Application Composer

TUTORIAL

Reference: APPCOMPOSER_TUTORIAL_043_EN

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

Application Composer

TUTORIAL

Reference: APPCOMPOSER_TUTORIAL_043_EN

Contents

Chapter 1

Prerequisites

This section sets outs the prerequisites that you should meet to successfully complete the steps in this tutorial.

In this section:

User profile

Application Composer users can choose from four predefined profiles depending on which features they are planning to use. The *Expert* profile is required to follow the steps in this tutorial.

The *Expert* profile should be specified in the preferences.

TO PERFORM THIS STEP

1 Select **Preferences** in the tool bar.

The *Preferences* window is displayed.

2 In the *General* tab, select Expert in the **Level** field.

3 Click **Validate**.

Java Development Kit

A JDK will be required to compile the application.

Your JDK installation folder should be specified in the preferences.

TO PERFORM THIS STEP

1. Select **Preferences** in the tool bar.
   The *Preferences* window is displayed.
2. In the *Java* tab, set the **JDK folder** field, e.g: *C:/Program Files/Java/JDKx.x.x*.
3. Click **Validate**.

Editor

A text editor will be required to make changes to the automatically generated specific code.

Your preferred editor should be specified in the preferences.

TO PERFORM THIS STEP

1. Select **Preferences** in the tool bar.
   The *Preferences* window is displayed.
2. In the *Java* tab, set the **Editor** field to the corresponding executable file.
3. Click **Validate**.

Tomcat

The Tomcat URL will be required to access the generated application.

The Tomcat URL should be specified in the preferences.

TO PERFORM THIS STEP

1. Select **Preferences** in the tool bar.
   The *Preferences* window is displayed.
2. In the *Tomcat* tab, set the **URL** field, e.g: *http://localhost:8080*.
3. Click **Validate**.

Structuring the data model

In this document we are assuming that Application Composer is installed on your system.

For further information on how to install Application Composer, please refer to the Application Composer user guide.

In the following sections we will demonstrate how to structure the data model of a hotel room reservation application.

NOTE To specify English as the language of the user interface, edit the *application_composer.ini* file located in the *Studio* folder of the setup directory and uncomment the following key: *LY_LANGUAGE=EN*

In this section:

2.1 Creating the application

TO PERFORM THIS STEP

1 Start Application Composer.

Application Composer’s main window is displayed.

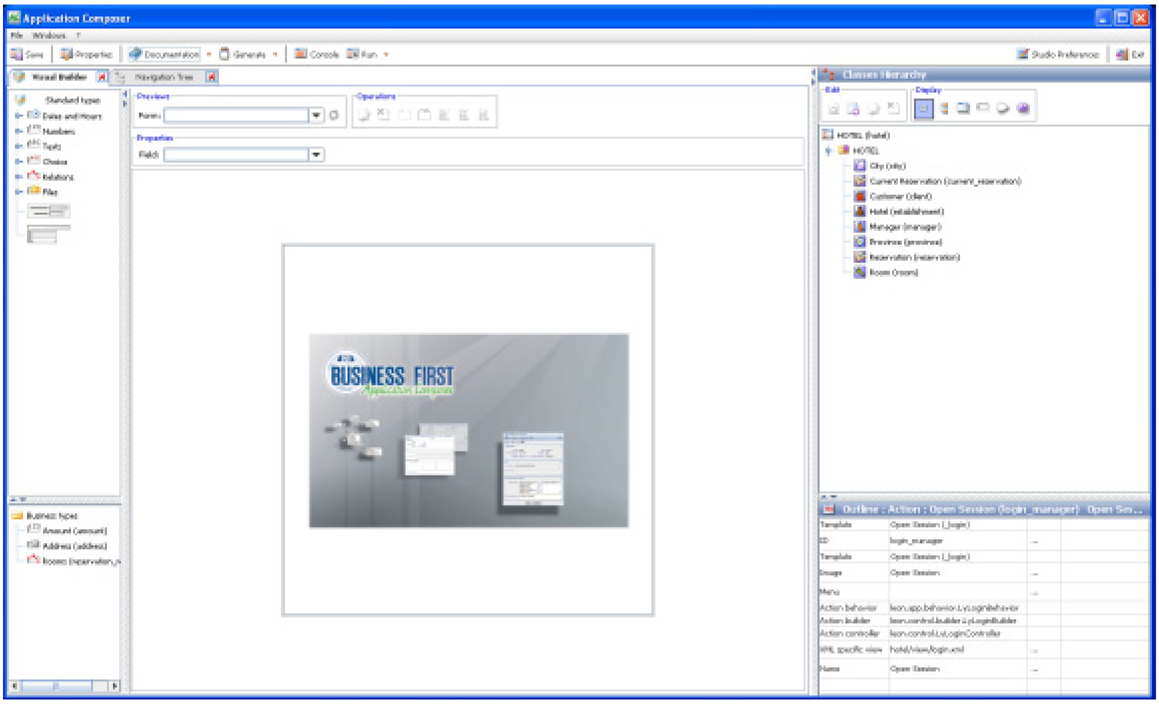
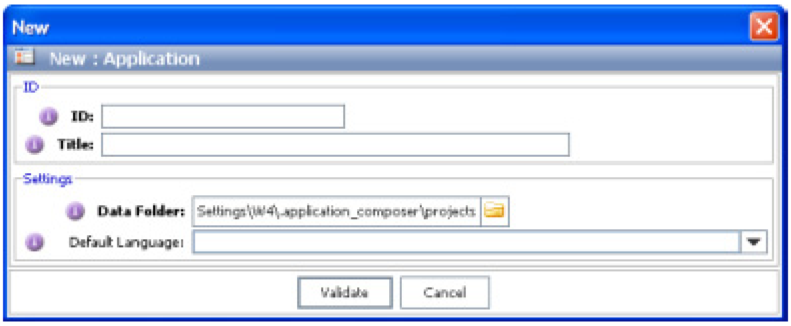


Fig 2.1 *Application Composer’s main window*

2 Select **File** ‣ **New...**

The *New : Application* window is displayed:



Data Folder: Settings\W4\.application_composer\projects

Fig 2.2 *Creating the application (2/2)*

3 In the **ID** field, type hotelresa.

NOTE Application identifiers are unique. Application Composer will not create the application if the ID you are providing is already in use.

4 In the **Title** field, type Hotel.

5 Review the default value for the **Data folder** field and change if appropriate. This is the location where the application files will be created.

6 In the **Default language** field, select English.

7 Click **Validate**.

2.2 Creating the application classes

The next step after the creation of the application is the creation of all the application's classes.

In this section:

2.2.1 *Region*

The *Region* class is the first application class that will be creating. It will be used to set the geographical locations of the hotels.

Creating this class involves the following actions:

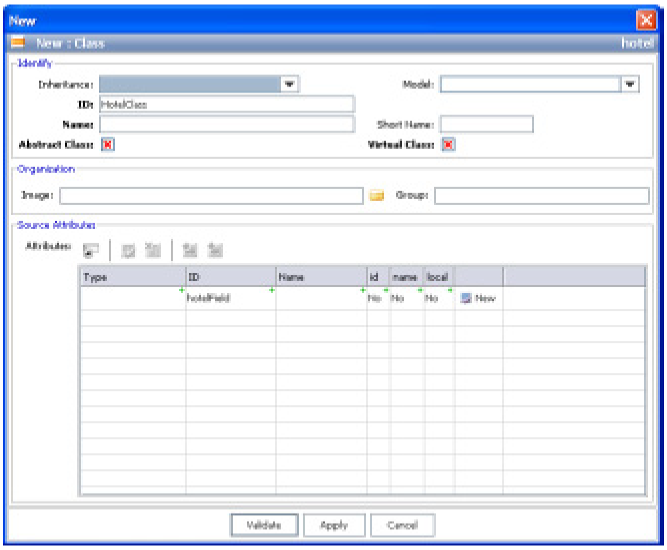
- Creating the class itself,
- Specifying its attributes,
- Specifying the available actions for the class.

TO PERFORM THIS STEP

1 Right-click the project (2nd level item in the *Class Hierarchy*) and select **Create class...** from the context menu.

[ALTERNATIVELY] Select the project and click **Create class...** in the *Edit* tool bar of the class hierarchy.

The *Create class* window is displayed:



ID: HotelClass

Type	ID	Name	id	name	local	
	hotelField		No	No	No	New

Fig 2.3 *Creating a new class*

2 In the **ID** field, type region.

NOTE Identifiers of fields, classes and attributes are mandatory, are unique within an application, and should not start with an underscore character. Identifiers are required to get objects programmatically.

3 In the **Name** field, type Region.

4 [Optional] Assign an image to the class:

4.1 Click 🗂 next to the **Image** field.

The file selection window is displayed.

4.2 Select an image to represent the *region* class.

You can for instance select the *province.gif* image to be found in the *Studio\examples\hotel\images*, in your Application Composer setup directory.

4.3 Click **Open**.

5 In the *Attributes* area, create an attribute:

5.1 In the **Type** column, select Text Attribute.

5.2 In the **ID** column, type region_id.

5.3 In the **Name** column, type Region ID.

5.4 Select the **id** check box.

NOTE The *id* mark specifies that when an object of the corresponding application class is created, the field value is used as an internal identifier for the object. At least one field should have the *id* mark within an application class.

NOTE A mark is a flag that can be applied to the fields that share the same behavior. For example in the *hotel* application, all fields with the *id* mark will be used to identify the objects in the corresponding application class. Certain marks, such as the *id* mark, can be applied in the class creation form although most of them need to be applied afterwards via a dedicated window.

5.5 Confirm the attribute creation by clicking **New** at the end of the row.

6 Create a second attribute:

6.1 In the **Type** column, select Text Attribute.

6.2 In the **ID** column, type region_name.

6.3 In the **Name** column, type Region Name.

6.4 Select the **name** check box.

NOTE The *name* mark specifies that when an object of the corresponding application class is created, the field value is displayed through the graphical user interface to easily identify the object.

6.5 Confirm the attribute creation by clicking **New** at the end of the row.

7 Click **Validate** in the *Create class* window.

The new *Region* class and the associated image are now displayed in the *class hierarchy*:

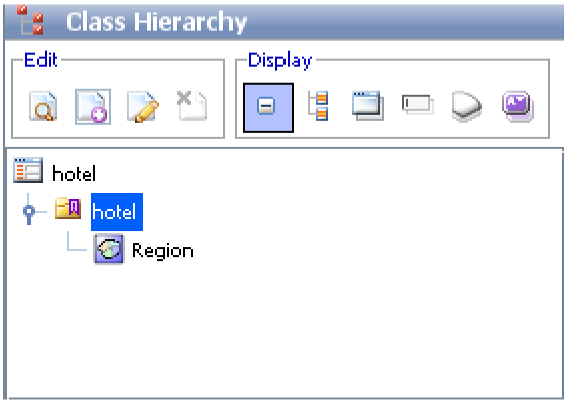


Fig 2.4 *Newly created class in the class hierarchy*

8 We will now set the marks for the class' attributes:

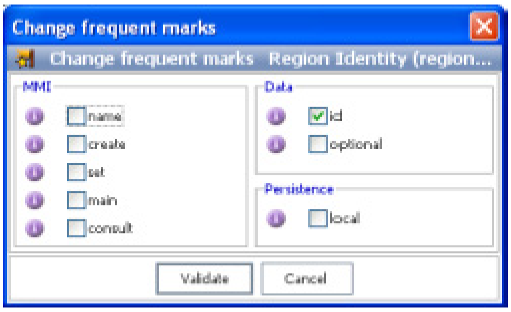
8.1 Select **Windows** ▸ **Data Model** ▸ **Attributes**.

The *Attributes* tab is displayed.

8.2 Select the *Region* class in the Class Hierarchy.

8.3 Right-click the *Region ID* attribute in the *Attributes* tab then select **Change frequent marks...**

The *Change frequent marks* window is displayed:



MMI
☐ name
☐ create
☐ set
☐ main
☐ consult

Data
☑ id
☐ optional

Persistence
☐ local

Fig 2.5 *Setting the frequent marks for an attribute*

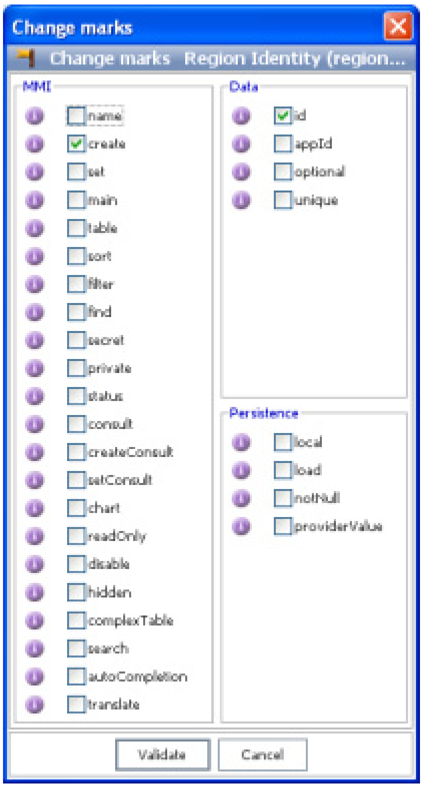
The *id* mark is already set up.

8.4 Select the *create* mark then click **Validate**.

NOTE The *create* mark flags the fields that will be included in the creation forms of the corresponding application objects.

8.5 Right-click the *Region ID* attribute and select **Change marks...**

The *Change marks* window is displayed:



MMI
☐ name
☑ create
☐ set
☐ main
☐ table
☐ sort
☐ filter
☐ find
☐ secret
☐ private
☐ status
☐ consult
☐ createConsult
☐ setConsult
☐ chart
☐ readOnly
☐ disable
☐ hidden
☐ complexTable
☐ search
☐ autoCompletion
☐ translate

Data
☑ id
☐ appId
☐ optional
☐ unique

Persistence
☐ local
☐ load
☐ notNull
☐ providerValue

Fig 2.6 *Setting the frequent marks for an attribute*

8.6 Select the *hidden* mark in the *MMI* area then click **Validate**.

NOTE The *hidden* mark flags the fields that will not be included in the read-only forms of the corresponding objects.

8.7 Set the following frequent marks for the Region Name attribute: create, set, and main.

NOTE The *set* mark flags the fields that will be included in the edit forms of the corresponding objects.

NOTE The *main* mark is a shortcut to the *table*, *sort*, *filter* and *find* marks.
The *table* mark flags the fields that will make up columns in table views.
The *sort* mark flags the fields that will be eligible as sort criteria.
The *filter* mark flags the fields that will be eligible as filter criteria.
The *find* mark flags the fields that will be eligible as search criteria.

8.8 Set the following marks for the Region Name attribute: unique (in the *Data* area).

NOTE The *unique* mark flags the fields for which there cannot be similar values within the same application class.

9 Now take a look at the possible actions for the *Region* class.

Which actions are possible for a class are listed in the *Actions* tab. Application Composer automatically generates a set of 6 generic actions for each created class: *Details (_consult)*, *New (_create)*, *Clone (_clone)*, *Modify (_set)*, *Delete (_delete)* and *Print (_print)*.

We will keep only the *new*, *modify* and *delete* actions for this class.

9.1 Make sure the *Region* class is selected in the Class Hierarchy then select **Windows** ▸ **Data Model** ▸ **Actions**

The *Actions* tab is displayed.

9.2 Hold down the CTRL key and select the _consult, _clone and _print actions.

9.3 Click **Delete** in the *Edit* tool bar.

9.4 Confirm your choice when prompted to do so (one dialog box is displayed per object to be removed).

The first class of our *hotel* room reservation application is now completed.

10 Click **Save** in the main tool bar, and remember to save your work at reasonable time intervals as you progress with this tutorial.

11 Run the application, for example by selecting **Run** ▸ **Display Web** in the main tool bar to preview the application in HTML.

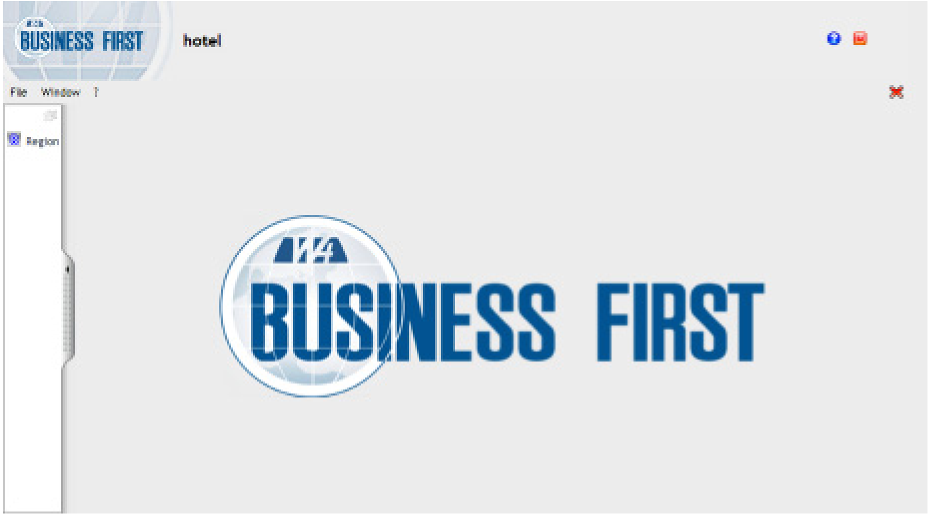


Fig 2.7 *Preview of the generated application (1/2)*

12 Click **Region** in the left menu.

The *Region* tab is displayed.

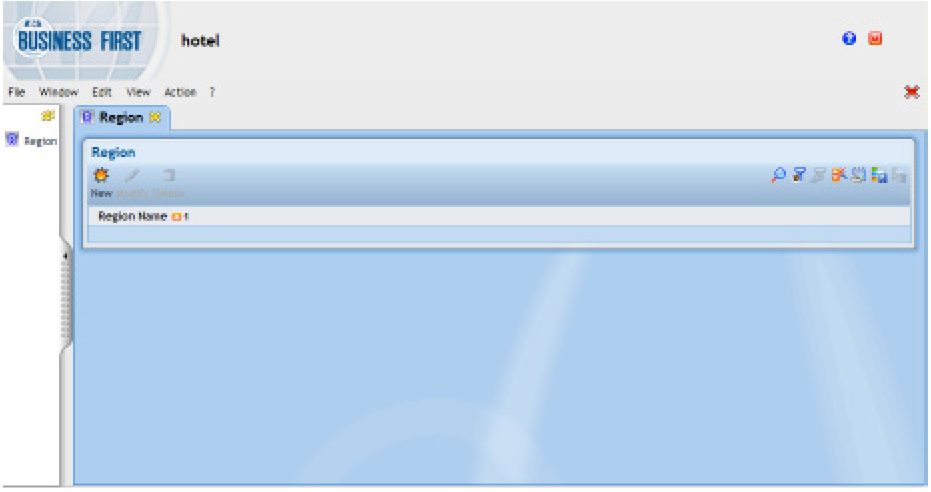


Fig 2.8 *Preview of the generated application (2/2)*

At this stage you can create, edit, and remove regions:

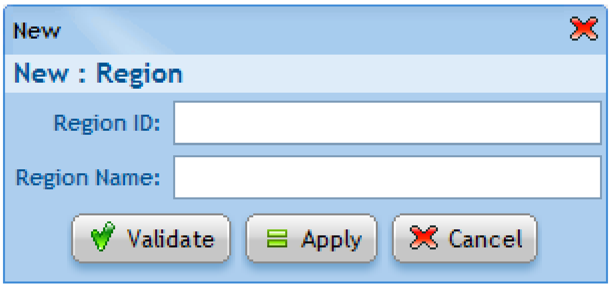
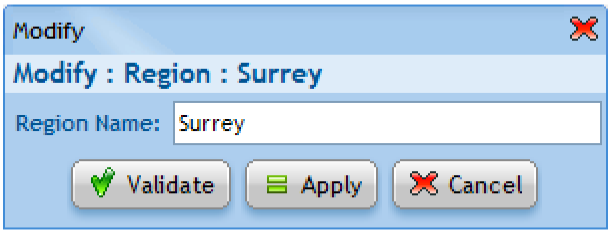


Fig 2.9 *The creation form for the Region class*


Modify

Modify : Region : Surrey

Region Name: Surrey

Fig 2.10 *The edit form for the Region class*

When manipulating regions you will notice the following:

- The *Regions ID* field is required to create a region. Also it is impossible to create two regions with the same region ID. This is because the *id* mark was applied to the *region_id* field.
- Both fields are displayed in the creation form. This is because both fields were applied the *create* mark. However the *region_name* field is also displayed in the edit form unlike the *region_id* field. This is because the *region_name* field was applied the *set* mark unlike the *region_id* field.
- You cannot create two regions with the same region name. This is because the *unique* mark was applied to the *region_name* attribute.
- The three remaining default actions are available from the toolbar in the *Regions* tab.

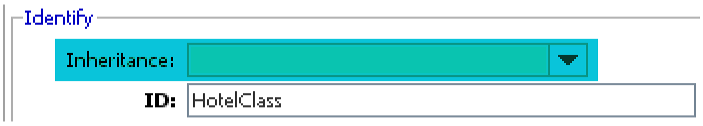
2.2.2 City

The *City* class will be used to further define the geographical locations of the hotels.

Creating this class involves the following actions:

- Creating the class itself,
- Specifying its attributes,
- Specifying the available actions for the class.

PLEASE NOTE There is no inheritance between classes in our application. Make sure that the *Inheritance* field is empty in the *Create class* window or, before creating a class, make sure the *Hotel* project (2nd level item) is selected in the class hierarchy.



