BUSINESS FIRST Application Composer

Application Composer

QUICK START TUTORIAL

Reference: APPCOMPOSER_QUICKSTART_041_EN

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

Application Composer

QUICK START TUTORIAL

Reference: APPCOMPOSER_QUICKSTART_041_EN

Contents

Overview

How to easily design a GUI, in rich, light, or fat client, and still not be familiar with the underlying technologies? The answer is: By focusing on business and working with application concepts. This approach is that of *Application Suite*, published by W4.

In this software suite, Application Composer and Application Engine provide a *Model-Driven* environment for rationalizing the development of complex, scalable applications in an agile way.

W4 website:

- www.w4.eu

In this section:

This tutorial is made up of the following steps:

Quick Start Tutorial

In this tutorial you will design your first application with Application Composer rapidly - in less than one hour - and with no previous technical skills.

This tutorial will walk you through successive iterations to gradually reach an elaborate application that will address all the features that can be expected from a personal video library management application.

Application Suite

Application Composer is part of a comprehensive software package, *BUSINESS FIRST Application Composer*, an application composition software suite.

BUSINESS FIRST Application Composer is segregated into two subgroups: Application Suite and *Process Suite*.

Within Application Suite, Application Composer, which is the cornerstone of this tutorial, is the graphic design workshop that takes care of modeling and GUI generation, and Application Engine is in charge of running the resulting application.

Within Process Suite, Process Composer and Process Engine provide a comprehensive BPM environment for designing and running business processes.

How it works

With Application Composer, the data model is at the heart of the developments and should be implemented in the first place. By just configuring the model, the designer specifies which views are displayed to the final user and how navigation is organized across the various screens.

All the technical constraints such as data presentation (generating the views, getting and displaying the data, preserving the consistency of the various application windows, etc) are taken care of by the framework itself, and especially by Application Engine, which analyzes the model to dynamically generate the application screens whenever they are required.

At this point the resulting application can already be executed, either in fat client, or in web mode. Afterwards it will be iteratively fine-tuned by adding specific functional behaviors as well as graphic resources (colors, images, fonts, etc) to obtain a comprehensive application available in a variety of modes: DHTML/AJAX (web), Swing, SWT, Eclipse plug-in, etc.

Step 1: Describing the data model

To compose the application in this tutorial, we will be using Application Composer. Application Composer is available either as a standalone desktop application, or as an Eclipse plugin. With Application Composer you can describe an application and this is a fully codeless process.

In this section:

NOTE To specify English as the language for the user interface, edit the `application_composer.ini` file in the `Studio` folder of the setup directory (by default: `C:\Program Files\w4\ApplicationComposer\Studio`) and uncomment the following key: LY_LANGUAGE=en

Creating the *Videolibrary* application

In this tutorial you will design a basic application for managing a personal videolibrary.

When Application Composer starts, the main window is displayed. To the right-hand side is the *Class Hierarchy*, which is empty so far.

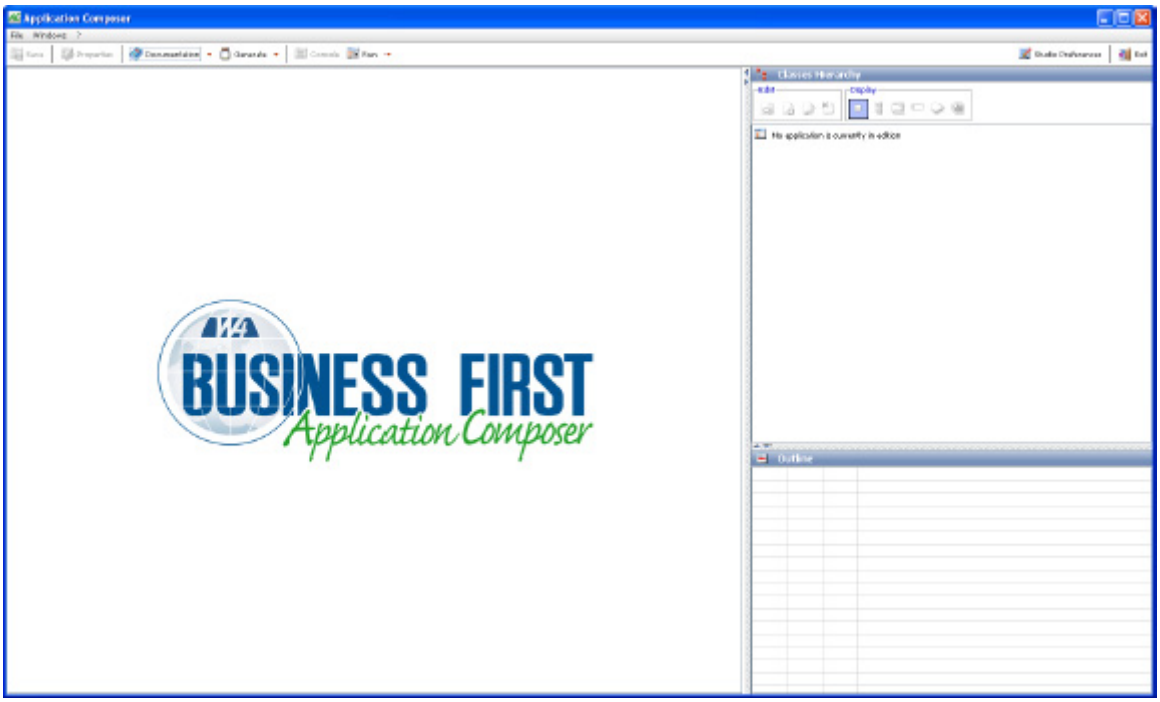

Fig 2.1 *Application Composer right after startup*

You will now create a new application.

TO CREATE THE APPLICATION

1 Select **File** ▸ **New**

The *New : Application* window is displayed.

2 Specify the identifier of the class: videolibrary.

This is an internal identifier. It is used by the designer.

3 Specify the name of the class: Videolibrary.

This name is external. It is visible by the final user.

The save directory can be changed as desired, although for this tutorial you can keep the default value.

