximum number of objects** - When the relation is multiple, you cannot specify a minimum or a maximum number of objects that can be selected.

Number of rows - For a multiple relation, specifies the height of the object selection area.

Abstract relation - Specifies whether the relation is abstract or not. An abstract relation cannot have a value and should be extended.

Ordered relation - Specifies whether their is a particular direction for the object selection and whether it should be kept. Please note: With relational databases, it is usual to use join tables that are not ordered and this option will only work if the data provider preserves the order of the data.

Tool bar - To specify that no tool bar linked to the relation input field should be dipslayed in create / edit forms.

Ignore context - To specify that the relation should ignore the context of the current form. By default, when creating or modifying an application object, if a context is available for the view, only the objects related to the view are available for the relation.

Contextual relation attribute - To specify that the context for this field is determined by the value of another relation in the form.

Representation of the relation

- **Default**
  [Simple association] A combo box
  [Multiple association] Two list boxes with the available values, and the currently selected values
  [Simple composition] The form of the target data item is included in the form of the main data item
  [Multiple composition] A table
- **As a table**
  [Simple relation] Not applicable
  [Multiple relation] Same display as with a multiple composition relation
- **As a link**
  [Simple relation] A link to the data item
  [Multiple relation] A list of links
- **As a form**
  [Simple relation] Same display as with a simple composition relation
  [Multiple relation] Not applicable

Direction - To specify how the relation can be used to calculate contexts between the objects. This parameter is used for the calculation of cross-references. By default all the relations are used for a calculation in both directions. You can specify in which case the relation should be considered.

- **Both directions** - Never ignored (default choice).
- **None** - Always ignored.
- **Upward direction** - Ignored for reverse calculations.
- **Backward direction** - Ignored for direct calculations.

Attribute indicating the class - To specify that the class to which the relation applies is determined by the value of another field (an enumeration whose option names are the names of the available application classes).

Formula

5.6 Creating a file attribute

A file attribute specifies the access path to a file.

The create form for a file attribute is as follows:

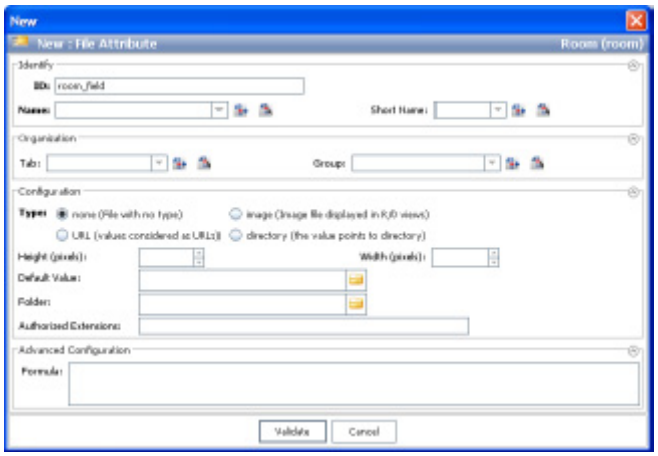

Fig 5.7 *Creating a file attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Type - To specify the type of data that is pointed to by the file attribute:

- **None (file with no type)**
- **Image (image file displayed in R/O views)**
- **URL (values considered as URLs)**
- **Directory (the value points to directory)**

Height - To specify the height for the preview area in the forms when a preview is available.

Width - To specify the width for the preview area in the forms when a preview is available.

Default value - To specify a default file.

Folder - The root folder for file selection.

Authorized extensions - The allowed extensions for file selection (e.g: *.doc).

Formula

5.7 Creating a table attribute

The create form for a table attribute is as follows:

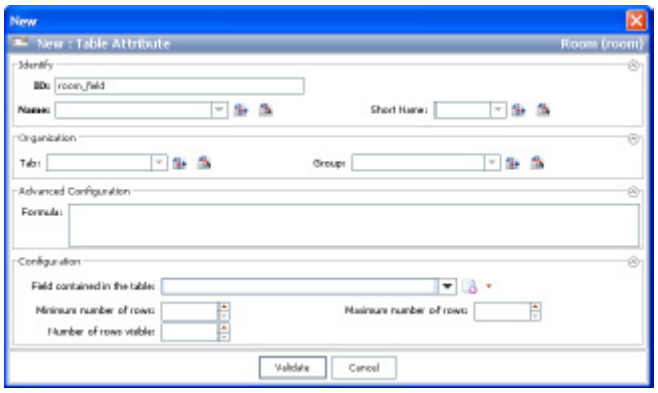

Fig 5.8 *Creating a table attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Formula

Field contained in the table - The field that makes up the table. You can have the table contain several fields by selecting a structure attribute.

Minimum number of rows - **Maximum number of rows** - To specify the minimum/maximum number of values required in the table.

Number of rows visible - To specify the number of rows to be displayed. This number does not need to be equal to the previous values.

5.8 Creating a structure attribute

A structure can be seen as a hidden composition relation to an internal class. In other words, it can be used to organize several fields into one.

The create form for a structure attribute is as follows:

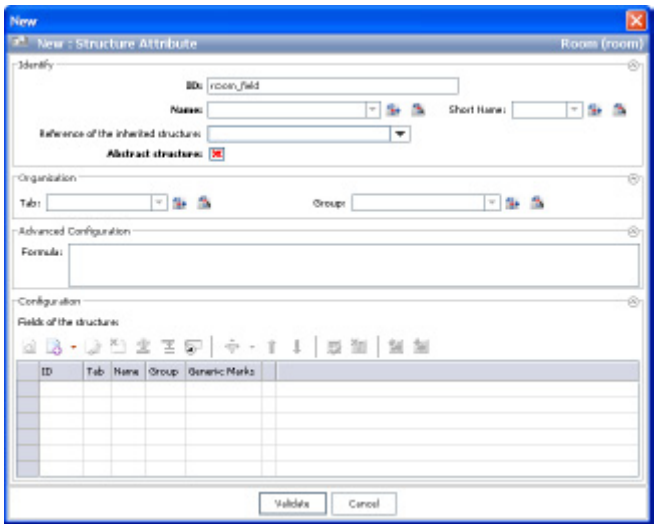

Fig 5.9 *Creating a structure attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Reference of the inherited structure - Parent structure.

Abstract structure - Specifies whether the structure is abstract. An abstract structure cannot be instantiated.

Formula

Fields of the structure - List of the fields in the structure.

5.9 Creating a typed field attribute

A typed field attribute allows you to define a field based on a model. The result is the same as using the model attribute when defining each of the field types.

A typed field has the same type as the field referenced by the type attribute.

The create form for a typed field attribute is as follows:

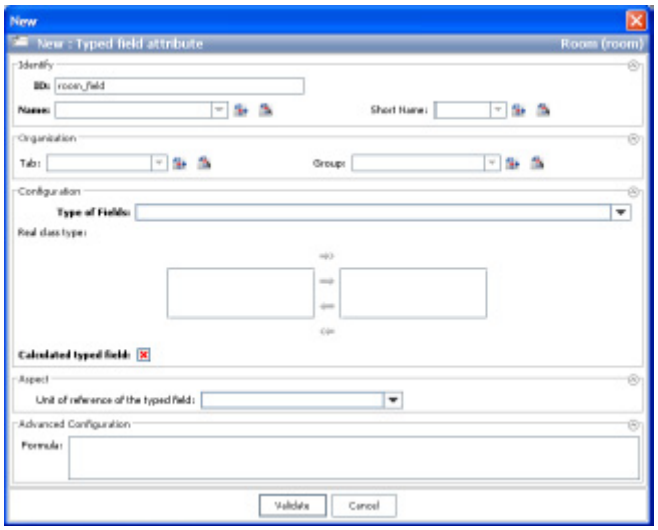

Fig 5.10 *Creating a typed field attribute*

For a description of the properties shared by all the attribute types:

> Creating an attribute: Overview, page 78

Type of fields - The model for the typed field.

Real class type - This attribute is used only with relations and structures to specify the real types of the supported relation classes or structures. Mainly used when extending abstract classes or abstract structures in order to specify the concrete types.

Calculated typed field - A calculated field will be declared as local when generating the default persistence model.

Reference unit of the typed field - To select the unit of the typed field from the type units.

Formula

5.10 Creating an attribute reference

You can create a pointer to an existing attribute. As a result, the target attribute becomes a shared attribute. No attribute is created.

Reference attributes are displayed on a yellow background in the *Attributes* tab.

The create form for an attribute reference is as follows:

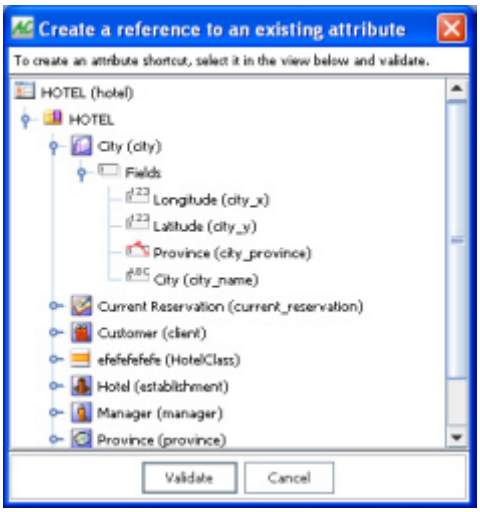

Fig 5.11 *Creating a reference to an attribute*

To create the reference, select the apropriate attribute then click **Validate**.

Creating an import attribute

An import attribute is a local attribute that is automatically calculated via a relation. It is automatically updated by Application Engine.

The create form for an import attribute is as follows:

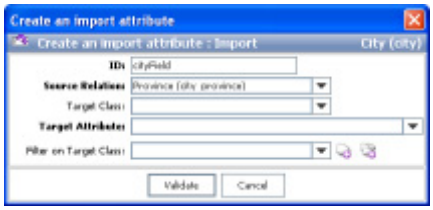

Fig 5.12 *Creating an import attribute*

ID - The attribute identifier.

Source relation - The relation that points to the classes containing the field to be imported.

Target class - The class targeted by the relation and containing the field to be imported.

Target attribute - The identifier of the field to be imported into the class(es) targeted by the relation.

Filter on target class

> 3.5 Creating filters for a project, page 46

5.11 Editing a numeric attribute

The edit form for a numeric attribute is similar to that of the create form.

For information on how to display this form:

> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:

> 5.1 Creating a numeric attribute, page 80

Available actions from within the edit form

Details - To display the attribute's read-only form:

> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.

> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.

> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.

> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.

> 5.36 Setting attribute labels, page 129

Set number unit - To specify one or more units for the attribute.

> 5.28 Setting units, page 114

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.12 Editing a text attribute

The edit form of a text attribute is similar to that of the create form.

For information on how to display this form:

> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:

> 5.2 Creating a text attribute, page 81

Available actions from within the edit form

Details - To display the attribute's read-only form.

> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.

> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.

> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.

> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.

> 5.36 Setting attribute labels, page 129

Set text unit - To set one or more units for the attribute.

> 5.28 Setting units, page 114

Define allowed characters - To set restrictions on which characters are allowed for the attribute.

> 5.22 Specifying allowed character sets, page 102

Encryption Java class - To encrypt the attribute value.

> 5.23 Encrypting a value, page 103

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.13 Editing a multiple choice attribute

The edit form of an enumeration attribute is similar to the creation form.

For information on how to display this form:

> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:

> 5.3 Creating a multiple choice attribute, page 82

Available actions from within the edit form

Details - To display the attribute's read-only form.
> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.
> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.
> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.
> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.
> 5.36 Setting attribute labels, page 129

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.14 Editing a time attribute

The edit form of a time attribute is similar to that of the create form.

For information on how to display this form:
> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:
> 5.4 Creating a time attribute, page 84

Available actions from within the edit form

Details - To display the attribute's read-only form.
> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.
> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.
> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.
> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.

> 5.36 Setting attribute labels, page 129

Set date unit - To set one or more units for the attribute.

> 5.28 Setting units, page 114

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.15 Editing a relation attribute

The edit form of a relation attribute is similar to that of the create form.

For information on how to display this form:

> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:

> 5.5 Creating a relation attribute, page 85

Available actions from within the edit form

Details - To display the attribute's read-only form.

> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.

> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.

> 5.25 Setting physical binding for a relation attribute, page 104

Set controls - To set one or more controls for the attribute.

> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.

> 5.36 Setting attribute labels, page 129

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.16 Editing a file attribute

The edit form of a file is similar to that of the create form.

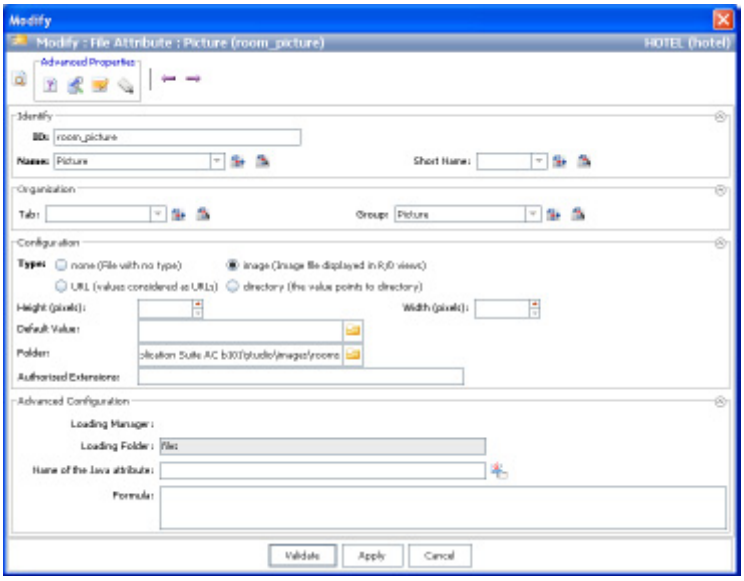

Fig 5.13 *Modifying a file attribute*

For information on how to display this form:

> Editing an attribute: Overview, page 79

The fields that are specific to the edit form are as follows:

Loading manager - The full name of a Java class in charge of managing attachments.

In thin client, the attached file can be stored on the server. If the value is not specified, the sending of attachments is disabled. You can set the field to *default* to specify the default manager (i.e. `leon.view.web.LyFileAttachmentHandler`). Alternatively you specify a specific manager.

Loading folder - The directory used to store the files on the server side.

Name of the Java attribute - The name of the corresponding Java field, when using interface class generation.

Generate - To automatically set the **Name of the Java attribute** field. The field identifier is used by default.

For further information regarding the other fields in the form:

> 5.6 Creating a file attribute, page 87

Available actions from within the edit form

Details - To display the attribute's read-only form.
> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.
> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.
> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.
> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.
> 5.36 Setting attribute labels, page 129

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.17 Editing a table attribute

The edit form of a table is similar to that of the create form.

For information on how to display this form:
> Editing an attribute: Overview, page 79

The fields that are specific to the edit form are as follows:

Name of the Java attribute - Name of the corresponding Java field, when using interface classes generation.

Generate - To automatically set the **Name of the Java attribute** field. The field identifier is used by default.

For further information regarding the other fields in the form:
> 5.7 Creating a table attribute, page 88

Available actions from within the edit form

Details - To display the attribute's read-only form.
> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.

> 5.26 Setting physical binding for a table attribute, page 111

Set physical binding - To set physical binding for the attribute.

> 5.26 Setting physical binding for a table attribute, page 111

Set controls - To set one or more controls for the attribute.

> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.

> 5.36 Setting attribute labels, page 129

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.18 Editing a structure attribute

A structure attribute's edit form is similar to the create form.

For information on how to display this form:

> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:

> 5.8 Creating a structure attribute, page 89

Available actions from within the edit form

Details - To display the attribute's read-only form.

> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.

> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.

> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.

> 5.35 Setting attribute controls, page 128

Set labels - To set one or more dynamic labels for the attribute.

> 5.36 Setting attribute labels, page 129

Cache management - To set a cache policy for the attribute.

> 5.37 Specifying a cache policy for an attribute, page 132

Set structure unit - To set one or more units for the attribute.

> 5.28 Setting units, page 114

Set structure rules - To set one or more rules for the attribute.

> 5.38 Setting attribute rules, page 133

Class behavior -

> 4.13 Specifying the class behavior, page 63

Interface class / Static identifier -

Previous - To display the previous attribute's edit form.

Next - To display the next attribute's edit form.

5.19 Editing a typed field attribute

The edit form of a typed field attribute is similar to that of the create form.

For information on how to display this form:

> Editing an attribute: Overview, page 79

For further information regarding the fields in the form:

> 5.9 Creating a typed field attribute, page 90

Available actions from within the edit form

Details - To display the attribute's read-only form.

> Viewing attribute details, page 79

Tooltip - To set a tooltip for the attribute.

> 5.24 Adding a tooltip to a field, page 104

Set physical binding - To set physical binding for the attribute.

> 5.27 Setting physical binding for other attribute types, page 113

Set controls - To set one or more controls for the attribute.

> 5.35 Setting attribute controls, page 128

Set number unit - To set one or more units for the attribute.

> 5.28 Setting units, page 114

⬅ **Previous** - To display the previous attribute's edit form.

➡ **Next** - To display the next attribute's edit form.

5.20 Editing an attribute reference

The edit form of a reference attribute is similar to that of the create form.

For further information regarding the fields in the form:

> 5.10 Creating an attribute reference, page 91

5.21 Setting the read-only control

The **Set R/O control** icon in the attribute modification form tool bar allows you to define the write protection level.

A field with a read-only control cannot be modified.

Write protection - Determines additional parameters for the protection. If the field is not specified, the priority level of the control will be empty and the type will be ALL.

Priority of the Control - Priority level of the control.

Type of the control - There are two types of control.

- **ALL** - The field cannot be modified, whatever the origin of the modification may be.
- **USER** - The field cannot be modified by the user (through the MMI or by code when processing is started). However, this field can be modified externally that is, when receiving a message.

5.22 Specifying allowed character sets

You can set restrictions on which characters are allowed in the fields. In fat client mode, unauthorized characters cannot be entered while in thin client mode they are rejected when the value is controlled.

If the text field has more than one character set, the character entered should be allowed by at least one of them.

TO PERFORM THIS STEP

1 Click **Define allowed characters...** in the edit form toolbar.

The *Define allowed characters* window is displayed:

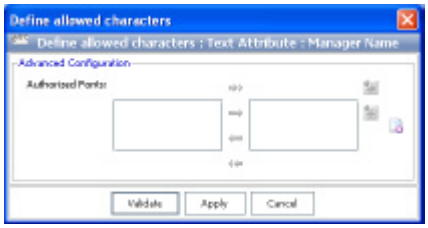

Fig 5.14 *Modifying the allowed character set (1/2)*

2 Click **New** to create a character set.

The *New : Authorized fonts* window is displayed:

Fig 5.15 *Modifying the allowed character set (2/2)*

3 Set the field:

Authorized character values - A regular expression that specifies the allowed characters.

4 Click **Validate**.

5 Back in the *Define allowed characters* window, create as many character sets as required.

Use the icons to decide which available character sets should be allowed:

- - To select a character set
- - To select all the character sets
- - To unselect a character set
- - To unselect all the character sets

6 Click **Validate**.

5.23 Encrypting a value

The value users enter in a text field can be encrypted.

TO PERFORM THIS STEP

1 Click **Encryption Java class...** in the edit form toolbar.

The *Encryption Java class* window is displayed:

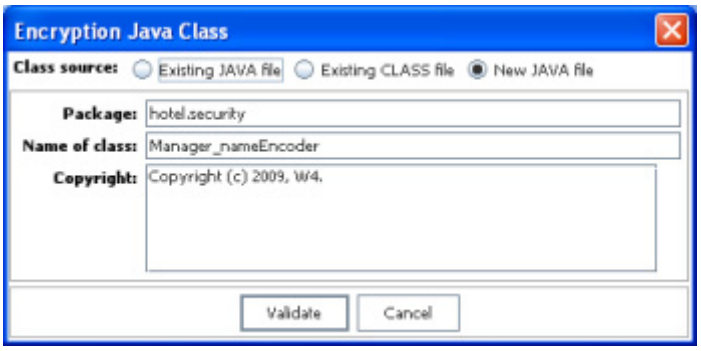

Fig 5.16 *Associating an encryption to a text*

2 Set the fields:

Existing JAVA file - To specify a preexisting Java file for encryption. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for encryption. The appropriate file should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new Java file for encryption.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. However you can change some parameters before generation via the next fields. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <application id>.security. The security directory will be created to the root of the application.

Name of the class - The class name, by default: *IdentifierEncoder* with the application identifier's first letter in uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the application_composer.ini file as the default value.

3 Click **Validate**.

5.24 Adding a tooltip to a field

Tooltips can be added to the fields. Tooltips are displayed when you hover over a field's help icon for a while.

