

## SpagoBI QuickStart

Authors

Luca Scarel  
Grazia Cazzin

## Index

<b>VERSION .....</b>	<b>4</b>
<b>1 DOCUMENT GOAL .....</b>	<b>4</b>
<b>REFERENCES .....</b>	<b>4</b>
<b>HELP FOR LECTURE.....</b>	<b>5</b>
1.1 STYLISTIC CONVENTIONS.....	5
1.2 SPECIAL SECTIONS .....	5
<b>2 SPAGOBİ - CONCEPTUAL OVERVIEW.....</b>	<b>6</b>
<b>3 RECURRING THEMES.....</b>	<b>12</b>
3.1 PORTLET LAYOUT .....	12
3.2 LIST AND DETAILED VIEW .....	12
<b>4 FUNCTIONALITY OVERVIEW.....</b>	<b>14</b>
4.1 SETTINGS.....	14
4.1.1 Engines Configuration.....	14
4.1.2 Roles synchronization.....	16
4.1.3 Import/Export.....	17
4.1.3.1 Export .....	18
4.1.3.2 Import .....	19
4.2 ADMINISTRATOR .....	21
4.2.1 Functionalities Management .....	22
4.2.2 Document Configuration .....	24
4.3 DEVELOPER.....	26
4.3.1 Predefined List of Value (LOV) .....	27
4.3.2 Predefined Values Constraints.....	29
4.3.3 Parameters Management.....	31
4.3.4 Document configuration .....	32
4.3.4.1 Parameters correlation .....	35
4.3.4.2 Links .....	37
4.3.4.3 Documents visibility .....	39
4.4 TESTER .....	40
4.5 END-USER.....	43
<b>5 GETTING STARTED WITH SPAGOBİ.....</b>	<b>44</b>
5.1 INSTALL EXOTOMCAT AND SPAGOBİ DEMO .....	44
5.2 REPORT .....	44
5.2.1 Create a Report template.....	45
5.2.1.1 Create a Report template using iReport .....	45
5.2.1.2 Create a Report template using Birt.....	46
5.2.2 Create a Parameter .....	48
5.2.2.1 Predefined List of Value (LOV) .....	48
5.2.2.2 Predefined Values Constraints .....	49
5.2.2.3 Parameters Management.....	49
5.2.3 Register the Analytical Document (the built report) into the platform .....	50
5.2.4 Test the Analytical Document .....	52
5.2.5 Execute the Analytical Document .....	52

5.3	OLAP ANALISYS .....	52
5.3.1	Create a template.....	53
5.3.2	Create Parameters.....	53
5.3.2.1	Predefined List of Value (LOV) .....	53
5.3.2.2	Predefined Values Constraints .....	54
5.3.2.3	Parameters Management .....	55
5.3.3	Register the Analytical Document (the built OLAP) into the platform .....	56
5.3.4	Test the Analytical Document .....	57
5.3.5	Execute the Analytical Document .....	57
5.4	DASHBOARD .....	57
5.4.1	Create a list of value .....	58
5.4.2	Create a template.....	58
5.4.3	Register the Analytical Document (the built DASHBOARD) into the platform.....	59
5.4.4	Test the Analytical Document .....	60
5.4.5	Execute the Analytical Document .....	60
5.5	DATA MINING .....	60
5.5.1	Create a Data mining process using Weka KnowledgeFlow.....	60
5.5.2	Parametrize process flow.....	62
5.5.3	Create a Parameter .....	63
5.5.3.1	Predefined List of Value (LOV) .....	63
5.5.3.2	Predefined Values Constraints .....	64
5.5.3.3	Parameters Management .....	64
5.5.4	Register the Analytical Document (the built Data-Mining Flow) into the platform .....	66
5.6	QUERY BY EXAMPLE.....	68
5.6.1	Field Selection .....	69
5.6.2	Condition .....	70
5.6.3	View Query .....	71
5.6.4	Save Query.....	72
5.6.5	Execute query.....	73
5.6.6	Generate Template.....	73
5.7	MAPS.....	74
5.7.1	How to get a geography svg map ?.....	75
5.7.2	Write a map template file.....	76
5.7.3	Define a Map document into SpagoBI.....	77
6	IN MORE DEPTH.....	78
6.1	PORTAL ADMINISTRATOR AND PORTLETS ORGANIZATION .....	78
6.2	ANALYTICAL DOCUMENT LIFE-CYCLE.....	78
6.3	USER ROLES .....	79
6.4	DOCUMENT ORGANIZATION AND SECURITY POLICY .....	79
6.5	USER DEFINITION AND ROLES MANAGEMENT.....	80
6.6	PORTAL DEFINITION.....	80
6.7	DATA MART (.JAR) DEVELOPMENT FOR QBE FEATURE .....	81
6.7.1	Brief QBE introduction.....	81
6.7.2	How to create database model jar file .....	81
7	GLOSSARY .....	91