Inheritance:

ID: HotelClass

Fig 2.11 *Inheritance field in the class creation window*

TO PERFORM THIS STEP

1 In the *Create class* window, specify the following fields:

- **ID**: city
- **Name**: City
- [Optional] **Image**: city (for instance the *city.gif* image in the *Studio\examples\hotel\images*, in your Application Composer setup directory).

2 Still in the *Create class* window, specify the following attributes:

Table 2.1: The attributes for the *City* class

TYPE	IDENTIFIER	NAME	ID	NAME	LOCAL
Text	city_name	City	**Yes**	**Yes**	No
Relation	city_region	Region	No	No	No
Numeric	city_x	Longitude	No	No	No
Numeric	city_y	Latitude	No	No	No

3 Click **Validate**.

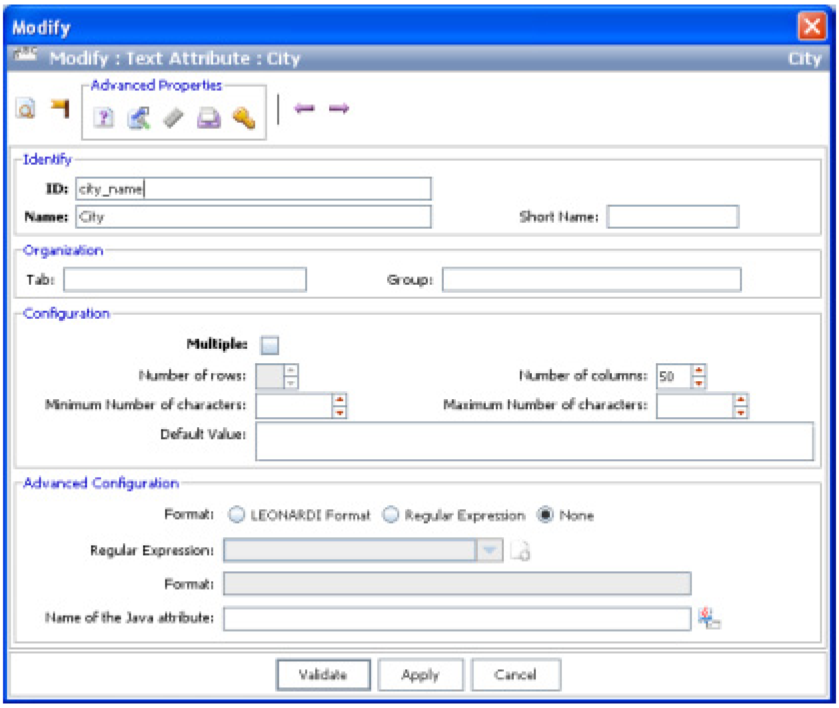
4 We will now modify the *city_name* attribute for a more appropriate component size:

4.1 In the *Attributes* tab, select the *city_name* attribute.

4.2 Click **Modify** in the tool bar.

[ALTERNATIVELY] Double-click the row of the *city_name* attribute.

The *Modify : Text Attribute : City* window is displayed:



ID: city_name

Name: City

Short Name:

Tab:

Group:

Multiple: ☐

Number of rows:

Number of columns: 50

Minimum Number of characters:

Maximum Number of characters:

Default Value:

Format: ○ LEONARDI Format ○ Regular Expression ● None

Regular Expression:

Format:

Name of the Java attribute:

Fig 2.12 *Modifying an attribute*

4.3 In the **Number of columns** field, select or type 50.

4.4 Click **Validate**.

5 Set the marks as listed in the following table:

Table 2.2: The marks for the *City* class attributes

IDENTIFIER	NAME	MARKS
city_name	City	id *, name *, create, main
city_region	Region	create, set, main
city_x	Longitude	create, set, main
city_y	Latitude	create, set, main

* This mark has already been added upon attribute creation.

6 Modify the *city_region* attribute:

6.1 Display the *Modify : Relation Attribute : Region* window.

6.2 In the *Target classes* area, select Region in the *Target class* column.

6.3 Confirm by clicking the **New** button at the end of the row.

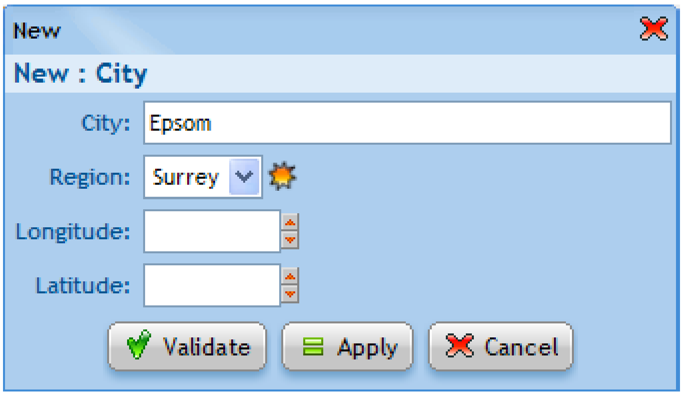
6.4 Click **Validate** in the *Modify : Relation Attribute : Region* window.

7 As we did for the *Region* class, we will remove the *consult*, *clone* and *print* actions from the *City* class.

8 Save the application.

9 Run the application.

At this stage, you can create, edit, and remove cities:



City: Epsom

Region: Surrey

Longitude:

Latitude:

Fig 2.13 *The creation form for the City class*

2.2.3 *Manager*

The *Manager* class will be used to describe the hotel managers.

Creating this class involves the following actions:

- Creating the class itself,
- Specifying its attributes,
- Specifying the available actions for the class.

TO PERFORM THIS STEP

1 In the *Create class : Class* window, specify the following fields:
  - **ID**: manager
  - **Name**: Manager
  - [Optional] **Image**: manager (for instance the *manager.gif* image in the *Studio\examples\hotel\images*, in your Application Composer setup directory).

2 Still in the *Create class* window, specify the following attributes:

Table 2.3: The attributes for the *Manager* class

TYPE	IDENTIFIER	NAME	ID	NAME	LOCAL
Text	manager_id	Manager ID	**Yes**	No	No
Text	manager_password	Password	No	No	No
Text	manager_lastname	Manager Last Name	No	**Yes**	No
Text	manager_firstname	Manager First Name	No	**Yes**	No
File	manager_picture	Manager Picture	No	No	No
Time	manager_date	Start Date	No	No	No
Relation	manager_sites *				
Relation	manager_clients*				

* The *manager_sites* and *manager_clients* attributes will be created at a later step as they are relational attributes whose target classes are not yet available.

3 Click **Validate**.

4 Modify the *manager_picture* attribute:

4.1 Display the *Modify : File Attribute : Manager Picture* window.

4.2 In the *Type* area, select the **Image** option button.

4.3 [Optional] In the **Default value** field, specify Studio\examples\hotel\images\manager.gif in your Application Composer setup directory.

4.4 In the **Folder** field, specify Studio\examples\hotel\images\managers in your Application Composer setup directory.

4.5 Click **Validate**.

5 The *manager_date* attribute is a class attribute of type time. It will be used to specify a starting date. Modify this attribute so as to display the current time by default:

5.1 Display the *Modify : Time Attribute : Start date* window.

5.2 In the *Configuration* area, select both the **Date** radio button and the **Default current date** check box.

5.3 Click **Validate**.

6 Set the marks as listed in the table below:

Table 2.4: The marks for the *Manager* class attributes

IDENTIFIER	NAME	MARKS
manager_id	Manager ID	id *, create
manager_password	Password	create, set, secret, private
manager_lastname	Manager Last Name	name *, create, set, main, complexTable
manager_firstname	Manager First Name	name *, create, set, main, complexTable
manager_picture	Manager Picture	create, set, main, optional, complexTable
manager_date	Start Date	main, consult, complexTable

* This mark has already been added upon attribute creation.

NOTE The *secret* mark flags a field which value is never displayed. Typically password fields are given this mark.

NOTE The *private* mark flags the fields whose value cannot be copied when duplicating the corresponding objects.

NOTE The *complexTable* mark flags a field which will be part of a complex table, i.e. a table whose values are set via relations to other tables.

NOTE The *optional* mark flags the fields which do not have to be specified for a form to be validated.

NOTE The *consult* mark flags the fields whose values are displayed although non editable in edit forms.

7 Specify the group as listed in the table below:

Table 2.5: The groups for the *Manager* class attributes

IDENTIFIER	NAME	GROUP
manager_id	Manager ID	Personal Details
manager_password	Password	
manager_lastname	Manager Last Name	
manager_firstname	Manager First Name	
manager_picture	Manager Picture	
manager_date	Start Date	Data

NOTE Specifying the group to which they belong allows certain attributes to be organized into the same frame, within the forms. To specify the group for an attribute, you need to display its modification form, via the *Attributes* tab.

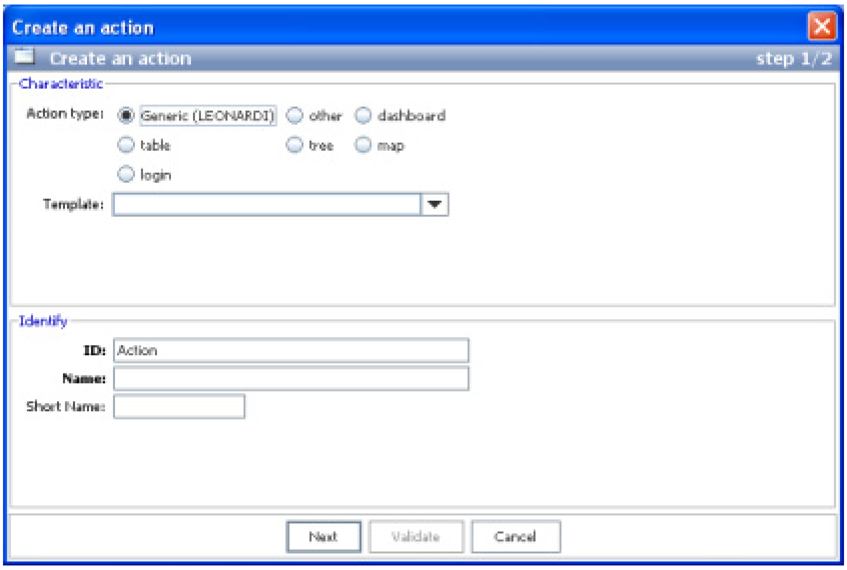
NOTE To specify the group for multiple attributes one after the other, it can be a good idea to click *Apply* instead of *Validate*. This will automatically display the form for the next attribute instead of closing the current form.

8 Contrary to what was done for the previous classes, we will not remove any actions. We will even create an additional action to allow users to generate a PDF file that will contain the manager details. The action for PDF file creation is not available in the free version of Application Composer.

8.1 Select the *Manager* class in the Class Hierarchy.

8.2 Click **Create an action...** in the *Creation* tool bar of the *Actions* tab.

The first screen of the *Create an action* window is displayed:



Action type: ● Generic (LEONARDI)

ID: Action

Fig 2.14 *Creating an action*

8.3 Specify the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Print Form (_printXSLFOForm)
- **ID**: printXSLFOForm_manager

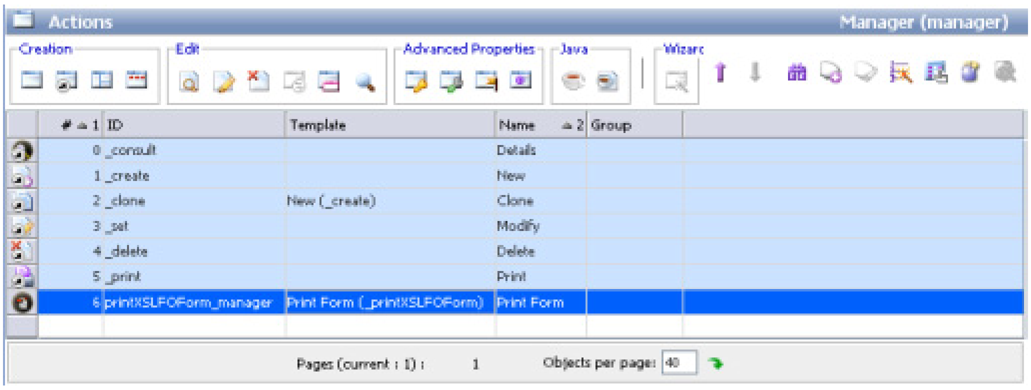
8.4 Click **Next**.

The second screen of the *Create an action* window is displayed:

8.5 Set the **Image** field to Studio\examples\hotel\images\pdf.gif in your Application Composer setup directory.

8.6 Click **Validate**.

The new action *printXSLFOForm_manager* is now available at the end of the list of the *Manager* class actions:



# 1	ID	Template	Name 2	Group
0	_consult		Details	
1	_create		New	
2	_clone	New (_create)	Clone	
3	_set		Modify	
4	_delete		Delete	
5	_print		Print	
6	printXSLFOForm_manager	Print Form (_printXSLFOForm)	Print Form	

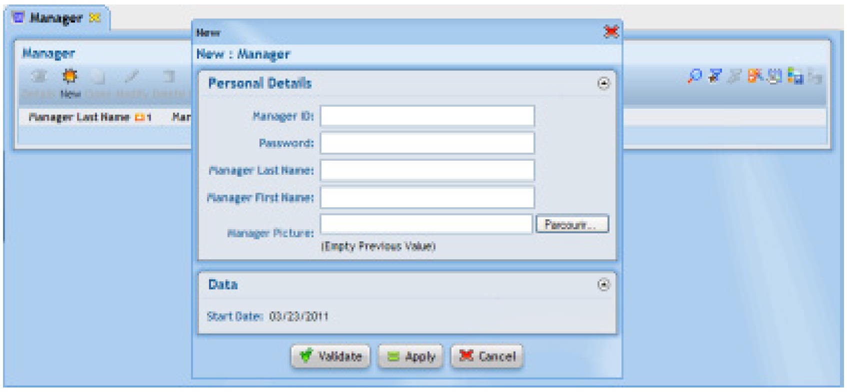
Pages (current : 1) : 1 Objects per page: 40

Fig 2.15 *Newly created action*

8.7 Save the application.

8.8 Run the application.

At this stage, you can create, clone, edit, and remove managers, and also view or print manager details.



Start Date: 03/23/2011

Fig 2.16 *The creation form for the Manager class*

2.2.4 Site

The *Site* class will be used to describe the hotels.

Creating this class involves the following actions:

- Creating the class itself,
- Specifying its attributes,
- Creating a structure attribute to specify all together the street, the zip code and the city of a hotel.

TO PERFORM THIS STEP

1. In the *Create class* window, specify the following fields:
    - **ID**: site
    - **Name**: Site
    - [Optional] **Image**: For instance the *establishment.gif* image in the *Studio\examples\hotel\images*, in your Application Composer setup directory.
2. Still in the *Create class* window, specify the following attributes:

Table 2.6: The attributes for the *Site* class

TYPE	IDENTIFIER	NAME	ID	NAME	LOCAL
Numeric	site_id	Site ID	**Yes**	No	No
Text	site_name	Site Name	No	**Yes**	No
Enumerated	site_category	Category	No	No	No
Relation	site_manager	Manager	No	No	No
Text	site_tel	Telephone	No	No	No
Structure	site_address *				
Text	site_description	Description	No	No	No
File	site_picture	Picture	No	No	No
Numeric	site_rooms_numbers *				
Relation	site_rooms *				

* Three attributes will be created at later steps: *site_address*, which is a structure attribute, *site_rooms_number* which is an import attribute, and *site_rooms*, which is a relation attribute.

3. Click **Validate**.
4. The class identifier field *site_id* is of type numeric. As a numeric identifier must be a positive integer, in its edit window, the **Type** field must be set to Integer and the **Minimum** field must be set to 0.
5. We will also specify the maximum number of characters available for the *site_name* attribute: The **Maximum number of characters** field must be set to 60.

6 Set the marks as listed in the table below:

Table 2.7: The marks for the *Site* class attributes

IDENTIFIER	NAME	MARKS
site_id	Site ID	id *, hidden, providerValue
site_name	Site Name	name *, create, set, main, private, unique, complexTable
site_category	Category	create, set, main, complexTable
site_manager	Manager	create, set, main, load
site_tel	Telephone	create, set, optional
site_description	Description	create, set, optional
site_picture	Picture	create, set, optional, private

* This mark has already been added upon attribute creation.

NOTE The *providerValue* mark flags the fields whose values will be automatically set by a data provider.

NOTE The *load* mark flags a field whose loading will be forced during load requests that do not specify the required fields.

7 The *site_name* has also a *specific* mark, i.e. a mark that does not exist as standard and needs to be created by the user: *SITE_COMPLEX_LIST*. To create this mark:

7.1 In the *Attributes* tab, right-click and select **Specific Marks...**

The *Specific Marks* window is displayed:

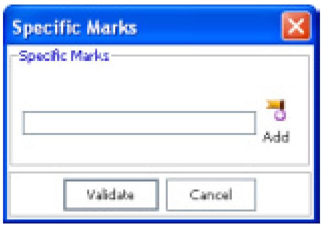


Fig 2.17 *Creating a specific mark (1/2)*

7.2 Type SITE_COMPLEX_LIST in the text field then click **Add**.

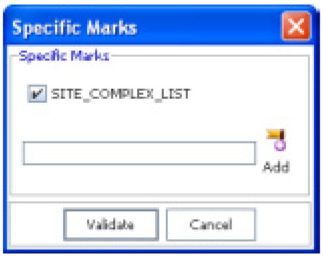


Fig 2.18 *Creating a specific mark (2/2)*

7.3 Click **Validate**.

7.4 Apply this specific mark to the fields *site_category* and *site_picture*.

To apply an already created specific mark, right-click the appropriate attribute and select **Specific Marks...**

In the *Specific marks* window just check the appropriate specific mark and validate.

NOTE This specific mark will also be applied to the fields *site_rooms_number* and *site_rooms* after their creation.

8 The *site_description* class attribute is a multiple text field. We will specify its behavior:

8.1 Display the *Modify : Text Attribute : Description* window.

8.2 In the *Configuration* area, select the **Multiple** check box.

8.3 Type 3 in the **Number of rows** field.

8.4 Type 30 in the **Number of columns** field.

8.5 Click **Validate**.

9 The *site_tel* class attribute is a text attribute with a short name and a particular Application Engine format:

9.1 Display the *Modify : Text Attribute : Telephone* window.

9.2 Type Phone in the **Short name** field.

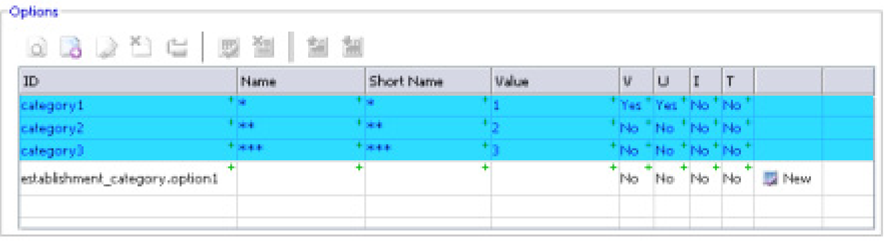
9.3 In the *Advanced configuration* area, select the LEONARDI format radio button and type [0-99][0-99][0-99][0-99][0-99] in the **Format** field.

9.4 Click **Validate**.

10 The *site_category* class attribute is an enumerative attribute. It allows you to specify the number of stars granted to a hotel. We will create three category types:

10.1 Display the *Modify : Enumerated Attribute : Category* window.

10.2 Specify the three categories in the *Options* area, as follows:



ID	Name	Short Name	Value	V	U	I	T	
category1	*	*	1	Yes	Yes	No	No	
category2	**	**	2	No	No	No	No	
category3	***	***	3	No	No	No	No	
establishment_category.option1				No	No	No	No	New

Fig 2.19 *Creating an enumerated attribute*

NOTE Specifying *Yes* for the *V* column means the corresponding option will be selected by default.

10.3 Click **Validate**.

11 Modify the *site_manager* attribute:

11.1 Display the *Modify : Relation Attribute : Manager* window.

11.2 In the **Target Class** column, select Manager.

11.3 Confirm by clicking **New** at the end of the row.

11.4 Click **Validate**.

12 Specify the group and tab as listed in the table below:

Table 2.8: The groups and tabs for the *Site* class attributes

IDENTIFIER	NAME	TAB	GROUP
site_id	Site ID	Description	Site Details
site_name	Site Name		
site_category	Category		
site_manager	Manager		
site_tel	Telephone		Phone & Addr.
site_description	Description		Overview
site_picture	Picture		

13 Modify the *site_picture* attribute:

13.1 Display the *Modify : File Attribute : Picture* window.

13.2 In the **Folder** field, specify the *Studio\examples\hotel\images\establishments* directory, in your Application Composer setup directory.

13.3 Leave the **Default Value** field empty.

13.4 Click **Validate**.

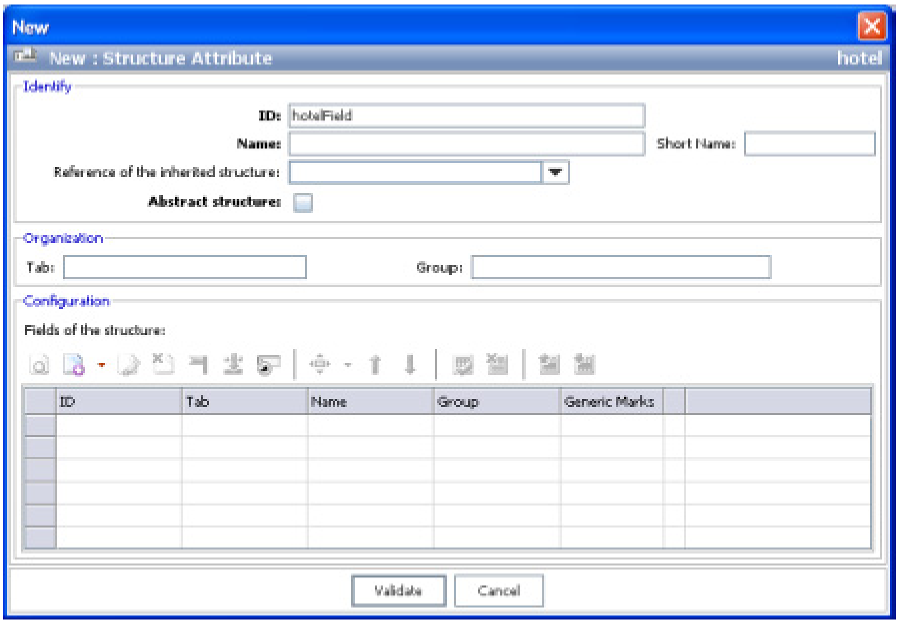
14 We will now create the *site_address* structure attribute. It allows us to specify the street, the zip code and the city of a hotel all together. In order to create this attribute, we will first

create the attributes of the structure at the level of the project. By doing so, the same attribute will be available as a model for many attributes in different classes.