TO PERFORM THIS STEP

1 Click **Tooltips...** in the edit form tool bar.

The *Tooltips* window is displayed:

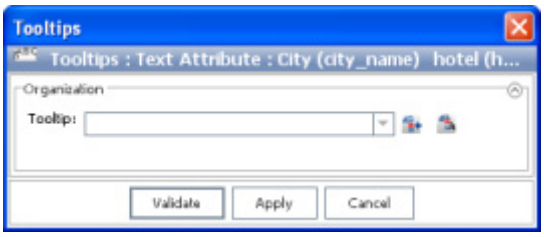

Fig 5.17 *Adding a tooltip to a field*

2 Set the fields:

Tooltip - The character string to be displayed when hovering over the field's help icon.

Create a new string

Choose a string

3 Click **Validate**.

5.25 Setting physical binding for a relation attribute

TO PERFORM THIS STEP

1 Click **Set physical binding...** in the edit form tool bar.

The *Set physical binding* window is displayed:

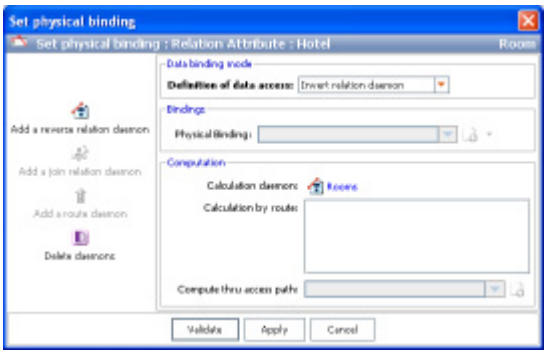

Fig 5.18 *Setting the physical link for a relation*

2 In the *Data binding mode* area, select the appropriate data binding mode in the **Definition of data access** drop-down list.

Direct binding - The field is directly linked to a physical field (e.g. a column in a database table).

Calculation by access path - To reference the value of a target object field from a source object through steps defined by relationships.

Calculation by daemon - Daemons are Java objects that allow you to be notified when events concerning logical objects are triggered. There are three types of daemon.

- **Invert relation daemon** - To represent the reverse simple relation by a direct multiple relation.
- **Joint daemon** - To manage a relation via a join.
- **Calculation by route** - To represent a direct relation (multiple or not) via several direct relations. A route is specified to access the physical field.

3 The subsequent actions to be performed depend on what option has been selected in the **Definition of data access** drop-down list.

3.1 **Direct binding** - In the *Bindings* area, select the appropriate binding in the **Physical binding** drop-down list.

Alternatively click **Select by name** next to the drop-down list to select the appropriate binding via an autocomplete text field.

You can also create new bindings:

- **Simple attribute binding**

> Creating simple attribute bindings, page 106

- **Multiple attribute binding**

> Creating multiple attribute bindings, page 107

- **LDAP directory attribute binding**

> Creating LDAP directory attribute bindings, page 107

3.2 **Calculation by access path** - In the *Computation* area, select the appropriate access path in the **Compute thru access path** drop-down list.

Alternatively, you can create new access paths:

> Creating access paths, page 131

3.3 **Calculation by daemon**

- **Reverse relation daemon**

> Creating reverse relation daemons, page 108

- **Joint relation daemon**

> Creating join relation daemons, page 108

- **Route daemon**

> Creating route daemons, page 109

4 Click **Validate**.

Creating simple attribute bindings

TO PERFORM THIS STEP

1 In the *Bindings* area of the *Set physical binding* window, click next to the icon.

2 Click **New : Simple attribute binding**.

The *New : Simple attribute binding* window is displayed:

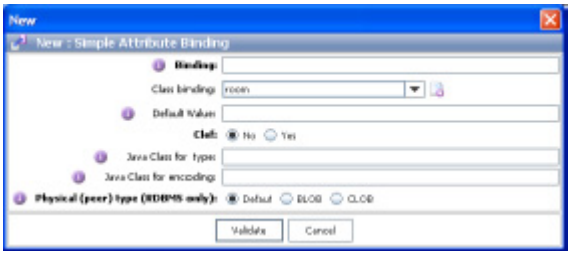

Fig 5.19 *Creating simple attribute bindings*

3 Set the fields:

Binding - The identifier of the physical field.

Class binding - The identifier of the physical class.

> 4.14 Setting physical binding for a class, page 64

Default value - The default value, which can be the value inserted in the database.

Primary key - Specifies whether the field has a primary key in the physical layer or not.

Java class for type - The full name of the Java class that specifies the type to be used for the field's physical values.

Java class for encoding - The full name of the Java class to be used for the encoding/decoding of the field's physical values.

Physical (peer) type (RDBMS only) - [Relational databases] Specifies whether data is stored as binary or character large objects. Use *Default* if database tables are not created using BLOB or CLOB. Only applies to text or file logical fields.

4 Click **Validate** to go back to the *Set physical binding* window.

Creating multiple attribute bindings

When using a relational database, the identifier of a table (the key) is often made up of several fields. If so, several foreign keys are imported so that records can refer one another.

TO PERFORM THIS STEP

1 In the *Bindings* area of the *Set physical binding* window, click next to the icon.

2 Click **New : Multiple attribute binding**.

The *New : Multiple attribute binding* window is displayed:

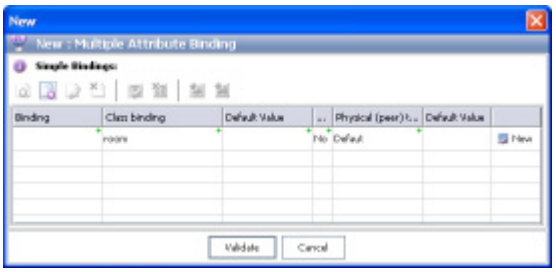

Fig 5.20 *Creating multiple attribute bindings*

3 Create as many simple bindings as required.

> Creating simple attribute bindings, page 106

4 Click **Validate** to go back to the *Set physical binding* window.

Creating LDAP directory attribute bindings

TO PERFORM THIS STEP

1 In the *Bindings* area of the *Set physical binding* window, click next to the icon.

2 Click **New : LDAP attribute binding**.

The *New : LDAP attribute binding* window is displayed:

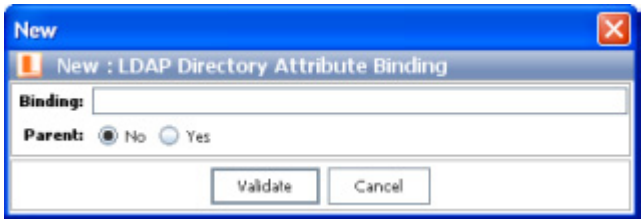

Fig 5.21 *Creating LDAP directory attribute bindings*

3 Set the fields:

Binding

Parent

4 Click **Validate** to go back to the *Set physical binding* window.

Creating reverse relation daemons

This action is only available with reverse relation daemons. It creates a reverse relation daemon from the Source relation.

TO PERFORM THIS STEP

1 In the *Set physical binding* window, click **Add a reverse relation daemon**.

The *New : Inverted daemon* window is displayed:

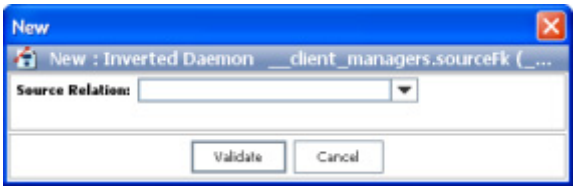

Fig 5.22 *Creating reverse relation daemons*

2 Set the fields:

Source relation

3 Click **Validate** to go back to the *Set physical binding* window.

The value that has been set for the source relation is displayed in the **Compute thru access path** field.

Creating join relation daemons

TO PERFORM THIS STEP

1 In the *Set physical binding* window, click **Add a joint relation daemon**.

The *New : Joint daemon* window is displayed:

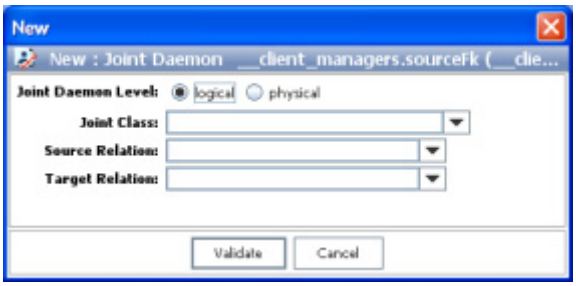

Fig 5.23 *Creating join relation daemons (1/2)*

2 Set the fields:

Logical - To specify a logical join class.

Physical - To specify a join database table.

Joint class - Logical join table.

Source relation - Relation of the logical join class used as source for the join.

Target relation - Relation of the logical join class used as target for the join.

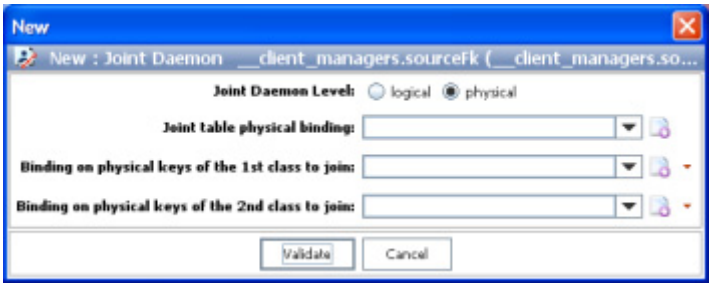

Fig 5.24 *Creating join relation daemons (2/2)*

Joint table physical binding

Binding on physical keys of the 1st class to join

Binding on physical keys of the 2nd class to join

3 Click **Validate** to go back to the *Set physical binding* window.

Creating route daemons

TO PERFORM THIS STEP

1 In the *Set physical binding* window, click **Add a route daemon**.

The *Add a route daemon* window is displayed:

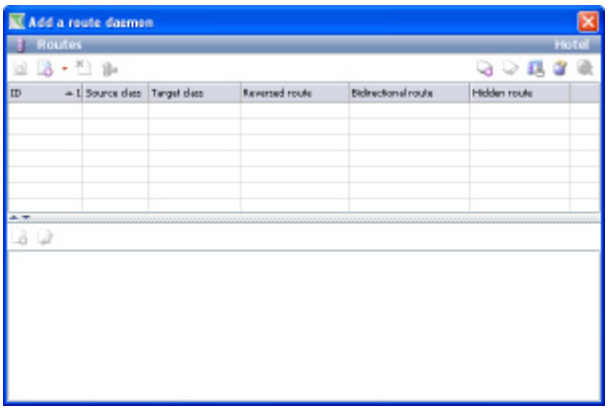

Fig 5.25 *Creating route daemons (1/2)*

2 Create as many routes and reverse routes as required.

2.1 Click ▾ next to the icon.

2.2 Choose to create either a route, or a reverse route.

The route creation window is displayed:

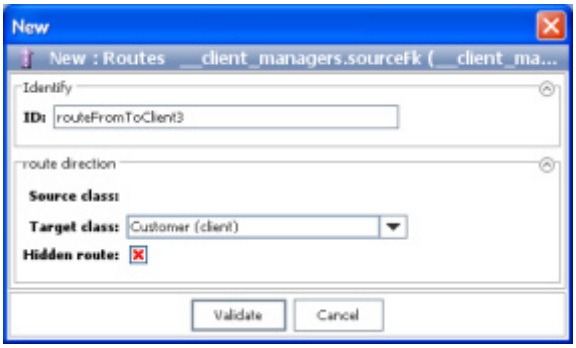

Fig 5.26 *Creating route daemons (2/2)*

2.3 Set the fields:

ID - Application Composer automatically sets the identifier as follows: *routeFrom<name of source class>To<name of target class>*. Change as appropriate

Target class - To select the target class.

Hidden route - To specify whether the route should be hidden, i.e. the route should not used for cross-reference calculation. Hidden routes are specified in the data model. They are used via programming only.

2.4 Click **Validate** to go back to the *Add a route daemon* window.

2.5 Repeat the steps above to create as many routes as required.

3 Close the *Add a route daemon* window to go back to the *Set physical binding* window.

The routes that have been created are displayed in the *Computation* area, in the **Calculation by route** field.

5.26 Setting physical binding for a table attribute

TO PERFORM THIS STEP

1 Click **Set physical binding...** in the edit form tool bar.

The *Set physical binding* window is displayed:

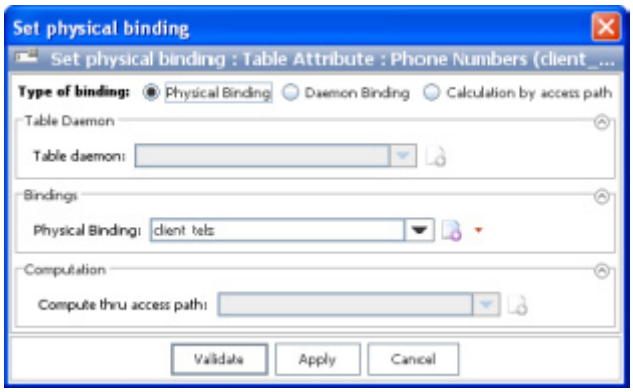

Fig 5.27 *Setting the physical binding of a table attribute*

2 Select the appropriate data binding mode in the **Type of binding** field:

Physical binding

Daemon binding

Calculation by access path

3 The subsequent actions to be performed depend on what option has been selected in the **Type of binding** field.

3.1 **Physical binding** - In the *Bindings* area, select the appropriate binding in the **Physical binding** drop-down list.

Alternatively click **Select by name** next to the drop-down list to select the appropriate binding via an autocomplete text field.

You can also create new bindings:

- **Simple attribute binding**

> Creating simple attribute bindings, page 106

- **LDAP directory attribute binding**

> Creating LDAP directory attribute bindings, page 107

3.2 **Daemon binding** - In the *Table daemon* area, select the appropriate daemon in the **Table daemon** drop-down list.

Alternatively you can create new daemons:

> Creating table daemons, page 112

3.3 **Calculation by access path** - In the *Computation* area, select the appropriate access path in the **Compute thru access path** drop-down list.

Alternatively, you can create new access paths:

> Creating access paths, page 131

4 Click **Validate**.

Creating table daemons

TO PERFORM THIS STEP

1 In the *Table daemon* area of the *Set physical binding* window, click **New**.

The *New : Array daemon* window is displayed:

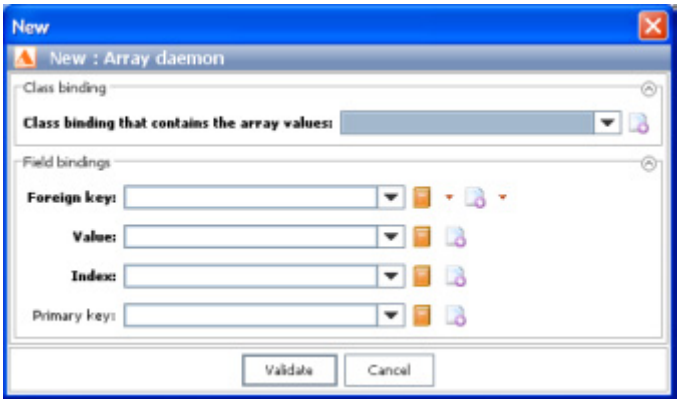

Fig 5.28 *Creating table daemons*

2 You can set the fields in the *Field bindings* area by:

- Selecting the appropriate value from the drop-down list
- Clicking **Select by name** next to the drop-down list to select the appropriate value via an autocomplete text field.
- Creating simple attribute bindings:

> Creating simple attribute bindings, page 106

- Creating multiple attribute bindings (for the **Foreign key** field only):

> Creating multiple attribute bindings, page 107

Class binding that contains the array values - The name of the table containing the values of the table field.

Foreign key - The foreign key allowing the identification of the original object that contains the table field.

Value - The name of the physical field containing the value.

Index - The name of the physical field containing the index of the value in the table.

Primary key - By default, the key is made up of the foreign key and the index of the value in the table.

3 Click **Validate** to go back to the *Set physical binding* window.

5.27 Setting physical binding for other attribute types

TO PERFORM THIS STEP

1 Click **Set physical binding...** in the edit form tool bar.

The *Set physical binding* window is displayed:

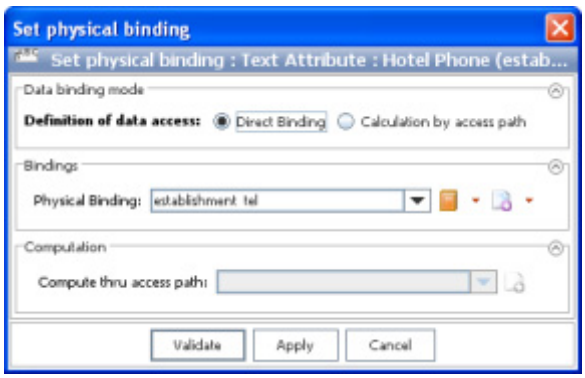

Fig 5.29 *Setting the physical binding for other attribute types*

2 Select the appropriate data binding mode in the **Definition of data access** field:

Direct binding

Calculation by access path

3 The subsequent actions to be performed depend on what option has been selected in the **Definition of data access** field.

3.1 **Direct binding** - In the *Bindings* area, select the appropriate binding in the **Physical binding** drop-down list.

Alternatively click **Select by name** next to the drop-down list to select the appropriate binding via an autocomplete text field.

You can also create new bindings:

- **Simple attribute binding**

> Creating simple attribute bindings, page 106

- **LDAP directory attribute binding**

> Creating LDAP directory attribute bindings, page 107

3.2 **Calculation by access path** - In the *Computation* area, select the appropriate access path in the **Compute thru access path** drop-down list.

Alternatively, you can create new access paths:

> Creating access paths, page 131

4 Click **Validate**.

5.28 Setting units

You can specify units for the date field, number, text and structure attributes.

TO PERFORM THIS STEP

1 Click **Set [...] Unit...** in the edit form tool bar.

The *Set [...] unit* window is displayed:

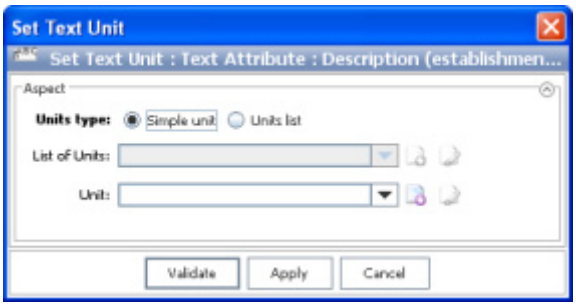

Fig 5.30 *Setting units*

2 Select the appropriate option in the **Unit type** field.

3 The subsequent actions to be performed depend on what option has been selected in the *Unit type* field.

3.1 **Simple unit**

Unit - To select an existing unit.

Alternatively click **New** to create a new unit.

> Creating simple units, page 115

You can also click **Modify** to edit the currently selected unit.

3.2 **Unit list**

List of units - To select an existing unit list.

Alternatively click **New** to create a new unit list.

> Creating simple units, page 115

You can also click **Modify** to edit the currently selected unit list.

Unit - To select an existing unit unit.

Alternatively click **New** to create a new unit.

> Creating simple units, page 115

You can also click **Modify** to edit the currently selected unit.

4 Click **Validate**.

Creating simple units

TO PERFORM THIS STEP

1 In the *Set [...] unit* window, click **New** next to the **Unit** field.

The *New : Unit attribute* window is displayed:

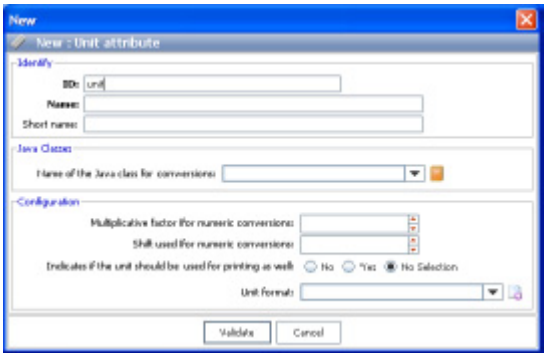

Fig 5.31 *Creating simple units (1/3)*

2 Set the fields:

Identifier - The unit identifier.

Name - The visible name.

Short name - The alternative name to be used when graphical components have limited available space.

Name of the Java class for the conversions - The full name of the Java class in charge of converting the unit into the reference unit and vice-versa. This class takes precedence over the class specified in the unit list. It must implement the `leon.info.infointerface.LyConverterInterface` interface.

Multiplying factor for the numerical conversions - When there is no conversion class, conversion for the numeric fields is performed automatically as soon as the desired unit is selected.

Shift used for the numerical conversions - The value that is added (or subtracted, if the value is negative) to convert the original value.

Example: To convert a value in degrees Fahrenheit into one in degrees Centigrade, the multiplying factor is 0,55555 and the shift is -32

Unit used for the printing - Possible values are **Yes**, **No** or **No selection**. By default, the reference unit is used, otherwise, the first unit in the unit list with this property is used.

Unit format - Select t he format that the entered value should match. A format can be associated either with a field of type text, date, or number, or with a unit.

Alternatively, you can create a new format.