**Version**

<b>Version/Release n°:</b>	0.6	<b>Data Version/Release:</b>	December, 14 <sup>th</sup> 2005
<b>Update description:</b>	First release – Draft		
<b>Version/Release n°:</b>	0.7	<b>Data Version/Release:</b>	January, 31 <sup>st</sup> 2006
<b>Update description:</b>	Second release – Draft		
<b>Version/Release n°:</b>	0.8	<b>Data Version/Release:</b>	April, 11 <sup>th</sup> 2006
<b>Update description:</b>	Third release – Draft (Added paragraph on datamart jar creation)		
<b>Version/Release n°:</b>	0.9	<b>Data Version/Release:</b>	July, 20 <sup>th</sup> 2006
<b>Update description:</b>	Fourth release – Draft		

**1 Document goal**

The document aim is to introduce the reader to the SpagoBI concepts by means of a full example based on the "SpagoBI demo distribution". The demo is freely downloading from the ObjectWeb forge ([http://forge.objectweb.org/project/showfiles.php?group\\_id=204](http://forge.objectweb.org/project/showfiles.php?group_id=204))

The document includes the following main chapters :

- **Conceptual overview.** Introduction of the core concepts of the SpagoBI free open source platform.
- **Functionality overview.** An explanation of the SpagoBI portlets and of the main tasks of the various users typologies, considering the SpagoBI demo as an example.
- **Getting started with SpagoBI.** How to build step-by-step an analytic application by means of the case study available in the SpagoBI demo distribution. Starting from the development of a template, you are introduced to the document parametrization and configuration following a logical path. The chapter uses simple examples to explain the SpagoBI main concepts.
- **In more depth.** How to build an analytical portal in more detail.

**References**

For further information about SpagoBI platform refer to the following documentation, available on the project site (<http://spagobi.eng.it>):

- [1] Cazzin G., Ruffatti G., *SpagoBI Overview*
- [2] Cazzin G., *SpagoBI Architectural Design*
- [3] Zoppello A., *SpagoBI Installation Manual*

## Help for lecture

Follows a short description of the most common views in SpagoBI.

### 1.1 STYLISTIC CONVENTIONS

LITTLE CAPITALS	The LITTLE CAPITALS references to the icon in a mask.
<i>italics</i>	The <i>italics</i> refers to fields of the masks.
<ITALIC CAPITALS>	In <ITALIC CAPITALS> the logical variables are suitable.
<b>boldface</b>	In <b>boldface</b> the main concepts.

### 1.2 SPECIAL SECTIONS



Note



Example



Reference to other section



In revision phase



Future implementation. To be done.



Advice for the reading of the section

## 2 SpagoBI - Conceptual overview

SpagoBI is a platform for the development of Business Intelligence projects: SpagoBI offers all the tools and the necessary components for the realization of analytical portals, whose designing and setting are the main project activities.

The development of an analytical portal with SpagoBI doesn't require the implementation of some J2EE services; you have only to set up the analytical documents and to register them correctly in the platform.

The **analytical documents** provide the end user with the needed information, in the most suitable way. SpagoBI allows to use many categories of analytical tools: Report, OLAP, Data Mining, Dashboard, Visual Inquiry, Maps.