14.1 Select the Hotel project (2nd level item) in the class hierarchy.

14.2 In the *Attributes* tab, click the **New : Structure Attribute** button.

The *New : Structure Attribute* window is displayed:



ID: hotelField

Fig 2.20 *Creating a structure attribute*

14.3 Set the **ID** field to *address* and the **Name** field to Address.

14.4 Create the 4 attributes that make up the structure, using the following information.

To create an attribute click **New...** in the *Configuration* area, then select the appropriate type from the tool bar that is displayed:

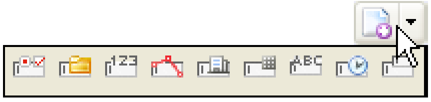
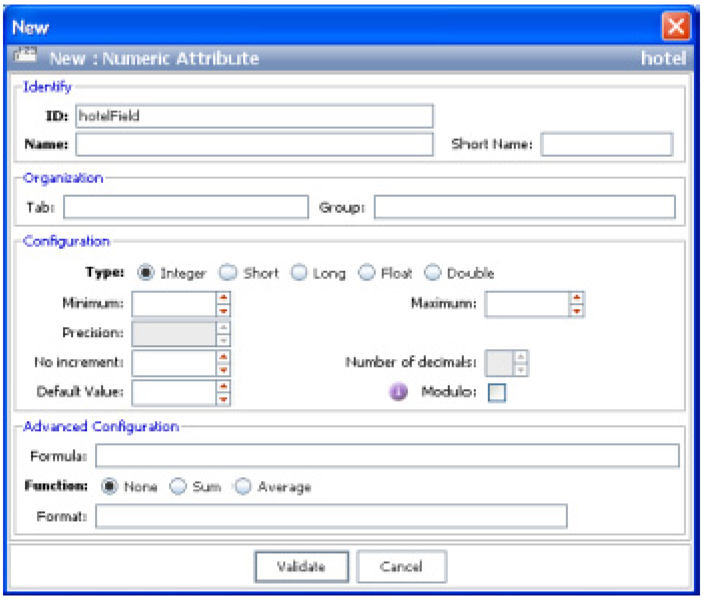


Fig 2.21 *Creating an attribute (1/2)*

The *New : <Type> Attribute* window is displayed:



ID: hotelField
Name:
Short Name:
Tab:
Group:
Type: ● Integer ○ Short ○ Long ○ Float ○ Double
Minimum:
Maximum:
Precision:
No increment:
Number of decimals:
Default Value:
Modulo: ☐
Formula:
Function: ● None ○ Sum ○ Average
Format:

Fig 2.22 *Creating an attribute (2/2)*

This is where you will specify the attribute parameters.

Table 2.9: The attributes making up the *site_address* attribute

TYPE	IDENTIFIER	NAME
Numeric	address_id	Address Identifier
Text	address_street	Street
Text	address_zipcode	Zip Code
Relation	address_city	City

The *address_city* attribute is a simple association relation targeting the *City* class.

NOTE *Association* is just one of the four available types for a relation attribute. In our example, the *Association* type can be justified by the fact that a city may belong to multiple addresses, and removing an address does not result in automatically removing the corresponding city.

The numeric identifier *address_id* is positive: Set the **Minimum** field to 0.

The *address_street* attribute has its **Number of columns** parameter set to 30, for the address to be readable.

The *address_zipcode* attribute, besides having the **Short name** Zip has the **LEONARDI format** [0-9][0-9][0-9][0-9][0-9].

14.5 When the attributes that make up the structure have been created, make sure you set their marks: Still in the *New : Structure Attribute* window, right-click the appropriate attribute and select **Change frequent marks** / **Change marks**.

Table 2.10: The marks for the attributes making up the *site_address* attribute

IDENTIFIER	NAME	MARKS
address_id	Address Identifier	id, hidden, providerValue
address_street	Street	create, set
address_zipcode	Zip Code	create, set
address_city	City	create, set

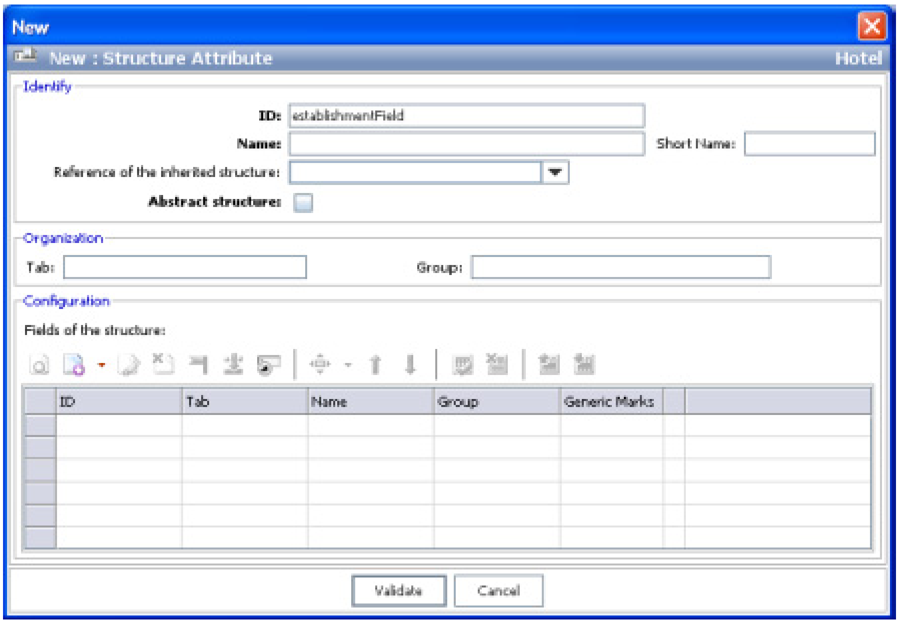
14.6 Click **Validate** in the *New : Structure Attribute* window to complete the creation of the structure attribute.

15 As soon as the structure attribute is available at the application level, we will use it in the *Site* class:

15.1 Select the *Site* class in the class hierarchy.

15.2 In the *Attributes* tab, select **New : Structure attribute** in the tool bar.

The *New : Structure Attribute* window is displayed:



ID: establishmentField

Fig 2.23 *Creating a structure attribute*

15.3 Set the **ID** field to site_address.

15.4 Set the **Name** field to Site Address.

15.5 Set the **Reference of the inherited structure** field to Address.

15.6 Set the **Tab** field to Description and the **Group** field to Phone & Addr.

15.7 Click **Validate**.

The 4 address attributes do not need to be re-created as they are inherited from the original *address* attribute.

15.8 Set its marks set to create, set and private.

15.9 Use the **Move up** ↑ and **Move down** ↓ buttons in the *Advanced* tool bar so *site_address* appears between *site_manager* and *site_tel*.

16 Save the application.

17 Run the application.

At this stage, you can create, clone, edit, and remove hotel sites, and also view or print hotel details.

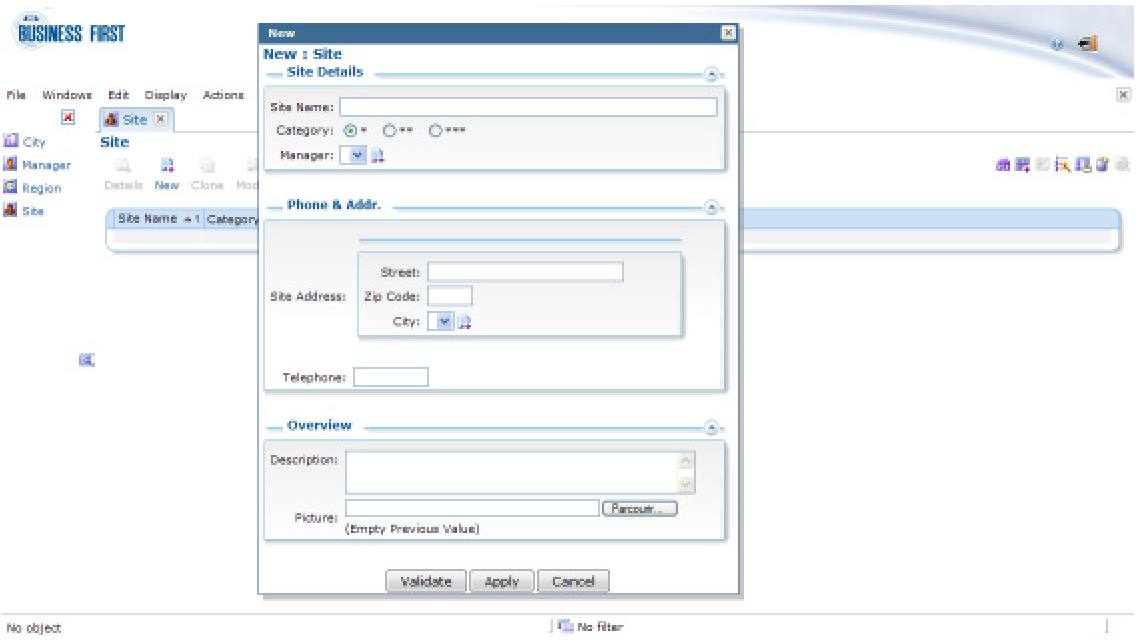


Fig 2.24 *The creation form for the Site class*

2.2.5 *Client*

The *Client* class will be used to describe the customers within our hotel room reservation application.

Creating this class involves the following actions:

- Creating the class itself,
- Specifying its attributes,
- Creating a default sort.

TO PERFORM THIS STEP

1 In the *Create class* window, specify the following fields:

  1.1 **Identifier**: client

  1.2 **Name**: Client

  1.3 [Optional] **Image**: client (for instance the *client.gif* image in the *Studio\examples\hotel\images*, in your Application Composer setup directory).

2 Still in the *Create class* window, specify the following attributes:

Table 2.11: The attributes for the *Client* class

TYPE	IDENTIFIER	NAME	ID	NAME	LOCAL
Numeric	client_number	Client #	**Yes**	No	No
Enumerated	client_title	Title	No	No	No
Text	client_lastname	Client Last Name	No	**Yes**	No
Text	client_firstname	Client First Name	No	**Yes**	No
Structure	client_address *				
Table	client_tels	Phone Numbers	No	No	No
Relation	client_reservation *				
Numeric	client_amount *				
Relation	client_manager *				

* These attributes will be created at later steps

3 Click **Validate**.

4 Set the three options of the *client_title* enumerated attribute: title_mister, title_miss and title_missus for respectively Mr. (the default option), Miss and Mrs. Their associated values going respectively from 0 to 2.

5 Set the tabs, groups, and marks for the *Client* class attributes.

Table 2.12: The tabs, groups, and marks for the *Client* class attributes

TYPE	IDENTIFIER	NAME	TAB	GROUP	MARKS
Numeric	client_number	Client #	Description	Personal Details	id *, main, private, unique, providerValue
Enumerated	client_title	Title			create, set, main
Text	client_lastname	Client Last Name			name *, create, set, main
Text	client_firstname	Client First Name			name *, create, set, main
Table	client_tels	Phone Numbers		Phone & Addr.	create, set, optional

* This mark has already been added upon attribute creation.

6 The structure attribute *client_address* is created in the same way as *site_address* earlier:

6.1 Select the *Client* class in the class hierarchy.

6.2 Click to create a structured attribute.

The *New : Structure Attribute* window is displayed.

6.3 Set the **ID** field to client_address.

6.4 Set the **Name** field to Client Address.

6.5 Set the **Reference of the inherited structure** field to Address.

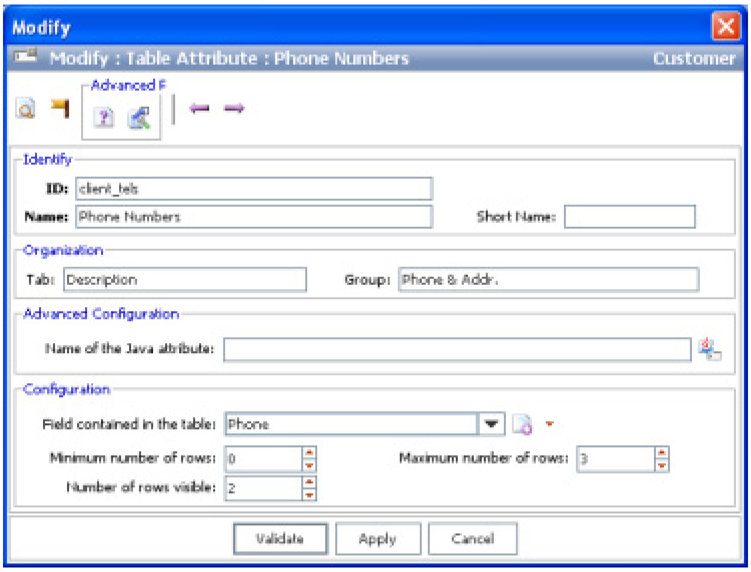
6.6 Set the **Tab** field to Description and the **Group** field to Phone & Addr.

6.7 Click **Validate**.

6.8 Set the marks to create, set, and private.

7 We will now complete the definition of the *client_tels* table attribute:

7.1 Display the *Modify : Table Attribute : Phone Numbers* window:



ID: client_tels
Name: Phone Numbers
Short Name:
Tab: Description
Group: Phone & Addr.
Name of the Java attribute:
Field contained in the table: Phone
Minimum number of rows: 0
Maximum number of rows: 3
Number of rows visible: 2

Fig 2.25 *Modifying a table attribute*

7.2 Click New... to the right of the **Field contained in the table** field and create a new text attribute. In the *New : Text Attribute* window that is displayed, specify the following fields and click **Validate**:

- **ID**: client_tel
- **Name**: Telephone
- **LEONARDI Format**: [0-99][0-99][0-99][0-99][0-99]

7.3 Back in the *Modify : Table Attribute : Phone Numbers* window, specify the following fields and click **Validate**:

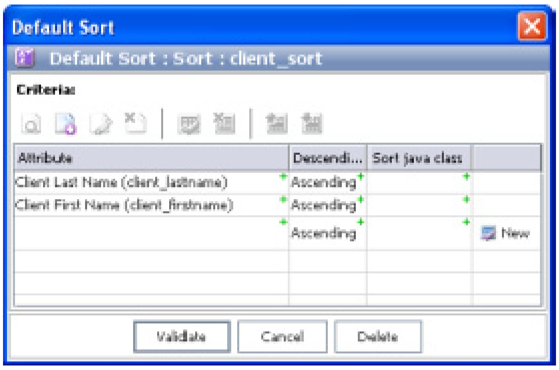
- **Minimum number of rows**: 0
- **Maximum number of rows**: 3
- **Number of rows visible**: 3

8 A default sort will be applied to the *client* class so the customers will be shown according to their last name then first name during a list consultation:

8.1 Right-click the *Client* class in the class hierarchy and select the **Default Sort...** option from the context menu

The *New : Sort* window is displayed.

8.2 Add two sort criteria: For the client last name then its first name.



Attribute	Descendi...	Sort java class	
Client Last Name (client_lastname)	Ascending		
Client First Name (client_firstname)	Ascending		
	Ascending		New

Fig 2.26 *Creating a sort criterion for a class*

8.3 Click **Validate**.

9 Save the application.

10 Run the application.

At this stage, you can create, clone, edit, and remove clients, and also view or print client details.

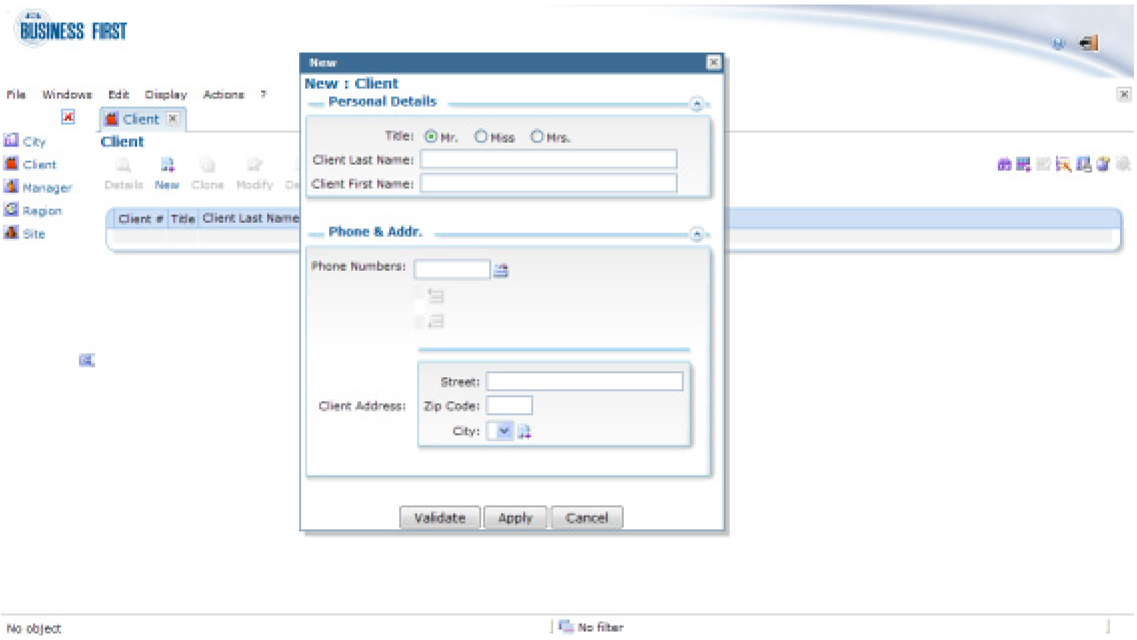


Fig 2.27 *The creation form for the Client class*

2.2.6 Room

The *Room* class will be used to describe the rooms in the hotels managed by the hotel room reservation application.

Creating this class involves the following actions:

- Creating the class itself,
- Specifying its attributes,
- Creating a default sort.

TO PERFORM THIS STEP

1 In the *Create class* window, specify the following fields:
  - **Identifier**: room
  - **Name**: Room
  - **Image**: room (for instance the *room.gif* image in the *Studio\examples\hotel\images*, in your Application Composer setup directory)

2 Still in the *Create class* window, specify its attributes:

NOTE The *room_site* attribute will be created at a later step as it is a reverse relation that needs the yet to be created *site_room* attribute.

NOTE The *room_rate* will also be created at a later step.

Table 2.13: The attributes for the *Room* class

TYPE	IDENTIFIER	NAME	ID	NAME	LOCAL
Numeric	room_id	Room ID	**Yes**	No	No
Relation	room_site *				
Numeric	room_number	Room #	No	**Yes**	No
Enumerated	room_type	Type	No	No	No
Numeric	room_rate *				
Enumerated	room_smoking	Smoking	No	No	No
Enumerated	room_bath	Bathroom	No	No	No
Enumerated	room_equipment	Amenities	No	No	No
File	room_picture	Picture	No	No	No

3 Click **Validate**.

4 For the *room_number* attribute, set the **Minimum** field to 0 (a room number cannot be negative), and the **Increment step** field to 1 (increment of 1 between each room).

5. For the *room_smoking* enumerated attribute, set the **Type of choice** field to Boolean in order to have graphical input components of type “yes/no”. This attribute has only two values, which you create in the **Options** area:
   - The default value no_smoking, corresponding to No, and with value 0
   - The option smoking for Yes, and with value 1
6. The *room_type* enumerated attributes has 3 options: room_type_single, room_type_double and room_type_suite.
7. The *room_bath* enumerated attributes has 2 values: shower and bath.
8. The *room_equipment* enumerated attribute has 4 different values that can be selected independently. We should check the Multiple choice check box and create the options equipment_air_conditioning, equipment_minibar, equipment_wifi and equipment_television, with no default value.
9. Specify the marks and groups as listed in the table below:

Table 2.14: The marks and groups for the *Room* class attributes

IDENTIFIER	NAME	GROUP	MARKS
room_id	Room ID	Description	id *, hidden, providerValue
room_number	Room #		name *, create, set, main, complexTable
room_type	Type		create, set, main, complexTable
room_smoking	Smoking	Details	create, set, main
room_bath	Bathroom		create, set, main
room_equipment	Amenities		create, set, optional
room_picture	Picture	Picture	create, set, table, optional

* This mark has already been added upon attribute creation.

10. We will now create the *room_rate* attribute. It is a numerical field having units. As we did for the *site_address* attribute, we will create a global amount attribute that describes a price amount at the application level.

10.1 Select the Hotel project in the class hierarchy.

10.2 Click **New : Numeric Attribute** in the main tool bar.

The *New : Numeric Attribute* window is displayed.