> Creating formats, page 116

3 Click **Validate**.

The following message is displayed:

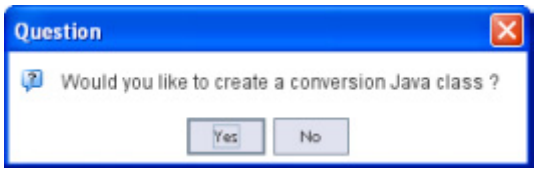

Fig 5.32 *Creating simple units (2/3)*

Clicking Yes displays the creation form for the conversion class:

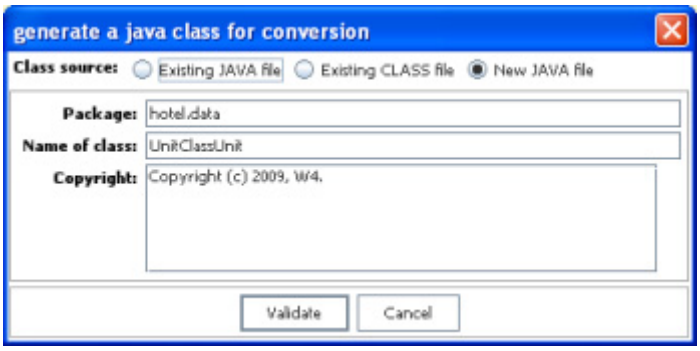

Fig 5.33 *Creating simple units (3/3)*

4 Set the fields:

Existing JAVA file - To select a preexisting Java file.

Existing CLASS file - To select a preexisting class file.

New JAVA file - To have Application Composer create a new Java file for the conversion.

Application Composer is going to generate the Java file in the application directory and create the relevant directories for the class packages. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <id of the application>.data. The data directory will be created to the root of the application.

Name of the class - The class name, by default: <*IdentifierUnit*>*ClassUnit* with the application identifier's first letter in uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the application_composer.ini file as the default value.

5 Click **Validate**.

Creating formats

TO PERFORM THIS STEP

1 In the *New : Unit attribute* window, click **New** next to the **Unit format** field.

The *New : Format* window is displayed:

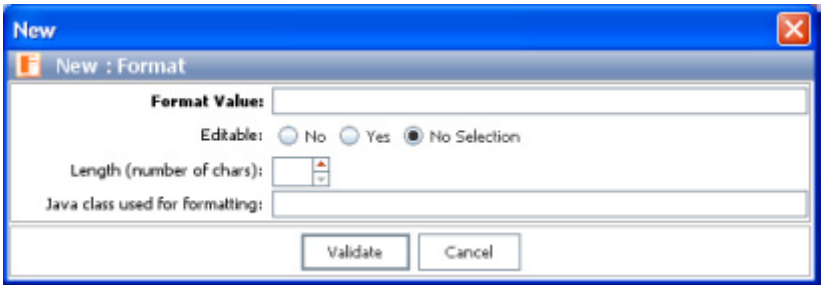

Fig 5.34 *Creating formats*

2 Set the fields:

Format value - A string containing fixed areas and editable numerical parts, character ranges or lists of choice. When the format is associated with a unit, the value is entered via a text field, editable or not, with or without input format. If so, the input format varies depending on the selected unit.

Editable - Specifies whether the text field can be edited or not.

Length (number of chars) - The width of the text field.

Java class used for the formatting - The full name of the Java class used for value formatting. This class must implement the interface `leon.info.infointerface.LyFormatterInterface`. This class is in charge of formating the entered value into a valid value.

3 Click **Validate** to go back to the *New : Unit attribute* window.

Creating unit lists

You can specify unit lists if there are several units for the date, number, text and structure attributes.

TO PERFORM THIS STEP

1 In the *Set [...] unit* window, click **New** next to the **List of units** field.

The *New : List of units* window is displayed:

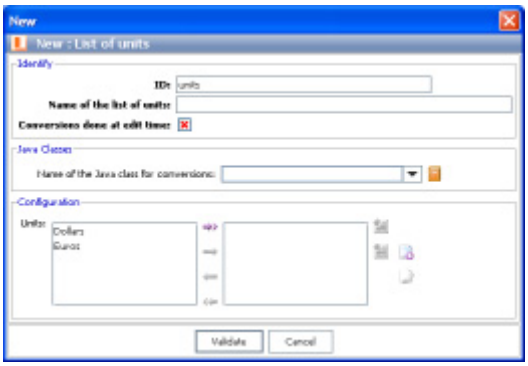

Fig 5.35 *Creating unit lists (1/2)*

2 Set the fields:

ID - The unit list identifier.

Name of the list of units - The list name.

Conversion done at edit time - Specifies whether or not conversion should be performed at input time.

Name of the Java class for the conversions - Select the Java class in charge of converting the selected unit into the field reference unit. This class must implement the interface `leon.info.infointerface.LyConverterInterface` that contains the methods used to convert the reference unit into the desired unit and vice-versa.

Alternatively click **Select by name** next to the drop-down list to select the appropriate class via an autocomplete text field.

Units: The units included in the list.

Use the icons to decide which available units should be used:

- To select a unit
- To select all the units
- To unselect a unit
- To unselect all the units

Alternatively click **New** to create a new unit.

> Creating simple units, page 115

You can also click **Modify** to edit the currently selected unit.

3 Click **Validate**.

The following message is displayed:

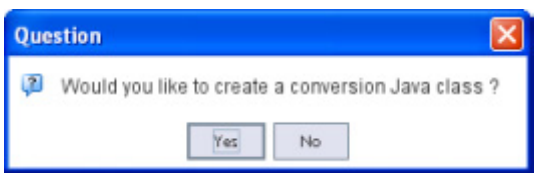

Fig 5.36 *Creating unit lists (2/2)*

Clicking Yes displays the **Generate Java class for the conversions** form.

For further information on how to set the fields in this form, please refer to the corresponding steps in:

> Creating simple units, page 115

5.29 Converting an attribute type

You can change the type of an attribute, for example a text identifier can be converted into a numeric identifier.

TO PERFORM THIS STEP

1 Right-click the attribute in the *Attributes* tab then select **Convert attribute type...** in the context menu option.

The *Convert attribute type* window is displayed:

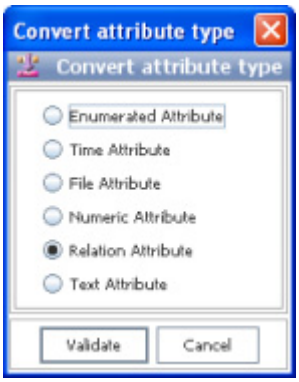

Fig 5.37 *Converting an attribute type*

2 Select the target type.

3 Click **Validate**.

The attribute's edit form is displayed so that you can review its parameters.

5.30 Setting attribute marks

Marks are predefined properties that can be applied to the attributes to specify how they should be processed in the application.

Marks fall into two categories:

- Most frequent marks
> Most frequent marks, page 120
- Other predefined marks
> Other predefined marks, page 122

In addition to the predefined marks, you can create custom marks:

> 5.31 Adding specific marks to an attribute, page 124

TO PERFORM THIS STEP

1. Right-click the attribute in the *Attributes* tab then select either **Change frequent marks**, or the **Change marks** in the context menu option.

   The *Change frequent marks* / *Change marks* window is displayed.
2. Select / clear the appropriate marks.
3. Click **Validate**.

Most frequent marks

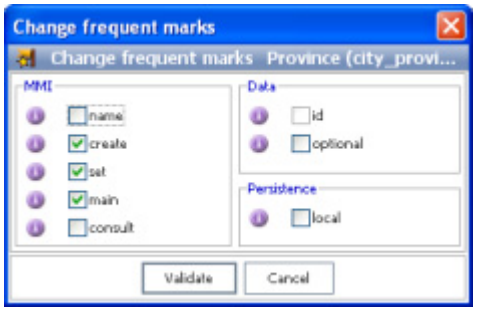

Fig 5.38 *Setting most frequent marks*

name - To specify that the attribute is part of (or is) the objects' visible names in the particular application class. Unlike the identifier, the name can be modified by the application and doesn't have to be unique. While the identifier represents the object internally, the name is displayed to the user when an action relates to the object (title bar, confirmation or error messages, label in the relations, etc). If there is no name mark, the identifier is displayed instead to the user.

create - To specify that the field is displayed to the user in a creation form (by default, the user is not prompted to enter data).

set - To specify that the field is displayed to the user in a modification form (by default the user is not allowed to modify data). Similar to the create attribute, but one does not necessarily modify the same fields. In particular, an identifier field (id) cannot have the set mark.

main - Shortcut equivalent to setting the table, sort, filter and find marks.

consult - To specify that the field is read-only in a create or edit form. This mark cannot be used with the create and set marks because their association does not make sense (a field cannot be both modifiable and read-only at the same time). It is equivalent to setting both the setConsult and the createConsult marks.

PLEASE NOTE The consult mark does not mean available in the read-only forms. By default, all the fields are available in read-only mode, except for the fields with the secret mark.

id - To specify the the attribute is part of (or is) the objects' identifier in the particular application class. Several attributes can have this mark. Being identifier implies that the field cannot be modified (*set='true'* is not a compatible mark) and that the combination of all the identifier fields of the object is unique for all of the concerned application class. An identifier field is the equivalent at application level to a primary key in a database. At least one field in the application class must have the id mark.

optional - To specify that the field is not mandatory and that the user can leave it blank. An optional field is displayed with a non-bold label in the forms while mandatory field labels are in bold. Not to be confused with the notNull mark which is on data provider side. An optional and notNull field for example is a field that can be specified by the user, and if no value is provided, the default value in the data model will be used instead.

local - To specify that the field is not obtained from a data provider but is calculated by the application.

Other predefined marks

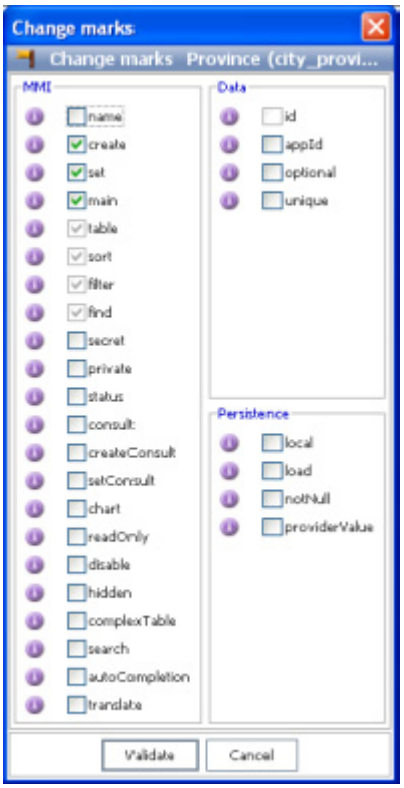

Fig 5.39 *Setting attribute mark*

table - To specify that the field value should be displayed in the list views (tables).

sort - To specify that the list view can be sorted according to the field. Clicking the column header allows you to change the sort criterion, and in fat client mode a menu entry is available in the **View / Sort** menu to apply this sort mode.

filter - To specify that the views can be filtered according to the field. The attribute will be available in the filter creation form. When no class field has the filter mark, the filter option is disabled and removed from the user interface for this class.

find - To specify that you can search for this attribute in a list view. The attribute will be available in the search creation form. When no class field has the find mark, the search option is disabled in the user interface for this class.

secret - To specify that the field value should never be displayed to the user. By default, all the fields are displayed in read-only forms EXCEPT for the fields with the *secret=true* mark, which are hidden. In the create / edit forms, the value input for the fields with the secret mark is hidden.

notNull - To specify that the field is required in create / edit forms. The field value is either provided by the application, or calculated via a default value. Not to be confused with the optional mark. The notNull mark does not mean that the field is required in a form. It just means that if the user does not provide a value, the default value will be used instead.

private - To specify that the field value is not duplicated for example when an action is copied.

unique - To specify that the field value is unique therefore no object of the same application class can have the same value for this field. For example, a user's login is unique.

status - To specify that an icon should be displayed in the views to represent the field status.

createConsult - To specify that the field is read-only in a create form. In creation mode, this makes sense with a field whose value is calculated. This mark cannot be used together with the create mark because their association does not make sense (a field cannot be modifiable in read-only mode and in creation mode at the same time).

setConsult - To specify that the field is read-only in an edit form. This mark cannot be used together with the set mark because their association does not make sense (a field cannot be both modifiable and read-only at the same time).

needPost - To specify that the modification of the field necessarily leads to the related form in the web display being submitted (POST) so as to perform a particular input control or with the aim of forcing the update of other fields in the same form. The generic cases (forms, rules, etc.) are managed automatically.

chart - To specify that the field is available in the statistic creation wizard.

readOnly - To specify that the field is read-only in a create / edit form. The field value is taken into account when the form is validated even if the field remains not editable.

disable - To specify that the field is displayed as inactive in a create / edit form. The field value is not taken into account when the form is validated, if the field remains disabled at this time.

hidden - To specify that the field is not displayed in read-only forms if the selected marks are not specified.

load - Indicates that the loading of the field is forced during load requests that do not specify the required fields.

ProviderValue - To specify that the field value is provided by a data provider.

appId - To create a secondary key whose uniqueness will also be checked. The secondary identifier of an object is the concatenation of the values corresponding to the fields with this mark, delimited by a character (a comma ',' by default).

complexTable - To specify that the field belongs to a complex table.

5.31 Adding specific marks to an attribute

In addition to the frequent marks and other predefined marks, you can create your own specifc marks and assign them to attributes.

Example: In a particular form, you want to display certain fields that share the same mark.

Specific marks are not used by the Application Engine code but can be used in the application's specific Java code.

TO PERFORM THIS STEP

1 Right-click the attribute in the *Attributes* tab then select **Specific marks** in the context menu option.

The *Specific marks* window is displayed:

Fig 5.40 *Creating a specific mark*

2 Select / clear the appropriate marks.

3 Type a name for the specific mark.

4 Click **Add**.

5 Click **Validate**.

5.32 Associating specific graphical components to an attribute

You can specify which graphical component will be used to represent a particular field.

Various components can be specified depending on whether the field is in read-only, edit or filter mode. Alternatively you can specify a generic component that is suitable for all modes.

TO PERFORM THIS STEP

1 Click **Specific graphical component** in the *Attributes* tab toolbar.

The *Specific graphical component* window is displayed:

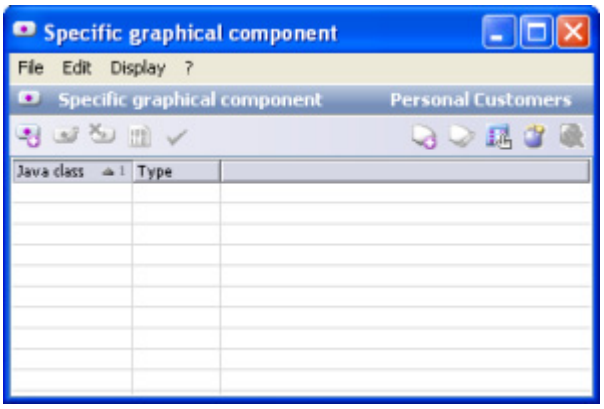

Fig 5.41 *Using specific graphical components for the attributes (1/2)*

2 Click **Generate graphical component**.

The following window is displayed:

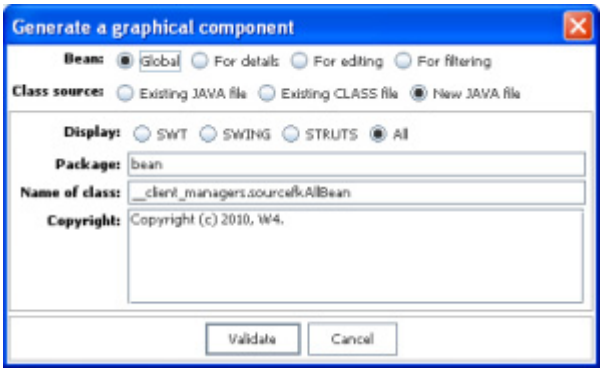

Fig 5.42 *Using specific graphical components for the attributes (2/2)*

3 Set the fields:

Bean - Specify for which purpose the component will be used (read-only, edit, filter, or general purpose).

Class source

- **Existing Java file** - To specify the Java file of an existing component.
- **Existing CLASS file** - To specify the Java class of an existing component.
- **New Java file** - Select this option if there is no preexisting component for the attribute.

Application Composer is going to generate a Java file for the component. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Displays - Select the appropriate viewer.

Package - Name of the corresponding package, by default: <application id>.bean.

Name of the class - Name of the generated class, by default: <field id>AllBean.

Copyright - Text to be included at the beginning of the Java file.

In the Specific graphical component window, you can:

- Edit the currently selected Java class

- Remove the currently selected Java class
- Add specific data to the currently selected graphical component

> 5.33 Adding specific data, page 126

- Compile the currently selected Java class

4 Click **Validate**.

5.33 Adding specific data

You can have the class manage data intended for the specific application code.

Specific data is specified as key/value pairs and, just as with specific marks, using specific data is up to application developers.

TO PERFORM THIS STEP

1 Right-click the attribute in the *Attributes* tab then select the **Application data** context menu option.

The *Application data* window is displayed:

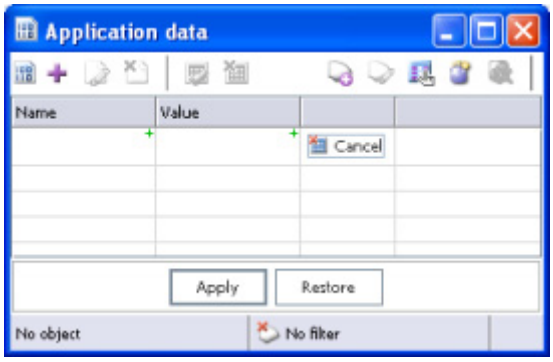

Fig 5.43 *Adding specific data*

2 Click **New...**

The *New : Data* window is displayed:

3 Set the fields:

Name - Name of the data item.

Value - Value of the data item.

4 Click **Validate**.

5 Back in the *Application data* window, click **Apply**.

5.34 Editing formatting constraints

You can set position constraint for the fields in the forms.

The Java equivalent to this functionality would be GridBagConstraint, associated to GridBagLayout.

TO PERFORM THIS STEP

1 Right-click the attribute in the *Attributes* tab then select the **Edit field display constraint** context menu option.

The *Modify : Formatting constraints* window is displayed:

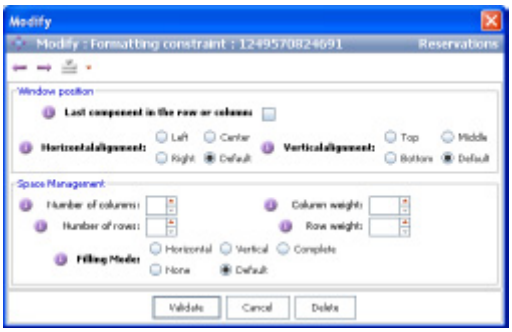

Fig 5.44 *Editing formatting constraints*

2 Set the fields:

Last component in the row or column - Specifies whether a line break should be added immediately after the component in the form.

Horizontal alignment - **Vertical alignment** - Specifies the align mode for the component, within the space that is allocated to it. If the default mode is selected, the align mode is determined by the form.

Number of columns - A form can be seen as a grid upon which labels and component take up a number of cells. If the number of columns is specified, the component or the label takes up the specified number of horizontal cells.

Column weight - If free space is available horizontally, this weight in percentage or in barycentric indicates how much free space should be allocated to the component.

Number of rows - A form can be seen as a grid upon which labels and component take up a number of cells. If the number of rows is specified, the component or the label takes up the specified number of vertical cells.

Row weight - If free space is available vertically, this weight in percentage or in barycentric indicates how much free space should be allocated to the component.

Filling mode - Specifies the component position in the form cell if the component size does not match the cell size. The complete mode attempts to change the component size so that it takes up all the cell space.

3 Click **Validate**.

5.35 Setting attribute controls

TO PERFORM THIS STEP

1 Click **Set controls...** in the edit form toolbar.

The *Set controls* window is displayed:

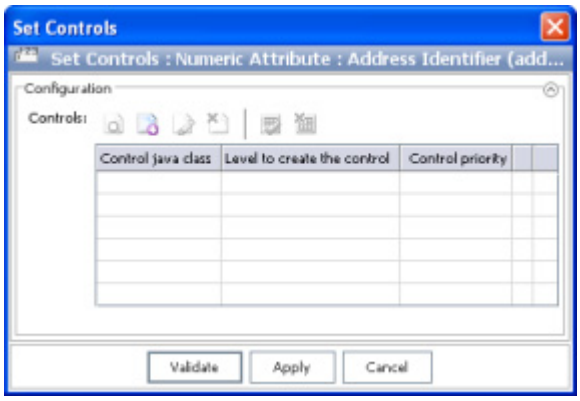

Fig 5.45 *Setting attribute controls (1/2)*

2 Click **New...**

The *New : Control* window is displayed:

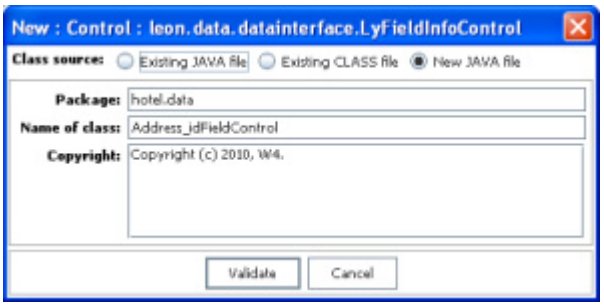

Fig 5.46 *Setting attribute controls (2/2)*

3 Set the fields:

Existing JAVA file - To specify a preexisting Java file for the application control. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class for the control. The appropriate class should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new file for the control.