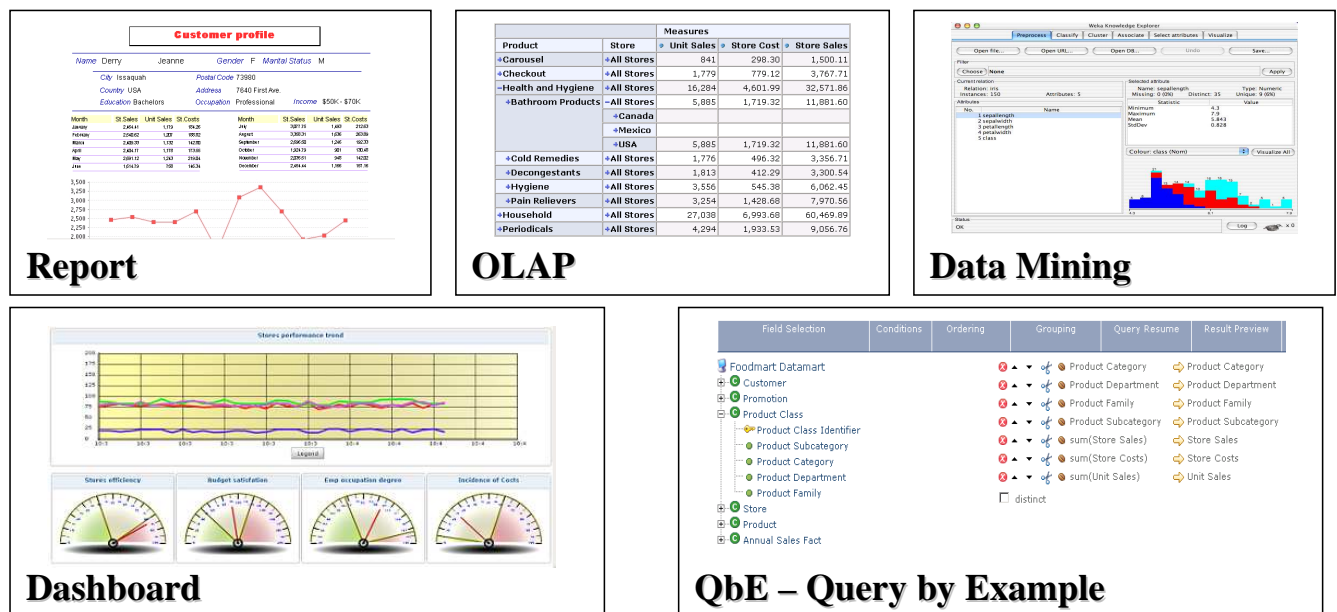
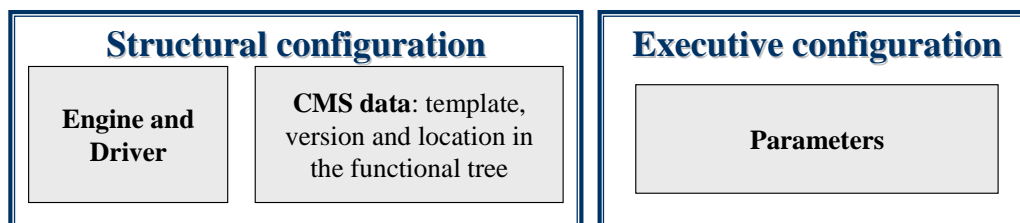
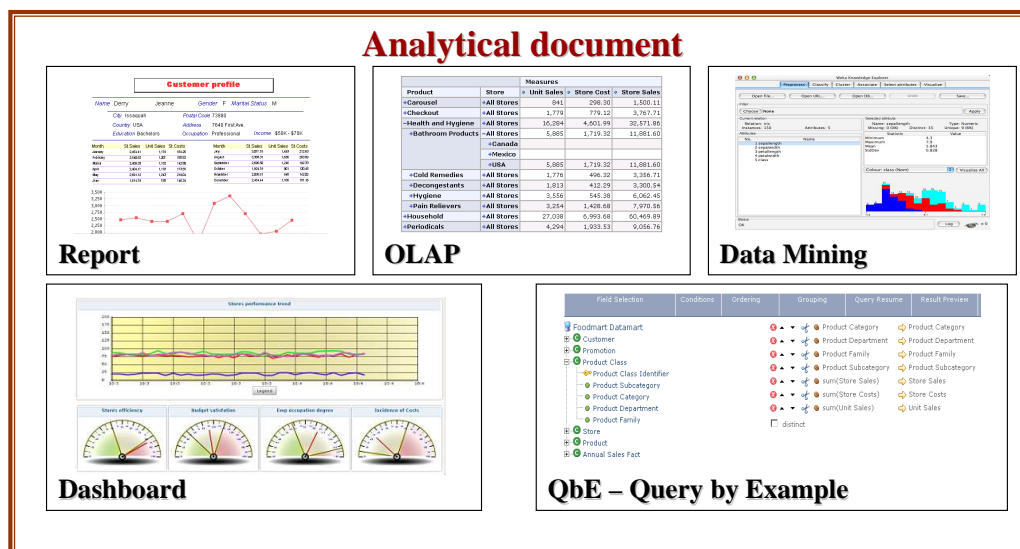


Figure 1 - Analytical documents

The Business Intelligence analyst and designer have to find the most suitable tool for every type of analysis and category of user. In fact, the building of an **analytical portal** is a balanced composition of different tools in order to give to each users' category the right degree of visibility and at the same time a freedom of movement through the information of his pertinence.

SpagoBI realizes both the structural and the executive support to the single analytical areas.



**Figure 2 – Analytical documents support**

Regarding to the **structural configuration**, every analytical documents' category refers to a particular (one or many) execution engine, interacting with SpagoBI through a specific driver. For every analytical document SpagoBI keeps the history of the templating version and organize them in the functional tree.

The SpagoBI demo has got a preloaded metadata environment allowing to manage:

- report on the JasperReport engine and Birt engine;
- OLAP on the Mondrian engine, with Jpivot interface;
- Dashboard on the OpenLazslo engine;
- free inquiry on a Hibernate implementation;
- data mining model on Weka engine.



The Dashboard implementation by means of OpenLazslo compiler is under a phase of greater integration through.