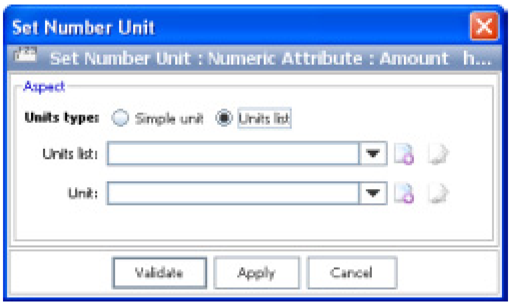
10.3 Set the following values:

- **ID**: amount
- **Name**: Amount
- **Type**: Double
- **Minimum**: 0.0
- **Increment step**: 10.0
- **Number of decimals**: 2
- **Function**: Sum

10.4 Click **Validate**.

10.5 Right-click the *Amount* attribute in the *Attributes* tab then select **Set Number Unit...**

The *Set Number Unit* window is displayed:



Units type: ○ Simple unit ● Units list

Units list:

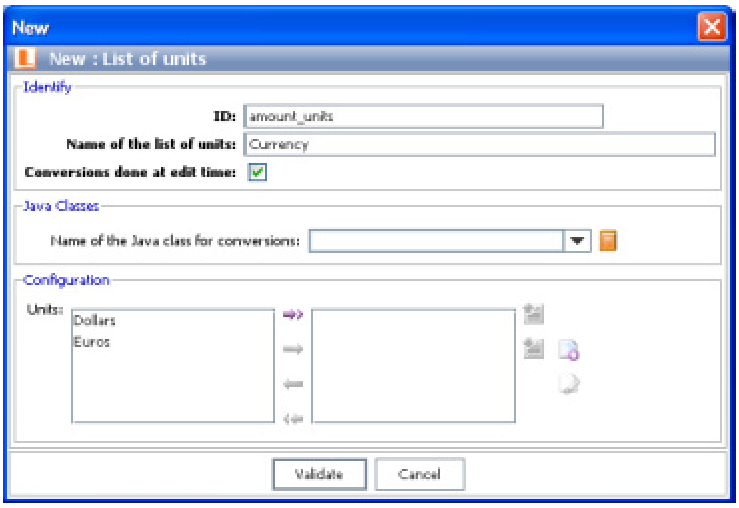
Unit:

Fig 2.28 *Setting number units (1/3)*

10.6 Select the **Unit list** radio button.

10.7 Click **New...** to the right of the **Unit list** text field.

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



ID: amount_units

Name of the list of units: Currency

Conversions done at edit time: ☑

Name of the Java class for conversions:

Units:
Dollars
Euros

Fig 2.29 *Setting number units (2/3)*

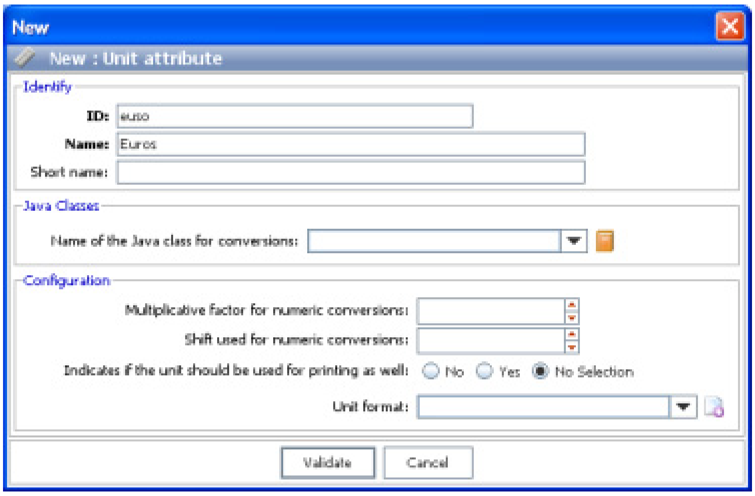
10.8 Set the **ID** field to amount_units and the **Name of the list of units** field to Currency.

10.9 Select the **Conversions done at edit time** check box.

NOTE Selecting this check box will result in automatically updating the *Amount* value when the unit changes from *Euro* to *Dollar* and vice versa.

10.10 Click **New...** to the right of the *Configuration* area to add new units.

The *New : Unit attribute* window is displayed:



ID: euro

Name: Euros

Short name:

Name of the Java class for conversions:

Multiplicative factor for numeric conversions:

Shift used for numeric conversions:

Indicates if the unit should be used for printing as well: ○ No ○ Yes ● No Selection

Unit format:

Fig 2.30 *Setting number units (3/3)*

10.11 The first unit we will create is *euro*: Set the **ID** field to euro and the **Name** field to Euros then click **Validate**. When prompted for *Would you like to create a conversion Java class?* choose **No**.

NOTE We will not be using a Java class as the conversion will be implemented by configuring a multiplicative factor in the next step. You resort to a Java class when the conversion requires a complex formula for instance.

Euro will be the default unit of the list and will be used as basis for the conversions.

10.12 Add a second unit to the list: Set the **ID** field to dollar, the **Name** field to Dollars, and the **Multiplicative factor for numeric conversions** field to 0.753 then click **Validate**. This will be used as multiplying factor for the Euros to Dollars conversion.

10.13 Back in the *New : List of units* window, click **Validate**.

10.14 Back in the *Set Number Unit* window, click **Validate**.

10.15 Once this global attribute is created, the *room_rate* attribute can be added to the *room* class. In the *Room* class, click **New typed field attribute**. In the *New : Typed field attribute* window, set the **ID** field to room_rate, the **Name** field to Price, the **Group** field to Description, and the **Type of Fields** field to Amount (Project: Hotel). Click **Validate**.

The *room_rate* attribute will be created based on the *Amount* attribute of the *Hotel* project.

10.16 Set the marks for the *room_rate* attribute: *create*, *set*, *main*, and *complexTable*.

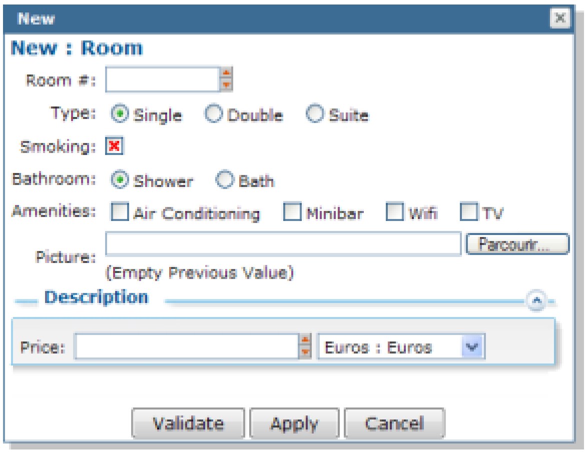
10.17 Finally, the specific mark *SITE_COMPLEX_LIST* can be set to the room_number, room_type and room_rate attributes.

11 We will set a default sort for the *Room* class in order to sort the rooms of a hotel by their number.

12 Save the application.

13 Run the application.

At this stage, you can create, clone, edit, and remove rooms, and also view or print room details.



Room #:

Type: ● Single ○ Double ○ Suite

Smoking: ☒

Bathroom: ● Shower ○ Bath

Amenities: ☐ Air Conditioning ☐ Minibar ☐ Wifi ☐ TV

Picture: (Empty Previous Value)

Price: Euros : Euros

Fig 2.31 *The creation form for the Room class*

2.2.7 *Reservation*

The *reservation* class will be used to describe the reservations for the sites that are managed by the room reservation application.

Creating this class involves the following actions:

- Creating the class itself,
- Specifying its attributes,
- Creating a default sort,
- Specifying the available actions for the class,
- Creating a class behavior.

TO PERFORM THIS STEP

1 In the *Create class* window, specify the following fields:

- **Identifier**: reservation
- **Name**: Reservation
- **Image**: reservation (for instance the `reservation.gif` image in the `Studio\examples\hotel\images`, in your Application Composer setup directory)

2 Still in the *Create class* window, specify its attributes.

The attributes of the reservation class are listed in the table below:

Table 2.15: The attributes for the *Reservation* class

TYPE	IDENTIFIER	NAME	ID	NAME	LOCAL
Numeric	reservation_number	Reservation #	**Yes**	**Yes**	No
Enumerated	reservation_state	State	No	No	No
Time	reservation_date	Date Reserved	No	No	No
Relation	reservation_client	Client	No	**Yes**	No
Time	reservation_checkin	Check-in	No	No	No
Time	reservation_checkout	Check-out	No	No	No
Numeric	reservation_days	Nights	No	No	No
Relation	reservation_site	Hotel	No	No	No
Relation	reservation_rooms	Rooms	No	No	No
Numeric	reservation_adults	Adults	No	No	No
Numeric	reservation_children	Children	No	No	No
Numeric	reservation_day_amount *				
Numeric	reservation_amount	Total Amount	No	No	**Yes**

* The *reservation_day_amount* attribute is an import attribute and will be created at a later step.

3 Create the *TREEMAP_TOOLTIP* specific mark then apply it to the *reservation_check_in*, *reservation_check_out* and *reservation_site* attributes.

4 Attributes *reservation_adults* and *reservation_children* have the following limitations: the increment is of 1 person, and there cannot be fewer than 0 or more than 4 people in a room. Finally, by default there must be one adult but no child for a reservation.

5 The *reservation_client* attribute is a relation attribute to the *Client* class similar to the ones previously created, but with an additional parameter allowing you to ignore the context of the view: Select the Ignore context check box.

6 The *reservation_rooms* attribute is a relational attribute to the *Room* class for which it is necessary to specify two new parameters: the context-sensitive relational attribute and the fact that the relationship can be of multiple type: Select the Multiple check box and set the **Contextual relation attribute** field to Hotel (reservation_site).

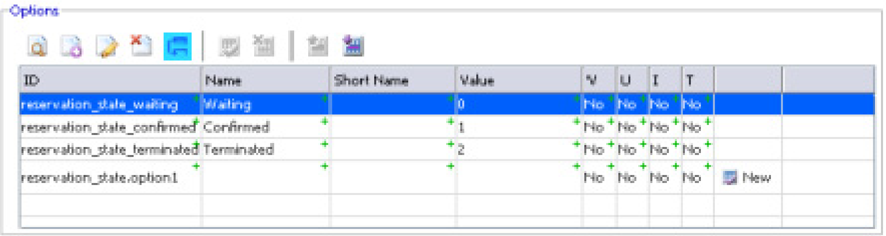
NOTE Setting this option will result in avoiding displaying rooms that do not belong to the current hotel.

7 The *reservation_state* attribute is an enumerated attribute whose options are as follows:

- *reservation_state_waiting* for the *Waiting* state
- *reservation_state_confirmed* for *Confirmed*
- *reservation_state_terminated* for *Terminated*

8 Once these options are created, we will add rules to the first one.

8.1 Select this option in the list then click **Rules...**

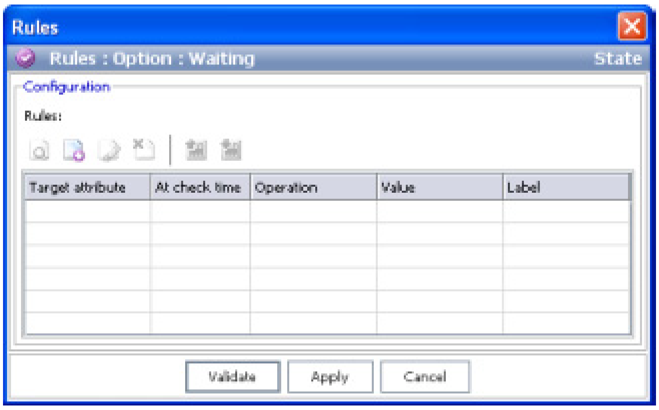


Options

ID	Name	Short Name	Value	V	U	I	T	
reservation_state_waiting	Waiting		0	No	No	No	No	
reservation_state_confirmed	Confirmed		1	No	No	No	No	
reservation_state_terminated	Terminated		2	No	No	No	No	
reservation_state.option1				No	No	No	No	New

Fig 2.32 *Setting rules (1/3)*

The *Rules : Options : Waiting* window is displayed:

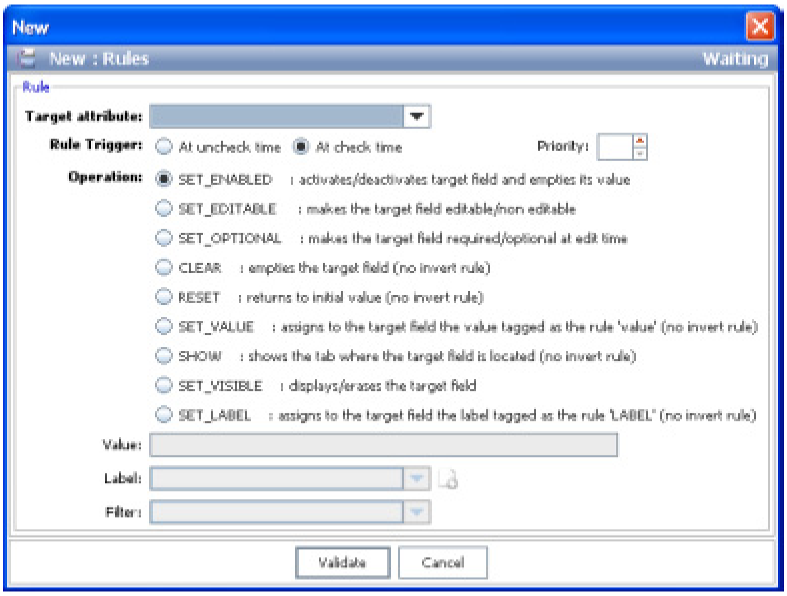


Target attribute	At check time	Operation	Value	Label

Fig 2.33 *Setting rules (2/3)*

8.2 Click **New...**

The *New : Rules* window is displayed:



Target attribute:

Rule Trigger: ○ At uncheck time ● At check time Priority:

Operation:
● SET_ENABLED : activates/deactivates target field and empties its value
○ SET_EDITABLE : makes the target field editable/non editable
○ SET_OPTIONAL : makes the target field required/optional at edit time
○ CLEAR : empties the target field (no invert rule)
○ RESET : returns to initial value (no invert rule)
○ SET_VALUE : assigns to the target field the value tagged as the rule 'value' (no invert rule)
○ SHOW : shows the tab where the target field is located (no invert rule)
○ SET_VISIBLE : displays/erases the target field
○ SET_LABEL : assigns to the target field the label tagged as the rule 'LABEL' (no invert rule)

Value:

Label:

Filter:

Fig 2.34 *Setting rules (3/3)*

8.3 Select Client from the **Target attribute** drop-down list then select the **SET_EDITABLE** radio button. Click **Validate**.

8.4 Proceed in the same way for the 7 other attributes: *Check-in*, *Check-out*, *Nights*, *Hotel*, *Rooms*, *Adults* and *Children*.

NOTE Creating the rules with the SET_EDITABLE option for *reservation_state_waiting* will result in enabling the target fields when the *Waiting* option button is selected, or disabling the target fields when the *Confirmed* or *Terminated* option button is selected.

8.5 Back in the *Rules : Options : Waiting* Window, click **Validate**.

8.6 Back in the *Modify : Enumerated Attribute: State* Window, click **Validate**.

9 Set the target class for the *reservation_site* attribute: *Site*.

10 Set the groups and marks for the attributes in the *reservation* class:

Table 2.16: The groups and marks for the *Reservation* class

TYPE	IDENTIFIER	NAME	GROUP	MARKS
Numeric	reservation_number	Reservation #	Reservation	id, name, main, setConsult, hidden, providerValue
Enumerated	reservation_state	State		set, sort, filter, find, notNull, status
Time	reservation_date	Date Reserved		consult
Relation	reservation_client	Client		name, create, main, load
Time	reservation_check_in	Check-in	Dates	create, set, main
Time	reservation_check_out	Check-out		create, set, main
Numeric	reservation_days	Nights		create, set, readOnly
Relation	reservation_site	Hotel	Rooms	create, set, main
Relation	reservation_rooms	Rooms		create, set
Numeric	reservation_adults	Adults	People	create, set, table
Numeric	reservation_children	Children		create, set, table
Numeric	reservation_day_amount *	Daily Rate	Price	filter, optional, local, consult, complexTable
Numeric	reservation_amount	Total Amount		optional, local, consult, main

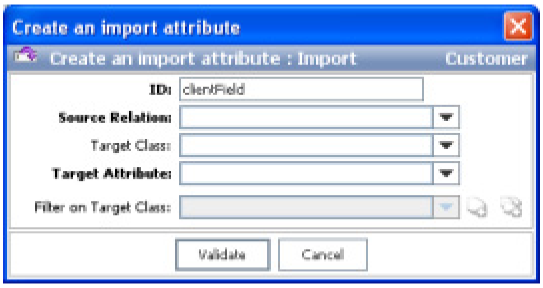
NOTE The *setConsult* mark flags the fields whose values are displayed although non editable in creation forms. Applying this mark only makes sense when the particular field value is computed or imported.

NOTE The *notNull* mark flags the fields that require a value for the form to be validated.

11 The *reservation_day_amount* attribute is an imported attribute. To create this attribute:

11.1 Select **Create an import attribute...** in the main tool bar.

The *Create an import attribute* window is displayed:



ID: clientField

Fig 2.35 *Creating an import attribute*

11.2 Specify the following attributes:

  - **Identifier**: reservation_day_amount
  - **Source Relation**: Rooms
  - **Target Class**: Room
  - **Target Attribute**: Price

11.3 Once the window validated, open its edit window to change its name, then apply the same configuration as the other amounts (**Type** to Double, **Minimum** to 0, **Increment step** to 10 and **Number of decimals** to 2, as well as a **Sum** function).

11.4 Set its marks: *filter*, *optional*, *local*, *consult*, *complexTable*.

12 The *reservation_amount* attribute contains a formula, to be specified in the **Formula** field, as follows: reservation_day_amount * reservation_days. Also, apply the same configuration as the other amounts (**Type** to Double, **Minimum** to 0, **Increment step** to 10 and **Number of decimals** to 2, as well as a **Sum** function).

The field value will be equal to the daily amount of the room's reservation multiplied by the number of nights for the reservation, which is the definition of the total amount of the reservation.

13 The *reservation_day_amount* and *reservation_amount* amounts specify prices which we already defined as a unit. Therefore, we can associate *Euros* as unit for these two fields.

13.1 Right-click the reservation_day_amount attribute then select **Set Number Unit...**

The *Set Number Unit* window is displayed.

13.2 Select the Unit list radio button.

13.3 Select Currency from the **Unit list** drop-down list.

13.4 Select Euros from the **Unit** drop-down list.

13.5 Click **Validate**.

13.6 Proceed in the same way for the *reservation_amount* attribute.

14 The *Reservation* class has a default sort according to the check-in dates.

15 We will remove the print action and create another one that will generate an invoice for the reservations.

15.1 Select the *Reservation* class in the class hierarchy.

15.2 Display the *Actions* tab.

15.3 Delete the *_print* action.

15.4 Click **Create an action...** in the *Creation* tool bar.

The *Create an action* window is displayed.

15.5 Specify the following fields:

- **Action type**: Generic (LEONARDI)
- **Template**: Print {_printVelocityModel}
- **Identifier**: print_model_reservation
- **Name**: Print the invoice

15.6 Click **Next** to display the action parameter configuration window and specify the following parameters:

- **Model file**: reservation_model.rtf **
- **Output file**: reservation.rtf **

** These files can be found in the *print* directory of the ready-to-use *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\print* folder in your Application Composer setup directory). Copy this folder to your *Hotel* application tree structure.

15.7 Select the **Print synchronous with GUI** check box.

15.8 Click **Validate**.

16 The *change_reservation_state* action allows to modify the state of a reservation with one click (pending => confirmed => finished). The procedure to create this action is similar to the one used for the previous action. Specify the following parameters:

- **Template**: _activate
- **ID**: change_reservation_state
- **Name**: Change the state
- **Image**: change_state.gif
- **Target attribute**: State

17 The *reservation_days* field contained in the *Reservation* class is a read-only field. In fact, its value is calculated based on the values specified for the *reservation_check_in* and *reservation_check_out* attributes. This calculation will be performed via a Java class defining the behavior of the *Reservation* application class. To generate this class:

17.1 Right-click the *Reservation* class in the class hierarchy then select **Class behavior...**

The *Class behavior* window is displayed.

17.2 Keep the default values and validate.

A *hotel.behavior.ReservationClassBehavior* class is then generated in the application directory. This class is visible in Application Composer in the list of the Java classes of the reservation application class. It must be implemented with the example code provided in the *hotel.behavior* package of the *Hotel* application shipped with Application Composer, and compiled in order to be used with the *Compile* action.

The *Compile* action may need certain parameters to be set in the *Preferences* window, such as the JDK folder, the compiler or the text editor that should be used.

2.2.8 *current_reservation*

The *current_reservation* class describes the reservations that are not yet completed (*pending* or *confirmed* status). This class is based on the reservation class with the difference that a filter will be added at the physical layer level at a later step.

TO PERFORM THIS STEP

1. In the *Create class* window, specify the following fields:
    - **Identifier**: current_reservation
    - **Model**: Reservation
    - **Name**: Current Reservation
    - **Image**: reservation
2. Click **Validate**.
3. We will create a sort for the *Check_in* field in this class as we did for the *reservation* class

2.3 Creating additional attributes

We will now proceed to the creation of a series of attributes that we left aside in previous sections.

In this section:

2.3.1 *site_rooms*

TO PERFORM THIS STEP

1. The *site_rooms* attribute is a multiple relational attribute of composition type targeting the *Room* class. This feature must be specified in the creation window of the relational attribute by selecting the **Composition** option of the **Type** field.

NOTE *Composition* is just one of the four available types for a relation attribute. In our example, the *Composition* type can be justified by the fact that a room belongs to just one hotel, and removing an hotel results in automatically removing the corresponding rooms.