Application Composer is going to generate the Java class in the application directory and create the appropriate directories for the class package. When the file has been generated, the class is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: *<application id>.data*. The data directory will be created to the root of the application.

Name of the class - The class name, by default: *IdentifierFieldControl* with the identifier's first letter in uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the `studio_composer.ini` file as the efault value.

4 Click **Validate**.

5 Back in the *Set controls* window, click **Validate**.

5.36 Setting attribute labels

A dynamic label is a label calculated at runtime. It contains a value that is a String with both fixed and variable parts, as well as parameters allowing, at runtime, to replace these variable parts by data of dynamic nature that depends on the use context. This label can change when the data to which it applies, is modified.

TO PERFORM THIS STEP

1 Click **Set labels...** in the edit form toolbar.

The *Set labels* window is displayed:

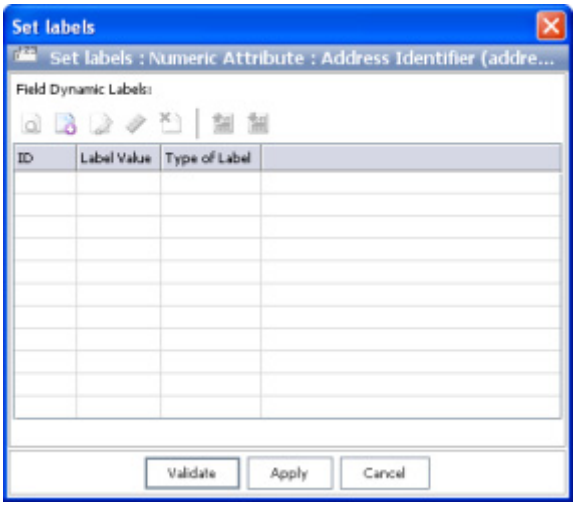

Fig 5.47 *Setting attribute labels (1/2)*

2 Click **New...**

The *New : Dynamic label* window is displayed:

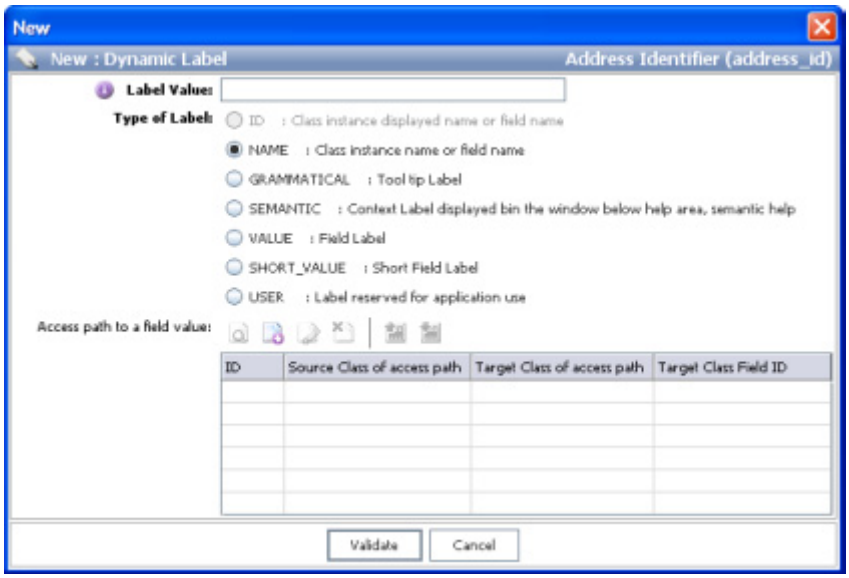

Fig 5.48 *Setting attribute labels (2/2)*

3 Set the fields:

Label value - The value of the label (or the corresponding entry in the dictionary). It should contain variable parts such as {i}, as with standard management of Java messages. The number of variable parts should match the number of access paths to be specified in the lower part of the form.

ID - The ID's visible name of the class instance. This label takes precedence over the combination of fields with the *id* mark. This label should be defined only with fields with the *id* mark.

NAME - The instance name of the class or the name of the fields. This label takes precedence over the combination of fields with the *name* mark and also over the *NAME* item.

GRAMMATICAL - The label displayed in the tooltips. This label takes precedence over the tooltip item.

SEMANTIC - Roll-over label displayed over the help line to the bottom of the forms.

VALUE - Presentation label for the field value.

SHORT_VALUE - Short presentation label for the field value.

USER - Application label.

4 To create an access path to a field value:

> Creating access paths, page 131

5 Click **Validate**.

Creating access paths

TO PERFORM THIS STEP

1 In the *New : Dynamic label* window, click **New**.

The *New : Access path* window is displayed:

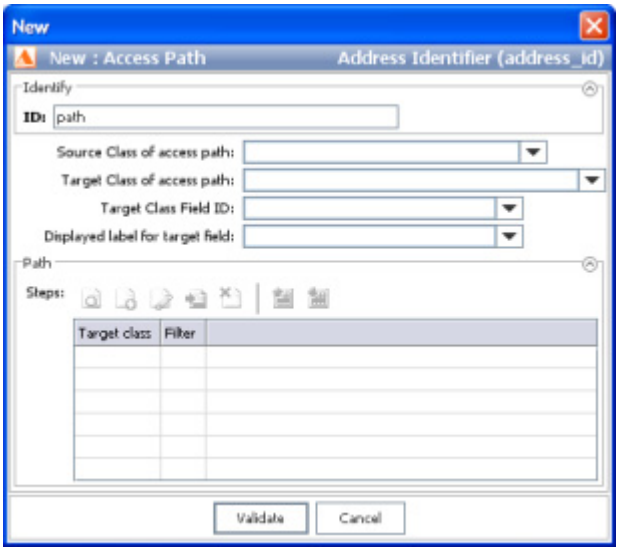

Fig 5.49 *Creating an access path*

2 Set the fields:

ID - The access path identifier.

Source class of access path - Source class of the access path, for an absolute access path.

Target class of access path - Target class of the access path. If not specified the current class will be used.

Target class of field ID - Final item of the access path. If not specified, the access path is equivalent to a route.

Displayed label for target field - Dynamic label to use for the target field.

3 To create steps for the access path:

3.1 Click **New** in the *Path* area.

The *Add step* window is displayed:

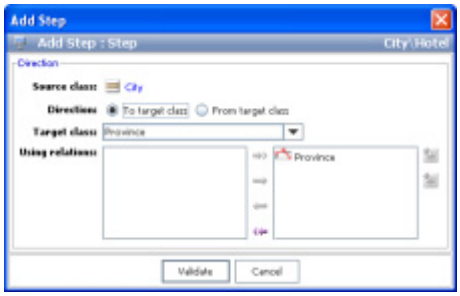

Fig 5.50 *Creating a step*

3.2 Set the fields:

To the target class - The target class of the step is the destination.

From the target class - The target class of the step is the origin.

Target class - Target class of the step.

Using relation - Relation field to link the source class to the target class.

3.3 Click **Validate**.

4 Back in the *New : Access path* window, click **Validate**.

5.37 Specifying a cache policy for an attribute

TO PERFORM THIS STEP

1 Click **Cache management...** in the edit form toolbar.

The *Cache management* window is displayed:

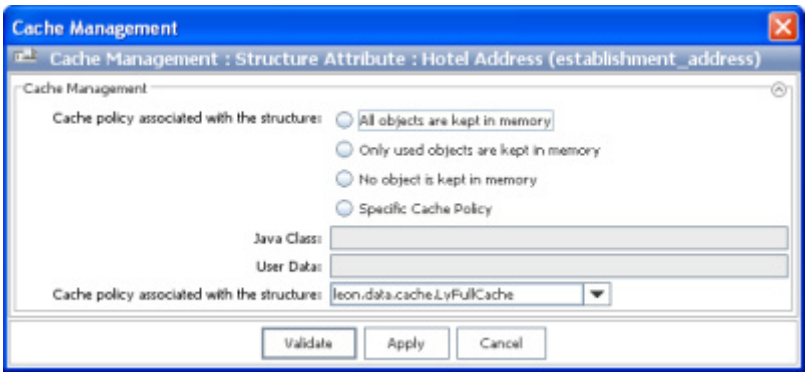

Fig 5.51 *Specifying a cache policy*

2 Set the fields:

All the objects are kept in memory (FullCache) - An object is loaded only if it is unknown, and it is never unloaded. This cache is efficient provided data volume is not too large.

Only used objects are kept in memory (Auto cache) - Only the objects used by the application code or displayed in the views are stored in the memory. When closing the view or when releasing the object list, the data is released from the memory.

No object is kept in memory (No Cache) - The data provider is accessed whenever necessary.

Specific cache policy - To specify a preexisting class for the cache. The appropriate class should be selected via the **Java class** field. The class should extend the `leon.data.LyCache` class.

Java class - The class name. Change as appropriate.

User data - This field can be used to configure a specific cache (number of managed objects, refresh delay, name of local cache, etc).

Generic marks

- **Local class** - The class is not linked to the data provider (*local* means that this class does not exist in the database). Therefore, the cache is always a full cache. There is no call to the physical link.
- **Load at start time** - Specifies whether data is loaded as soon as the application starts, or the first time the user invokes it.

Cache policy associated with the structure

3 Click **Validate**.

5.38 Setting attribute rules

TO PERFORM THIS STEP

1 Click **Set [...] rules...** in the edit form toolbar.

The *Set [...] rules* window is displayed:

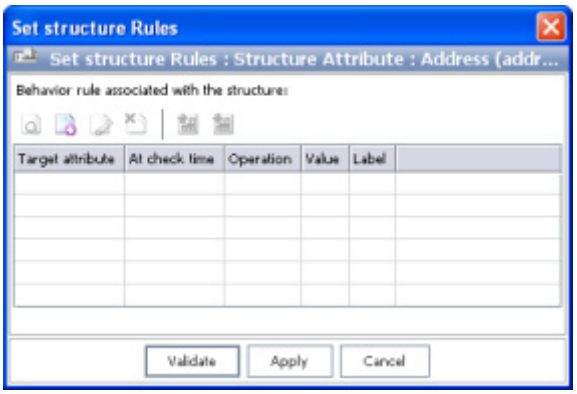

Fig 5.52 *Setting attribute rules (1/2)*

2 Click **New**.

The *New : Rules* window is displayed:

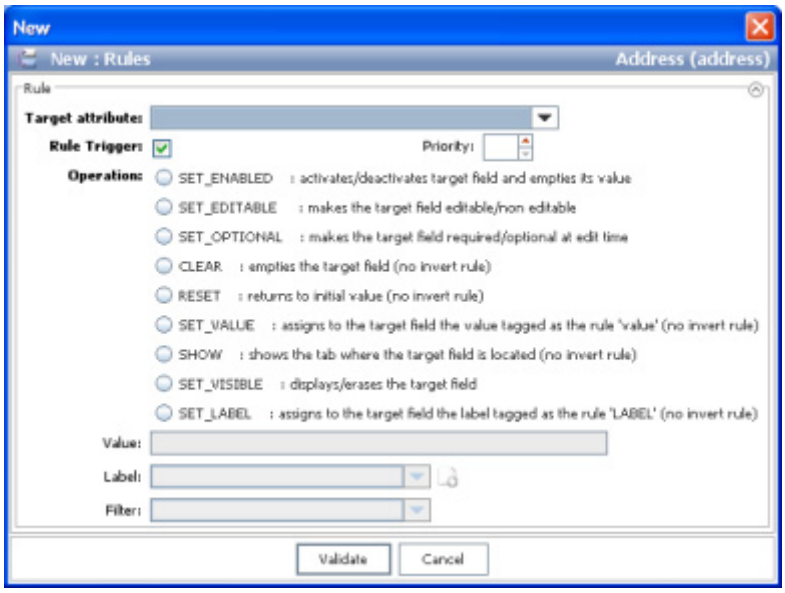

Fig 5.53 *Setting attribute rules (2/2)*

3 Set the fields:

Target attribute - Specifies the field to which the rule applies.

Role trigger

Priority - Priority level (relative to other rules).

Operation

- **SET_ENABLED** - Enables/disables the field and clears its value.
- **SET_EDITABLE** - Makes the field editable/not editable.
- **SET_OPTIONAL** - Makes the field optional/mandatory.
- **CLEAR** - Empties the field (no reverse rule).
- **RESET** - Returns the field to its original value (no reverse rule).
- **SET_VALUE** - Assigns to the field the value specified for the **Value** field (no reverse rule).
- **SHOW** - Displays the field to the user, i.e. displays the tab in which the field is located (no reverse rule).
- **SET_VISIBLE** - Displays/Hides the field.
- **SET_LABEL** - Assigns to the field the label that is specified for the **Label** field (no reverse rule).

Value - The value to be specified with the SET_VALUE operation.

Label - The value to be specified with the SET_LABEL operation.

Alternatively, click **New** to create a new label.

For further information on how to create a label:

> 5.36 Setting attribute labels, page 129

Filter - To set a filter for the control data.

4 Click **Validate**.

5 Back in the *Set [...] rules* window, click **Validate**.

Working with routes

Routes are context-sensitive relations between application classes.

A route is used to set up a link between two application classes to obtain a target class object based on the source object context.

In this chapter we will cover the following topics:

To display the **Routes** tab, select **Windows** ▸ **Routes**.

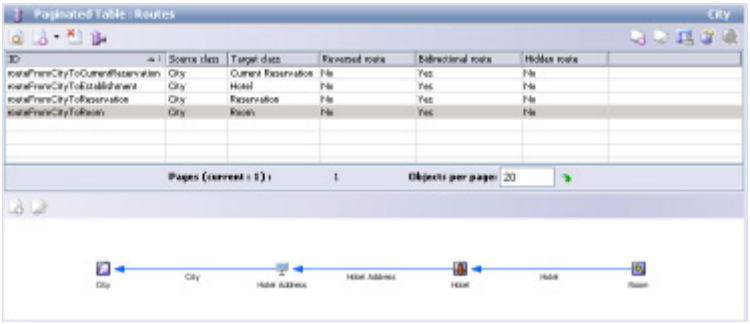

Fig 6.1 *The Routes tab*

The lower area provides a graphical representation of the route steps and allows you to check whether the route is complete.

Viewing details of a route

The **Details** icon in the *Routes* tab tool bar displays the details of the currently selected route.

Deleting a route

The **Delete** icon in the *Routes* tab tool bar deletes the currently selected route.

Making a route bi-directional

A route only applies to one direction. The **Make the route bidirectional** icon in the *Routes* tab tool bar allows you to make a route valid in the opposite direction i.e. from the target class to the current class.

Applying a filter to a step

The steps to follow are similar to those for creating a project filter:

> 3.5 Creating filters for a project, page 46

6.1 Creating a route

TO PERFORM THIS STEP

1 Click **New** in the *Routes* tab tool bar.

The *New : routes* window is displayed.

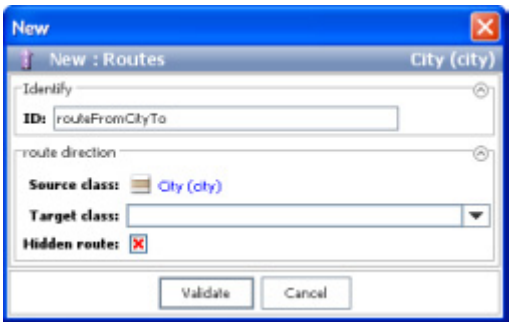

Fig 6.2 *Creating a route*

2 Set the fields:

ID - Application Composer automatically sets the fields to: routeFrom<original class name>To<target class name>. Change as appropriate.

Target class - To specify the target class for the route.

Hidden route - To specify whether the route is hidden, i.e. the route is not used for cross-reference calculation. This option is used to specify routes in the meta-model, and to be used via programming only.

3 Click **Validate**.

6.2 Creating a reverse route

A route is reversed if the calculation starts from the target class, or from the current class. The form is the same as the create form of a non-reverse route.

> 6.1 Creating a route, page 137

TO PERFORM THIS STEP

1 Click **Create a reverse route** in the *Routes* tab tool bar.

The *Create a reverse route* window is displayed.

2 Set the fields.

> 6.1 Creating a route, page 137

3 Click **Validate**.

6.3 Creating a step

TO PERFORM THIS STEP

1 Select the route in the table view of the *Routes* tab.

The route is displayed in the lower area.

The dotted line means the route is not fully specified.

2 Click **Add step...**

The *Add step* window is displayed:

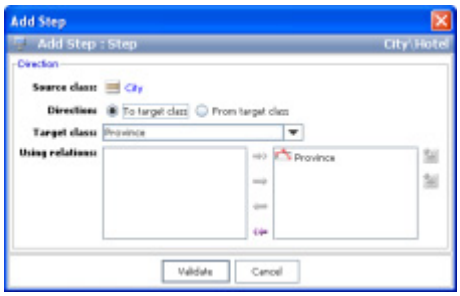

Fig 6.3 *Creating a step*

3 Set the fields:

Direction - The step direction: To the target class or From the target class.

Target class - The step target.

Using relations - The list of the relation fields used to link the target class to the original class.

4 Click **Validate**.

Working with actions

Actions are listed in both the *Navigation tree* and the *Actions* tab.

Whereas the Actions tab displays the actions associated with logical classes, the navigation tree displays the actions associated with the application's sequence of actions.

As opposed to the Navigation tree, which lets you add an action at a specific location, the Actions tab lets you create actions at the class, project or application level.

In this chapter we will cover the following topics:

- How to create actions:
  - 7.1 Creating a simple action, page 146
  - 7.2 Creating a composite action, page 150
  - 7.3 Creating a tab action, page 152
  - 7.4 Creating a reference to an existing action, page 153
  - 7.5 Creating child actions, page 154
- How to perform basic actions:
  - Viewing details of an action, page 142
  - Editing an action, page 142
  - Removing an action, page 143
  - Specifying tooltips for an action, page 143
  - Specifying keyboard shortcuts for an action, page 143
  - Changing action order, page 143
  - Copying the target action, page 143
  - Generating the default actions, page 144
  - Setting the default action, page 144
  - Adding specific marks for an action, page 144
  - Specifying application data for an action, page 145
  - Specifying the root action, page 145
  - Specifying an XML view of the action, page 145
  - Previewing an action, page 145

How to implement advanced action-related features:

To display the Navigation tree, select **Windows** ▸ **Navigation tree**.

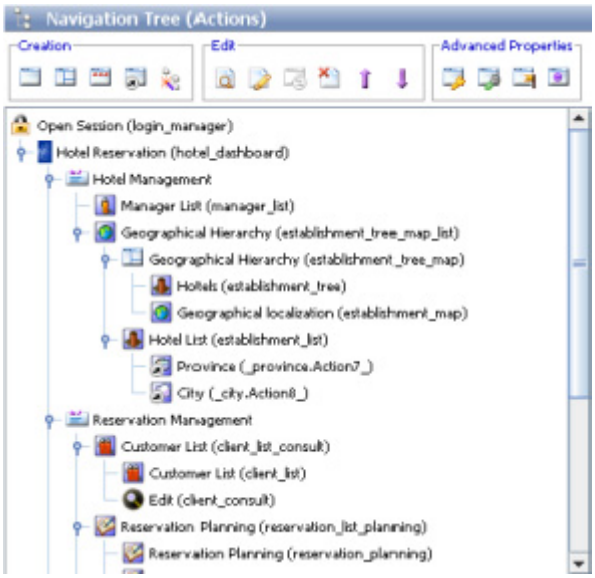

Fig 7.1 *The navigation tree*

To display the **Actions** area, select **Windows** ▸ **Data model** ▸ **Actions**.