Many other alternative engines in every analytical area will be integrated in the SpagoBI platform in the future.

Regarding to the **executive configuration**, SpagoBI manages some parameters as autonomous and independent entities. The parameters include the behaviour rules (presentation and validation) according to the end-user roles. Through the parameters,

SpagoBI builds an executive environment which places in the middle the operative model referred to the particular reality in use.

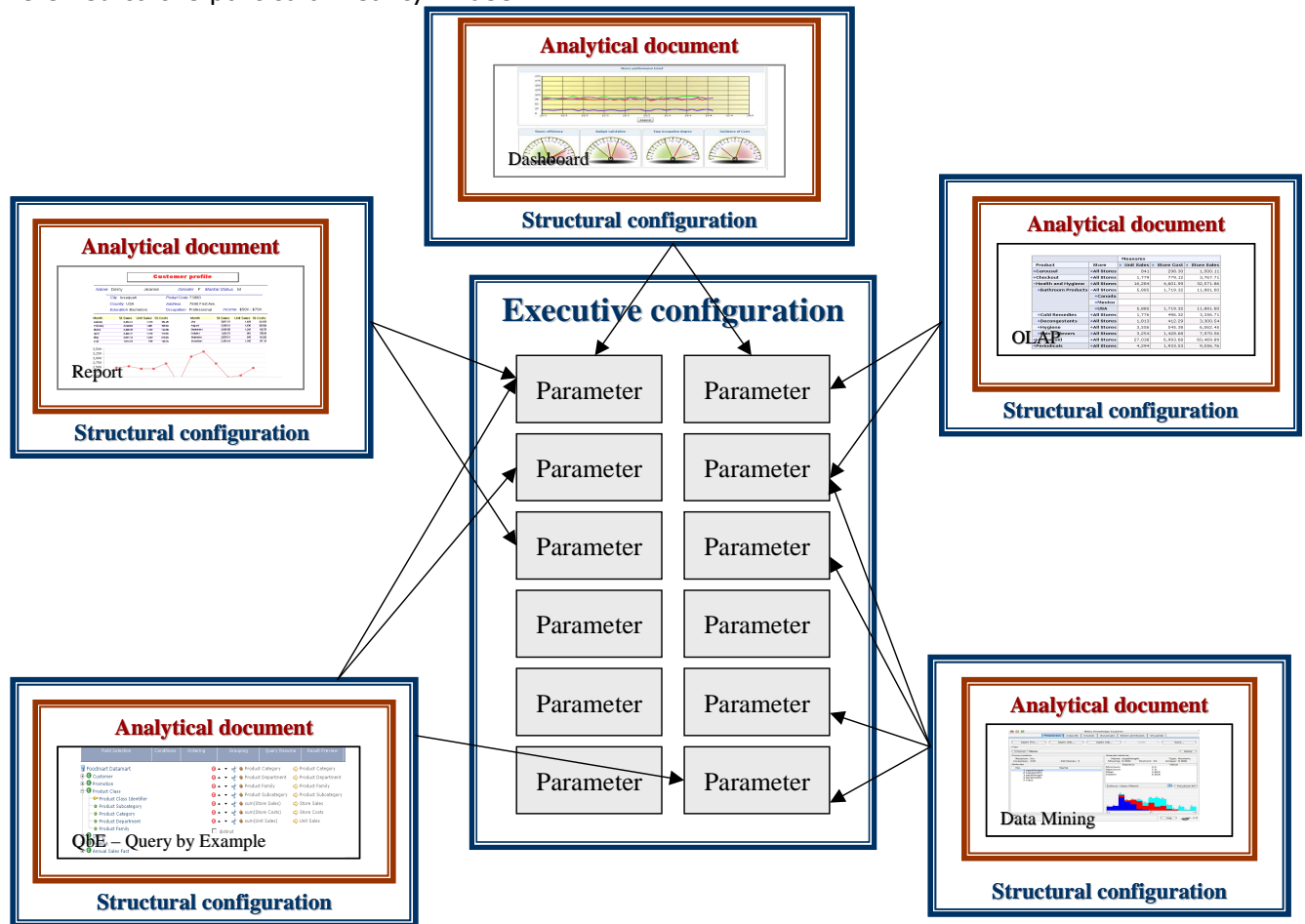


Figure 3 – Executive configuration

A **parameter** models a concept or a datum frequently used as discriminant on the global data context.

Every parameter can be used in many different ways, according to the different end-users' roles. So, every use mode refers to an initial visualization method, to one or more validation rules and to one or more end-user roles.

Follow a parameter abstract schema and an implementation example.