4 Specify the language: English.

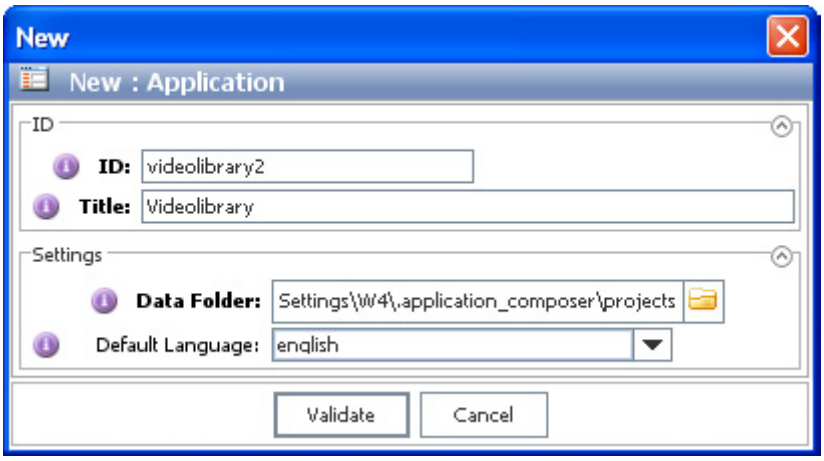

Fig 2.2 *Creating the application*

5 Click **Validate**.

The application has now been created. This initializes the tree structure in the *Class hierarchy*:

Fig 2.3 *The Class hierarchy after the application creation*

6 At that point it can be a good idea to save your work: Click the **Save** button below the menu bar, and remember to save your work at reasonable time intervals as you progress with the application.

Creating the class *Category*

You will now create your first application class, which represents the categories of films in the video library. Categories can be: comedy, drama, science-fiction, etc.

TO CREATE THE CLASS CATEGORY

1 In the *Class hierarchy*, click **Create a class...**

The *Create a class* window is displayed.

This window lets you specify the information related to the class to be created, however for this tutorial, you can specify no further detail than its name.

2 Specify the class name: Category.

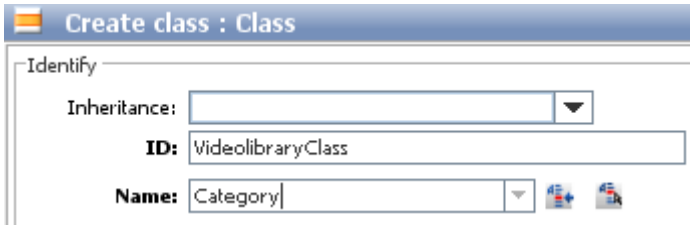

Fig 2.4 *Creating a class*

3 Click **Validate**.

A new node is displayed in the *Class hierarchy*:

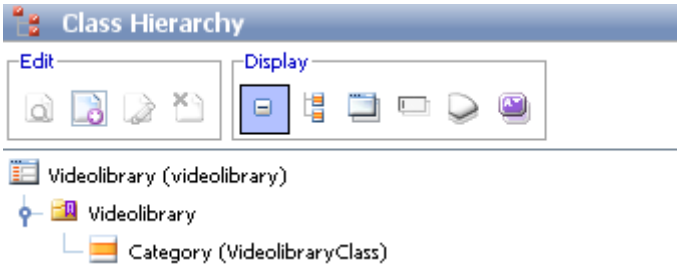

Fig 2.5 *The Class hierarchy after the creation of the class category*

Selecting a node in the *Class hierarchy* displays a preview of the related forms in the *Visual builder* tab.

The form which is displayed in the central pane of the *Visual builder* is WYSIWYG: It is similar to the one that will be displayed at runtime. It is contextual, firstly to the currently selected class, and secondly to the given action.

If you select the *Category* class in the *Class hierarchy*, you will see that so far the form is empty as the class has no attributes:

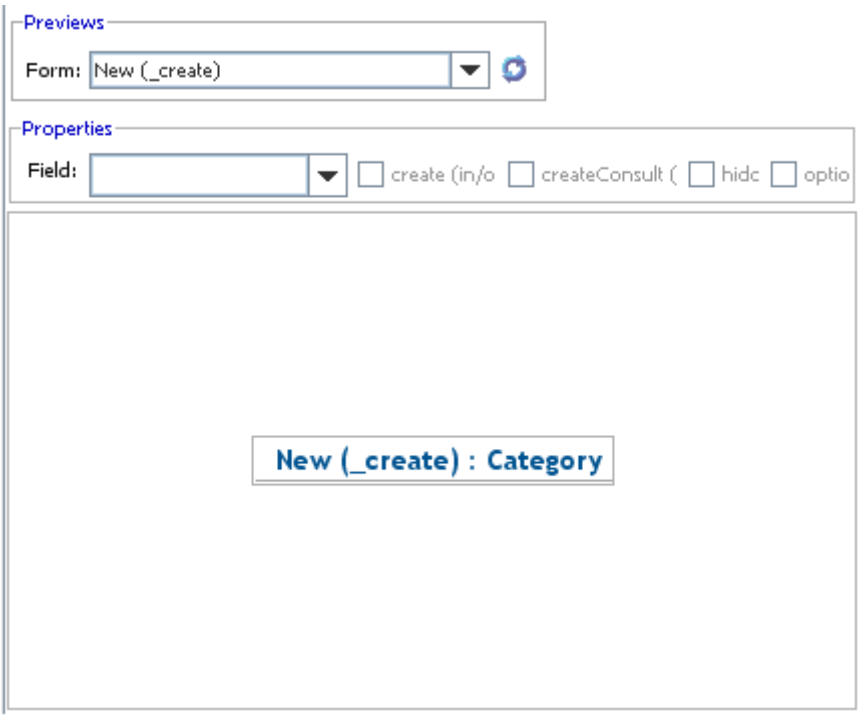

Fig 2.6 *Preview of the class in the Graphic builder*

4 In the *Standard types* area of the *Visual builder*, to the left, add a simple text field for the film genre, either via drag-and-drop, or by double-clicking:

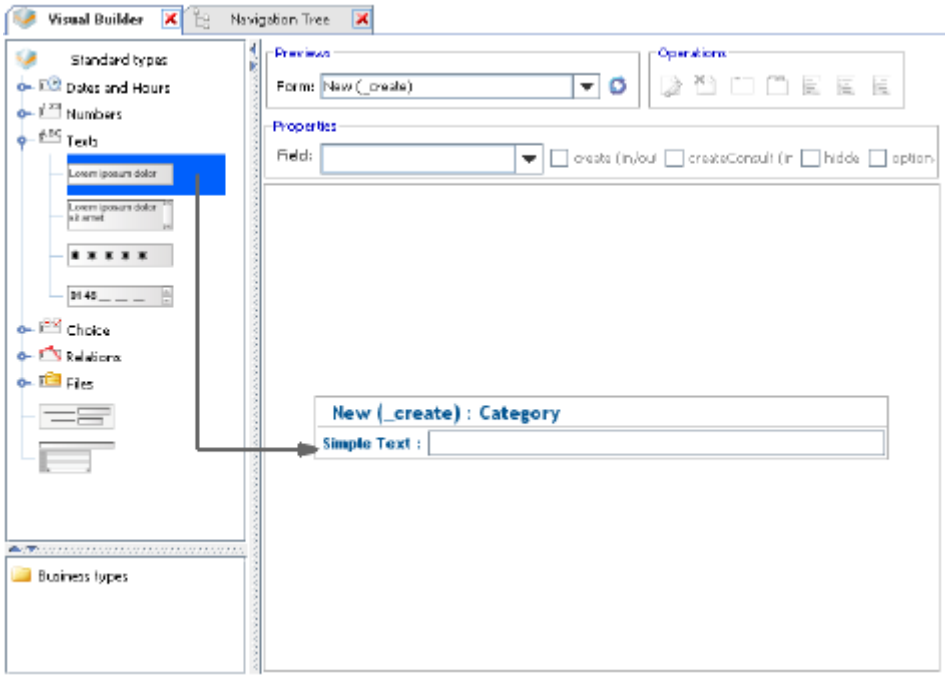

Fig 2.7 *Using the Visual builder to add a field*

5 Double-click the label of the field in the form then give the field a more eloquent name: Genre.

6 Right-click the field then select **Change frequent marks...** from the context menu.

7 In the popup window, select the **name** mark:

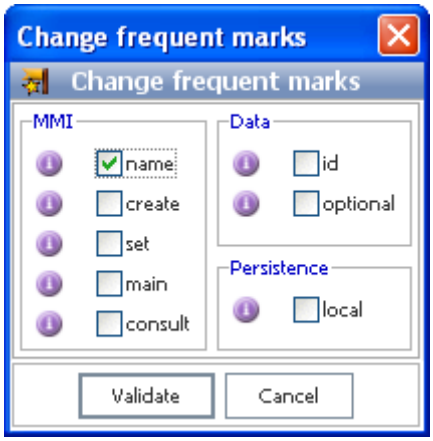

Fig 2.8 *Changing the marks for a field*

8 Click **Validate**.

The *Genre* field will be used for displaying the categories in the various views.

Creating the class *Film*

Like you did for the *category* class, you will now create the second class in your application: *Film*.

TO CREATE THE CLASS FILM

1 The *Category* class is now the currently selected class in the *Class hierarchy*. Before creating the *Film* class, select the second node in the *Class hierarchy*, which represents the *Videolibrary* project.

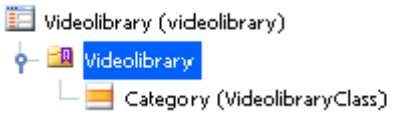

Fig 2.9 *Class hierarchy before the creation of the Film class*

2 Create a new class.

3 Specify the name of the class: Film.

4 Click **Validate**.

A film is described by a title, a category, and a release date:

- *Title* is a simple text field, like the one used when specifying the category.
- *Genre* is a simple relation to the previously defined *Category* class.
- *Release date* is a date field (DD/MM/YYYY format).

5 Create the *Title* field:

5.1 In the *Class hierarchy*, select the *Film* class.

5.2 From within the *Standard types* area of the *Visual builder*, add a simple text field.

5.3 Rename the field.

5.4 As the name of a film matches its title, add the *name* mark to the field.

6 Create the *Genre* field:

6.1 Add a simple relation field.

The *Select target class* window is displayed.

6.2 Select the *Category* class from the drop-down list then click **Validate**.

6.3 Rename the field.

6.4 As *Genre* is not a required attribute for defining a film, add the *optional* mark to the field.

7 Create the *Release date* field:

7.1 Add a date field.

7.2 Rename the field.

7.3 As *Release date* is not a required attribute for defining a film, add the *optional* mark to the field.

The class should now be as follows:

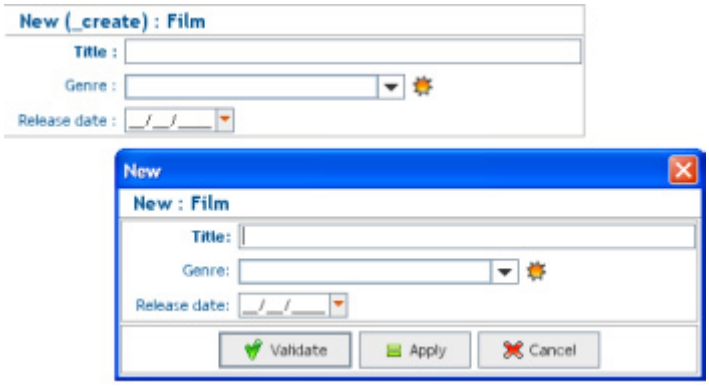

Fig 2.10 *film creation form, in the ediitor and preview in the final application*

NOTE You can change the characteristics of the business classes and their attributes from within the *Outline* area (to the right-bottom of the main window), which is contextual relative to which graphical object is currently selected in the form.
You can for instance change the image associated to the class. When the class (and not its fields) is selected in the editor, double-click the *Image* field in the *Outline* area.
If no image is specified, a default image is used, which displays the first letter of the class name in upper case (*F* for *films*, and *C* for *categories*).
You can see that this field is optional as it is displayed in normal style, in contrast to

bold style.

The data model description of your videolibrary application is now completed. You application is now fully operational.

Running the application in fat client (SWING viewer)

Before running the application, you will specify that the data that will be manipulated will need to be saved when a user session comes to an end. This will allow for persistent, later reusable data (films and categories). To this end, you will need to edit the application properties.

Also, as you will run the application in SWING, you will need to specify the JDK in use in the preferences.

TO SET THE PROPERTIES FOR STEP 1

1. Click the **Properties** button, below the menu bar.

   The *Modify : Application : Videolibrary* window is displayed.
2. Display the *Environment* tab.
3. Edit the LY_SAVE_FILES variable:
   - 3.1 Click the *Value* column next to the LY_SAVE_FILES variable.
   - 3.2 Substitute the current value by true then press ENTER on your keyboard.
   - 3.3 Click *Validate* next to the LY_SAVE_FILES variable.
   - 3.4 Validate the window.

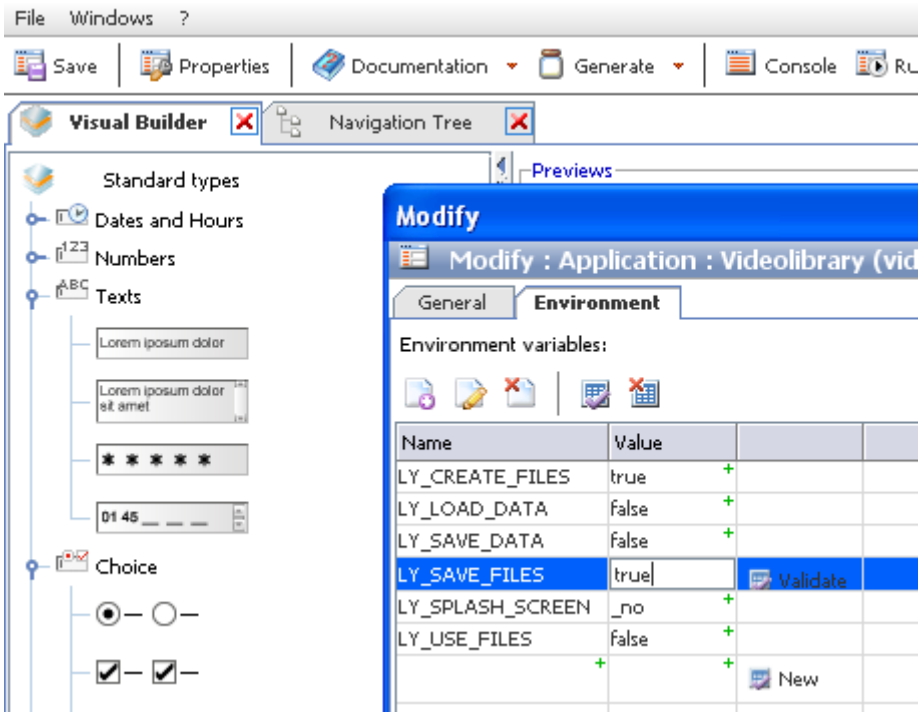

Fig 2.11 *Updating the properties for data persistence*

TO SET THE PREFERENCES FOR STEP 1

1 Click the **Preferences** button, below the menu bar.

The *Preferences* window is displayed.