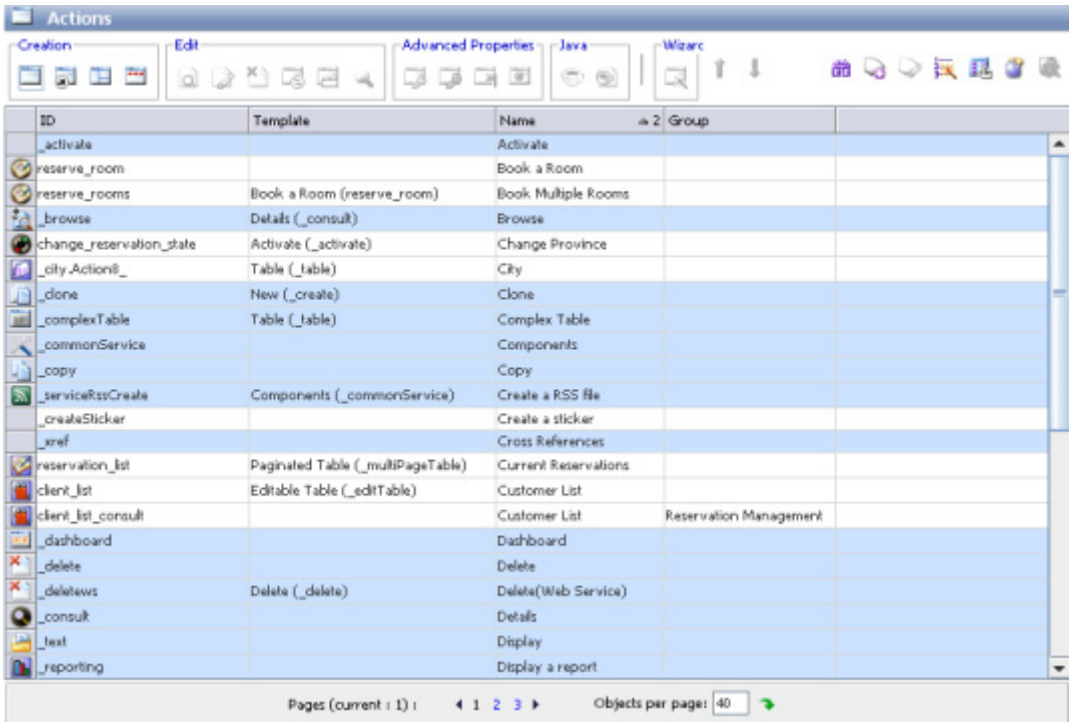

Fig 7.2 *The Actions tab*

NOTE Action background colors:

- Blue: A default Application Engine action, i.e. a reference to a generic Application Engine action
- Yellow: A reference to a specific application action
- White: A local action

Viewing details of an action

You can view the details of a preexisting action via **Details**, either in the Navigation tree tool bar, or in the Actions tab tool bar.

Editing an action

You can edit a preexisting action via **Modify**, either in the Navigation tree tool bar, or in the Actions tab tool bar.

Removing an action

You can remove an action and its child actions via **Delete**, either in the Navigation tree tool bar, or in the Actions tab tool bar.

Specifying tooltips for an action

You can add a tooltip to be displayed when the user hovers over an action for a while, via the **Tooltips...** context menu option, either in the Navigation tree, or in the Actions tab.

Specifying keyboard shortcuts for an action

You can specify a keyboard combination such as CTRL + <KEY> to run the action, via the **Keyboard shortcut...** context menu option, either in the Navigation tree, or in the Actions tab.

Changing action order

You can change the order of the actions in the application class via **Move up** and **Move down** either in the Navigation tree tool bar, or in the Actions tab tool bar.

Copying the target action

If the action is a generic Application Composer action or if the selected action is a link to an action shared by several application classes (displayed on a yellow background) you can make a local copy of the target action, via **Copy referenced action** either in the Navigation tree tool bar, or in the Actions tab tool bar.

The generic actions and the shared actions targeted by links are defined only once in the application: All the application classes using them point to the same action. If you want to modify a shared action for an application class, you can request a local copy of the target

action by using this feature. This creates a local action with the same properties as the target action. However it is not shared and can be modified.

Generating the default actions

You can generate the default actions for your application via  **Generate default GUI** either in the Navigation tree tool bar, or in the Actions tab tool bar.

For generation to run successfully, make sure that the classes have been defined and that each class has an identifier field.

The MMI generator produces a dashboard as the root action. It contains a table action for each application class.

Setting the default action

You can set the mark default=true for an action, via the **Set the default action...** context menu option, either in the Navigation tree, or in the Actions tab.

This triggers the action when you double-click the class containing this action.

Adding specific marks for an action

You can add specific marks to an action, via the  **Specific marks...** context menu option, either in the Navigation tree, or in the Actions tab.

Specific marks are not used by the Application Engine code but can be used in the application's specific Java code.

The approach is similar as when adding specific marks to a class:

> 4.10 Adding specific marks for a class, page 60

Specifying application data for an action

You can add to the action data that is intended for the application's specific code, via the **Application data...** context menu option, either in the Navigation tree, or in the Actions tab.

Application data is specified as key/value pairs and, just as with specific marks, using application data is up to application developers.

The approach is similar as when specifying application data for a class:

> 4.11 Specifying application data, page 61

Specifying the root action

You can specify the application's root action, via the **Define a root action...** context menu option, either in the Navigation tree, or in the Actions tab.

The root action is the first action to be executed at Application Engine startup. it is usually a login action, a dashboard-type view, etc.

Specifying an XML view of the action

You can specify an XML view as a substitute for the builder, via the **Action view...** context menu option, either in the Navigation tree, or in the Actions tab.

This allows you to define the form and the specific content of the view.

Application Composer automatically generates the XML file of the default action view.

Previewing an action

You can preview an action, via the **Preview** context menu option, either in the Navigation tree, or in the Actions tab.

This displays a preview of the action with no data.

7.1 Creating a simple action

TO PERFORM THIS STEP

1 Click **Create an action** either the Navigation tree tool bar, or the Actions tab tool bar.

The first screen of the *Create an action* wizard is displayed:

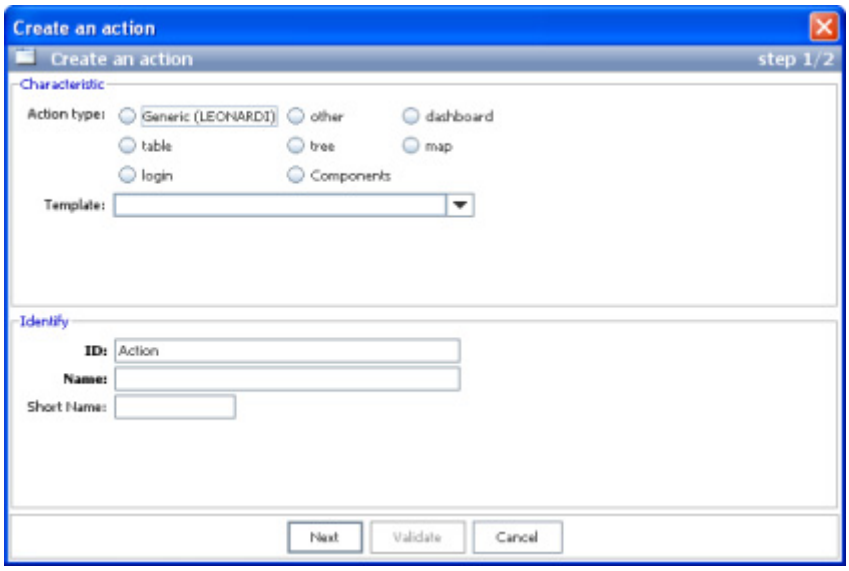

Fig 7.3 *Creating a simple action*

2 Set the fields:

Action type

- **Generic** - To select one of the Application Engine's generic actions.
- **Other** - To select a user action that will be used as a template.
- **Dashboard**
- **Table**
- **Tree**
- **Map**
- **Login**
- **Components**

Template - The action template.

ID - The action identifier. Its default value depends on the selected type.

Name - The action name.

Short name - The action alias.

3 Click **Next**.

The second screen of the *Create an action* wizard is displayed.

This screen depends on the action type. Please refer to the appropriate section:

> Creating a simple action (generic / other / component), page 147
> Creating a simple action (dashboard), page 147
> Creating a simple action (table), page 148
> Creating a simple action (tree), page 148
> Creating a simple action (map), page 149
> Creating a simple action (login), page 149

Creating a simple action (generic / other / component)

TO PERFORM THIS STEP

1 In the second screen of the *Create an action* wizard, set the fields:

Group - The group the action belongs to.

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

2 Click **Validate**.

Creating a simple action (dashboard)

TO PERFORM THIS STEP

1 In the second screen of the *Create an action* wizard, set the fields:

Group - The group the action belongs to.

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Open actions in the same view

Action displayed...

Use expand bars

Expand bars opened

Open actions in tab

Make a snapshot

- **ALL**
- **MINIMUM**
- **NO**
- **NONE**
- **SKIP**

2 Click **Validate**.

Creating a simple action (table)

TO PERFORM THIS STEP

1 In the second screen of the *Create an action* wizard, set the fields:

Group - The group the action belongs to.

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Target class

Attribute marks

Status field mark

Class filter

Filter

Sort

Column width

Number of objects per page

Target sub attribute

Show filter...

Make a snapshot

- **ALL**
- **MINIMUM**
- **NO**
- **NONE**
- **SKIP**

DHTML mode

_allowedNavigation

2 Click **Validate**.

Creating a simple action (tree)

TO PERFORM THIS STEP

1 In the second screen of the *Create an action* wizard, set the fields:

Group - The group the action belongs to.

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Target class

Filter

Make a snapshot

- **ALL**

- **MINIMUM**
- **NO**
- **NONE**
- **SKIP**

_discoverOnclick

2 Click **Validate**.

Creating a simple action (map)

TO PERFORM THIS STEP

1 In the second screen of the *Create an action* wizard, set the fields:

Group - The group the action belongs to.

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Target class

XML file to describe background objects

Make a snapshot

- **ALL**
- **MINIMUM**
- **NO**
- **NONE**
- **SKIP**

2 Click **Validate**.

Creating a simple action (login)

TO PERFORM THIS STEP

1 In the second screen of the *Create an action* wizard, set the fields:

Group - The group the action belongs to.

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Login class

Login attribute

Password attribute

Maximum login retry

Allow users creation

Allow session reconnection

Display with SSO mode

- **Always**
- **IF_NOT_LOGGED**

Make a snapshot

- **ALL**
- **MINIMUM**
- **NO**
- **NONE**
- **SKIP**

2 Click **Validate**.

7.2 Creating a composite action

A compound action displays two actions simultaneously.

As an action in the compound action can itself be a compound action, you can display multiple views with more than two actions.

TO PERFORM THIS STEP

1 Click **Create a compound action** either the Navigation tree tool bar, or the Actions tab tool bar.

The *Create a compound action* window is displayed:

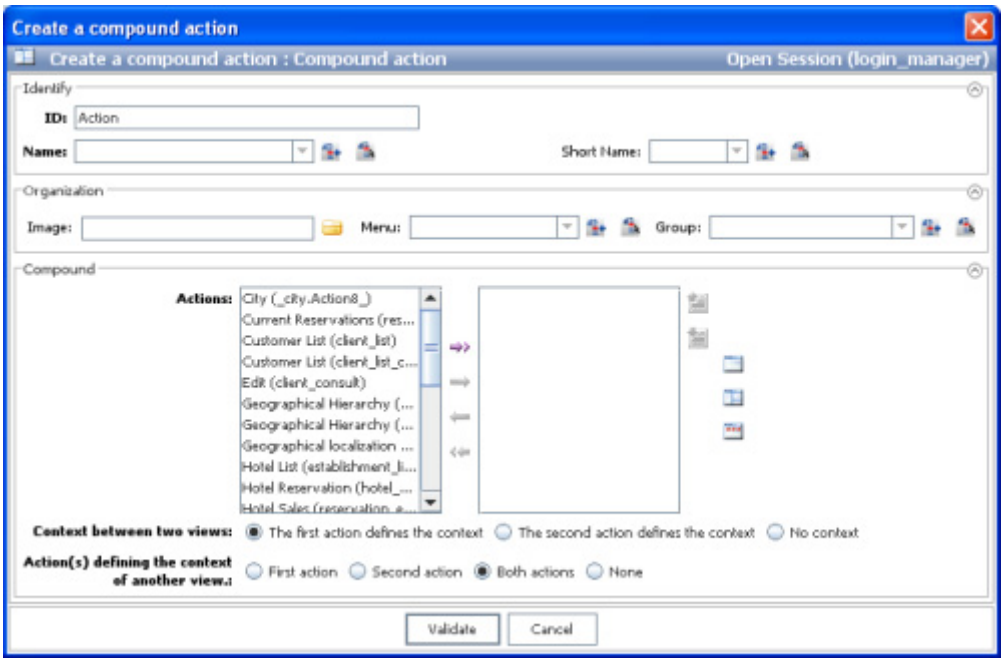

Fig 7.4 *Creating a composite action*

2 Set the fields.

ID

Name

Short name

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Group - The group the action belongs to.

Actions

Context between the views

- **The first action defines the context** - If you select this option, the second view will depend on the object selected in the first one.
- **The second action defines the context** - If you select this option, the first view will depend on the object selected in the second one.
- **No context** - If you select this option, both actions are fully independent of one another.

Action(s) defining the context for another view - To specify which actions are contextual (context-dependent).

3 Click **Validate**.

7.3 Creating a tab action

A tab action is represented by a view made up of several tabs, where each tab provides a specific action.

TO PERFORM THIS STEP

1 Click **Create a tab action** either the Navigation tree tool bar, or the Actions tab tool bar.

The *Create a tab action* window is displayed:

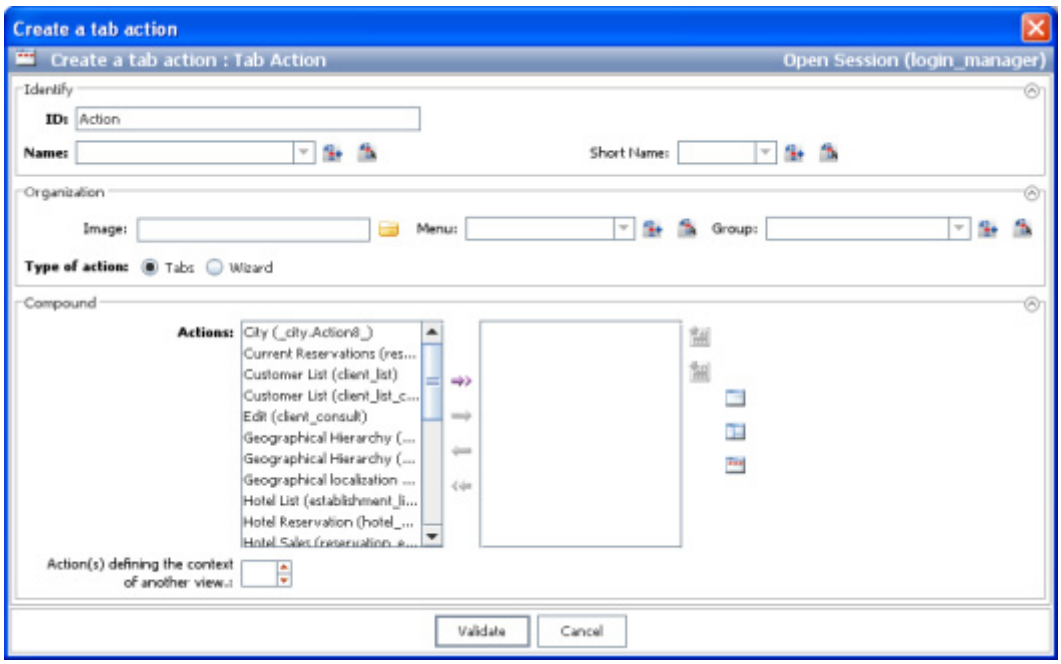

Fig 7.5 *Creating a tab action*

2 Set the fields:

ID

Name

Short name

Image - The icon representing the action. Click to browse for the appropriate image.

Menu - The menu within which the action will be available.

Group - The group the action belongs to.

Type of actions

- **tabs**
- **Wizard**

Actions - Select which actions should be displayed in the various tabs.

Action(s) defining the context for another view - To specify which actions are context-dependent. By default, all of them are context-dependent. If no action is context-dependent, set the field to *-1*.

3 Click **Validate**.

7.4 Creating a reference to an existing action

You can add a link to an action that is already specified either in the application, or as part of the generic actions.

Action references are displayed on a yellow background in the Actions tab.

TO PERFORM THIS STEP

1 Click **Create action shortcut** either the Navigation tree tool bar, or the Actions tab tool bar.

The *Create an action reference* window is displayed:

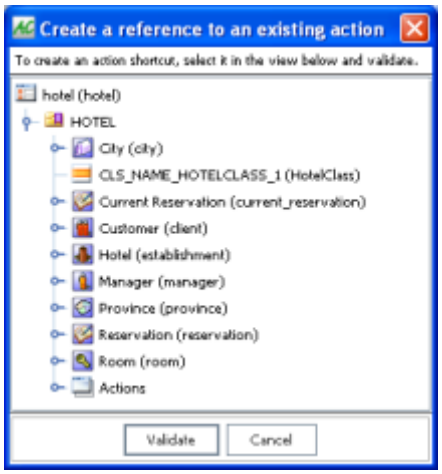

Fig 7.6 *Creating a reference on an existing action*

2 Select the target action.

3 Click **Validate**.

7.5 Creating child actions

The **Child actions** option in the context-sensitive menu (navigation tree) or in the **Actions** tab tool bar allows you to create child actions for the selected action. A table of the actions opens, similar to the one for the class actions, where the tool bar is similar to the one for the navigation tree.

TO PERFORM THIS STEP

1 Click **Child actions** either the Navigation tree tool bar, or the Actions tab tool bar.

The *Children actions* window is displayed:

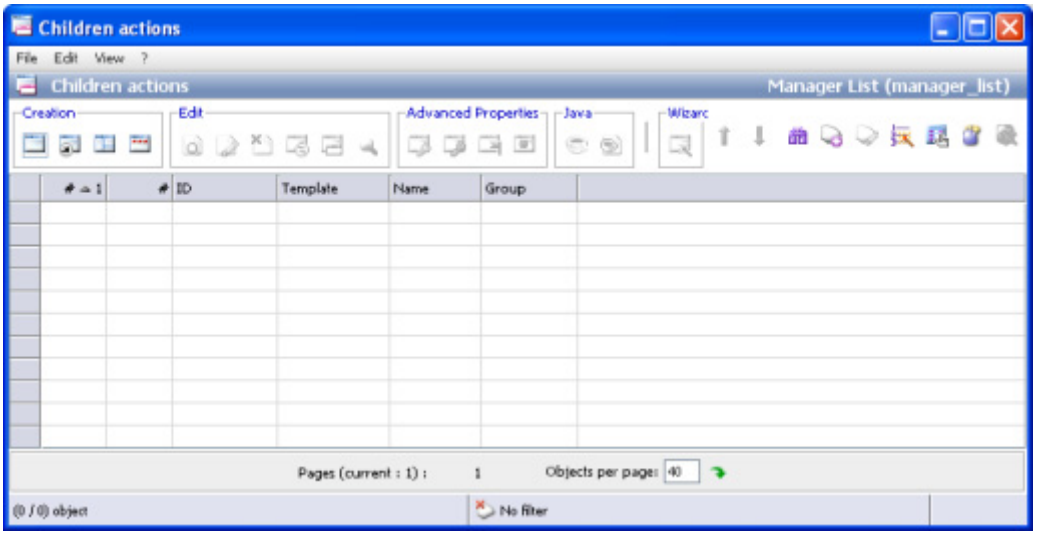

Fig 7.7 *Creating child actions*

2 Create the child actions.

You can create:

- Simple actions
  > 7.1 Creating a simple action, page 146
- Compound actions
  > 7.2 Creating a composite action, page 150
- Tab actions
  > 7.3 Creating a tab action, page 152
- Action references
  > 7.4 Creating a reference to an existing action, page 153

7.6 Customizing the view for an action

When invoked, some actions generate a view (e.g. a form, a main view, etc), which can be customized.

TO PERFORM THIS STEP

1 Click **View structure...** either the Navigation tree tool bar, or the Actions tab tool bar.

The *View structure* window is displayed:

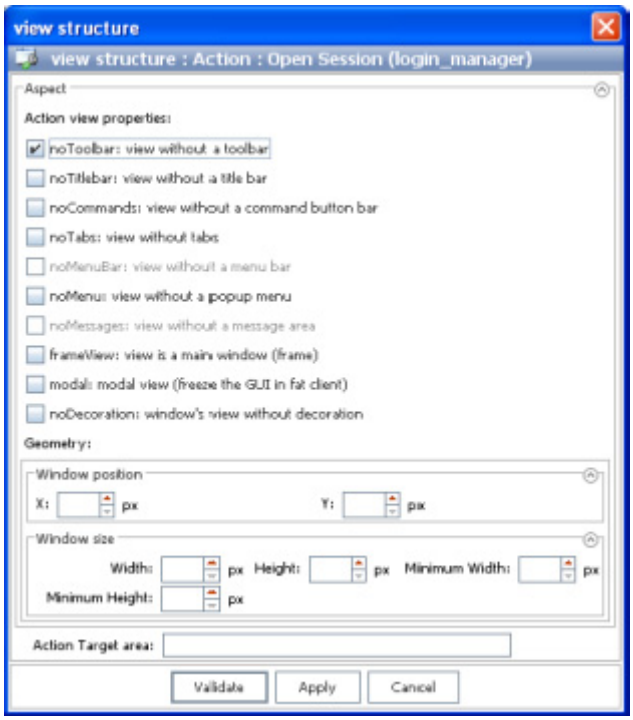

Fig 7.8 *Customizing the view for an action*

2 Set the fields:

Action view properties

- **noToolbar** - To specify that the view should have no tool bar.
- **noTitlebar** - To specify that the view should have no title bar.
- **noCommands** - To specify that the view should have no command bar. The command bar can be removed only from the views where it does not have any input validation role (e.g. a read-only view).
- **noTabs** - To specify that the view should have no tab if some fields have been set to be displayed in tabs.
- **noMenuBar** - To specify that the view should have no context-sensitive menus (popup menu).
- **noMenu** -
- **noMessages** - To specify that the view should have no message bar. This is only relevant for the views of type Frame.