2 Set its other properties:

Table 2.17: The properties for the *site_rooms* attribute

TYPE	IDENTIFIER	NAME	TAB	GROUP	ID	NAME	LOCAL
Relation	site_rooms	Rooms	Rooms	Overview	No	No	No

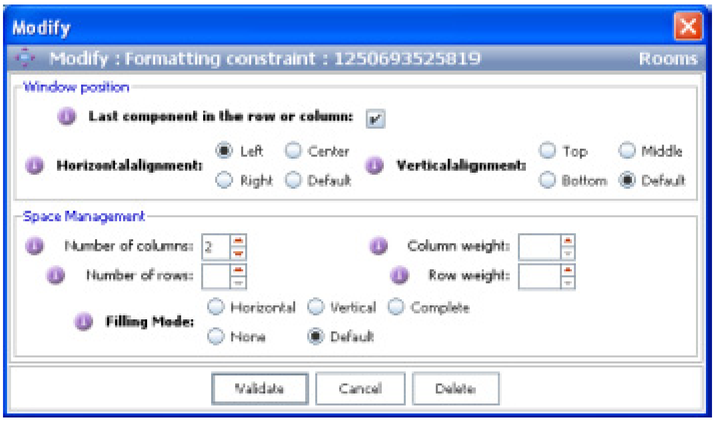
3 Set its marks: create, set, optional, private, load.

4 Add the *SITE_COMPLEX_LIST* to the attribute.

5 Once the attribute has been created, we will add a positioning constraint to it so that the field positions in the different forms can be adjusted.

5.1 Right-click this attribute and select **Edit field display constraint** ▸ **Label display constraints...**

The *Modify : Formatting constraint* window is displayed:



Modify

Modify : Formatting constraint : 1250693525819 Rooms

Window position

Last component in the row or column: ☑

Horizontalalignment: ● Left ○ Center ○ Right ○ Default

Verticalalignment: ○ Top ○ Middle ○ Bottom ● Default

Space Management

Number of columns: 2

Number of rows:

Column weight:

Row weight:

Filling Mode: ○ Horizontal ○ Vertical ○ Complete ○ None ● Default

Fig 2.36 *Modifying a formatting constraint for a label*

5.2 Select the **Last component in the row or column** check box.

NOTE Selecting the *Last component in the row or column* option will result in forcing the component to be displayed below its label. If the option is not selected the component is displayed next to the label.

5.3 Set the **Horizontal alignment** field to Left.

5.4 Set the **Number of columns** field to 2.

5.5 Click **Validate**.

2.3.2 *manager_sites*

The *manager* class contains a relational attribute (*manager_sites*) that we left aside in a previous section.

We will now deal with the particularities of this local attribute. The *manager_sites* attribute contains, for a given manager, the list of the hotels that they manage. This list is calculated automatically based on the values of the *site_manager* attribute of the *Site* class.

TO PERFORM THIS STEP

1 First create a multiple and local relation attribute targeting the *Site* class.

2 Set its other properties:

Table 2.18: The properties for the *manager_sites* attribute

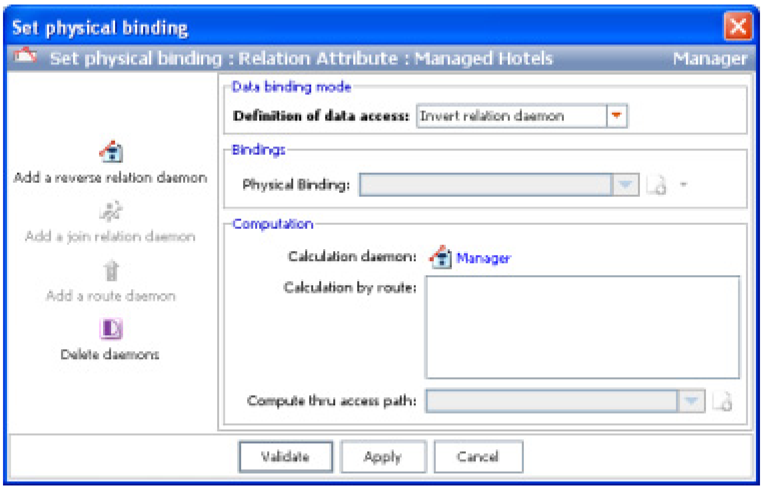
TYPE	IDENTIFIER	NAME	GROUP	ID	NAME	LOCAL
Relation	manager_sites	Managed Hotels	Data	No	No	Yes

3 Set its marks: complexTable.

4 We will now specify the physical binding of the *manager_sites* attribute:

4.1 Right-click the *manager_sites* attribute then select **Set physical binding...**

The *Set physical binding* field window is displayed:



Set physical binding

Set physical binding : Relation Attribute : Managed Hotels   Manager

Data binding mode

Definition of data access: Invert relation daemon

Bindings

Physical Binding:

Computation

Calculation daemon: Manager

Calculation by route:

Compute thru access path:

Fig 2.37 *Setting physical binding, inverted daemon*

4.2 In the **Definition of data access** drop-down list, select Invert relation daemon.

NOTE Resorting to a reverse daemon is essential for the *manager_sites* relation attribute as a manager may be in charge of multiple sites. Earlier in this tutorial when configuring the *city_region* relation attribute there was no need for a daemon as a city belongs to just one region.

4.3 Click the **Add a reverse relation daemon** button to create a new relation daemon.

The *New : Inverted Daemon* window is displayed.

4.4 In the **Source Relation** drop-down list, select the only available source relation: Manager.

4.5 Click **Validate**.

4.6 Back in the *Set physical binding* window, click **Validate**.

2.3.3 *client_reservations and room_site*

These attributes are created in the same way as for the *manager_sites* attribute.

For the *client_reservations* attribute, set its other properties:

Table 2.19: The properties for the *client_reservations* attribute

TYPE	IDENTIFIER	NAME	TAB	ID	NAME	LOCAL
Relation	client_reservations	Reservations	History	No	No	Yes

For the *room_site* attribute, set its other properties:

Table 2.20: The properties for the *room_site* attribute

TYPE	IDENTIFIER	NAME	GROUP	ID	NAME	LOCAL
Relation	room_site	Hotel	Description	No	Yes	Yes

2.3.4 *manager_clients and client_managers*

The *manager_clients* and *client_managers* attributes contain, for a given manager, the list of the clients he/she personally manages and, for a given client, the list of his/her personal managers.

Both lists are symmetrical and are contained in the database in the same table ensuring the join between these two application classes.

In order to create these attributes, we will proceed as we did for the *manager_sites* attribute, but this time use a join daemon.

NOTE Resorting to a join daemon is essential for the *manager_clients* and *client_managers* relation attributes as in the database a join table is used to build the list of the clients and their corresponding managers. A client may be linked to multiple managers, and vice versa.

TO PERFORM THIS STEP

1 Create two multiple local attributes targeting each other's class.

For the *manager_clients* attribute, properties are as follows:

Table 2.21: The properties for the *manager_clients* attribute

TYPE	IDENTIFIER	NAME	GROUP	ID	NAME	LOCAL
Relation	manager_clients	Personal Customers	Data	No	No	Yes

For the *client_managers* attribute, properties are as follows:

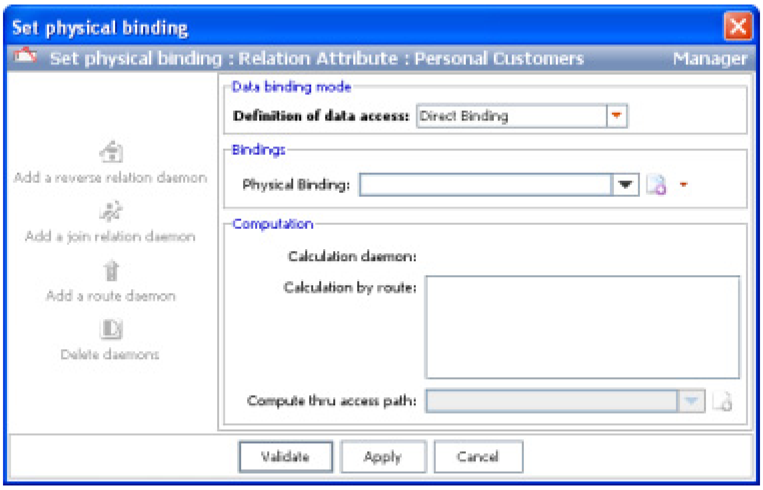
Table 2.22: The properties for the *client_managers* attribute

TYPE	IDENTIFIER	NAME	TAB	ID	NAME	LOCAL
Relation	client_managers	Staff Managers	History	No	No	Yes

2 We will now specify the physical binding of the *manager_clients* attribute:

2.1 Right-click the *manager_clients* attribute then select **Set physical binding...**

The *Set physical binding* window is displayed:



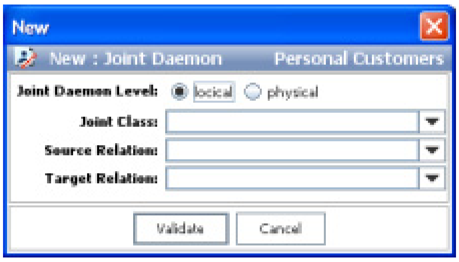
Definition of data access: Direct Binding

Fig 2.38 *Setting physical binding, joint daemon (1/6)*

2.2 In the **Definition of data access** drop-down list, select Joint daemon.

2.3 Click the **Add a join relation daemon** button.

The *New : Joint Daemon* window is displayed:

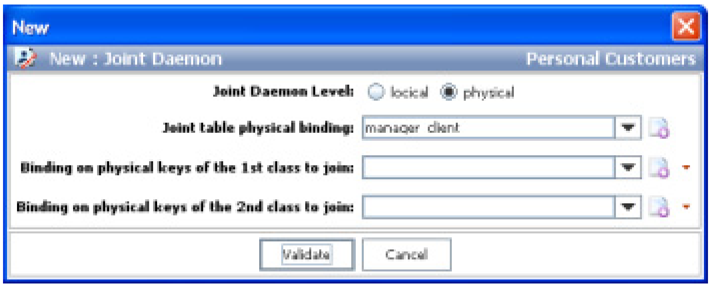


Joint Daemon Level: ● logical ○ physical

Fig 2.39 *Setting physical binding, joint daemon (2/6)*

2.4 Select the Physical radio button to create a new joint daemon at a physical level.

The *New : Joint Daemon* window changes to the following:


New
New : Joint Daemon
Personal Customers
Binding on physical keys of the 1st class to join:
Binding on physical keys of the 2nd class to join:
Validate
Cancel

Joint Daemon Level: ○ logical ● physical

Joint table physical binding: manager client

Fig 2.40 *Setting physical binding, joint daemon (3/6)*

2.5 Click **New** next to **Joint table physical binding**.

The *New : Simple binding* window is displayed:

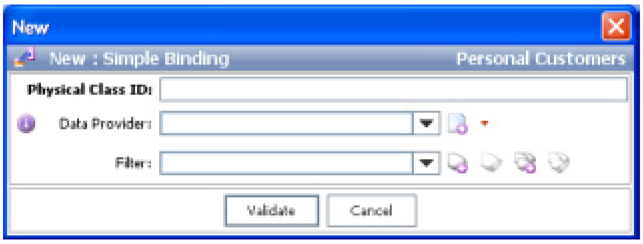
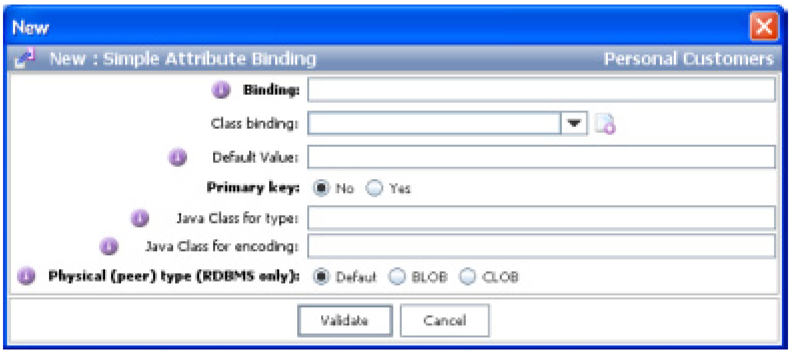


Fig 2.41 *Setting physical binding, joint daemon (4/6)*

2.6 In the **Physical class ID** field, type the name of the join table in the database then click **Validate**.

2.7 Back to the *New : Joint Daemon* window, click **New** next to the **Binding physical keys of the 1st class to join** field, then **New : Simple Attribute Binding** from the popup menu to create the simple links for the classes to join.

The *New : Simple Attribute Binding* window is displayed.



Binding:

Class binding:

Default Value:

Primary key: ● No ○ Yes

Java Class for type:

Java Class for encoding:

Physical (peer) type (RDBMS only): ● Default ○ BLOB ○ CLOB

Fig 2.42 *Setting physical binding, joint daemon (5/6)*

2.8 In the **Binding** field, specify the joint table foreign key column to the current column, i.e. the column where manager ids are stored.

2.9 In the **Class binding** field, select the join table.

2.10 Click **Validate**.

2.11 Back to the *New : Joint Daemon* window, click the **New** icon next to the **Binding physical keys of the 2nd class to join** field, then **New : Simple Attribute Binding** from the popup menu to create the simple links for the classes to join.

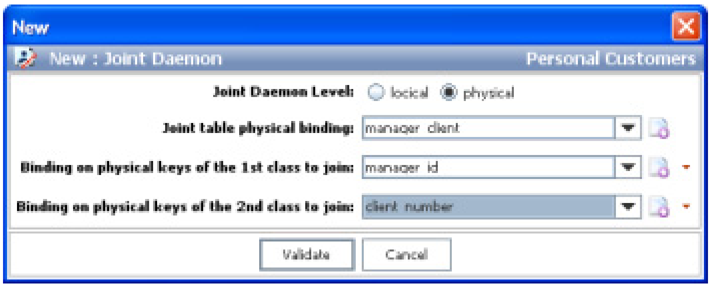
The *New : Simple Attribute Binding* window is displayed.

2.12 In the **Binding** field, specify the joint table column which is the foreign key to the table of the logical class that is referenced by the binding, i.e. the column where client ids are stored.

2.13 In the **Class binding** field, select the join table.

2.14 Click **Validate**.

The *New : Joint Daemon* window is displayed again:



Joint Daemon Level: ○ logical ● physical

Joint table physical binding: manager_client

Binding on physical keys of the 1st class to join: manager_id

Binding on physical keys of the 2nd class to join: client_number

Fig 2.43 *Setting physical binding, joint daemon (6/6)*

2.15 Click **Validate**.

2.16 Back to the *Set physical binding* window, click **Validate**.

3 Proceed in a similar way for the joint daemon of the *client_managers* attribute, but with one difference: The *manager_id* and *client_number* will need to be swapped.

2.3.5 *site_rooms_number*

TO PERFORM THIS STEP

1 The *site_rooms_number* attribute is an imported attribute from the *room_type* attribute. This attribute can now be created in the same way as the *reservation_amount* attribute was created. In the *Create an import attribute* window, set the following fields:

- **ID**: site_rooms_number
- **Source Relation**: Rooms
- **Target Class**: Room
- **Target Attribute**: Type

2 Set its other properties:

Table 2.23: The properties for the *site_rooms_number* attribute

TYPE	IDENTIFIER	NAME	TAB	GROUP	ID	NAME	LOCAL
Numeric	site_rooms_number	Number of Rooms	Rooms	Overview	No	No	**Yes**

3 Set its marks: optional, main, consult.

2.3.6 *client_amount*

TO PERFORM THIS STEP

1 The *client_amount* attribute is an imported field from the *Reservation* relation, targeting the *reservation_amount* attribute. Once created, set its unit along with the other amount properties:

- **Type**: Double
- **Minimum**: 0
- **Increment step**: 10
- **Number of decimals**: 2
- **Function**: Sum

2.4 Creating an additional action

We will now create a specific action for the *Site* and the *Room* classes: The *reserve_room* action.

TO PERFORM THIS STEP

1 This action must be firstly defined for the *Site* class by clicking **Create an action...**

1.1 In the *Create an action* window, set the following fields:

- **Action type**: Generic (Application Engine)
- **ID**: reserve_room
- **Name**: Book a room
- **Short Name**: Book

1.2 Click **Next**.

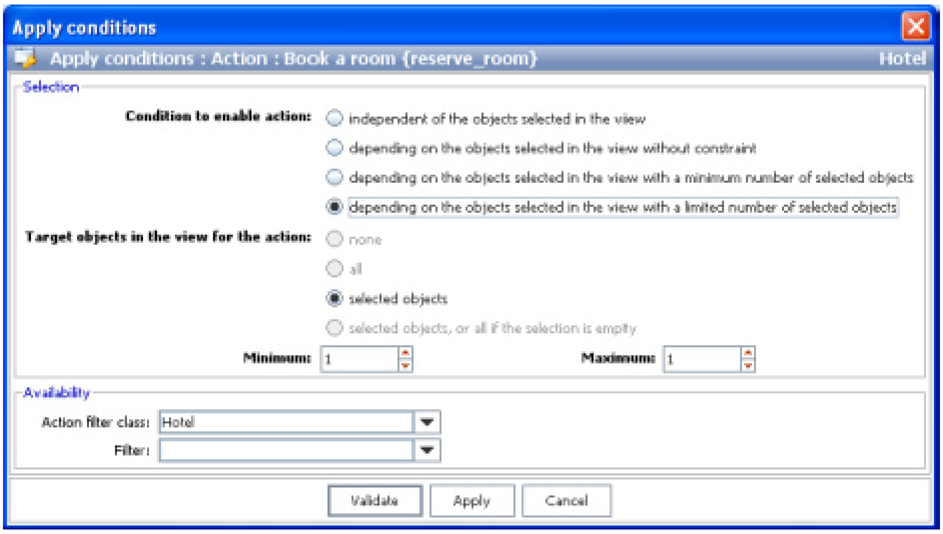
1.3 Set the **Image** field to reserve.gif.

1.4 Click **Validate**.

1.5 Once the action has been created, you need to set its marks and its apply conditions in order to specify that the action can only be initiated for an object, and for one object only.

1.5.1 Right-click the newly created action in the *Actions* tab then select **Apply conditions...**

The *Apply conditions* window is displayed:



Condition to enable action:
○ independent of the objects selected in the view
○ depending on the objects selected in the view without constraint
○ depending on the objects selected in the view with a minimum number of selected objects
● depending on the objects selected in the view with a limited number of selected objects

Target objects in the view for the action:
○ none
○ all
● selected objects
○ selected objects, or all if the selection is empty

Minimum: 1 **Maximum:** 1

Action filter class: Hotel
Filter:

Fig 2.44 *Specifying apply conditions for an action*

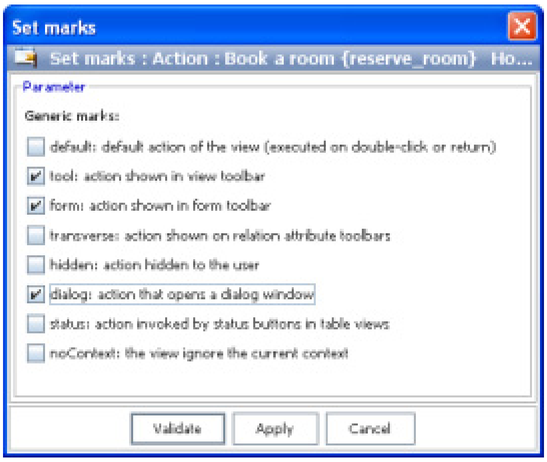
1.5.2 Select the Depending on [...] with a limited number of selected objects option.

1.5.3 Click **Validate**.

1.6 Setting the action marks will allow you to specify that the action will be available in the main tool bar and in the window tool bar, and also that it will display a dialog box.

1.6.1 Right-click the newly created action in the *Actions* tab then select **Set marks...**

The *Set marks : Action* window is displayed:



Generic marks:

☐ default: default action of the view (executed on double-click or return)
☑ tool: action shown in view toolbar
☑ form: action shown in form toolbar
☐ transverse: action shown on relation attribute toolbars
☐ hidden: action hidden to the user
☑ dialog: action that opens a dialog window
☐ status: action invoked by status buttons in table views
☐ noContext: the view ignore the current context

Fig 2.45 *Setting marks for an action*

1.7 Select the tool, form, and dialog options.

1.8 Click **Validate**.

2 Now that the action has been defined at the *Site* class level, we will use it as a model for another action - *reserve_rooms* - for the *Room* class.

2.1 In the *Create an action* window, set the following fields:

- **Action type**: Other
- **Template**: Book a room (reserve_room)
- **ID**: reserve_rooms
- **Name**: Book Multiple Rooms

2.2 We will now change the maximum number of selected objects required to start the action: In the Apply conditions window, set the **Maximum** field to 5.

3 Because the *reserve_room* action is not based on any generic Application Engine action, you will need to define its behavior via a Java class. To generate this class:

3.1 Right-click the root node of the class hierarchy then select **Session behavior...**

The *Session behavior* window is displayed.

3.2 Keep the default values and click **Validate**.

This generates the *HotelSessionBehavior.java* file to the *behavior* folder of the in the application directory.

Implement the code provided with the ready-to-use *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\behavior* folder in your Application Composer setup directory.

3.3 We will now compile the behavior:

3.3.1 Select **Windows** ▸ **Java classes**.

The *Java classes* window is displayed.

The *hotel.behavior.HotelSessionBehavior* class is prefixed by a red bullet, meaning that it hasn't been compiled yet.

3.3.2 Select the file and click ✓ **Compile**.

The red bullet changes to a green bullet.

2.5

Creating the routes

In this section we will create the routes connecting:

- cities (*City* class) to the *Hotel*, *Room* and *Reservations* classes
- regions (*Region* class) to the *Site*, *Room* and *Reservation* classes

NOTE Why create routes?
You create a route when you need to display data originating from two or more tables that do not have direct relations between them.