2 Display the *Java* tab.

3 Specify the directory of the JDK to be used.

4 Click **Validate**.

TO RUN THE APPLICATION IN SWING

1 To run your business application with no effort, just select a graphic environment such as the SWING viewer: Select **Run** ▸ **Display SWING**.

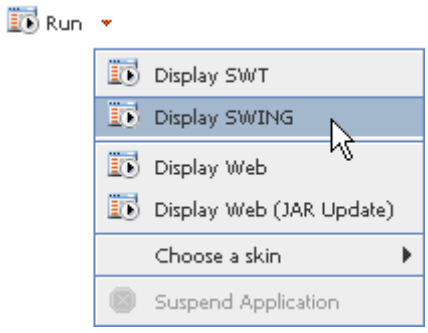

Fig 2.12 *Running the application in SWING (1/2)*

The model is interpreted by the engine, which at this point is able to generate a default view tree, based on your data model, and which will let you access the various screens for managing your data. You can create categories and films.

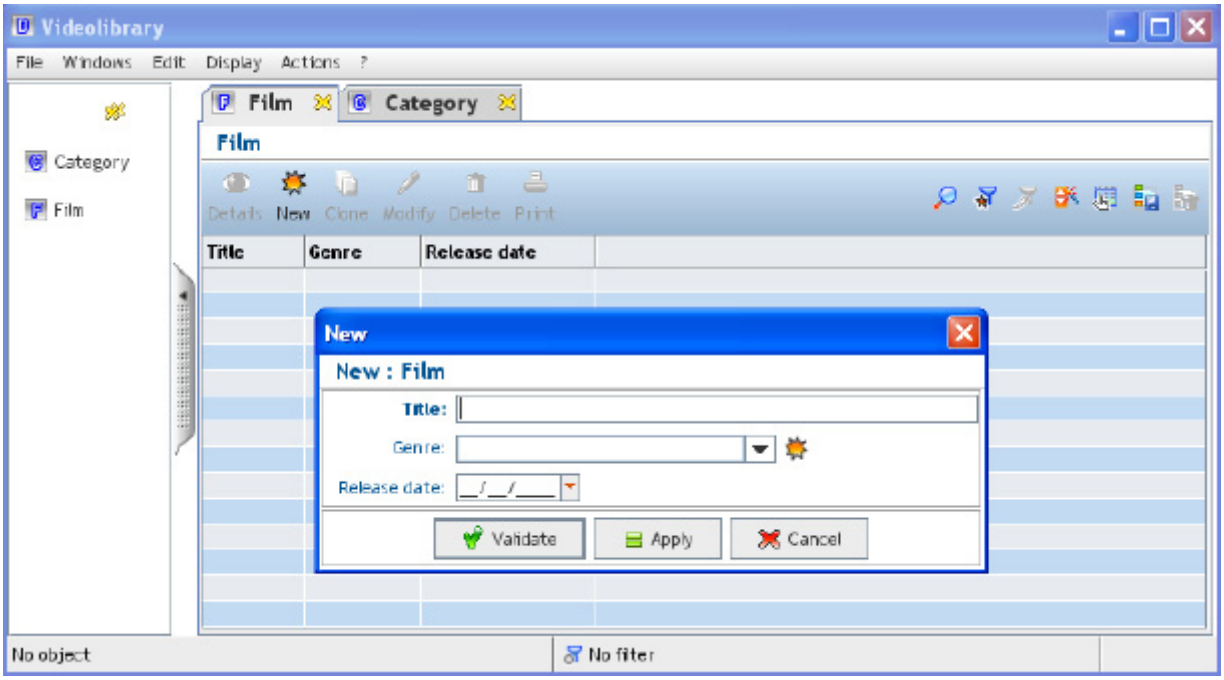

Fig 2.13 *Running the application in SWING (2/2)*

From now on the services for viewing, creating, modifying, and deleting data can be used, as can be the sorting and filtering services.

Step 2 (optional): Describing the navigation

As discussed earlier, this step is not essential for rapidly obtaining an operational application. However it is necessary for customizing the views and it can be useful to address this in the context of this tutorial.

In this section:

Actions and navigation tree

Let us now focus on the navigation tree definition. The *Navigation tree* tab lets you specify how the application data will be displayed to the final user, and how the screen to screen navigation will be organized. So far the tree has no actions in it. First you will just use the GUI automatic generation button.

TO GENERATE THE DEFAULT GUI

1. Display the *Navigation tree*.
2. Click **Generate default GUI**.

   This option creates a root action, i.e. the first action that is run when Application Engine starts the application, and by which you access the screens related to the various business classes, which are displayed as tables.

   The navigation tree generated in our example therefore has a main window via which you can display the list of films and the list of categories:

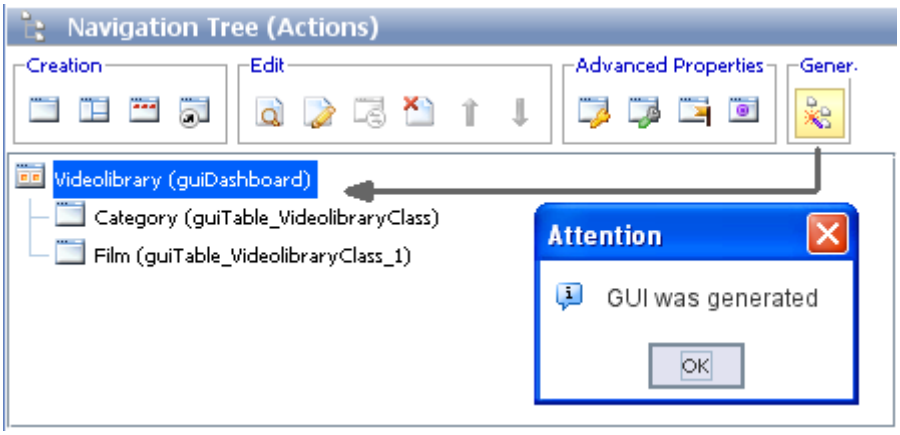

Fig 3.1 *Navigation tree*

Selecting an action in the navigation tree displays a preview of the corresponding view in the central pane.