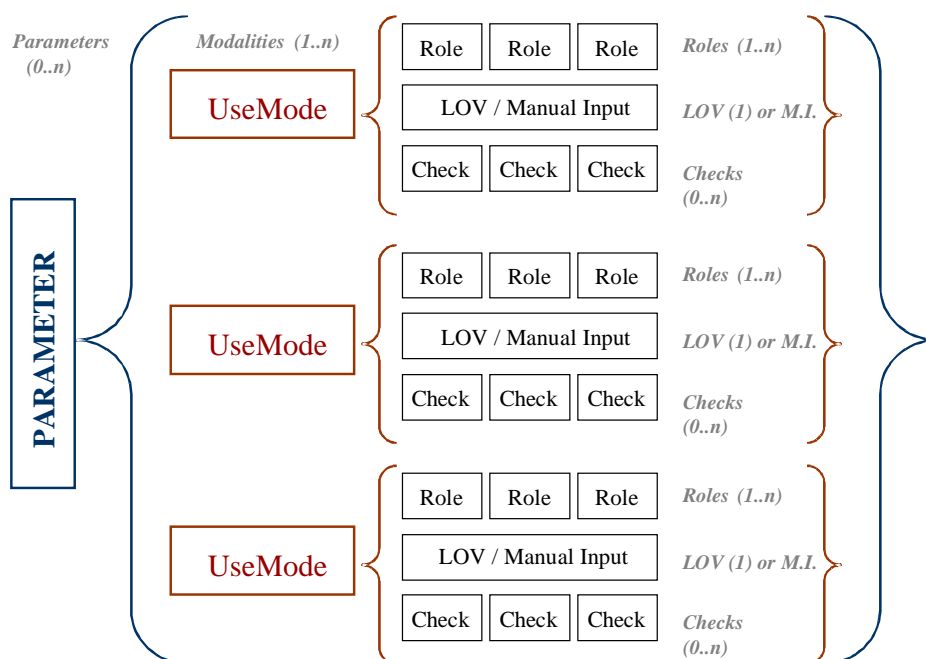


Figure 4 – Parameter's abstract structure

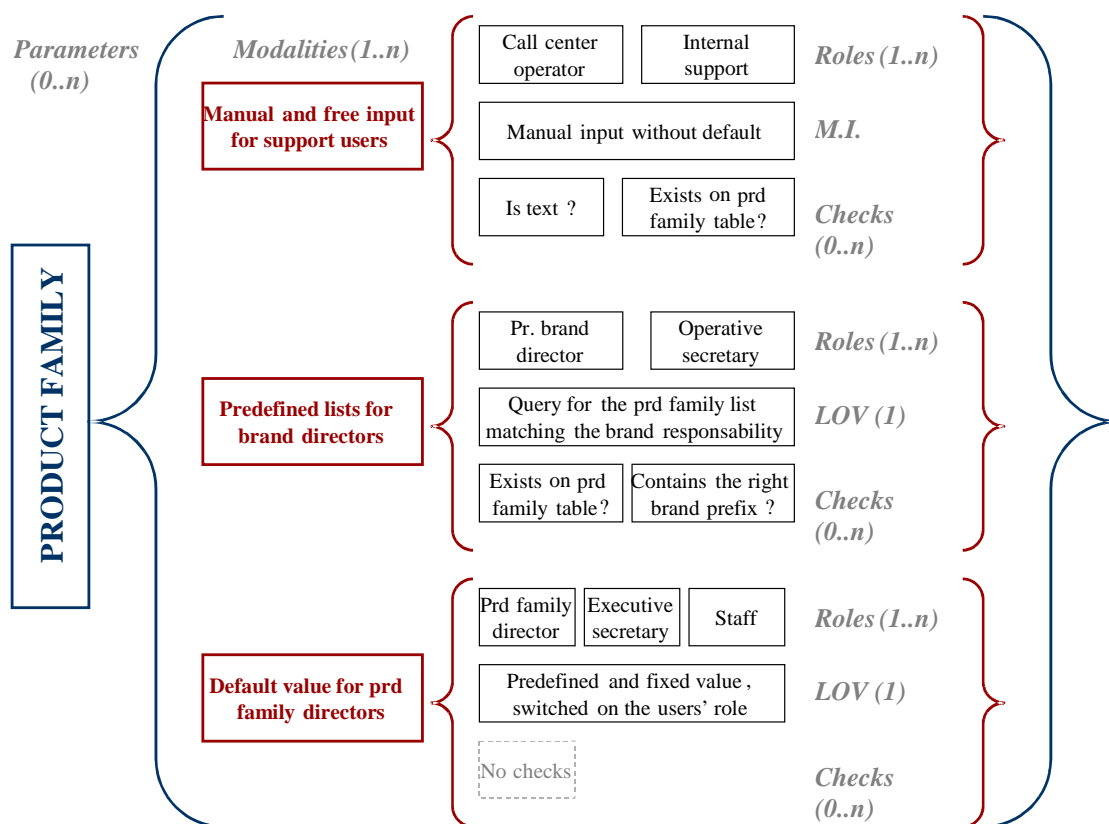


Figure 5 – Parameter's implementation example

Every parameter can be associated to many different analytical documents (also for category) driving their behaviour according to its rules.