- **frameView** - To specify that the view should be of type Frame (main view). Otherwise, the view is of type Dialog (secondary view). This is only relevant for the upper level actions.
- **modal** - To specify that the view is modal, i.e. the view must be completed for the application to continue. This is only relevant for the viewers of type fat client (AWT, SWING, SWT), for the views of type dialog (secondary views) and for the upper level actions.
- **noDecoration** -

Window position - To specify the position of the view when it is displayed. If x remains empty, the view will be centered horizontally (and centered vertically if y remains empty).

Window size - To specify the size of the view. If the width remains empty or equal to zero, the view will take up all the available width of the screen (and all the available height of the screen if the height remains empty or equal to zero).

Position (available only for composite actions) - The position of the views that make up the action. If horizontal position is selected, the views will be placed one next to the other. If vertical position is selected they will be placed one above the other.

Sizes (available only for composite actions) - The proportion (as a percentage) of the space taken up by each of the views. For example *40 60* means that the first view takes up 40% of the total space available in the composite view and the second 60%.

Action target area - The area in which the action will be executed. This is only relevant for the Eclipse plugin viewer.

7.7 Specifying apply conditions for the actions

You can specify the conditions that should be met for the action to be allowed (minimum or maximum number of selected objects, context, filter, etc).

TO PERFORM THIS STEP

1 Click **Apply conditions...** either the Navigation tree tool bar, or the Actions tab tool bar.

The *View structure* window is displayed:

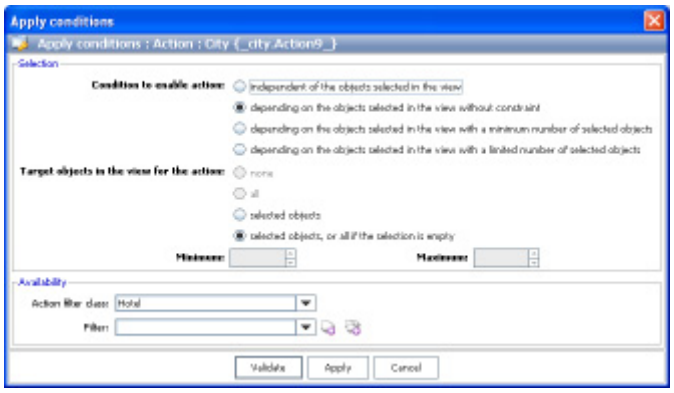

Fig 7.9 *Specifying apply conditions for the actions*

2 Set the fields:

The fields **Condition to enable action**, **Target objects in the view for the action**, **Minimum** and **Maximum** are bound together in the following way:

If you first select the option **Independent of the objects selected in the view** you can choose between:

- None
- All

If you first select the option **Dependent of the objects selected in the view without constraint** you can choose between:

- Selected objects
- Selected objects or all if the list is empty

If you first select the option Dependent of the objects selected in the view with a minimum number of selected objects, specify the minimum number of selected objects in the **Minimum** field.

If you first select the option **Dependent of the objects selected in the view with a limited number of selected objects**, specify the minimum and the maximum number of selected objects in the **Minimum** and **Maximum** fields.

In the **Availability** area, you can specify the following:

- **Action filter class** - The class to which the filter applies.
- **Filter** - The filter to determine the objects for which the action can be triggered.

3 Click **Validate**.

7.8 Setting action marks

TO PERFORM THIS STEP

1 Right-click the appropriate action either in the Navigation tree, or in the Actions tab then select **Set marks...** from the context menu.

The *Set marks* window is displayed:

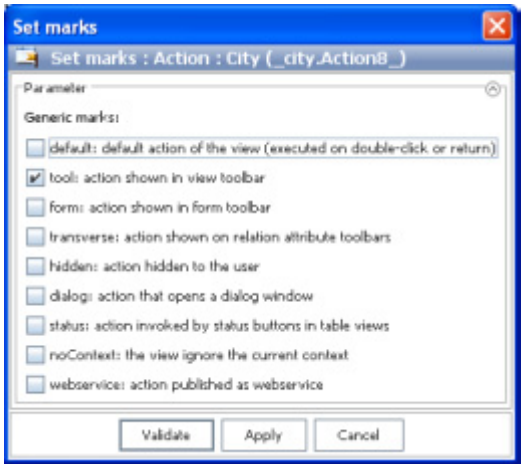

Fig 7.10 *Setting action marks*

2 Set the fields:

default - When you double-click an object in a view, the action that is run is by default the read-only action. However you can choose another action, by applying the *default* mark to it.

tool - To specify that the action can be displayed as an icon on the tool bar. The default value is false.

form - To specify that the action can be displayed as an icon in forms with a tool bar (read-only, edit).

transverse - To specify that the action can be used as a transverse action, i.e. as an action associated with a relation, and which is available via the create/edit forms.

hidden - To specify that the action is not visible for the user and therefore is not available in the views that are displayed to the user.

dialog - To specify that the action opens up a dialog. Therefore, the action label should be followed by ellipses (…).

status - To specify the action that is invoked when selecting the status icon of an object in a table. If no action has this mark, the default action in the view is started.

noContext

webservice

3 Click **Validate**.

7.9 Specifying specific resources for an action

You can customize the application's look and behavior. Resources are specified for the whole application via the **Windows** ▸ **Resources** menu.

Resources can also be modified for a particular action.

TO PERFORM THIS STEP

1 Click **Specific resources...** either the Navigation tree tool bar, or the Actions tab tool bar.

The *Specifc resources* window is displayed:

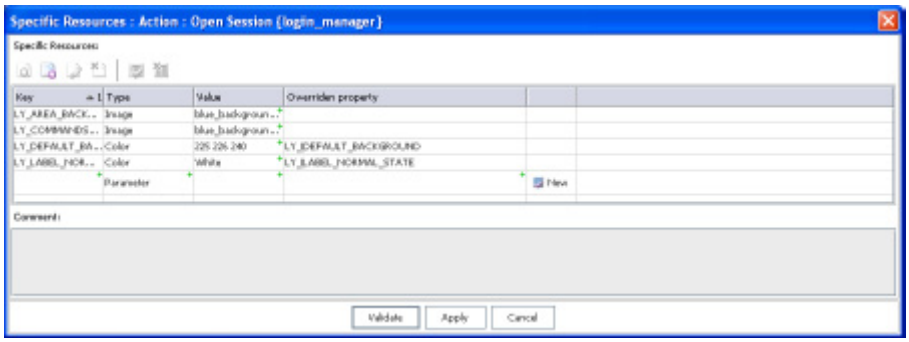

Fig 7.11 *Specifying specific resources for an action*

2 Click **New...** in the toolbar.

The *New : Specific resources* window is displayed:

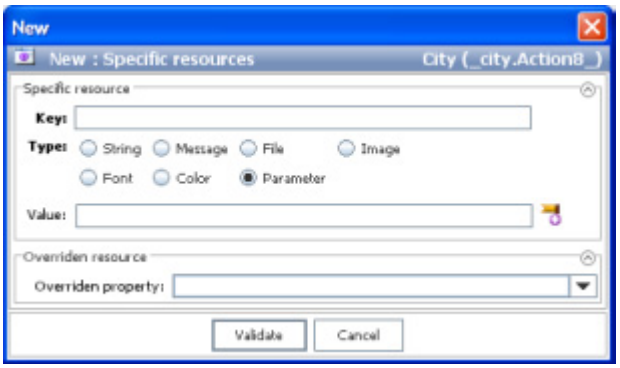

Fig 7.12 *Specifying specific resources for an action*

3 Set the fields:

Key

Type

Value

Overriden property

4 Click **Validate**.

5 Back in the *Specifc resources* window, click **Validate**.

7.10 Setting specific parameters for an action

TO PERFORM THIS STEP

1 Right-click the appropriate action either in the Navigation tree, or in the Actions tab then select **Specific parameters...** from the context menu.

The *Specific parameters* window is displayed:

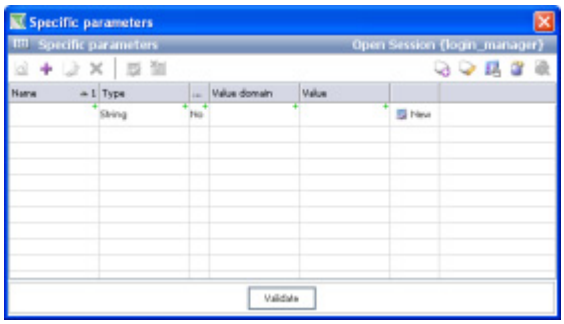

Fig 7.13 *Setting specific parameters for an action (1/2)*

2 Click **New...** in the toolbar.

The *New : Parameter declaration* window is displayed:

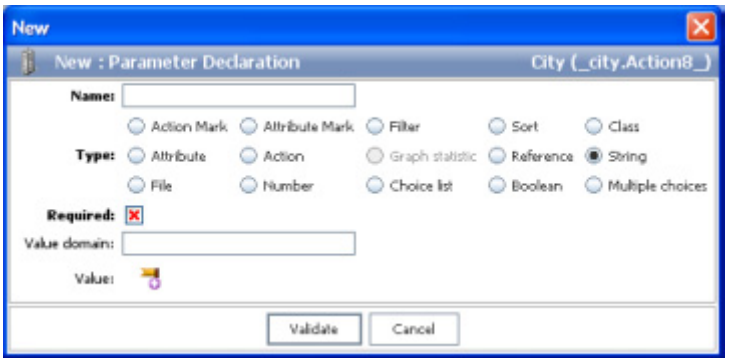

Fig 7.14 *Setting specific parameters for an action (2/2)*

3 Set the fields:

Name

Type

Required

Value domain

Value - Click to specify the parameter value.

4 Click **Validate**.

5 Back in the *Specifc parameters* window, click **Validate**.

7.11 Specifying an action behavior

You can specify a behavior for the non generic actions or for the generic actions with particular application code.

TO PERFORM THIS STEP

1 Right-click the appropriate action in the Navigation tree then select **Action behavior...** from the context menu.

Alternatively, click **Action behavior...** in the Actions tab tool bar.

The *Action behavior* window is displayed:

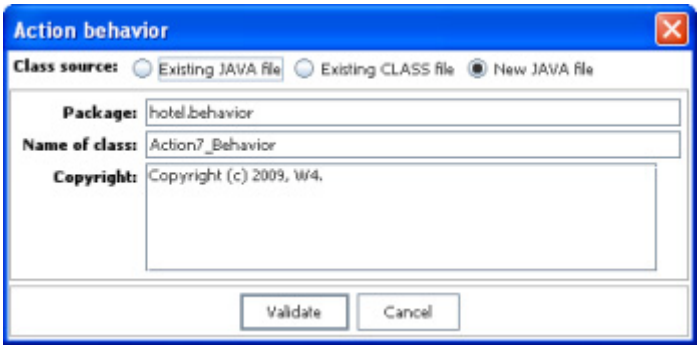

Fig 7.15 *Specifying an action behavior*

2 Set the fields:

Existing JAVA file - To specify a preexisting existing Java file. The appropriate file should be selected via the **Existing implementation** field

Existing CLASS file - To specify a preexisting class. The appropriate file should be selected via the **Class** field

New JAVA file - To have Application Composer create a new Java file.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. However you can change some parameters before generation via the next fields. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package: By default: <application id>.behavior. The behavior directory will be created to the root of the application.

Name of the class - The class name, by default: *IdentifierActionBehavior* with the application identifier's first letterin uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the application_composer.ini file as the default value.

7.12 Specifying an action builder

TO PERFORM THIS STEP

1 Right-click the appropriate action in the Navigation tree then select **Action builder...** from the context menu.

Alternatively, click **Action builder...** in the Actions tab toolbar.

The *Action builder* window is displayed:

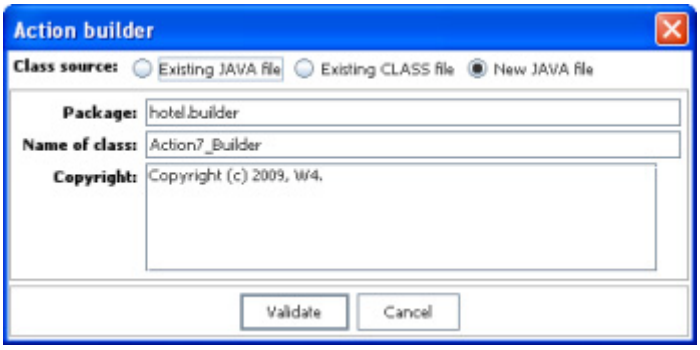

Fig 7.16 *Specifying an action builder*

2 Set the fields:

Existing JAVA file - To specify a preexisting Java file. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class. The appropriate file should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new Java file.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. However you can change some parameters before generation via the next fields. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <application id>.builder. The builder directory is created to the root of the application.

Name of the class - The class name, by default: *IdentifierActionBuilder* with the application identifier's first letter in uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the application_composer.ini file as the default value.

7.13 Specifying code for actions with no template

For the actions with no template, you can specify the Java class containing the code for the action's invocation method.

This class must implement the interface *LyProcessActionInterface*.

TO PERFORM THIS STEP

1 Right-click the appropriate action (an action created with the *Other* type) either in the Navigation tree, or in the Actions tab then select **Action behavior** ▸ **Action process** from the context menu.

The *Action process* window is displayed:

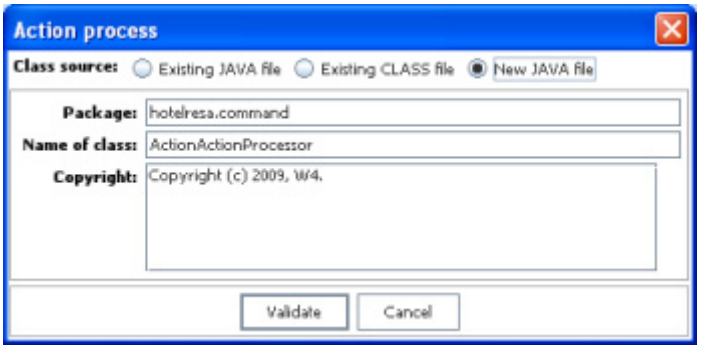

Fig 7.17 *Specifying the code for an action with no template*

2 Set the fields:

Existing JAVA file - To specify a preexisting Java file. The appropriate file should be selected via the **Existing implementation** field.

Existing CLASS file - To specify a preexisting class. The appropriate file should be selected via the **Class** field.

New JAVA file - To have Application Composer create a new Java file.

Application Composer is going to generate the Java class in the application directory and create the relevant directories for the class packages. However you can change some

parameters before generation via the next fields. When the file has been generated, it is opened in the text editor that has been specified in the preferences.

Package - The class package, by default: <application id>.builder. The builder directory will be created to the root of the application.

Name of the class - The class name, by default: *IdentifierActionProcessor* with the application identifier's first letter in uppercase.

Copyright - The class comment, with the value of the resource LY_DEFAULT_COPYRIGHT set in the application_composer.ini file as the default value.

Managing Java classes and libraries

The *Java classes* tab displays all the Java classes in the application and allows you to manage the CLASSPATH used for compilation.

The *Libraries* tab lists the access paths to the libraries and Java sources (CLASSPATH). You can modify the order and the content of the CLASSPATH from this tab.

8.1 Java classes

To display the *Java classes* tab, select **Windows** ▸ **Java classes**.

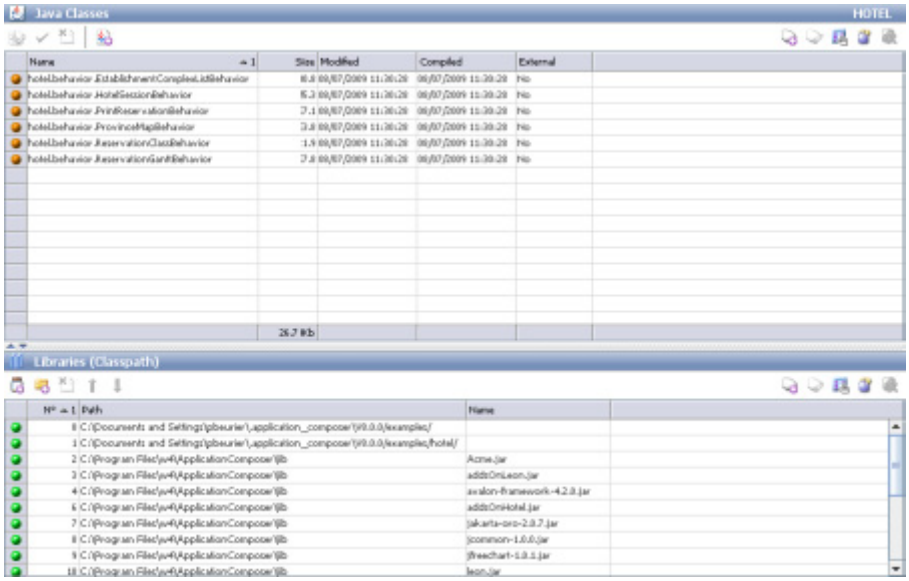

Fig 8.1 *The Java classes tab*

NOTE Libraries that cannot be found are displayed on a pink background.

The possible actions are as follows:

- **Edit a Java class** - To review / edit the source code in the text editor specified in the preferences.
- **Compile** - To compile the class.
- **Delete** - To remove the selected Java class(es). Please note: The corresponding .java files are removed from the disk.
- **Add a Java class** - To add an external Java class that contains application code required by the application.

The status icons specifies whether the class needs to be compiled or not.

- A red bullet flags a Java class that has not been compiled yet.
- An orange bullet flags a Java class that has been modified since last compilation and that needs to be compiled again.
- A green bullet flags a compiled version that is up-to-date.

8.2 Libraries

The *Libraries* tab can be used to manage the CLASSPATH used for compilation.

The *Libraries* tab displays the paths to the libraries and Java sources that are required to run the application. It can be used to edit the application's classpath.

To display the *Libraries* tab, select **Windows** ▸ **Java classes**.

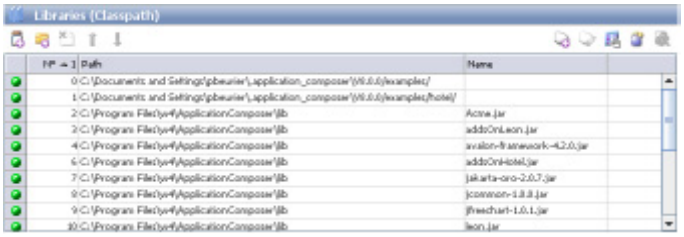

Fig 8.2 *The Libraries tab*

NOTE A red bullet flags a library that could not be found at the specified location.

The possible actions are as follows:

- **Add an archive** - To add a .jar library to the CLASSPATH.
- **Add a directory** - To add a directory to the CLASSPATH.
- **Delete** - To delete the currently selected library/libraries.
- **Move up** - **Move down** - To reorganize the list order. Order is significant when generating the classpath, especially for the addson (patches), which include corrections. You should specify addsOnXX.jar files before XX.jar files.

Managing data sources

Application Composer allows you to configure connections to data providers referred to as *locations* in Application Engine. For example a location can be a connection to a database via a JDBC driver, a connection to flat files, etc.

9.1 Adding a data source

TO ADD A DATA SOURCE

1 Select **File** ▸ **Data sources...**

The **Configure data sources** window is displayed:

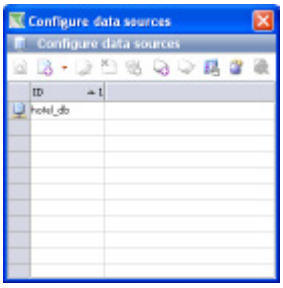

Fig 9.1 *Adding a data source (1/2)*

This window lists the available data sources and connectors.