Creating the routes for the *city* class

We will first focus on the route connecting a city to a hotel.

Later on in this tutorial we will create a compound view comprising the list of hotels and a geographical map. Selecting a city (*city* class) will display the corresponding hotels (*site* class) on the map.

However there is no direct relation going from *city* to *site* and the only attribute that *indirectly* links both classes is the *site_address* structure attribute in the *site* class. This attribute comprises a relation attribute targeting the *city* class (*address_city*).

Therefore the link between both classes can be represented as follows:
site > *site_address* > *address_city* > *city*

The route we will build will go from *city* to *site*. It will comprise two steps: one from *city* to *address_city*, and the other one from *site_address* to *site*. As this route will be *reverse* relative to the actual link between both classes, we will need to make it *bidirectional*.

TO PERFORM THIS STEP

1 Select the *City* class in the class hierarchy then click **New** in the *Routes* tab:

NOTE If the *Routes* tab is not displayed, select **Windows** ▸ **Data Model** ▸ **Routes**.

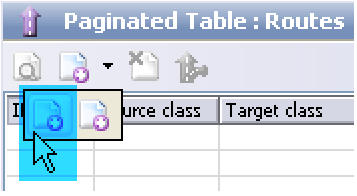
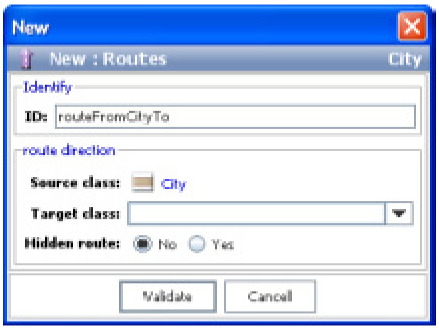


Fig 2.46 *Creating a route (1/6)*

The *New : Routes* window is displayed:



ID: routeFromCityTo

Source class: City

Target class:

Hidden route: ● No ○ Yes

Fig 2.47 *Creating a route (2/6)*

2 Set the following fields:
  - **ID**: routeFromCityToSite
  - **Target class**: Site

3 Click **Validate**.

The newly created route is displayed in the list of the routes of the *City* class. Selecting this route displays its graphical representation in the lower part of the window:

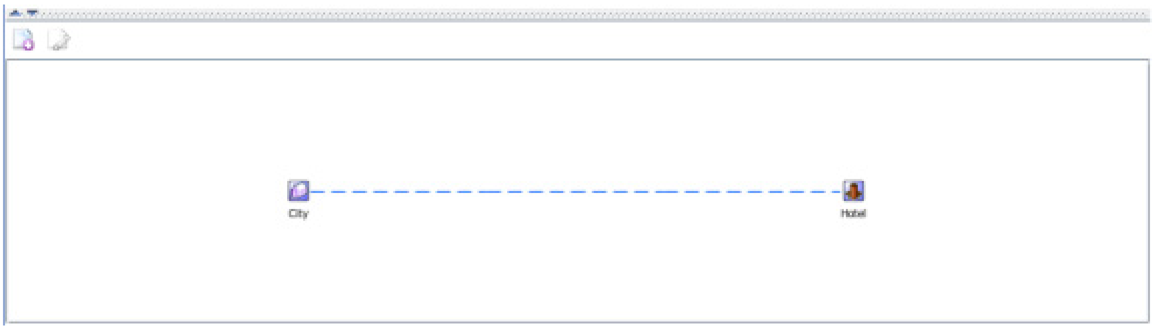


Fig 2.48 *Creating a route (3/6)*

4 Create the step going from *city* to *address_city*.

4.1 Click **Add step**.

The *Add step* window is displayed.

4.2 Set the following fields:
  - **Direction**: From target class

- **Target class**: Site address

4.3 Click **Validate**.

The graphical representation is updated to reflect the changes:

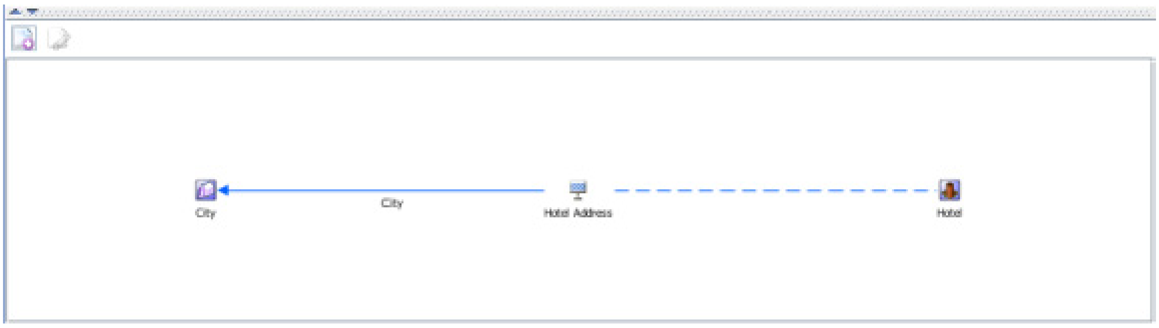


Fig 2.49 *Creating a route (4/6)*

5 Create the step going from *site_address* to *site*.

5.1 Click **Add step**.

The *Add step* window is displayed.

5.2 Set the following fields:

- **Direction**: From target class
- **Target class**: Site

5.3 Click **Validate**.

The graphical representation is updated to reflect the changes:

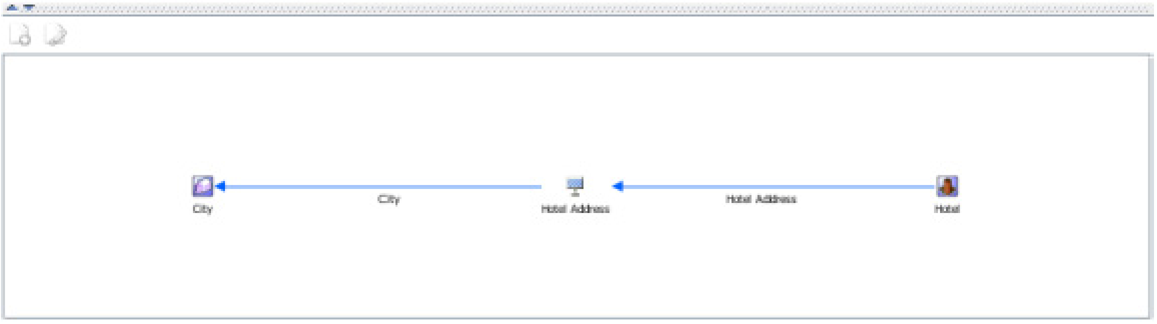
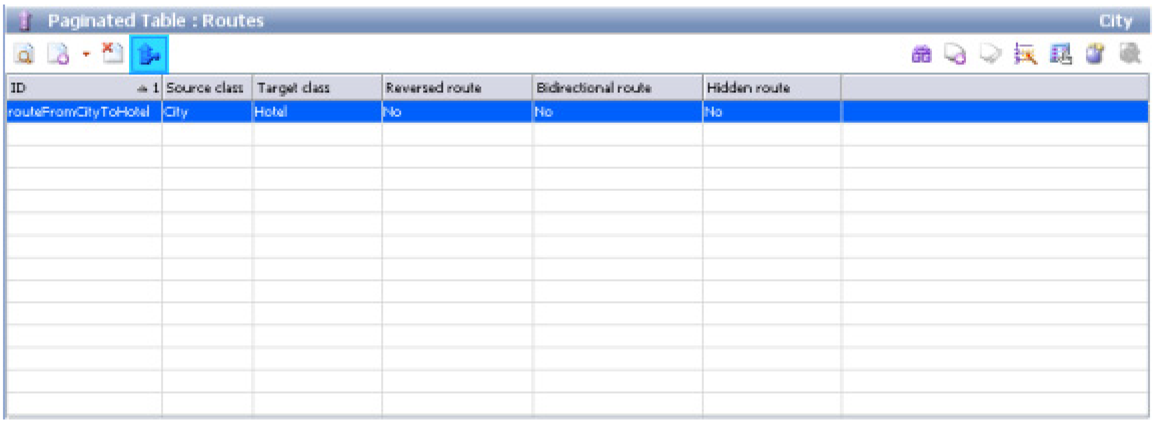


Fig 2.50 *Creating a route (5/6)*

6 Make sure the route is selected and click **Make the root bidirectional** in the tool bar:



ID	Source class	Target class	Reversed route	Bidirectional route	Hidden route
routeFromCityToHotel	City	Hotel	No	No	No

Fig 2.51 *Creating a route (6/6)*

7 Repeat these steps to create routes between the *City* class and the *Room*, *Reservation* and *Current_reservation* classes.

The routes to be created can be represented as follows:

- City > Hotel Address > Site > Room
- City > Hotel Address > Site > Room > Reservation
- City > Hotel Address > Site > Room > Current Reservation

Creating the routes for the *Region* class

We will now create the routes connecting the *Region* class the *Site*, *Room* and *Reservation* classes.

The routes to be created can be represented as follows:

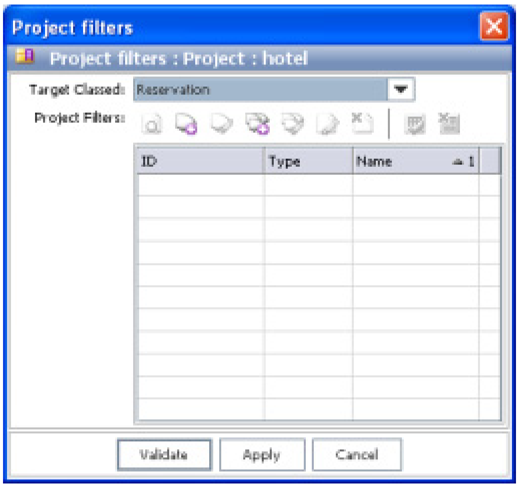
- Region > City > Hotel Address > Site
- Region > City > Hotel Address > Site > Room
- Region > City > Hotel Address > Site > Room > Reservation
- Region > City > Hotel Address > Site > Room > Current Reservation

2.6 Declaring the application filters

TO PERFORM THIS STEP

1 Right-click the *Hotel* project in the class hierarchy, then select **Project filters...**

The *Project filters : Project : Hotel* window is displayed:



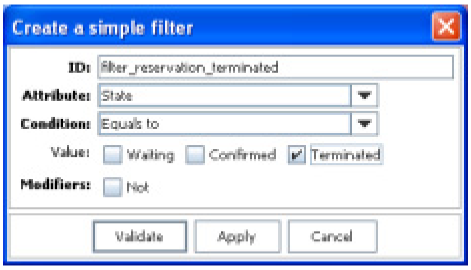
Target Classes: Reservation

Fig 2.52 *Creating a simple filter (1/2)*

2 Select Reservation from the **Target class** drop-down list.

3 Click **Create a simple filter...**

The *Create a simple filter* window is displayed:



ID: filter_reservation_terminated

Attribute: State

Condition: Equals to

Value: ☐ Waiting ☐ Confirmed ☑ Terminated

Modifiers: ☐ Not

Fig 2.53 *Creating a simple filter (1/2)*

4 Set the following fields:

- **ID**: filter_reservation_terminated
- **Attribute**: State
- **Condition**: Equal to
- **Value**: Terminated

5 Click **Validate**.

6 Back in the *Project filters : Project : Hotel* window, click **Validate**.

Chapter 3

Building the navigation tree

Now that the application classes have been defined, we will focus on the application's navigation tree, i.e. we will create the available views and specify how they follow on from one another.

In this section:

3.1 Login action (login_manager)

We will first create a root action that will be a login action for our hotel room reservation application.

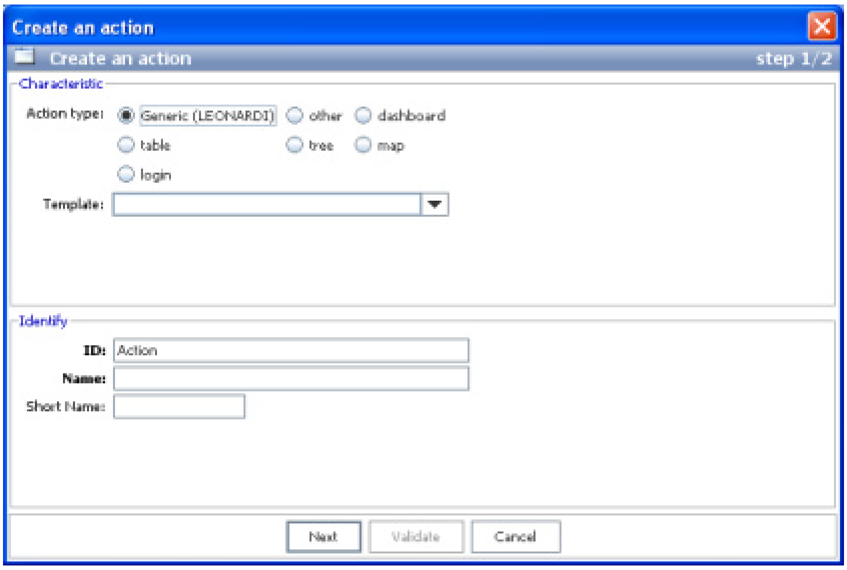
TO PERFORM THIS STEP

1 Select the *Navigation tree* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Navigation tree**.

2 Click **Create an action...** in the *Creation* tool bar.

The first screen of the *Create an action* window is displayed:



Action type: ● Generic (LEONARDI) ○ other ○ dashboard

ID: Action

Fig 3.1 *Creating the root action (1/2)*

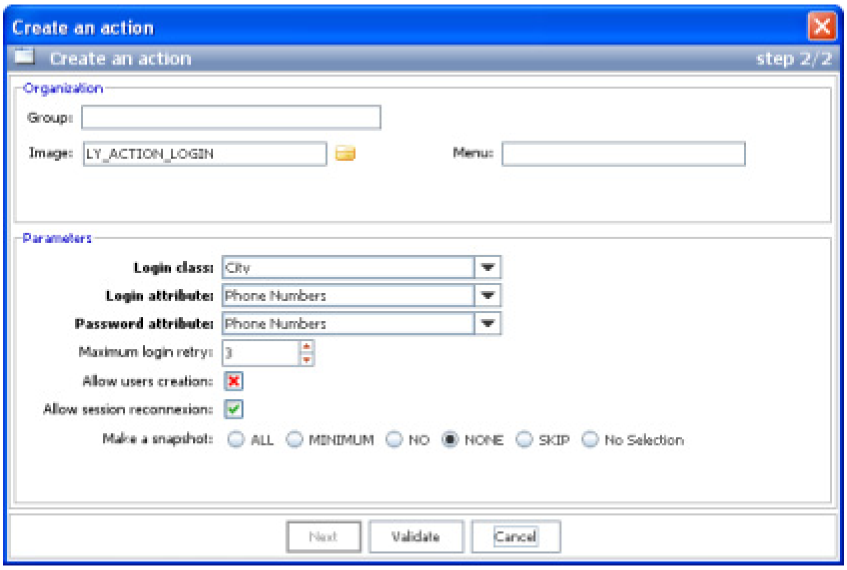
3 Choose login as the action type.

4 Set the following fields:

- **ID**: login_manager
- **Name**: Open Session

5 Click **Next**.

The second screen of the *Create an action* window is displayed:



Group:

Image: LY_ACTION_LOGIN

Menu:

Login class: City

Login attribute: Phone Numbers

Password attribute: Phone Numbers

Maximum login retry: 3

Allow users creation: ☒

Allow session reconnexion: ☑

Make a snapshot: ○ ALL ○ MINIMUM ○ NO ● NONE ○ SKIP ○ No Selection

Fig 3.2 *Creating the root action (2/2)*

6 Set the following fields:

- **Login class**: Manager
- **Login attribute**: Manager ID
- **Password attribute**: Password

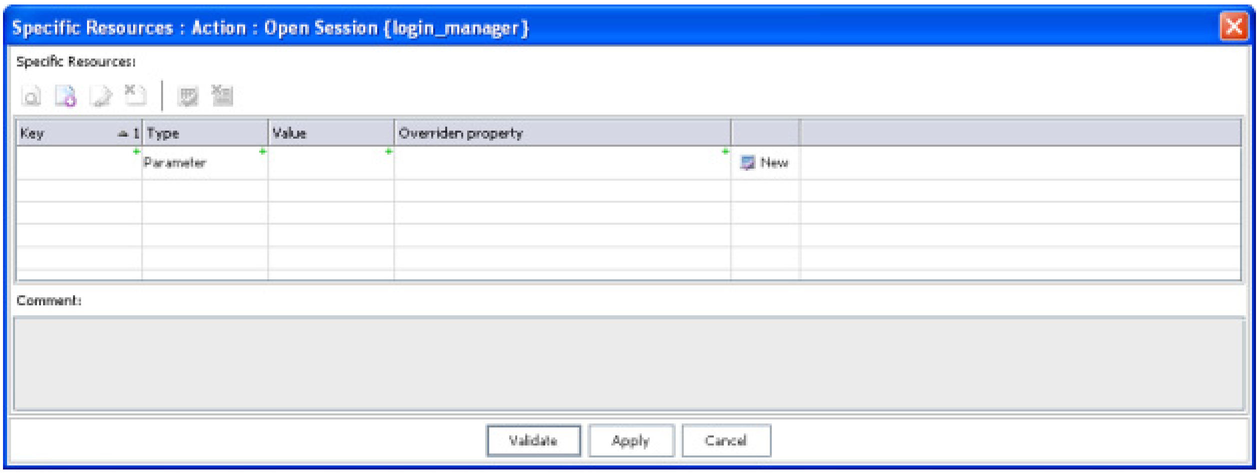
7 Click **Validate**.

Once the action has been created, it is displayed as the root node in the application's navigation tree.

8 We will change two parameters to customize this action.

8.1 Right-click the action and select **Specific Resources...**

The *Specific Resources : Action : Open Session* window is displayed:

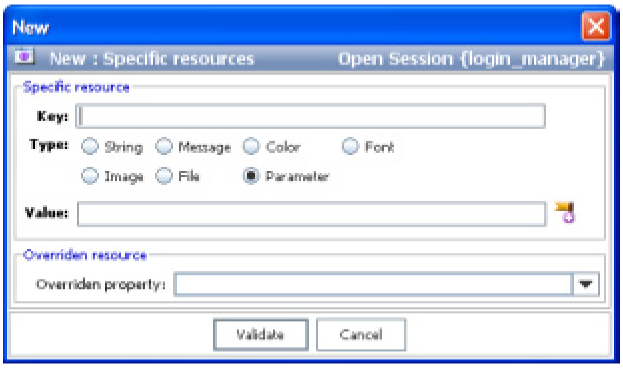


Key	Type	Value	Overriden property	
	Parameter			New

Fig 3.3 *Customizing an action (1/2)*

8.2 Click **New...** in the tool bar.

The *New : Specific resources* window is displayed:



Type: ● Parameter

Fig 3.4 *Customizing an action (2/2)*

8.3 Set the following fields:

- **Key**: LY_DEFAULT_BACKGROUND
- **Type**: Color
- **Overridden property**: LY_DEFAULT_BACKGOUND

8.4 We will now specify the RGB code for the background color.

8.4.1 Click next to the **Value** field.

The window for specifying the RGB code is displayed.

8.4.2 Specify the RGB code: 225 226 240

8.4.3 Click **Validate**.

8.5 Back in the *New : Specific resources* window, click **Validate**.

8.6 Back in the *Specific Resources : Action : Open Session* window, create another specific resource:

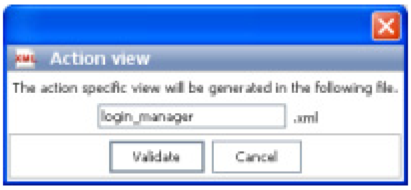
- **Key**: LY_LABEL_NORMAL_STATE
- **Type:** Color
- **Overridden property**: LY_LABEL_NORMAL_STATE
- **Value**: White

8.7 Back to the *Specific Resources : Action : Open Session* window, click **Validate**.

9 We will define a specific view for this action.

9.1 Right-click the *login_manager* action then select **Action view...**

The **Action view** window is displayed:



The action specific view will be generated in the following file.

login_manager .xml

Fig 3.5 *Specifying a specific view for an action*

9.2 Specify login.xml as the name of the file to be generated.

9.3 Click **Validate**.

This generates the *login.xml* file to the *view* folder of the in the application directory.

Implement the code provided with the *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\view* folder in your Application Composer setup directory.

The *appli.gif* image is required for the view. This file is available in the *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\images* folder in your Application Composer setup directory.

10 The *appli.gif* image has a blue background. However by default the login window background has a different color. Two specific color resources need to be added to this action: *LY_AREA_BACKGROUND_IMAGE* and *LY_COMMANDS_BACKGROUND_IMAGE* which represent the same *blue_background.gif* image. They are available from the same location as *appli.gif*.

11 The text color needs to be lightened via the *LY_LABEL_NORMAL_STATE* color property.

NOTE If you generate your application at that point you will notice that no login form will be displayed. This is because this form authenticates the connecting users against the database table that is bound to the *manager* class, and so far this table contains no entries.

3.2 Dashboard action (hotel_dashboard)

We will now create the main action for our application: The dashboard that will allow you to display all the views.

TO PERFORM THIS STEP

1 Be sure to select the root action (*login_manager*) in the class hierarchy and then click **Create an action...** in the *Creation* tool bar.

The *Create an action* window is displayed.