The recording phase of an analytical document must therefore set its structural (driver, engine, CMS) and executive (parameters) information.

When a user (with its role) runs an analytical document, the structural information are read and then a custom page for the parametric input is produced on the basis of the execution information. At the end the document is produced on the basis of the inserted values (explicitly or implicitly).

A sample follows in the picture below:

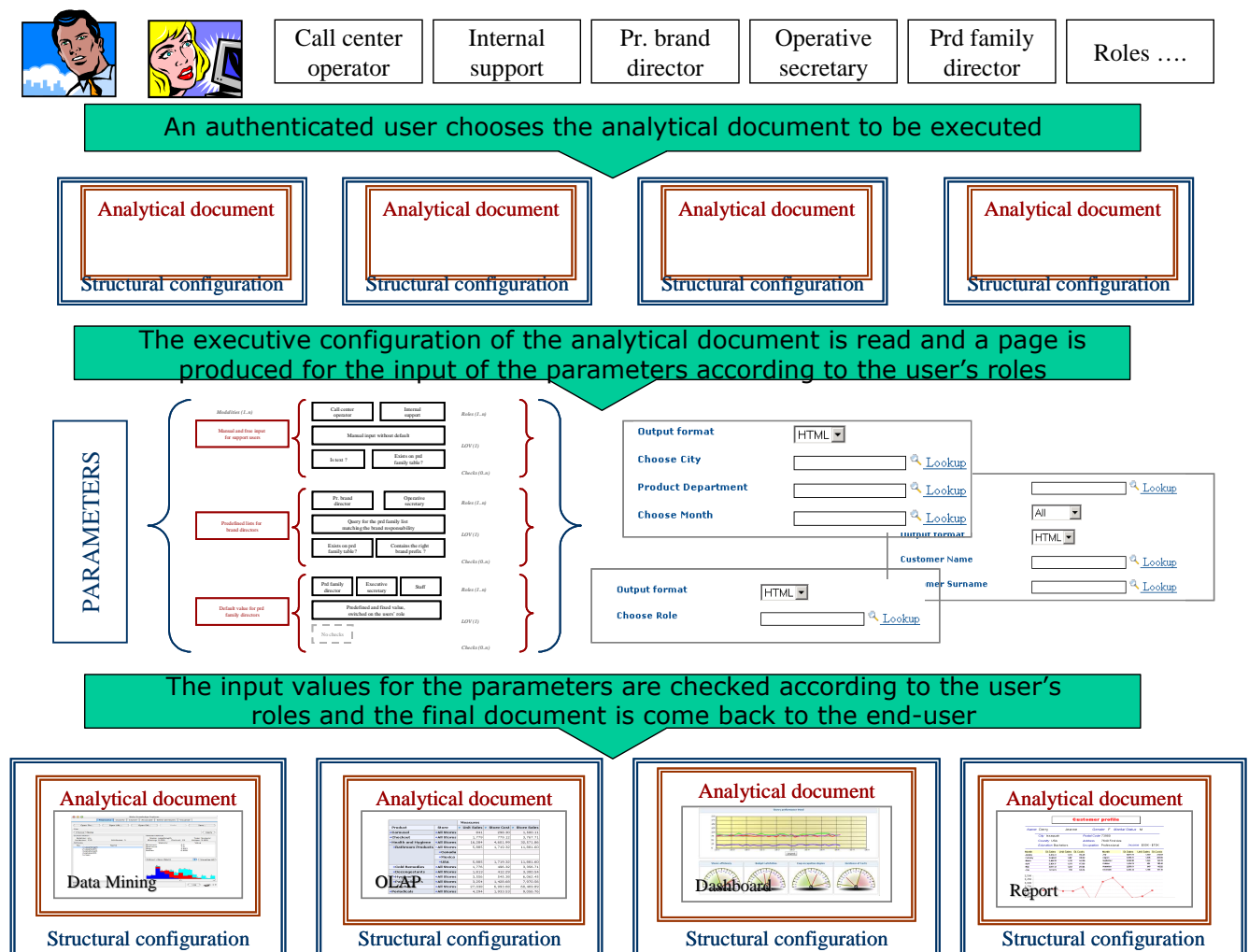
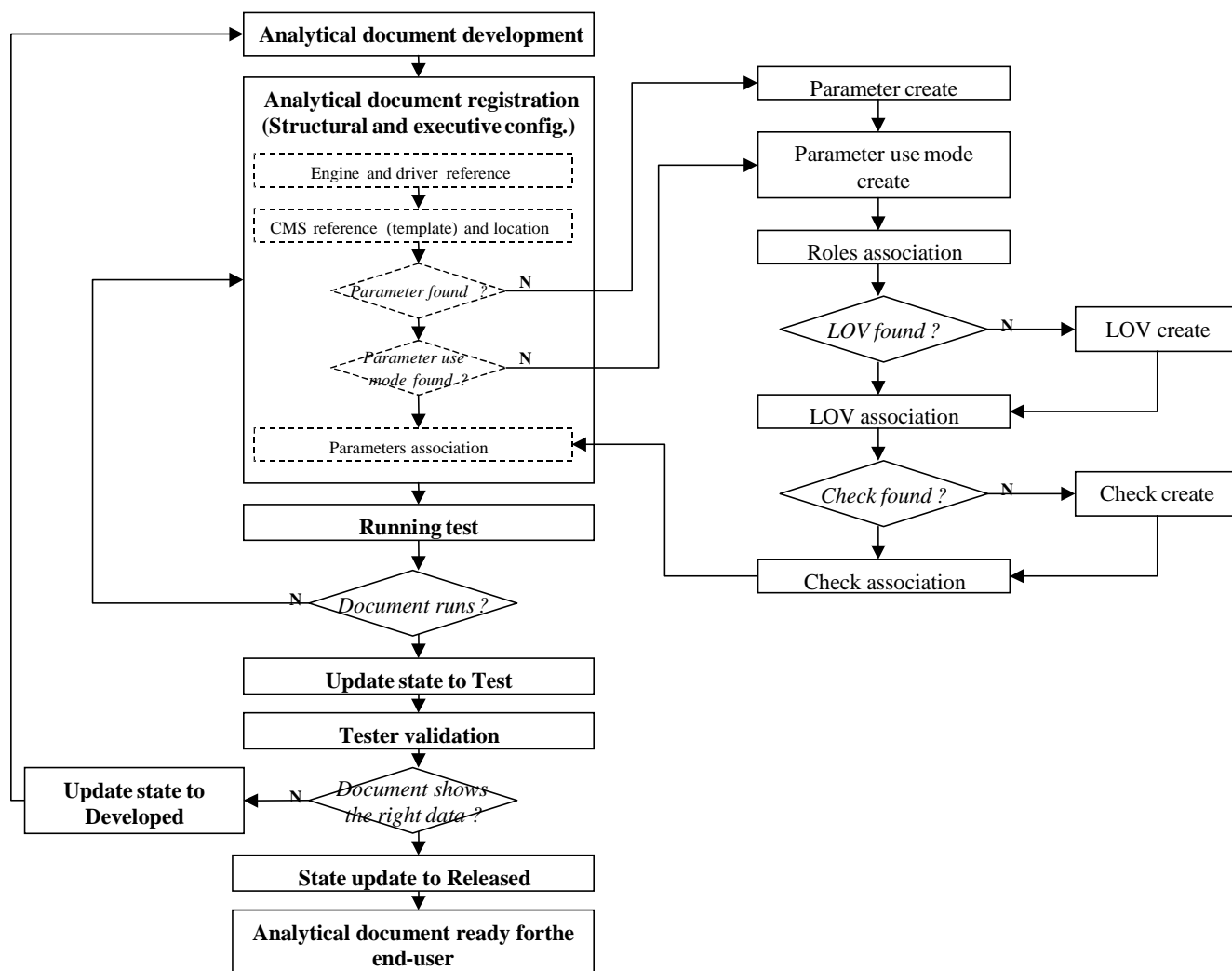