2 Click **Create**...

A sub tool bar is displayed to let you select the appropriate data source type:

Fig 9.2 *Adding a data source (2/2)*

The possible data source types are as follows:

- **RDBMS**: Relational database
  > 9.1.2 RDBMS, page 172
- **LDAP**: Active directory
  > 9.1.3 LDAP, page 173
- **File location**: Flat files
  > 9.1.1 File location, page 171
- **Generic**: Generic data source used to define, for example, a link to a specific data source
  > 9.1.4 Generic data provider, page 174

9.1.1 File location

A file data source can be used for application development phases, for demo or prototypes, or in some cases for standalone applications.

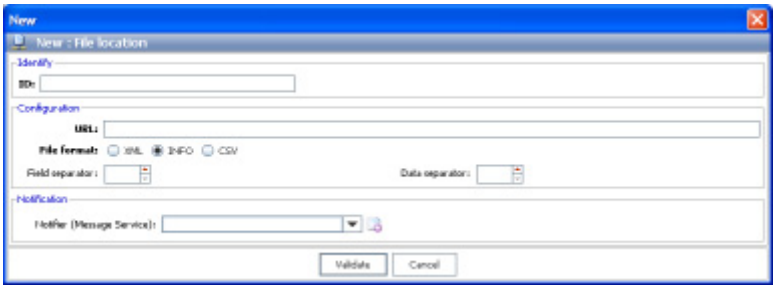

Fig 9.3 *Adding a data source - file location*

URL - The access path to the directory containing the files to be saved.

File format - The possible formats are as follows:

- **XML** - XML flat files.
- **INFO** - Flat files in Ithe NFO format (Application Engine's default format). Data is stored in rows with separators.
- **CVS** - File of type CSV (format of type Excel) with separators.

Field separator - The code of the character to be used to delimit the fields. The character corresponding to the current code is displayed to the right of the edit area.

Data separator - The code of the character to be used to delimit data.

Notifier (message service) - The notification service is a software bus (basing on a MOM: Message Oriented Middleware) that makes it possible to send notifications when objects are created, deleted and modified between several applications running on different machines. The notification service invokes JMS to send and receive messages.

9.1.2 RDBMS

An RDBMS data source is a JDBC connector to a relational database. The databases that are currently supported by Application Engine are: Microsoft Access, MySQL, Oracle, Polyhedra, Microsoft SQL Server, Sybase, Instant DB and PostgreSQL.

For other databases, Application Engine uses standard SQL.

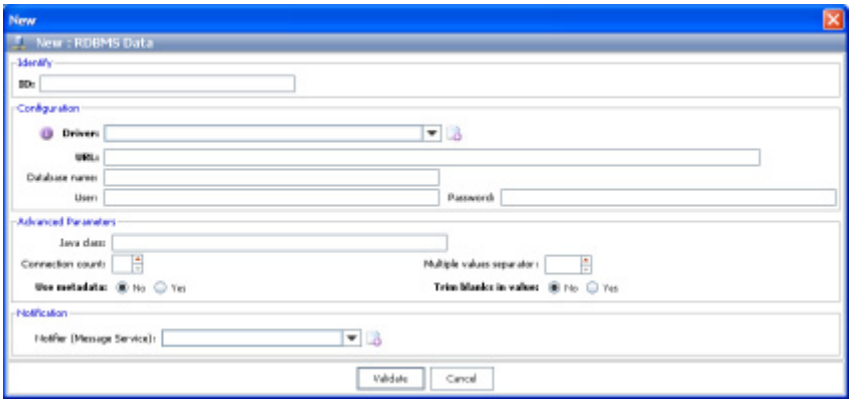

Fig 9.4 *Adding a data source - RDBMS*

Driver - The Java class of the JDBC driver.

URL - The URL to be provided to the JDBC driver.

Database name - **User** - **Password** - Database, user and password to be used to open a database connection.

Java Class - This item is optional. By default, the RDBMS data source uses standard SQL. As some databases require some changes to the SQL queries, you can specify the Java classes to be used.

NOTE For the databases supported by Application Engine, this parameter should not be set.

Connection count - Number of connections that are simultaneously opened by the application to the database. If the value is greater than one, Application Engine can manage a pool of connections to the database.

Multiple values separator - In theory, a relational database cannot manage multiple values for the relations directly. The usual way to proceed is via a join table, which is supported by Application Engine. For simple cases, you can also store the multiple values in a field in the database by using a separator to separate the values. Using a join table is however strongly recommended.

Use Discovery - To use a discovery to know the type of data stored in the RDBMS. If this option is enabled, Application Engine will first retrieve the metadata and use the types internally.

Trim blanks in value - Some databases sometimes store values with a blank character at the beginning or at the end. This may interfere with string comparisons and/or filtering in the applications. Enabling this option ensures that these blank characters are deleted when data is retrieved from the database.

Notifier (Message service)

> 9.2 Specifying an additional notification service, page 175

9.1.3 LDAP

LDAP directory data source.

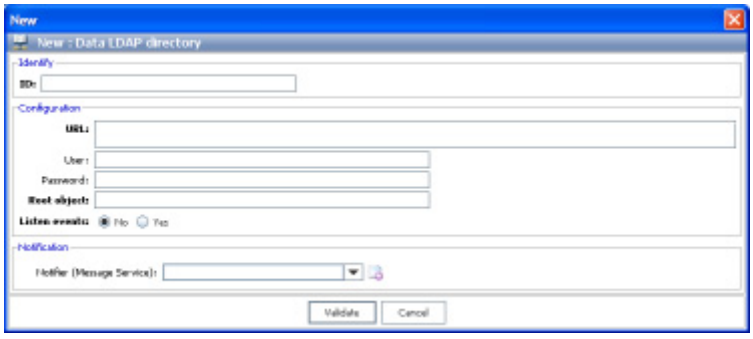

Fig 9.5 *Adding a data source - LDAP*

URL - The address/port of the LDAP server used for connection.

User - The user name for connecting to the directory.

Password - The password for connecting to the directory.

Root object - The identifier of the base object for accessing the data in the directory.

Listen events - To specify whether the application should listen for directory events.

Notifier (Message service)

> 9.2 Specifying an additional notification service, page 175

9.1.4 Generic data provider

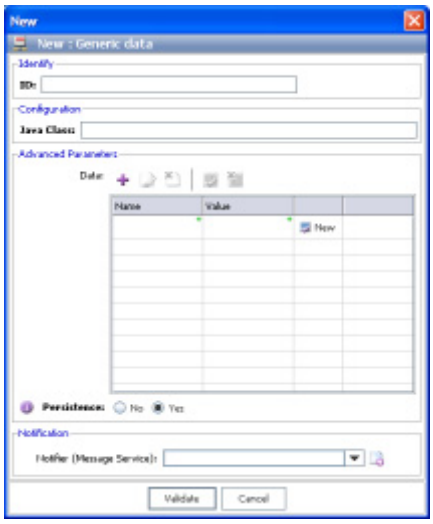

Fig 9.6 *Adding a data source - Generic data*

Java Class - To specify the name of the Java class that extends `leon.peer.LyData Provider` and that is the data provider to be used.

Data - To specify the parameters for configuring the connector.

Persistence - To specify whether the connector whose updates manage data persistency, should be used or not.

Notifier (Message service)

> 9.2 Specifying an additional notification service, page 175

9.2 Specifying an additional notification service

You can configure an additional notification service.

Notification allows you to address the restrictions of some connector types that fail to notify the applications when data is updated. Typically, when data is updated in a relational database by a client application, the other client applications are not notified.

To remedy this problem, the notification service sends events such as create/edit/delete, potentially along with the modified values.

The available implementation for this software bus uses a JMS connector.

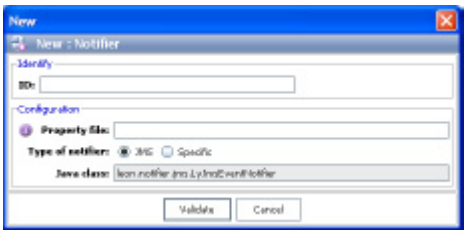

Fig 9.7 *Creating an additional notification service*

Property file - The property file that contains the parameters for configuring the notification service.

Java class - The name of the Java class that extends `leon.notifier.LyEventNotifier` and that is the notification service to use. The *JMS* value is an alias for the generic JMS notification bus `leon.notifier.jms.LyJmsNotifier`

Working with Discovery

Discovery is used to explore relational databases, Java files (Bean, EJB), CSV files, W4 procedure models, or ECM systems, in order to extract their structure and implement their data as part of an application.

In this chapter we will cover the following topics:

To display **Discovery**, select **Windows** ▸ **Discovery**.

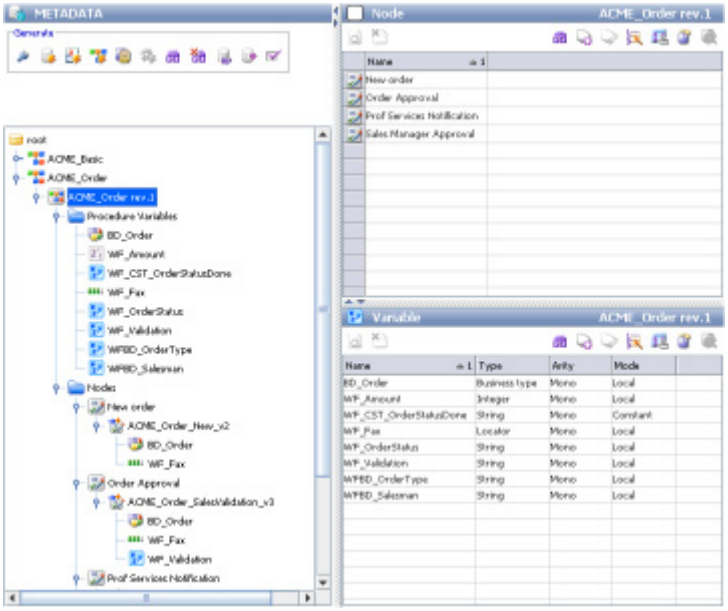

Fig 10.1 *Discovery*

The navigation tree, to the left, displays a hierarchical view of the discovery's target.

The list view, to the right, provides details of the currently selected object in the navigation tree.

10.1 Specifying the drivers to be used

The **List of drivers** icon in the Discovery toolbar displays the JDBC driver used to connect to the database.

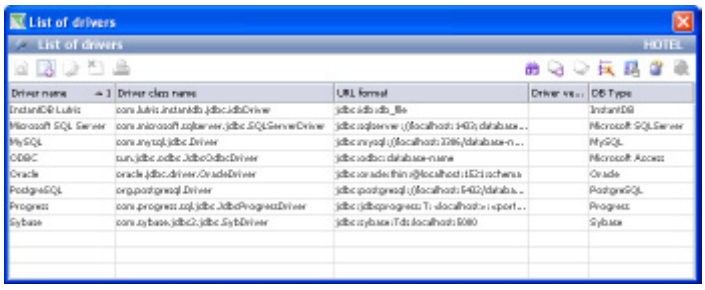

Fig 10.2 *Viewing the list of available drivers*

All the drivers specified for the application are displayed in this window.

Via this window you can:

- **Details** - View the details of the currently selected driver
- **New** - Specify a new driver for use
- **Modify** - Change the details of the currently selected driver
- **Delete** - Remove the currently selected driver from the list of drivers to use

To add a new driver, the following fields must be specified:

- **Driver class name** - The name of the driver's Java class: It is used as the identifier. Only one database has this name.
- **URL format** - It is used as a prefix for the new connections to this type of databases (is automatically displayed in the connection URL after selecting the driver). This field is not mandatory.

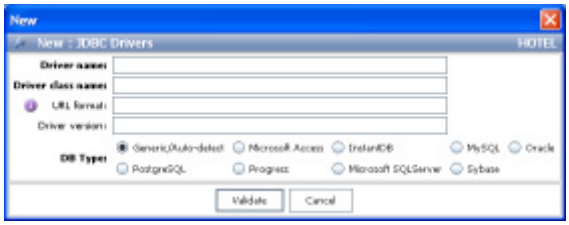

Fig 10.3 *Adding a new driver*

PLEASE NOTE The driver's Java class must be declared in the application's CLASSPATH. If database connection fails (driver not found), you just need to add the library to the Application Composer application's CLASSPATH.

For further information regarding libraries:

> 8.2 Libraries, page 167

10.2 Discovering an SQL database

The **SQL base structure discovery** icon in the Discovery toolbar allows you to discover the structure of an SQL database.

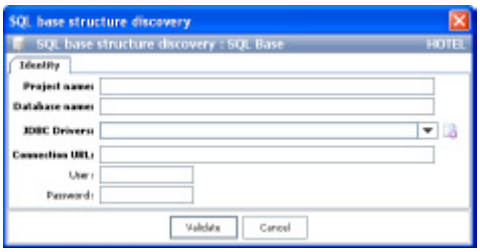

Fig 10.4 *Discovering an SQL database*

Project name - The name of the Application Engine project.

Database name - The name of the database. It is used as an identifier. Only one database should have this name.

JDBC drivers - The JDBC driver used to connect to the database. All the drivers specified for the application are displayed in a drop-down list.

Connection URL - The URL to connect to the database. The format of this URL depends on which driver is selected.

User - **Password** - A user name with permissions to connect to the database, along with the corresponding password. These fields may be optional depending on the type of database.

When the form has been validated, the application attempts to connect to the database. If connection is successful, the metadata is extracted and the corresponding objects are created.

When extraction is completed, Discovery displays a list of columns identified by the system as potentially suitable for conversion to an Application Engine enumeration type. You then need to validate the columns for which the enumeration type is justified.

For detailed information about this feature:

> 10.12 Viewing the columns compatible with the enumerate type, page 187

10.3 Discovering a Java file

The **Java files structure discovery** icon in the Discovery toolbar allows you to explore compiled Java files in order to extract their data.

You may need to open a communication between any modeling tool and Discovery by using Java classes. Most of the modeling tools allow to generate Java classes based on an Object Oriented Model (OOM):

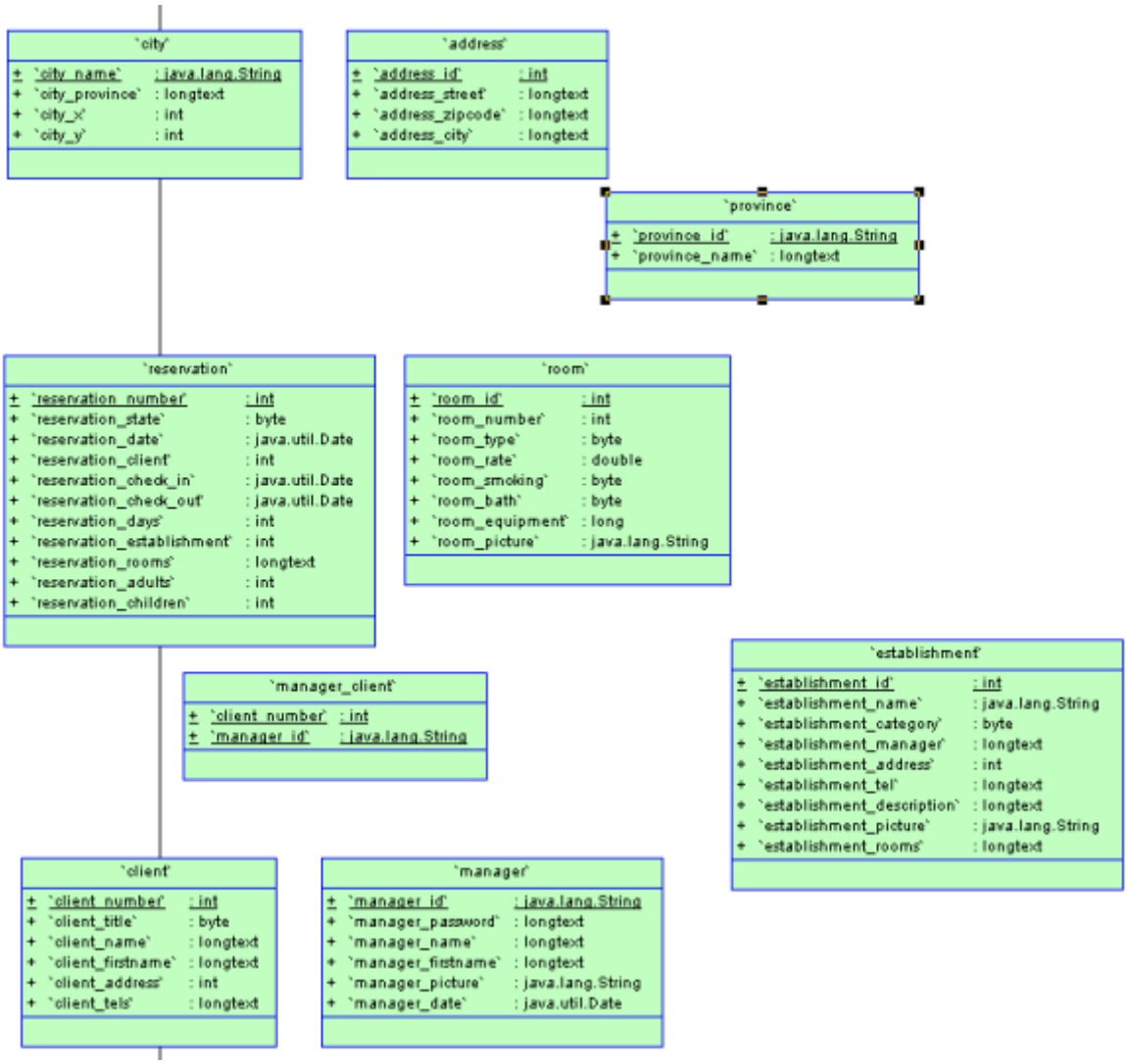

Fig 10.5 *Object Oriented Model*

The introspection mechanism used to identify the tables and the columns works as follows:

- Tables represent the JAVA classes that are found.
- Columns represent the public attributes of the class.
- A property is specified for the columns and attribute types of the class. A conversion is applied between the attribute's JAVA type and the type of the column in Discovery.

Table 10.1: Java attribute type and matching column type in Discovery

TYPE OF THE JAVA ATTRIBUTE	TYPE OF THE COLUMN IN DISCOVERY (TYPE SQL)
java.lang.String, java.lang.StringBuffer	VARCHAR
java.lang.Character	CHAR
java.lang.Integer	INTEGER
java.lang.Short	SMALLINT
java.lang.Long	BIGINT
java.lang.Double, java.lang.Number	DOUBLE
java.lang.Float	FLOAT
java.lang.Byte, java.lang.Boolean	BYTE
java.lang.Date	DATE

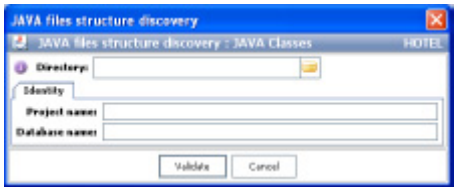

Fig 10.6 *Discovering a Java file*

Directory - The directory where the compiled files are located.

10.4 Discovering a CSV file

The **CSV discovery** icon in the Discovery toolbar allows you to explore compiled Java files in order to extract their data.

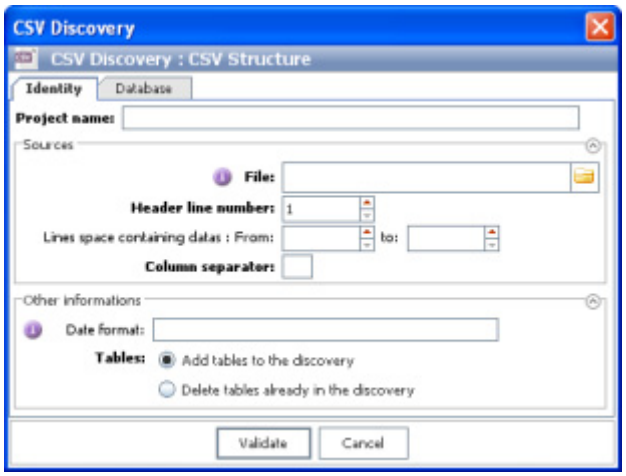

Fig 10.7 *Discovering a CSV file (1/2)*

Project name

Files

Header line number

Line space...