Creating the compound action *Films by genres*

You will now compose an additional view to display the films by genres, for which you will create a compound view made up of two basic views:

- A tree for viewing the categories, to the left (this is a new action to be created)
- A table for viewing the films matching the selected category, to the right (this is an existing action that will be reused)

TO CREATE THE COMPOUND ACTION

1 In the *Navigation tree*, right-click the root node then select **Create a compound action...** from the context menu.

The *Create a compound action* window is displayed.

2 Rename the action: Films by genres.

3 Create the first view (the tree of the categories):

3.1 In the *Composition* area, click **Create an action**.

The *Create an action 1/2* window is displayed.

3.2 In the *Characteristics* area, select **Tree**.

3.3 In the *Identification* area, substitute the default name by Genres

3.4 Click **Next**.

The *Create an action 2/2* window is displayed.

3.5 From the **Target class** drop-down list, select Category.

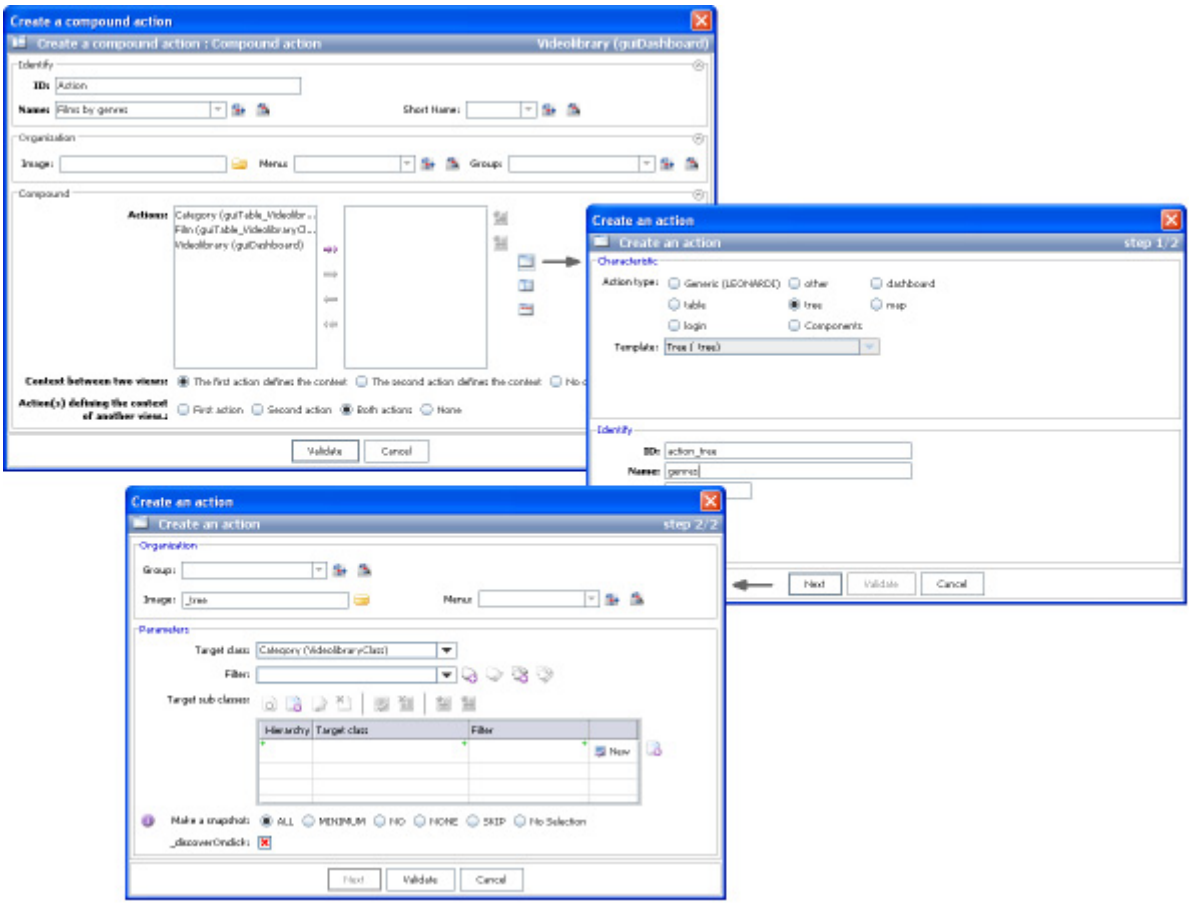

Fig 3.2 *Creating a tree action within a compound action*

3.6 Click **Validate**.

Your first action is now created and is displayed in the right-hand side of the *Composition* area:

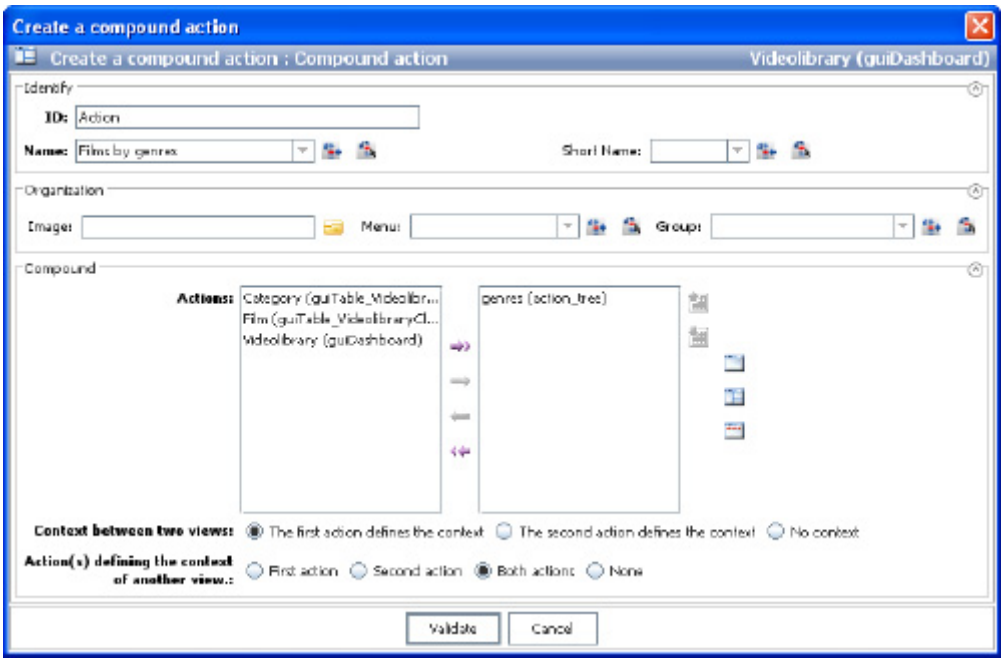

Fig 3.3 *First action of the compound action*

4 For the second action, just reuse the *Film* action, which was automatically generated when the default GUI was generated: Select the *Films* actions in the left area then click ➡ **Select**.