2 Set the following fields:

- **Action type**: dashboard
- **ID**: hotel_dashboard
- **Name**: Hotel Reservation
- **Image**: appli_small.gif

3 Configure the action via the following specific resources:

- LY_WRITE_TOOL_LABEL, parameter, true
- LY_TITLEBAR_BACKGROUND, color, blue_background.gif
- LY_DEFAULT_IMAGE_WIDTH, parameter, 25
- LY_DEFAULT_IMAGE_HEIGHT, parameter, 25

Now that the dashboard has been created, we will define all the actions it will contain. The actions are listed in the table below:

Table 3.1: The dashboard actions

IDENTIFIER	NAME	TYPE
manager_list	Manager List	Complex table
client_list_consult	Customer List	Compound action
reservation_list_planning	Reservation Planning	Compound action
reservation_treemap	Reservation Distribution	Tree map
establishment_tree_map_list	Geographical localization	Compound action
terminated_reservation_filter_list	Reservation History	Compound action
reservation_establishment_chart	Hotel Sales	Statistics
reservation_month_chart	Reservations per month	Statistics
user_manual	User Tutorial	URL

3.3 Manager list (manager_list)

Manager List is a complex table action.

TO PERFORM THIS STEP

1 Be sure to select the *hotel_dashboard* action in the navigation tree and then click **Create an action...** in the *Creation* tool bar.

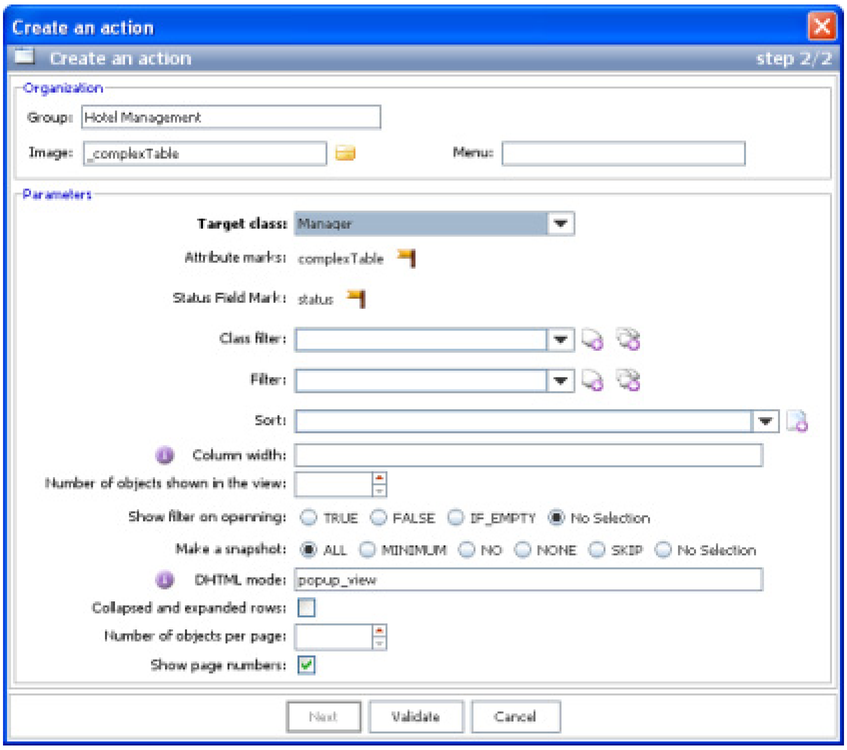
The first screen of the *Create an action* window is displayed.

2 Set the following fields:
- **Action type**: Generic (Application Engine)
- **Template**: Complex Table
- **ID**: manager_list
- **Name**: Manager List

3 Click **Next**.

The second screen of the *Create an action* window is displayed.

4 Set the following fields:
- **Group**: Hotel Management
- **Target class**: Manager



Group: Hotel Management

Image: _complexTable

Menu:

Target class: Manager

Attribute marks: complexTable

Status Field Mark: status

Class filter:

Filter:

Sort:

Column width:

Number of objects shown in the view:

Show filter on openning: ○ TRUE ○ FALSE ○ IF_EMPTY ● No Selection

Make a snapshot: ● ALL ○ MINIMUM ○ NO ○ NONE ○ SKIP ○ No Selection

DHTML mode: popup_view

Collapsed and expanded rows: ☐

Number of objects per page:

Show page numbers: ☑

Fig 3.6 *Creating a complex action*

5 Click **Validate**.

NOTE This action needs to be represented by a complex table as in the list of managers we will need to display multiple values within the same raw because a manager may be responsible for multiple hotels.

3.4 Customer list (client_list_consult)

Customer List is a compound action. The purpose of a compound action is to display two views one besides the other, either horizontally or vertically. Typically the second view displays the details of the currently selected item in the first view.

The first action will display a list of the clients where the fields will be editable.

The second action will display a read-only form with the details of the selected client.

TO PERFORM THIS STEP

1 Make sure the *Hotel Reservation (hotel_dashboard)* action is selected in the navigation tree then click **Create a compound action...** in the *Creation* tool bar.

The *Create a compound action* window is displayed.

2 Set the following fields:

- **ID**: client_list_consult
- **Name**: Customer List
- **Group**: Reservation Management

3 We will now create the first sub-action:

3.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

3.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Editable Table
- **ID**: client_list
- **Name**: Customer List

3.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

3.4 Set the following fields:

- **Target class**: Client
- Click **Validate**.

4 Back in the *Create a compound* action window, create the second sub-action:

- **Action type**: Generic (Application Engine)
- **Template**: Details (_consult)
- **ID**: client_consult
- **Name**: Edit
- **Target class**: Client

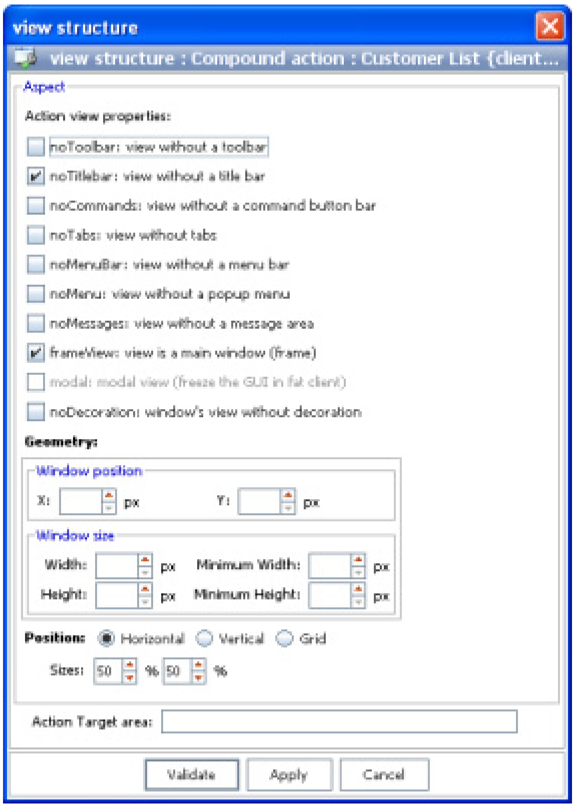
5 Back in the *Create a compound* action window, click **Validate**.

The newly created action is displayed in the navigation tree.

6 We will now set up the structure of the compound action.

6.1 Right-click the *client_list_consult* action then select **View structure...**

The *View structure* window is displayed:



Action view properties:

☐ noToolbar: view without a toolbar
☑ noTitlebar: view without a title bar
☐ noCommands: view without a command button bar
☐ noTabs: view without tabs
☐ noMenuBar: view without a menu bar
☐ noMenu: view without a popup menu
☐ noMessages: view without a message area
☑ frameView: view is a main window (frame)
☐ modal: modal view (freeze the GUI in fat client)
☐ noDecoration: window's view without decoration

Geometry:

Window position
X: px Y: px

Window size
Width: px Minimum Width: px
Height: px Minimum Height: px

Position: ● Horizontal ○ Vertical ○ Grid

Sizes: 50 % 50 %

Action Target area:

Fig 3.7 *Creating a compound action*

6.2 Set the **Sizes** field to 55 % and 45 %.

6.3 Select the Horizontal radio button.

6.4 Click **Validate**.

7 Set up the structure of the *client_consult* action: In the *View structure* window, select the *noToolbar* and *noTitlebar* options.

NOTE By selecting the *noToolbar* option, we will prevent customer information from being edited in the view.

3.5 Geographical Hierarchy (establishment_tree_map_list)

Geographical Hierarchy is a compound action itself comprising a compound action - *Geographical Hierarchy (establishment_tree_map)* - and a complex table - *Hotel List (establishment_list)*.

establishment_tree_map is made of a tree action - *establishment_tree* - and a map - *establishment_map*.

establishment_list is a complex table made up of two actions.

NOTE If, when an action has been created, it happens to be incorrectly placed in the navigation tree, just drag and drop it to the appropriate location.

TO PERFORM THIS STEP

1 Create the *establishment_tree_map_list* action:

1.1 Click **Create a compound action...**

The *Create a compound action* window is displayed.

1.2 Set the following fields:

- **ID**: establishment_tree_map_list
- **Name**: Geographical Hierarchy
- **Group**: Hotel Management
- **Image**: map

2 Create the sub compound action: *establishment_tree_map*.

2.1 Click **Create a compound action...** in the *Compound* area.

The *Create a compound action* window is displayed.

2.2 Set the following fields:

- **ID**: establishment_tree_map
- **Name**: Geographical Hierarchy

2.3 Create the first sub-action in the *establishment_tree_map* sub-action: *establishment_tree*.

2.3.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

2.3.2 Set the following fields:

- **Action type**: Tree
- **ID**: establishment_tree
- **Name**: Hotels

2.3.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

2.3.4 Set the following field:

- **Target class**: Region

2.3.5 Specify the first target sub class: click **New...** and in the window that is displayed, set the following fields:

- **Hierarchy**: 1
- **Target class**: City

2.3.6 Specify the second target class:

- **Hierarchy**: 2

- **Target class**: Site

2.3.7 Click **Validate**.

2.4 Create the second sub-action in the *establishment_tree_map* sub-action: *establishment_map*.

2.4.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

2.4.2 Set the following fields:

- **Action type**: Map
- **ID**: establishment_map
- **Name**: Geographical Localization

2.4.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

2.4.4 Set the following field:

- **Target class**: Site

2.4.5 Click **Validate**.

2.5 Back in the *Create a compound action* window, click **Validate**.

2.6 Back in the *Create a compound action* window, create the sub complex table action: *establishment _list*.

2.6.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

2.6.2 Set the following fields:

- **Action type**: Generic (LEONARDI)
- **Template**: Complex Table (_complexTable)
- **ID**: establishment_list
- **Name**: Hotel List

2.6.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

2.6.4 Set the following fields:

- Image: establishment
- Target class: Site

2.6.5 Set the marks: Click ⚑ next to **Attribute marks** to display the *Set marks for the fields processed by the action* window. In the **Field specific marks** area, select the SITE_COMPLEX_LIST mark then click ➡. In the **Field generic marks** area, remove the *complexTable* mark. Click **Validate**.

2.6.6 Set the **Status field mark** field to Status.

2.6.7 Enable the **Collapsed and expanded rows** field.

2.6.8 Enable the **Show page numbers** field.

2.6.9 Click **Validate**.

3 Back in the *Create a compound action* window, click **Validate**.

4 We will now add two actions under the *Hotel List (establishment_list)* action.

4.1 Select the *Hotel List (establishment_list)* action in the navigation tree and click **Create an action...**

The first screen of the *Create an action* window is displayed.

4.2 Set the following fields:

- **Action type**: Table
- **ID**: establishment_list_province
- **Name**: Province

4.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

4.4 Set the following fields:

- Image: province
- Target class: Province

4.5 Click **Validate**.

4.6 Proceed similarly to create another action:

- **Action type**: Table
- **ID**: establishment_list_city
- **Name**: City
- Image: city
- Target class: City

5 We will now specify the behavior for certain actions.

5.1 Right-click the *establishment_list* action in the navigation tree and select **Action behavior** ▸ **Action behavior...**

The *Action behavior* window is displayed.

5.2 Keep the default values and click **Validate**.

This generates the *EstablishmentComplexListBehavior.java* file to the *behavior* folder of the in the application directory.

Implement the code provided with the ready-to-use *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\behavior* folder in your Application Composer setup directory.

5.3 Display the *Java classes* tab and compile the behavior.

5.4 Specify a behavior for the *establishment_map* action.

The file to be used for this behavior is *ProvinceMapBehavior.java*

The *establishment_category*.gif* and *france256.gif* images are required for the view. These files are available in the *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\images* folder in your Application Composer setup directory.

6 We will add some specific resources to the *establishment_tree* action.

6.1 Right-click the *establishment_tree* action in the navigation tree then select **Specific resources...**

The *Specific resources* window is displayed.

6.2 Create the following resources:

Table 3.2: The specific resources for the establishment_tree action

KEY	TYPE	VALUE	OVERRIDDEN PROPERTY
LY_DIR_CLOSED	Image	map.gif	LY_DIR_CLOSED
LY_DIR_OPEN	Image	map.gif	LY_DIR_OPEN
LY_TREE_ROOT	String	France	root

7 We will now set up the structure for the *establishment_tree_map_list* action:

7.1 Right-click the *establishment_tree_map_list* action in the navigation tree then select **View structure...**

The *View structure* window is displayed.

7.2 Set the **Sizes** field to 60 % and 40 %.

7.3 Select the Vertical radio button.

7.4 Click **Validate**.

8 We will now set up the structure for the *establishment_tree_map* action: horizontal, 30%, 70%

3.6 Reservation planning (reservation_list_planning)

reservation_list_planning is a compound action comprising a Gantt diagram (*reservation_planning*) and a paginated table (*reservation_list*).

TO PERFORM THIS STEP

1 Create the *reservation_list_planning* action:

1.1 Click **Create a compound action...**

The *Create a compound action* window is displayed.

1.2 Set the following fields:

- **ID**: reservation_list_planning
- **Name**: Reservation Planning
- **Short Name**: Planning
- **Group**: Reservation Management
- **Image**: reservation

2 Create the sub compound action: *establishment_tree_map*.

2.1 Click **Create a compound action...** in the *Compound* area.

The *Create a compound action* window is displayed.

2.2 Set the following fields:

- **ID**: establishment_tree_map
- **Name**: Geographical Hierarchy

2.3 Create the *reservation_planning* sub action:

2.3.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

2.3.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Gantt (_gantt)
- **ID**: reservation_planning
- **Name**: Reservation Planning
- **Short Name**: Planning

2.3.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

2.3.4 Set the following fields:

- **Image**: reservation
- **Target class**: Hotel (establishment)
- **Target sub class**: Room (room)
- **Begin date attribute**: Check-in (reservation_check_in)
- **End date attribute**: Check-out (reservation_check_out)
- **Update period (seconds)**: 60

2.3.5 Click **Validate**.

2.4 Create the second sub-action in the *establishment_list_planning* sub -action: *reservation_list*.

2.4.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

2.4.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Paginated Table (_multiPageTable)
- **ID**: reservation_list
- **Name**: Current Reservations

2.4.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

2.4.4 Set the following field:

- **Image**: reservation
- **Target class**: Current Reservation (current_reservation)

2.4.5 Click **Validate**.

2.5 Back in the *Create a compound action* window, click **Validate**.

2.6 We will now specify the behavior for the *reservation_planning* action.

2.6.1 Right-click the *reservation_planning* action in the navigation tree and select **Action behavior** ▸ **Action behavior...**

The *Action behavior* window is displayed.

2.6.2 Keep the default values and click **Validate**.

This generates the *ReservationGanttBehavior.java* file to the *behavior* folder of the in the application directory.

Implement the code provided with the ready-to-use *Hotel* application that is shipped with Application Composer (*Studio\examples\hotel\behavior* folder in your Application Composer setup directory.

2.6.3 Display the *Java classes* tab and compile the behavior.

3 We will now set up the structure for the *reservation_list_planning* action.

3.1 Right-click the *reservation_list_planning* action in the navigation tree then select **View structure...**

The *View structure* window is displayed.

3.2 Set the **Sizes** field to 55 % and 45 %.

3.3 Select the Vertical radio button.

3.4 Click **Validate**.

3.7 Reservation distribution (reservation_treemap)

reservation_treemap is based on Tree Map (*_treemap*) model.

TO PERFORM THIS STEP

1 Create the *reservation_treemap* action:

1.1 Click **Create an action...** in the navigation tree.

The first screen of the *Create an action* window is displayed.

1.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Tree Map (_treeMap)
- **ID**: reservation_treemap
- **Name**: Reservation Distribution
- **Short Name**: Distribution

1.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

1.4 Set the following field:

- **Group**: Reservation Management
- **Target class**: Current Reservation (current_reservation)
- **Use global configuration**: Enabled
- **Marks of fields used into tooltip**: TREEMAP_TOOLTIP

- **Marks of fields used to display small images**: status
- **Class hierarchy fields**: Enabled
- **Choose the hierarchy level**: Enabled
- **Choose the size field**: Enabled
- **Choose fieldinfo size**: Total Amount (current_reservation_amount)
- **Choose the color field**: Enabled
- **Color field**: Daily Amount (current_reservation_day_amount)
- **Choose aggregation type**: Enabled
- **Aggregation type**: BALANCED
- **Threshold colors**: Enabled
- **Border sizes**: 0
- **Select border size**: 2
- **Label style**: TOPLEFT
- **Make a snapshot**: NONE

1.5 Click **Validate**.

2 We will now add two color resources but also remove the status image for confirmed reservations In order to lighten the display.

We will also specify the font and color to be used for the various areas, as well as the border color.

The font and outline colors are specified with the *LY_MAP_FONT_COLOR* and *LY_MAP_OUTLINE_COLOR* resources, whose values are respectively R: 46, G: 54, B:77 and R: 92, G: 108, B: 154.

2.1 Right-click the *reservation_treemap* action in the navigation tree then select **Specific resources...**

The *Specific resources* window is displayed.

2.2 Create the following resource:

Table 3.3: The specific resources for the reservation_treemap action

KEY	TYPE	VALUE	
reservation_state_confirmed	Image	*one blank character*	
LY_TREEMAP_MAX_COLOR_CODE	String	CornFlowerBlue	Green
LY_TREEMAP_MIN_COLOR_CODE	String	IndianRed2	Red
LY_MAPNODE_FONT	Font	Arial-bold-11	LY_MAPNODE_FONT
LY_MAP_FONT_COLOR	Color	46 54 77	LY_MAP_FONT_COLOR
LY_MAP_OUTLINE_COLOR	Color	92 108 154	LY_MAP_OUTLINE_COLOR

3.8 Reservation history (terminated_reservation_filter_list)

TO PERFORM THIS STEP

1 Click **Create a compound action...** in the navigation tree.

The *Create a compound action* window is displayed.

2 Set the following fields:

- **ID**: terminated_reservation_filter_list
- **Name**: Reservation History
- **Group**: History
- **Image**: history

3 We will now create the first sub-action:

3.1 Click **Create an action...** in the *Compound* area.

The first screen of the *Create an action* window is displayed.

3.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: New Simple Filter (__filter_new)
- **ID**: terminated_reservation_filter
- **Name**: New Simple Filter

3.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

3.4 Set the following fields:

- **Target class**: Reservation (reservation)
- Click **Validate**.

4 Back in the *Create a compound* action window, create the second sub-action:

- **Action type**: Generic (Application Engine)

    - **Template**: Paginated Table (_multiPageTable)
    - **ID**: reservation_history
    - **Name**: Reservation History
    - **Target class**: the class filter we have just created, i.e: filter_reservation_terminated
    - **Class filter**: Reservation (reservation)

5 Back in the *Create a compound* action window, click **Validate**.

6 We will now set up the structure for the *terminated_reservation_filter_list* action.

6.1 Right-click the *terminated_reservation_filter_list* action in the navigation tree then select **View structure...**

The *View structure* window is displayed.

6.2 Set the **Sizes** field to -1 % and 45 %.

6.3 Select the Vertical radio button.

-1 indicates that the first sub-action must use its preferred size, and the second action will use the remaining space. If both actions have enough space, the second action will use 45% of the available space, as specified, and the first action will use the remaining 55%.

6.4 Click **Validate**.

3.9 Charts

The dashboard will also display two types of graphics:

- Hotel Sales (*reservation_establishment_chart*)
- Reservations per month (*reservation_month_chart*)

TO PERFORM THIS STEP

1 Create the *reservation_establishment_chart* action:

1.1 Click **Create an action...** in the navigation tree.

The first screen of the *Create an action* window is displayed.

1.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Predefined Statistic (_graphicalChart)
- **ID**: reservation_establishment_chart
- **Name**: Hotel Sales
- **Short Name**: Per Hotel

1.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

1.4 Set the following field:

- **Group**: History

- **Target class**: Current Reservation (current_reservation)
- **Parameters for the statistic**: Hotel Sales (chart_reservations_establishment)

1.5 Click **Validate**.

2 Create the *reservation_month_chart* action:

2.1 Click **Create an action...** in the navigation tree.

The first screen of the *Create an action* window is displayed.

2.2 Set the following fields:

- **Action type**: Generic (Application Engine)
- **Template**: Predefined Statistic (_graphicalChart)
- **ID**: reservation_month_chart
- **Name**: Reservations per month
- **Short Name**: Per Month

2.3 Click **Next**.

The second screen of the *Create an action* window is displayed.

2.4 Set the following field:

- **Group**: History
- **Target class**: Current Reservation (current_reservation)
- **Parameters for the statistic**: Reservations per month (chart_reservations_month)

2.5 Click **Validate**.

3.10 User tutorial

user_manual will display a user guide of our application in the web browser.

TO PERFORM THIS STEP

1 Click **Create an action...** in the navigation tree.

The *Create an action* window is displayed.