Column separator

Date format

Add tables to discovery

Delete tables already in discovery

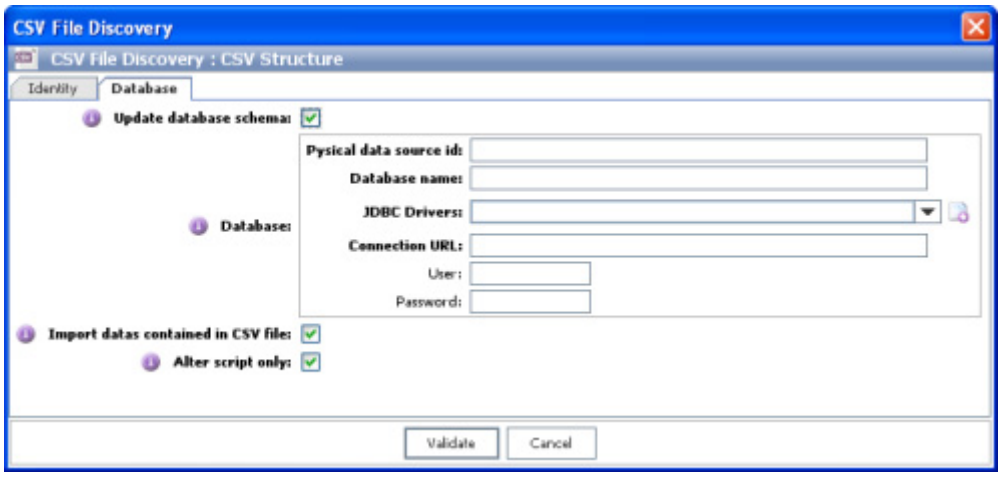

Fig 10.8 *Discovering a CSV file (2/2)*

Update database schema

Physical data source id

Database name

JDBC drivers

Connection URL

User

Password

Import data contained in CSV files

Alter script only

10.5 Discovering a W4 model

The **Discover a W4 Suite model** icon in the Discovery toolbar allows you to create an Application Engine metamodel based on a W4 process. All the W4 process activities will be converted to Application Engine application forms.

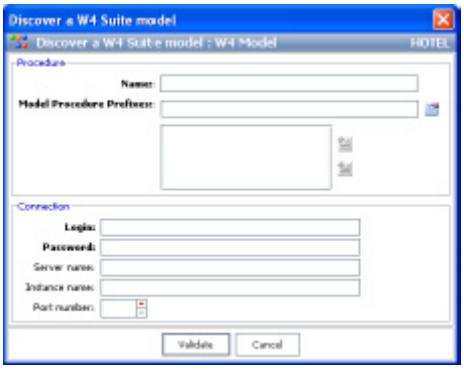

Fig 10.9 *Discovering a W4 model*

Name - This name will be assigned to the top level directory of the discovery process in the Discovery tree view.

Model procedure prefixes - Type the name or prefix of the W4 procedure to be discovered and click on the icon to validate.
Repeat this step if more than one procedure is to be discovered.

Login - **Password** - An authorized login and password for accessing the W4 Engine server where the W4 model is stored.

Server name - The W4 Engine server where W4 Extension Bus resides (defaults to localhost).

Instance name - The W4 Engine instance of the server where the W4 model is stored.

Port number - RMI port for W4 Extension Bus connection (defaults to 7507).

10.6 Discovering an ECM system

The **ECM discovery** icon in the Discovery toolbar allows you to browse the contents of ECM systems connected to W4 Extension Bus, to import ECM objects into metamodels.

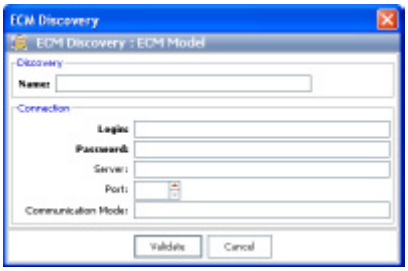

Fig 10.10 *Discovering an ECM system*

Name - This name will be assigned to the top level directory of the discovery process in the Discovery tree view.

Login - **Password** - An authorized login and password to access the ECM system.

Server - The server where W4 Extension Bus resides (defaults to localhost).

Port - RMI port for W4 Extension Bus connection (defaults to 7507).

10.7 Performing another discovery

The **Rediscover** icon in the Discovery toolbar allows you to synchronize the metamodel with the process when changes have been made to the process.

This feature can be used only for an existing database in the application. The parameters may be updated, or you can validate the form if no change is required.

The re-connection should be used when the user wishes to fully recreate the database tables. All the database tables are removed and recreated via an automatic reconnection to the database, as for a normal connection. After this operation, the state of the database in Discovery is identical to that of the actual database.

10.8 Comparing structures

The **Show differences with real structure** icon in the Discovery toolbar allows you to compare the version of the database in Discovery with the actual state of the database at the time when the action is running, in order to validate the changes or to ignore them, as appropriate. For example, if only the changes relating to a particular table are relevant, you can update this table alone.

10.9 Cancelling the comparison

The **Hide differences with real structure** icon in the Discovery toolbar allows you to go back to the original presentation of the database objects. The objects present in the database and missing in Discovery are deleted. The changes made to certain objects are not validated and the objects that no longer exist in the actual database are maintained. The state preceding the comparison action is therefore restored.

10.10 Updating the discovery

The **Update discovery from real structure** action is the logical continuation of the comparison action. The update function works only for objects whose state is different from *Identical to the real database*, the comparison function must therefore be executed beforehand.

This allows you to look at the differences between the databases, and update only part of the database (one or several tables or columns). With a full database update, only the detected changes are considered.

10.11 Exporting data to Application Composer

The **Create / Update the business model**  icon in the Discovery toolbar allows you to export data from Discovery to Application Composer.

Following the data export, the objects in the application can be modified and new objects can be created. However, no object in Application Composer can be deleted. Furthermore, modifications are made only to the attributes that can be specified by the information contained in the Discovery objects. As a result, the customizations made by the user to these objects are not corrupted.

A summary informs the user of the changes made. It gives a view on the created objects, the attributes of the objects that have been modified and their new value.

The list of the tables exported from Discovery is displayed as a class in the Application Composer application.

The columns exported from Discovery are displayed as class attributes.

During the export, a list of Application Engine generic actions is added by default to each created class.

The reverse operation is performed when opening Discovery from Application Composer. This allows you to update the objects of Discovery according to the changes made in Application Composer and as a result, to maintain consistency between the objects of both applications.

In any case, when opening the Discovery application, the object comparison function with the actual database is performed ensuring that any differences are highlighted at once when the application is opened.

10.12 Viewing the columns compatible with the enumerate type

The **Update discovery from real structure** icon in the Discovery toolbar is used to identify the columns whose characteristics indicate that they may be of Application Engine enumeration type, in order to compensate the lack of enumeration type in some RDBMS. The following criteria are considered to select the columns:

- The column status: A primary or foreign key cannot be of type enumeration.
- The column size: A text column of size 252 will probably not be of type enumeration.
- The total number of records in the corresponding table: If a table has only few records, all its columns will be detected as being of enumerate type as there are few different values.
- The number of different values taken up by the column in these records.

A pre-selection of the columns is displayed in a list.

To set the column type to enumerate click or to restore the original type click . The first action converts the Application Engine type into a CHOICE type.

Values - The values available for this field of enumerate type.

Importing a UML model

UML import allows you to load a diagram of UML (Unified Modeling Language) classes into Application Composer. Application Engine's data model is built based on the UML model. Using this feature you can obtain a default MMI based on your data model simply and quickly.

In this chapter we will cover the following topics:

The import is performed via a file in XMI format (XML Metadata Interchange). This format is the standard exchange format for UML models. For further information regarding these OMG (Object Management Group) standards, you may visit the following Web sites:

- OMG:
  http://www.omg.org/
- XMI:
  http://www.omg.org/technology/documents/formal/xmi.htm
- UML:
  http://www.uml.org/

Some options are available to obtain a prototype more quickly. They allow you to limit the number of manual changes required following an import.

- 11.2 Importing an XMI File, page 190

11.1 Supported versions and tools

UML and XMI Versions

The UML versions supported so far are versions 1.3 and 1.4.

The XMI versions supported so far are versions 1.0, 1.1, 1.2. The specifications of the version XMI 1.0 being ambiguous on several points, they may have been interpreted in different ways by the different modeling tools. As a result some errors may occur when reading XMI 1.0 files.

Modeling tools

The modeling tools with best support are those using and/or supported by the libraries of the MetaData Repository (MDR) by NetBeans.
http://mdr.netbeans.org/

They should allow you to export models in UML version 1.3 or 1.4, in XMI format 1.1 or 1.2.

Below are some sample tools (there are many others):

- Poseidon for UML by Gentleware:
  gentleware.com
- A free version intended for non-commercial use is available.
- Enterprise Architect by Sparx Systems:
  http://www.sparxsystems.com/
- Together Developer by Borland:
  http://www.borland.com/fr/products/together/index.html
- The IBM Rational Rose products:
  http://www-306.ibm.com/software/info/ecatalog/fr_FR/category/SW710.html

Any tool that produces XMI files that conform to the specifications is compatible.

To export UML models into XMI, some tools offer a variety of options. You should choose a UML version 1.4 and an XMI version 1.1 or 1.2 whenever possible. Choose to exclude the information on the diagram and any other item specific to the tool.

Finding version and tool information in the file

The XMI version is always specified in the root tag of the file.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2'>
...
</XMI>
```

The UML version is not always specified in the XMI file. If it is present, then it is described in the meta model information.

```
<XMI xmi.version = 'X.Y'>
<XMI.header>
...
<XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
</XMI.header>
...
</XMI>
```

The name of the tool used to export the UML model to XMI format is specified as follows:

```
<XMI xmi.version = 'X.Y'>
<XMI.header>
<XMI.documentation>
<XMI.exporter>Netbeans XMI Writer</XMI.exporter>
<XMI.exporterVersion>1.0</XMI.exporterVersion>
</XMI.documentation>
...
</XMI.header>
...
</XMI>
```

11.2 Importing an XMI File

PLEASE NOTE Importing a new model overwrites all previous data.

Select **File** ▸ **XMI import** to perform this action.

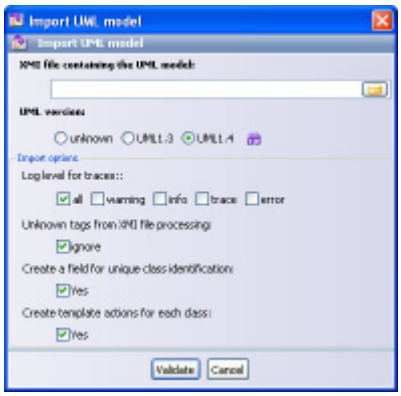

Fig 11.1 *Importing an XMI File*

XMI file containing the UML model - The file containing the XMI model. The file extension should be either.xmi or .xml.

UML version - The version of the UML model contained in the XMI file. The version 1.4 is selected by default because this is the most common version. The icon on the right allows you to search automatically for the UML version in the file.

Log level for traces - This option allows you to define the desired trace level during the importation.

- **all** - All the traces.
- **warning** - Recoverable errors.
- **info** - Processing information.
- **trace** - Calls to methods and call stacks.
- **error** - Errors.

Unknown tags from XMI file processing - This option allows you to specify how the tags that are not recognized when the file is read should be processed. There are two possible options:

- **Ignore** - The unknown tag is ignored and a warning is written to the trace, indicating its name.
- **Throw an exception**: An error is displayed to the user and the import is stopped.

Create a field for unique class identification - This option allows you to create a field identifier in all the classes, thus enabling you to avoid setting it manually. The presence of such a field is necessary in the Application Engine classes.

Create template actions for each class - This option allows you to create the basic actions in all the classes: Consult, Create, Duplicate, Modify, Delete and Print.

11.3 Transformation of the UML Model

As Application Engine's data model has a class structure, we will take a look at the UML class diagrams. The other diagrams are not transformed.

11.3.1 Transformed UML Items

The following table shows how the items of a UML model that can be present in a class diagram, are transformed. A dash means that the item is not transformed.

Table 11.1: Mapping table between UML items and Application Engine items

UML ITEM	APPLICATION ENGINE ITEM
Association	One or two relation fields depending on the the association ends, simple or multiple relations depending on multiplicity.
Attribute	Text, number, date, typed or table field depending on the type and on attribute multiplicity.
Class	Class, structure field or choice field depending on the stereotype or the references to the class.
Dependency	Relation field
Generalization	Class inheritance
Interface	_
Method	_
Model	Project
Operation	Action
Package	Project
Parameter	_
DataType	Structure or enumeration field
Object	_
Parameterized class	Class (transformation identical to a *normal* class)

A UML model contains all the information regarding existing diagrams. The items originating from diagram types other than class diagrams are ignored during transformation.

11.3.2 Overview of transformations

Model, packages, classes, attributes and operations

As the models are a particular type of package, they are transformed as such.

The UML packages are transformed into Application Engine projects. The tree structure of the packages is reflected in the project tree.

As a rule the UML classes are transformed into Application Engine classes and their organization into packages is preserved. In some cases a class can be transformed into a field of the root project:

- When the class is of <<enumeration>> stereotype, it is transformed into a choice field and the enumeration literals into choice options.
- When the class is of <<type>> or <<dataType>> stereotype or when this is the type of at least one attribute of the model, it is transformed into a structure field, its attributes being transformed into fields of the structure.

The attributes are transformed according to their type and to their multiplicity. If no multiplicity is defined, we consider that it is equal to 1..1. The following table summarizes the transformation according to the type and to the multiplicity.

Table 11.2: UML Transformation of attributes according to their type and multiplicity

TYPE | MULTIPLICITY	UPPER LIMIT = 1	UPPER LIMIT > 1 OR INFINITE
STRING, CHAR	Text field	Table field made up of items of type text
BYTE, DOUBLE, FLOAT, INT, INTEGER, LONG, REAL, SHORT	Number field	Table field made up of items of type number
DATE	Date field	Table field made up of items of type date
BOOLEAN	Typed field, whose type is a choice field defined in the root fields.	Table field made up of items of type typed field
A CLASS OF THE MODEL, AN ENUMERATION OR A DATA TYPE	Typed field, whose type is the root field derived from the type transformation.	Table field made up of items of type typed field
UNKNOWN	Text field	Table field made up of items of type text

If the lower limit of the multiplicity of an attribute is equal to 0 then, it will have the **optional** mark.

The operations are transformed into actions of the class in which they are defined. The methods and the parameters of the operations are not taken into account.

Generalizations

Generalizations are transformed into inheritances.

Import of models with multiples extends is not supported: An error is returned to the user and the import is stopped.

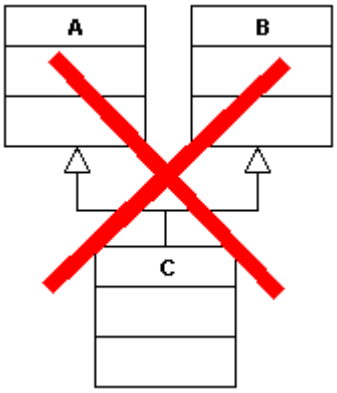

Fig 11.2 *Generalization: Multiple inheritances are forbidden*

We recommend caution with inheritances between classes coming from different packages. When reading the data model, inheritances are resolved in their definition order. A class cannot extend a class that is defined after it in the model.

Below is an example of a UML model that, once transformed, will not be read correctly.

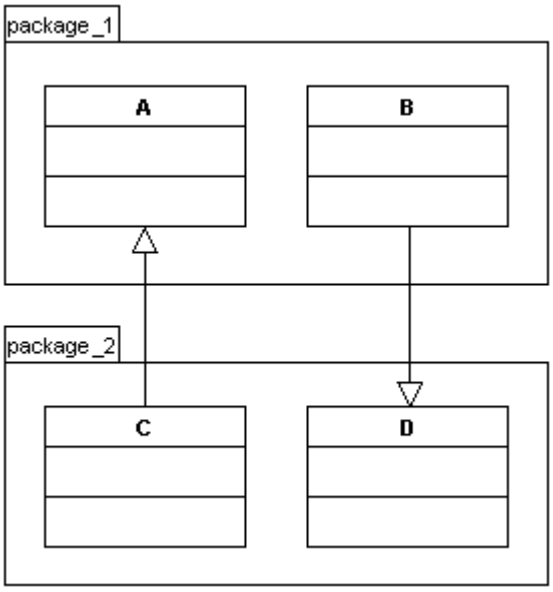

Fig 11.3 *Inheritance that is not supported by Application Engine*

To specify that class C extends class A, package_1 must be defined first. If we also specify that B extends D, it will be impossible to read the data model. At this point in time, when

Application Engine reads that class B extends D, D is still unknown. By specifying package_2 first, the same error type occurs because class A is unknown when reading the inheritance of class C.

Associations

An association is transformed into a relation field. The navigability indicates both the original class of the field and its target class. A double direction relation is translated into two relation fields. The type of aggregation (i.e. aggregation, composition or none) indicates the type of relation (i.e. association or composition).

The multiplicity provides additional information: An interval whose lower limit is equal to 0 gives the optional mark to the field. An interval whose upper limit is greater than 1 indicates that the relation is multiple.

The associations in UML can link together at least two Classifiers. We transform only the binary associations between two Classes.

Particular cases - The associations of reflexive compositions are not supported in Application Engine, we transform them into relations of type Association.

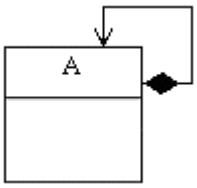

Fig 11.4 *Particular case of association that cannot be modeled in Application Engine*

The Associations between a class A and a class B, whose end points do not have a navigability type defined (AssociationEnds), are interpreted as follows:

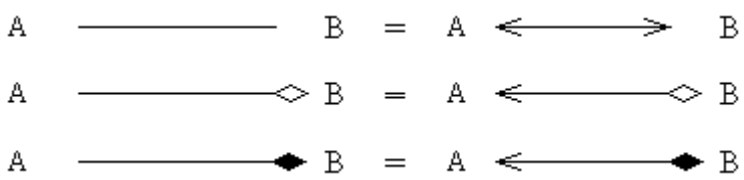

Fig 11.5 *Interpretation of the associations whose navigability is null*

The associations in the figure below are not transformed because they do not make sense in a UML description.

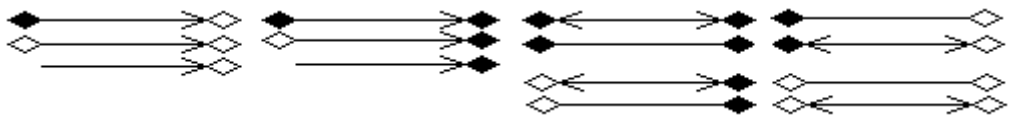

Fig 11.6 *Not pertinent UML association*

Dependencies

Dependencies are transformed into relation fields from the client to the provider.

Only the dependencies between two classes are transformed.

11.4 Most Frequent Errors

In this section, we will deal with the most frequent errors. For any other unexplained error, please contact W4 Support. Provide the trace file and the XMI file, or an example of XMI file corresponding to the generated error.

1. Libraries are not located in the right place

In this section $APPENGINE$ is the access path to where Application Engine is located.

The libraries required by the XMI import module are located in the $APPENGINE$/install/mdr and $APPENGINE$/leon/tools/studio/lib/uml directories.

The java.lang.ClassNotFoundException and java.lang.NoClassDefFoundError exceptions indicate that some libraries are not bound. Make sure that the libraries umlloader.jar, uml14loader.jar, uml13loader.jar, uml14.jar and uml13.jar are located in the directory $APPENGINE$/leon/tools/studio/lib/uml and that the libraries jmi.jar, jmiutils.jar, mdrapi.jar, mof.jar, nbmdr.jar and openide-util.jar are in the $APPENGINE$/install/mdr directory.

2. The XMI file is malformed

a. Structural malformation

When the XMI file is malformed, the following exceptions are thrown.

```
javax.jmi.xmi.MalformedXMIException
org.netbeans.lib.jmi.util.DebugException
```

The associated error message offers a good indication of the source of the malformation.

For further information regarding the source of the exception, see the console or the trace file available at $APPENGINE$/xmi_import.log.

The following message is an example of malformation.

```
java.lang.NumberFormatException: For input string: "*"
```

This generally occurs during the definition of the multiplicity where * is left for the particular infinite value. You just need to replace the * by -1, -1 being the representation of this particular infinite value.

b. Malformation due to the processing option of unknown tags

If the processing of unknown tags is defined as *throw an exception* then, the exception will be thrown as soon as the first unknown tag is read. The exceptions that can appear are the same as those previously mentioned. The name of the unknown tag will be indicated in the message of the source of the error. Below is an example of a message where the UML:Diagram tag is unknown.

```
org.netbeans.lib.jmi.util.DebugException: Name cannot be resolved:
UML:Diagram
```

Unknown tags are those that are not defined in the metamodel of the UML language available on the Web site of the OMG:

http://www.omg.org/docs/ad/01-02-15.xml

This metamodel is described in the MOF language and exported in XMI.

3. An error occurs at read time

If an error occurs at read time, an exception of the following type is thrown.

```
leon.tools.studio.xmi.MLUmlReadingException
```

Some information and recommendations are often associated to the exception and are included in the error message that is displayed.

4. An error occurs during transformation

If an error occurs during transformation, an exception of the following type is thrown.

```
leon.tools.studio.xmi.MLUmlTransformationException
```

Such errors may occur if the model does not correspond to our modeling recommendations, notably regarding multiple inheritances.

A

B

C

D

E

F

G

I

J

K

L

M

N

P

R

S

T

U

V

W

X

Application Composer

USER GUIDE

Reference: APPCOMPOSER_USER_046_EN

Should you have any comment or suggestion related to this document, please contact W4 Customer Support providing the document reference:

- Via the W4 SupportFlow case management tool on MyW4.com at http://support.myw4.com
- By email: support@w4.eu
- By telephone: +33 (0) 820 320 762