The second action is displayed in the right area:

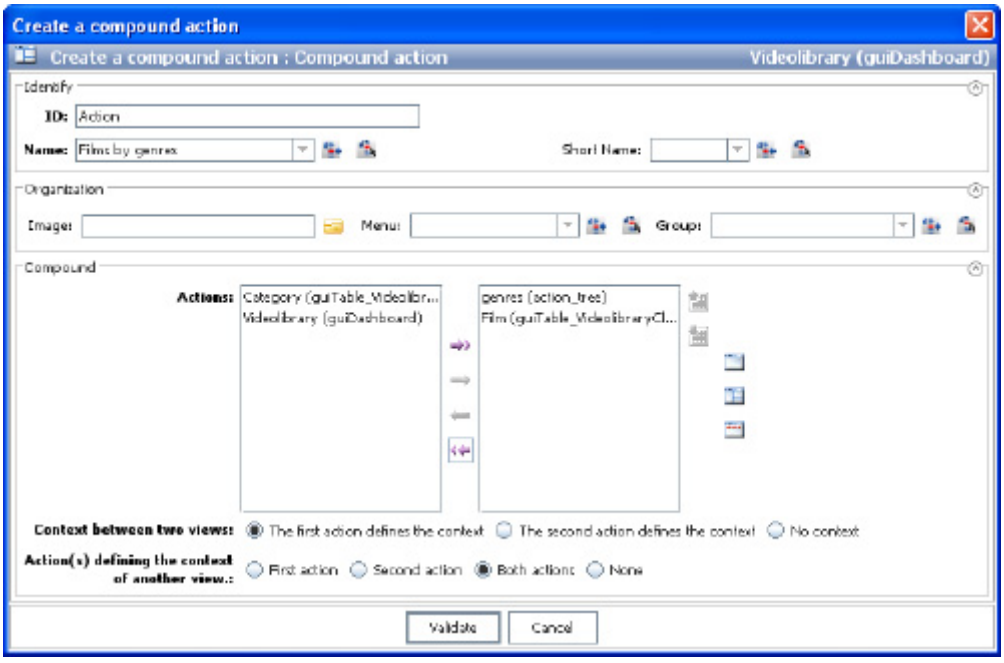

Fig 3.4 *Second action of the compound action*

Note that the context between the views can be specified. In our example, selecting a film genre in the tree decides which films are displayed. Therefore you can leave the default option *The first action defines the context*.

5 Click **Validate**.

The *Navigation tree* is refreshed and displays the newly created compound action:

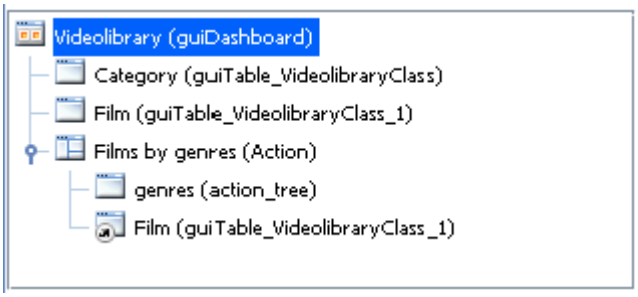

Fig 3.5 *The Navigation tree after the creation of the compound view*

6 At this point you can run the application again, and you may take the opportunity to display it in web mode: Select **Run** ▸ Display Web

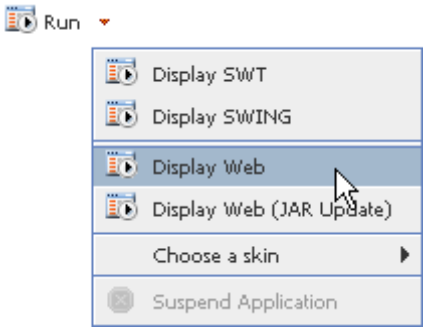

Fig 3.6 *Running the application in web mode*

At runtime you can see that the main window now has a *Film* menu with two options: *Film* (the automatically generated action) and *Films by genre*, the compound action, which you have created.

NOTE To obtain separate entries, just associate distinct images to the actions.

After you have entered a set of data for both the films and the categories, you can use the application like would any final user do, and you will see that selecting a genre in the left does update the list of films displayed in the right. The result should be as follows:

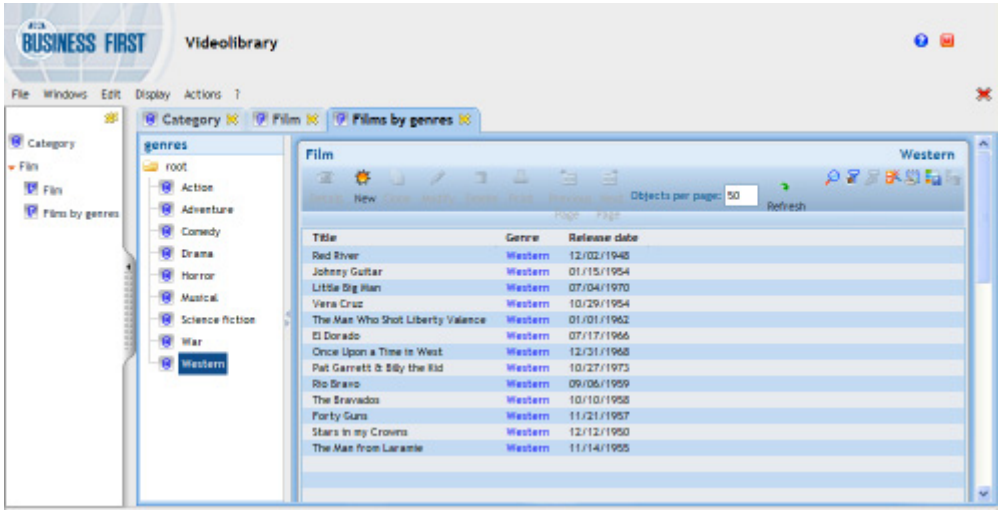

Fig 3.7 *Your videolibrary application in web mode*

Step 3 (optional): Fine-tuning the application

Although your application is fully operational, it is very basic at that point and it should be fine-tuned, for which you will now design two specific behaviors in Java, one for conforming to a business rule, and the other for fine-tuning a graphic behavior.

In this section:

Before starting this step, you will need to review your Application Composer preferences, as defining specific behaviors requires that a default text editor is specified, and also that Application Composer is used by a profile other than beginner.