2 Set the following fields:

- **ID**: user_manual
- **Name**: User Tutorial
- **Group**: Help
- **Image**: user_manual

3 Click **Validate**.

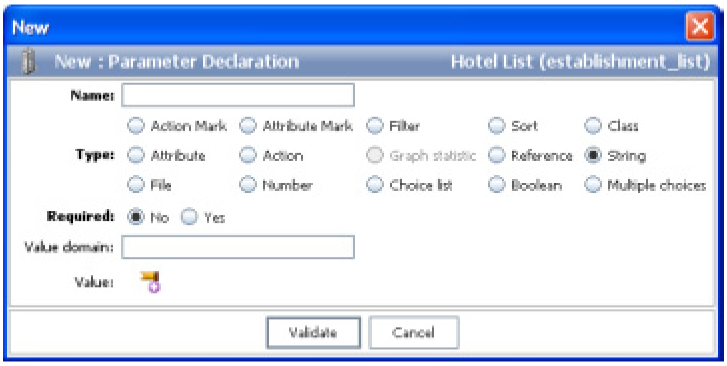
4 We will now create a specific parameter for the action:

4.1 Right-click the *user_manual* action in the navigation tree then select **Specific parameters...**

The Specific parameters window is displayed.

4.2 Click ➕ **New...** in the toolbar.

The *New : Parameter Declaration* window is displayed:



New

New : Parameter Declaration   Hotel List (establishment_list)

Name:

Type: ○ Action Mark ○ Attribute Mark ○ Filter ○ Sort ○ Class ○ Attribute ○ Action ○ Graph statistic ○ Reference ● String ○ File ○ Number ○ Choice list ○ Boolean ○ Multiple choices

Required: ● No ○ Yes

Value domain:

Value:

Fig 3.8 *Creating a specific parameter for an action*

4.3 Set the fields as follows:

- **Name**: _url
- **Type**: String
- **Required**: yes
- **Value**: HOTEL_HELP_FILE

4.4 Click **Validate**.

The *HOTEL_HELP_FILE* resource specifies the entry point html file.

NOTE We will associate an HTML file with this resource later on in this tutorial.

Chapter 4

Specifying the specific resources

In this section we will define specific resources in order to customize our hotel room reservation application.

Resources allow you to set application preferences (strings, messages, files, images, colors, fonts and parameters).

In this section:

4.1 Overloading parameters

We will now overload some generic application parameters.

TO PERFORM THIS STEP

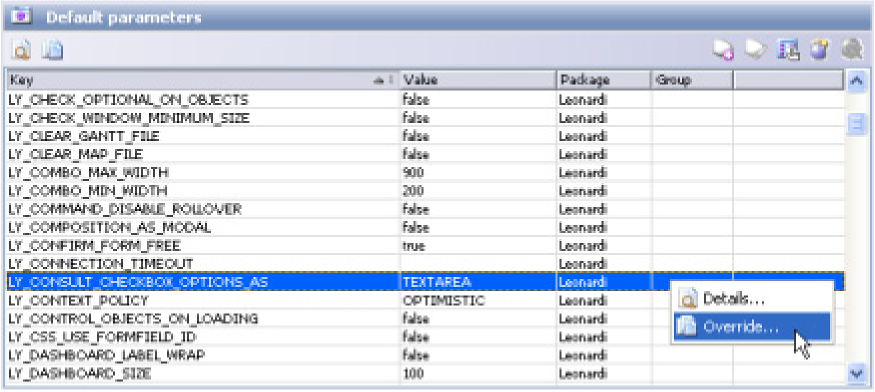
1 Select the *Parameters* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Resources** ▸ **Parameters**.

2 In the *Default parameters* area, select the generic resource *LY_CONSULT_CHECKBOX_OPTIONS_AS* and click **Override...** in the toolbar.

ALTERNATIVELY Right-click *LY_CONSULT_CHECKBOX_OPTIONS_AS* and select **Override...**

NOTE The *LY_CONSULT_CHECKBOX_OPTIONS_AS* resource is used to specify a graphical representation for the multiple check boxes in read-only forms.



Key	Value	Package	Group	
LY_CHECK_OPTIONAL_ON_OBJECTS	false	Leonardi		
LY_CHECK_WINDOW_MINIMUM_SIZE	false	Leonardi		
LY_CLEAR_GANTT_FILE	false	Leonardi		
LY_CLEAR_MAP_FILE	false	Leonardi		
LY_COMBO_MAX_WIDTH	900	Leonardi		
LY_COMBO_MIN_WIDTH	200	Leonardi		
LY_COMMAND_DISABLE_ROLLOVER	false	Leonardi		
LY_COMPOSITION_AS_MODAL	false	Leonardi		
LY_CONFIRM_FORM_FREE	true	Leonardi		
LY_CONNECTION_TIMEOUT		Leonardi		
LY_CONSULT_CHECKBOX_OPTIONS_AS	TEXTAREA	Leonardi		
LY_CONTEXT_POLICY	OPTIMISTIC	Leonardi		
LY_CONTROL_OBJECTS_ON_LOADING	false	Leonardi		
LY_CSS_USE_FORMFIELD_ID	false	Leonardi		
LY_DASHBOARD_LABEL_WRAP	false	Leonardi		
LY_DASHBOARD_SIZE	100	Leonardi		

Fig 4.1 *Overloading the default parameters*

The *Override : Default parameters* window is displayed.

3 In the **Value** field, change the preexisting value (*TEXTAREA*) for STRING.

4 Click **Validate**.

The parameter is now displayed in the *Parameters* area.

5 Proceed in a similar way for four other attributes:

- *LY_DASHBOARD_SIZE*: 200
- *LY_DEFAULT_IMAGE_HEIGHT*: 20
- *LY_FREE_ON_CLOSE*: true
- *LY_NO_DRIVER_MANAGER*: true

6 Click **Validate**.

4.2 Overloading colors

We will now overload a generic color.

TO PERFORM THIS STEP

1 Select the *Colors* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Resources** ▸ **Colors**.

2 In the *Default colors* area, select the generic resource *LY_MAP_BACKGROUND* and click **Override...** in the toolbar.

ALTERNATIVELY Right-click *LY_MAP_BACKGROUND* and select **Override...**

The *Override : Default colors* window is displayed.

3 Set the **Value** field to: 17 111 183.

4 Click **Validate**.

LY_MAP_BACKGROUND is now also displayed in the *Colors* area, which list the specific colors.

4.3 Creating new image resources

We will now create three specific images to represent all three possible reservation states (terminated, waiting, confirmed).

TO PERFORM THIS STEP

1 Select the *Images* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Resources** ▸ **Images**.

2 Click **New...** in the toolbar of the *Images* area.

The *New : Images* window is displayed.

3 Set the following fields:

- **Key**: reservation_state_terminated
- **Value**: green2.gif shipped with Application Composer (*Studio\examples\hotel\images* folder in your Application Composer setup directory)

4 Proceed in the same way to create the remaining images:

- *reservation_state_waiting*, `red2.gif`
- *reservation_state_confirmed*, `yellow2.gif`

4.4 Specifying the files required for the application

Our application requires a number of external files, such as a file to be associated with the *HOTEL_HELP_FILE* resource that we previously defined for the *user_manual* action.

This file is shipped with Application Composer (*Studio\examples\hotel\html\help_en.html* folder in your Application Composer setup directory.

Copy the *html* folder to your application tree structure.

TO PERFORM THIS STEP

1. Select the *Files* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Resources** ▸ **Files**.

2. Click **New...** in the toolbar of the *Files* area.

   The *New : Files* window is displayed.
3. Set the following fields:
   - **Key**: HOTEL_HELP_FILE
   - **Language**: English
   - **Value**: The path to the *help_en.html* file
4. Click **Validate**.
5. Proceed in the same way for another file - *help.html* - which contains the documentation in French.

We will also need to overload the following files:

- *LY_LOGIN_PAGE* is the entry point for the login action
- *LY_EXIT_PAGE* is the page that is displayed when a user logs out
- *LY_UP_PAGE* is the upper side of the application, which contains the application title and its icon

These are JSP files used in web mode. They are also contained in the *html* folder (*login_page_hotel.jsp*, *exit_hotel.jsp* and *up_hotel.jsp*).

exit_hotel.jsp requires the *background.gif* file for the title to be visible in the upper side of the application.

4.5 Internationalizing the application

So far, every text message we have created has been in English. However they can be made available easily in other languages, and you can also easily translate any existing applications.

TO PERFORM THIS STEP

1 Select the *Messages* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Resources** ▸ **Messages**.

2 In the *Language* area, select the target language.

3 Click **New...** in the toolbar of the *Messages* area to create a message or **Override...** in the toolbar of the *Default messages* area to overload a message.

4.6 Specifying resources for the login_manager action

We previously created a specific view via the *login.xml* file. Numerous resources are used in this file. Some of them, such as *LY_LOGIN_USER* or *LY_LOGIN_PASSWORD*, do not need to be modified, and others do.

We will now override the *HOTEL_LOGIN_INFO*, *HOTEL_LOGIN_USER_VALUE* and *HOTEL_LOGIN_PASSWORD_VALUE* string resources.

HOTEL_LOGIN_INFO provides some help on how to connect to the application.

HOTEL_LOGIN_USER_VALUE and *HOTEL_LOGIN_PASSWORD_VALUE* are the login and password of the test account.

TO PERFORM THIS STEP

1 Select the *Strings* tab.

NOTE If this tab is not displayed, select **Windows** ▸ **Resources** ▸ **Strings**.

2 In the *Language* area, select English.

3 We will now create the *HOTEL_LOGIN_INFO* resource:

  3.1 In the *Strings* area, click **New...**

The *New : Strings* window is displayed.

3.2 Set the following fields:

- **Key** - HOTEL_LOGIN_INFO
- **English** - Demo application. Use the following account info to connect:

3.3 Click **Validate**.

4 Proceed in a similar way to create the *HOTEL_LOGIN_USER_VALUE* and *HOTEL_LOGIN_PASSWORD_VALUE* resources.

Set the **English** field to *adm* for both of them.

Connecting the application to a data source

We will now connect our application to flat files. These are provided with the ready-to-use *Hotel* application that is shipped with Application Composer.

To perform the steps in this section, copy the content of the *data* folder in the ready-to-use *Hotel* application to the *data* folder in your application.

In this section:

5.1 Configuring the data source

We will now set up a *file location* to connect the application to flat files.

TO PERFORM THIS STEP

1 Select the root package in the class hierarchy.

2 Select **File** ▸ **Data sources...**

The *Data sources* window is displayed:

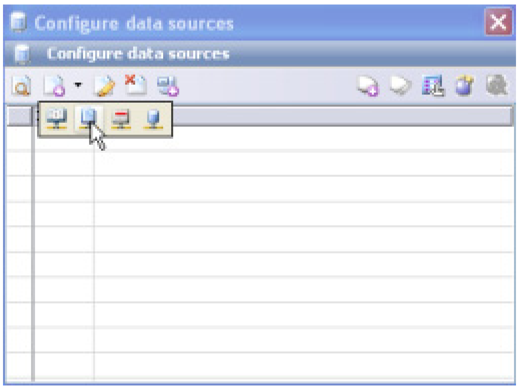
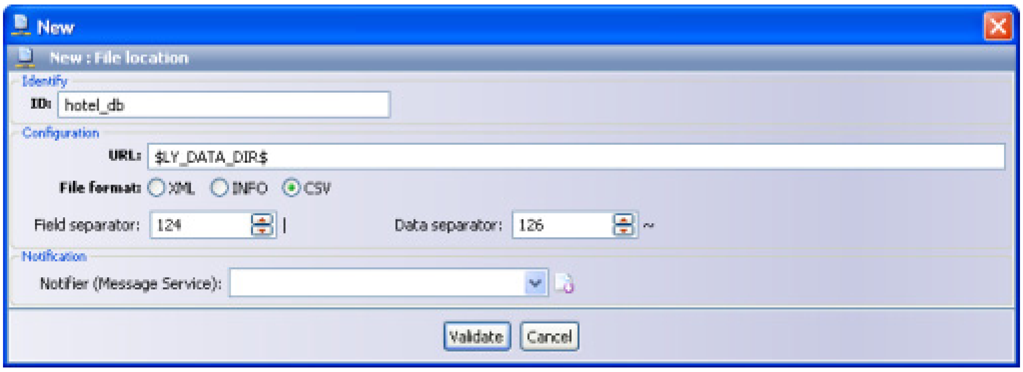


Fig 5.1 *Configuring the data source (1/2)*

3 Select the **New : File location** icon on the tool bar.

The *New : File location* window is displayed.



ID: hotel_db

URL: LY_DATA_DIR

File format: ○ XML ○ INFO ● CSV

Field separator: 124 |

Data separator: 126 ~

Notifier (Message Service):

Fig 5.2 *Configuring the data source (2/2)*

4 Set the following files:

- **ID** - *hotel_db*
- **URL** - LY_DATA_DIR

NOTE LY_DATA_DIR is a predefined variable that points to the *data* folder of the current application.

- **File format** - *CSV*

5 Click **Validate**.

The flat file connection is now displayed in the list of the data sources.

5.2 Setting the physical binding for the classes

Now that the data source has been configured, we will now modify the link to the physical layer of all the application classes.

Prior to performing this step, you will need to initialize the LY_CREATE_FILES, LY_SAVE_FILES, and LY_USE_FILES variables in the application's *env.res* file.

TO INITIALIZE THE VARIABLES

1 Click **Properties** in the tool bar.

The *Modify : Application* window is displayed.

2 Select the *Environment* tab.

3 Set the LY_CREATE_FILES, LY_SAVE_FILES, and LY_USE_FILES variables to *true*.

NOTE The LY_CREATE_FILES variable specifies whether or not a file should be created for data persistence.
The LY_SAVE_FILES variable specifies whether or not the data should be saved to the specified file.
The LY_USE_FILES variable specifies whether or not the file specified for data persistence should be used.

4 Click **Validate**.

TO MODIFY THE LINKS OF THE CLASSES

1 Click **Properties** in the tool bar.

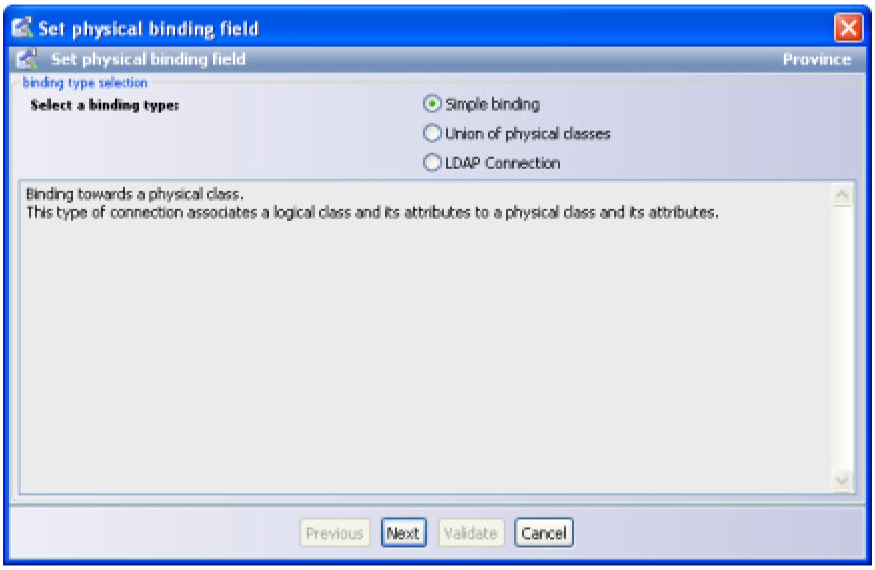
The *Modify : Application* window is displayed.

2 Make sure the **Source** field is set to Real Data.

NOTE *Real data* means the configured data source will be used to save data.

3 Right-click the *Province* class in the class hierarchy and select **Set physical bindings...**

The first screen of the *Set physical bindings* window is displayed:



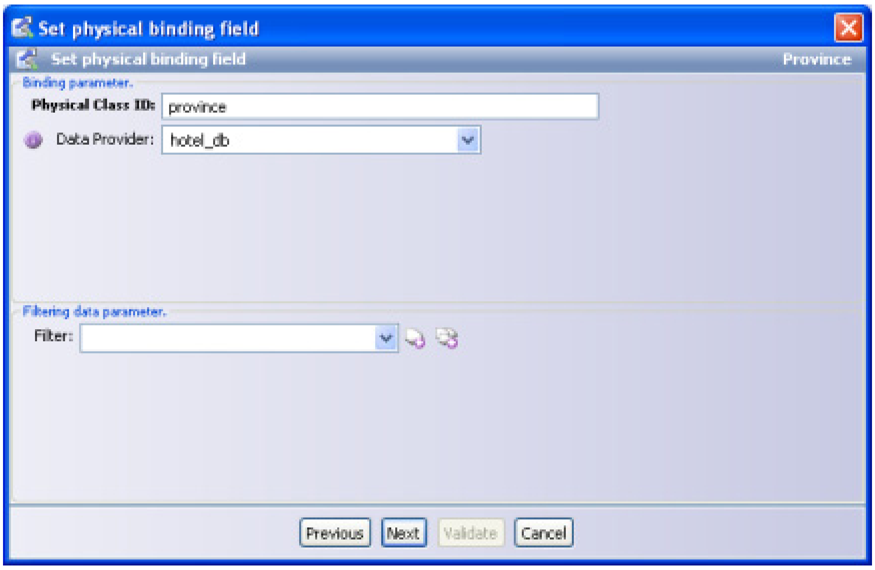
Select a binding type:
● Simple binding
○ Union of physical classes
○ LDAP Connection

Binding towards a physical class.
This type of connection associates a logical class and its attributes to a physical class and its attributes.

Fig 5.3 *Configuring the physical layer of the class (1/4)*

4 Keep the default option (*simple binding*) then click **Next**.

The second screen of the *Set physical bindings* window is displayed:


Set physical binding field
Set physical binding field
Province
Binding parameter.
Filtering data parameter.
Previous
Next
Validate
Cancel

Physical Class ID: province
Data Provider: hotel_db
Filter:

Fig 5.4 *Configuring the physical layer of the class (2/4)*

5 Set the following fields:

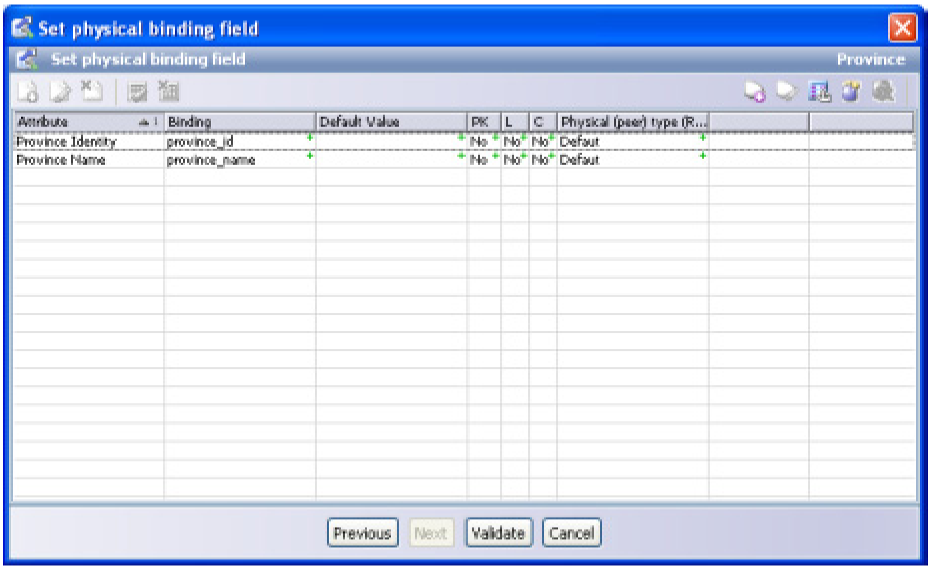
- **Physical class ID** - province

NOTE The value of the **Physical Class ID** field must match the name of the table in the database.

- **Data provider** - hotel_db

6 Click **Next**.

The third screen of the *Set physical bindings* window is displayed:



Attribute	Binding	Default Value	PK	L	C	Physical (peer) type (R...		
Province Identity	province_id		No	No	No	Defaut		
Province Name	province_name		No	No	No	Defaut		

Fig 5.5 *Configuring the physical layer of the class (3/4)*

Click **Validate**.

7 Proceed in the same way for the other application classes, except for the *Current Reservation* class.

8 For the *Current Reservation* class, we will establish a link to the database using a filter.

8.1 Right-click the *Current Reservation* class in the class hierarchy and select **Set physical bindings...**

The first screen of the *Set physical bindings* window is displayed.

8.2 Keep the default option (*simple binding*) then click **Next**.

The second screen of the *Set physical bindings* window is displayed.

8.3 Click **Create a simple filter** to the right of the **Filter** field.

The *Create a simple filter* window.

8.4 Set the following fields:

- **ID** - filter_current_reservation
- **Attribute** - State
- **Condition** - Equal to
- **Value** - Waiting, Confirmed

8.5 Click **Validate**.

8.6 Click **Next**.

The third screen of the *Set physical bindings* window is displayed.

8.7 Click **Validate**.

5.3 Setting the physical binding for the Address attribute

The address attribute will not be bound to any specific class but to the corresponding values in the database.

TO PERFORM THIS STEP

1. Select the Hotel project in the class hierarchy.
2. Select the *Attributes* tab.
3. Right-click the address attribute then select **Set physical binding field** ▸ **Set Structure Class Binding...**

   The *Set Structure Class Binding* window is displayed.
4. Click **New...** to the right of the **Physical Binding** field.

   The *New Simple Binding* window is displayed:
5. Set the following fields:
   - **Physical Class ID**: address
   - **Data Provider**: hotel_db
6. Click **Validate** to close the *New : Simple Binding* window.
7. Back in the *Set Structure Class Binding* window, click **Validate**.

Illustrations

Index

Application Composer

TUTORIAL

Reference: APPCOMPOSER_TUTORIAL_043_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