TO CONFIGURE THE PREFERENCES FOR STEP 3

1. Click the **Preferences** button.
2. In the *General* tab, select a profile other than *beginner*: *standard*, *advanced* or *expert* level.
3. Display the *Java* tab.
4. In the *Editor* field, specify the full path for the executable of the text editor to be used by default.
5. Click **Validate**.
6. Restart Application Composer.

First behavior: Business rule for the film release date

Let us assume that you wish to restrict the films in your videolibrary to the films released in the 20th century. Your first specific behavior will therefore check that the film release date entered by the user is valid, i.e. it is earlier than January, 1st 2001.

When the actions, classes, and fields have been generated at step 1 those items have been automatically given identifiers, which we have kept as is so far in order to save time. However these can be adapted so that the items are easier to use.

TO CHANGE THE IDENTIFIER OF CLASS FILM

1. In the *Class hierarchy*, right-click the *Film* class then select **Modify...** from the context menu.
   The *Modify : Class : Film* window is displayed.
2. Substitute the existing identifier by: film
3. Click **Validate**.

TO CHANGE THE IDENTIFIER OF THE RELEASE DATE FIELD

1. In the *Class hierarchy*, select the *Film* class.
2. Display the *Visual builder* tab.
3. Right-click the *Release date* field then select **Modify...** from the context menu.
   The *Modifiy : Time attribute : Release date* window is displayed.
4. Substitute the existing identifier by: film_date.
5. Click **Validate**.

TO ADD THE DATE CONTROL

1. In the *Class hierarchy*, right-click the *Film* class then select **Class behavior...** from the context menu.
   The *Class behavior* window is displayed:

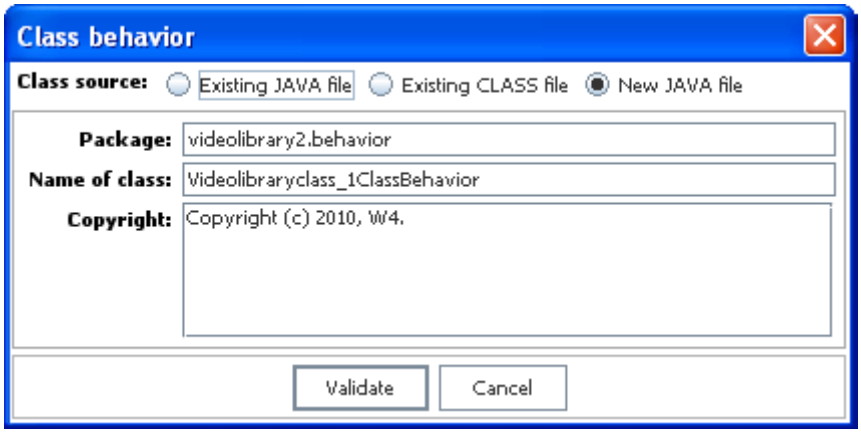

Fig 4.1 *Generating a behavior class*

2. Click **Validate**.

Validating the default options in the class behavior creation window automatically creates a Java class - *FilmClassBehavior.java* - which is displayed in the editor configured in the preferences.

In this class, you will override the *controlNewValues()* method. This method is invoked by Application Engine when an object of the related application class (in this case: a film) is created or modified. It also performs an overall control of the object data before the creation or modification is validated.

Full documentation for the Java API and the methods that can be specialized can be found in the *W4 Documentation Browser*.

3 To be able to use dates in Java, add the following instructions at the beginning of the file:

```
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
```

4 To implement the control on the film release date, place the following code in the Java file, instead of the autogenerated method skeleton:

```
public short controlNewValues(LySetController set, LyValueSet
newValues)
{
   // Retrieve the date
   LyValue val = newValues.getFieldValue("film_date");
   if ((val != null) && (val.getValue() != null))
   {
      // Get the film date
      Date filmDate = ((Date)val.getValue());

      // Test if the film date is before 2001
      DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
      Date limitDate = null;
      try {limitDate = df.parse("2000-12-31");}
      catch (Exception e){e.printStackTrace();}
      if (filmDate.compareTo(limitDate) > 0)
      {
         // Display error message and return an error
         set.showError("The release date is invalid!");
         return STATUS_KO;
      }
   }
   // Generic processing
   return super.controlNewValues(set, newValues);
}
```

5 Save the Java file.

6 Compile the Java file:

6.1 In the *Class hierarchy*, select the *Film* class.

6.2 Display the Java behavior window: Select **Windows** ▸ **Java classes**.

6.3 In the *Java classes* window, select the class *videolibrary.behavior.FilmClassBehavior*

6.4 Click **Compile** (or compile the corresponding Eclipse project).

The bullet in front of the library should switch from red to green, meaning the class has been successfully compiled.

7 Restart the application using the viewer of your choice then check that the film release date control is performed when creating or modifying a film. Now try to create a film with a release date in the 21st century!

Second behavior: Fine-tuning the graphic representation for certain films

You will now use a second Java class for customizing your application. You will create an action behavior for acting upon the presentation: In the film table you will highlight the films released in the year 2000.

TO ADD THE DATE CONTROL

1 In the *Navigation tree*, right-click the class representing the list of films then select **Action behavior...** from the context menu.

The *Action behavior* window is displayed.

2 For readability purposes, substitute the existing class name by: FilmTableBehavior

3 Click **Validate**.

Validating the default options in the action behavior creation window automatically creates a Java class - *FilmTableBehavior.java* - which is displayed in the editor configured in the preferences.

In this class, you will override the *getRowColor()*. This method is invoked by Application Engine for specifying the color to be used when displaying a row in the table of the films.

4 To be able to use dates in Java, add the following instructions at the beginning of the file:

```
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
```

5 To implement the change in a row color, place the following code in the Java file, instead of the autogenerated method skeleton:

```
public        String        getRowColor(LySimpleTableController
tableController, LyObject object)
{
   // Retrieve the film date
   LyValue val = object.getFieldValue("film_date");
   if ((val != null) && (val.getValue() != null))
   {
      Date filmDate = ((Date)val.getValue());

      // Test if the film date is before 2001 and after 2000
      DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
      Date limit1 = null, limit2 = null;
      try
```

```
        {
           limit1 = df.parse("2000-01-01");
           limit2 = df.parse("2001-01-01");
        }
        catch (Exception e){e.printStackTrace();}

        // Check the date to return adequate color
        if      (filmDate.compareTo(limit2)      <      0      &&
filmDate.compareTo(limit1) >= 0)
           return "yellow";
     }
     // Generic processing
     return super.getRowColor(tableController, object);
}
```

6 Save the Java file.

7 Select the first node (node of the application) in the *Class hierarchy*.

8 Compile the Java file.

9 Restart the application using the viewer of your choice then check that the graphic behavior specified for the table of films is applied when creating or modifying a film released in 2000.

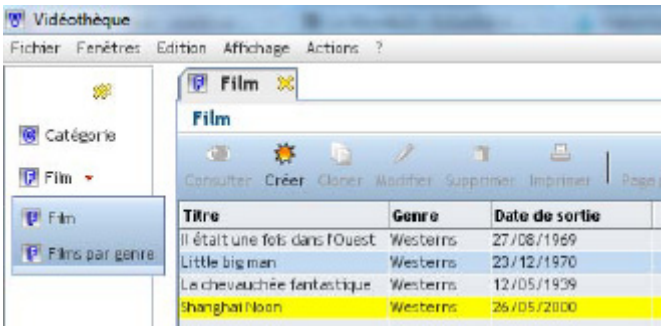

Fig 4.2 *Action behavior: Highlighting certain films*

Step 4 (optional): Connecting the application to a MySQL database

So far the application has been used with flat files, i.e. the data has been saved in files stored on your hard drive, separate from the application. However the data cannot be easily shared with other people: So that it can be shared, the application' *data* directory would need to be copied across machines, at the expense of data consistency.

Using a database lets you maintain consistency across all client applications. You will now create a database for the application, using MySQL.

In this tutorial we will assume that you have an operational MySQL database server. A catalogue may exist in the database, although this is not required.

Application Composer requires the JDBC drivers of the databases to which it can connect. In our example, we will need to get the JDBC connector for MySQL, which can be downloaded at:

> http://dev.mysql.com/downloads/connector/j/

TO ADD THE MYSQL DRIVER

1. Display the *Java classes* tab.
2. In the *Libraries (Classpath)* area, click **Add an archive...** to add a Java archive to the libraries in use.
3. Select the Jar file of the MySQL connector.

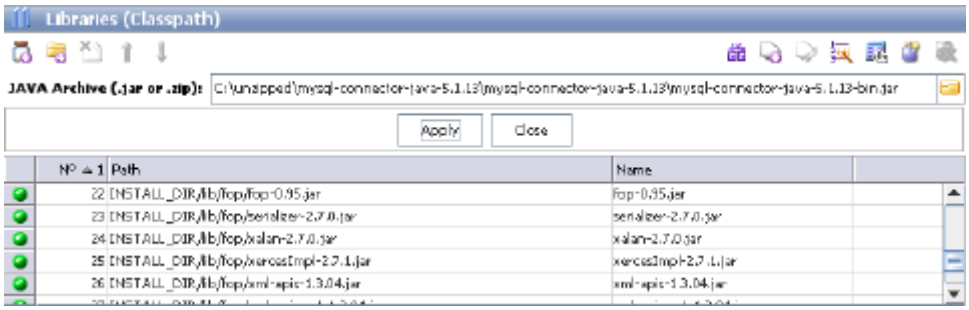

Fig 5.1 *Adding the MySQL driver to the libraries (Java classpath)*

4 Click **Apply**.

5 Select **Generate** ▸ **Generate / Alter database**.

The *Generate / alter database* window is displayed.

6 Specify the **Database type** field: mysql.

7 You will now create a database *location*. This is a configuration with all the details required for a database connection.

7.1 Click **Create RDBMS data** next to the **RDBMS location** field.

The *Create : DBMS data* window is displayed.

7.2 Specify the identifier: videolibrary_db.

7.3 Specify the driver.

This is the entry Java class of the JDBC driver.

7.4 Specify the connection URL. This URL is dependent on the database and the JDBC driver. Typically for MySQL, the URL is such as:
jdbc:mysql://[host]:[port]/[catalogue].

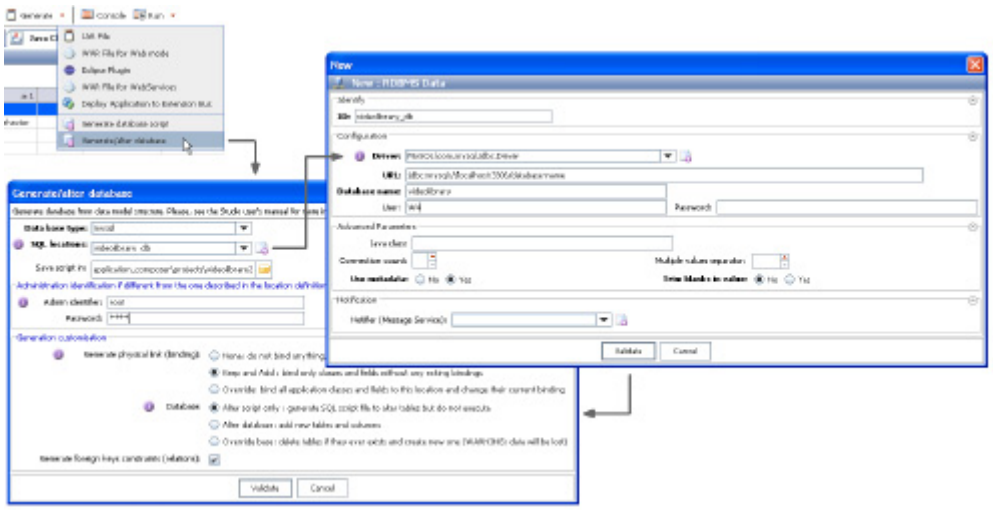

Fig 5.2 *Creating the MySQL database*

7.5 If the catalogue specified at database creation does not exist yet, it will not be possible to associate it to the specified user. Therefore it is necessary to specify the identifier and password of the administrator user who will be in charge of creating the new database along with the required tables. To be on the safe side, it can be a good idea to require the overwriting of the previous entries in the database as well as any previous links with other databases in Application Composer. This decision is up to the user.

7.6 Click **Validate**.

8 Click **Validate**.

When the action has been validated, the database will be created and automatically used by the application. Do not forget to create and assign a user with restricted permissions if the database has been created by the administrator.

Where to go from here?

We have been developing a basic application for managing a personal videolibrary, which lets you perform a variety of actions directly in a database, such as creating, editing and reviewing objects, and also perform standard operations such as searching, sorting, filtering, printing, etc. All this has been achieved by writing optional functional code: Performing a functional control on the data (checking the film release date) and customizing the colors in a list (highlighting the films released in 2000).

The first operational version, or *prototype* so to speak, uses just a few of the Application Composer features. To achieve a real world application, you could connect the classes to advanced data sources: LDAP directories, application servers, etc. and maybe add a JMS bus to take advantage of real-time update.

You could also complement your model by adding new application classes (via UML import, via XMI, or by peforming the discovery of an existing database), by adding new behaviors for the data, and by adding more complex views (complex tables, map views, compound views, Gantt diagrams, paginated tables, Excel import/export, PDF printing, etc).

Finally, you could customize your application according to the desired visual style guidelines, images, colors, position constraints for the fields in the views, etc.

Illustrations

A

B

C

D

G

J

L

M

N

O

P

R

S

W

Application Composer

QUICK START TUTORIAL

Reference: APPCOMPOSER_QUICKSTART_041_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