Figure 6 – Analytical document execution

A new analytical document requires a process handling as the one shown in the following schema:



**Figure 7 – Analytical document handling**

The schema points out also the management of the approval flow for the analytical document passing from the developer to the tester, reaching the end user when certified only.

Please see in the next paragraphs how SpagoBI allows to run all these operations.








### 3 Recurring themes



Both the development and the administration interface are under revision to achieve a better usability.

#### 3.1 PORTLET LAYOUT

Every user portlet points out some common characteristics:

- On the top, there is the title identifying the portlet meaning.
- On the right side of the title, some icons allow the access to the general functions acting on the portlet's content. The main functions are (where admitted):
  -  going back to the previous page without saving changes; Every portlet
  -  creating a new element; Every portlet
  -  switching from the list view to the tree view; Document config.
  -  switching from the tree view to the list view; Document config.
  -  saving information; Details pages
  -  saving information and going back to the previous page; Details pages
  -  testing before saving. LOV details
- The '\*' character identifies the required fields.

#### 3.2 LIST AND DETAILED VIEW



One of the most common view in SpagoBI is a simple table showing a list of elements.



Common characteristics are:

- On the top, the title identifying the table meaning.
- The first row shows a label for each column displayed.
- The list can be divided into pages that can be turn over using the two arrows on the bottom row.
- The current page and the total number of pages are displayed in the middle of the bottom row.
- Every list has a detailed page showing and allowing to modify all the data about a single element.

Every list is alphabetically ordered on the first column's content (the label) and each row shows the essential data of an element, always identified by a unique label or title.

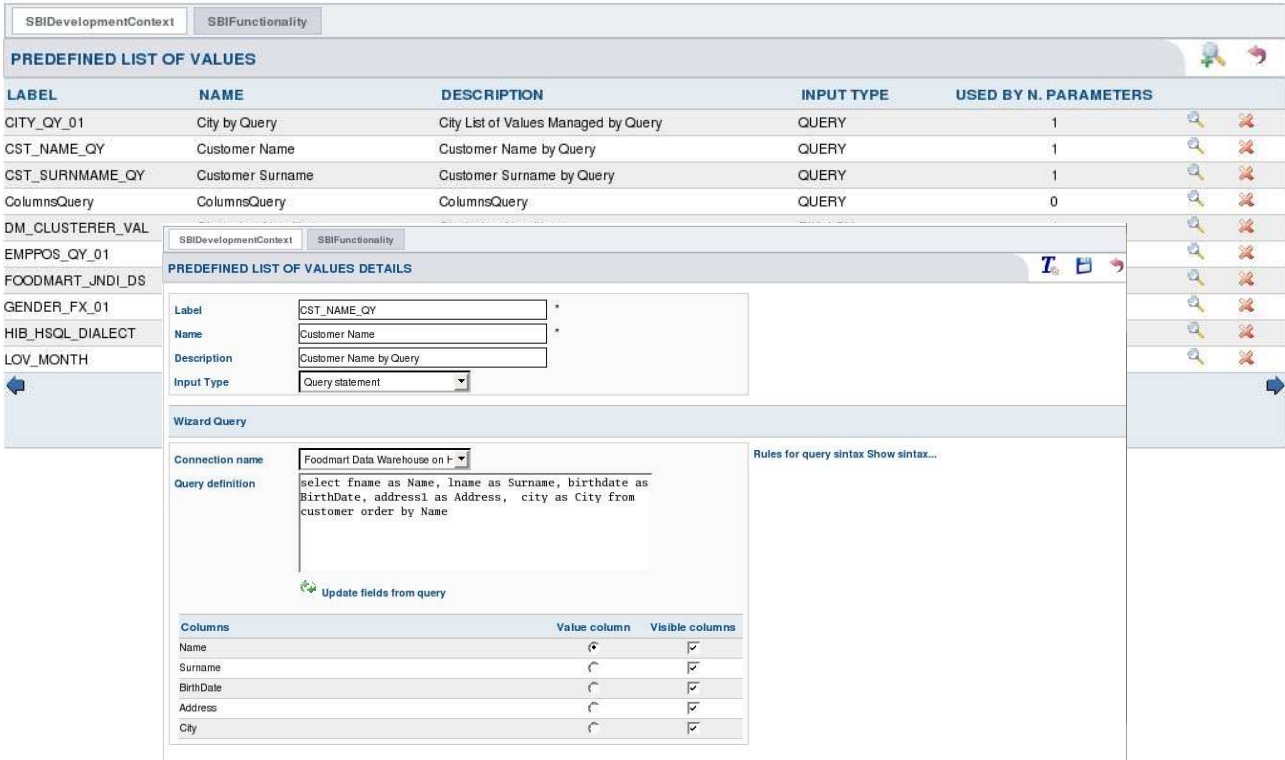
On the right side of every row, some icons drive the operativeness on the single element (row) of the list. The main possible functions are (where admitted):

-  accessing the details page for the selected element (row). Every list
-  deleting the corresponding element (row); Every list

-  executing the corresponding element (row);
-  Selecting all.

Analytical Doc. list only  
Tree management

A standard view of a list and detailed page follows.



The screenshot displays the SpagoBI interface. At the top, there are tabs for 'SBIDevelopmentContext' and 'SBIFunctionality'. Below them is a section titled 'PREDEFINED LIST OF VALUES' containing a table with columns: LABEL, NAME, DESCRIPTION, INPUT TYPE, and USED BY N. PARAMETERS. The table lists several predefined values, including 'CITY\_QY\_01', 'CST\_NAME\_QY', 'CST\_SURNAME\_QY', and 'ColumnsQuery'. The 'CST\_NAME\_QY' row is selected, and its details are shown in the 'PREDEFINED LIST OF VALUES DETAILS' section. This section includes fields for Label, Name, Description, and Input Type, all populated with values from the selected row. Below this is the 'Wizard Query' section, which shows the 'Connection name' as 'Foodmart Data Warehouse on h...' and the 'Query definition' as a SQL query: 'select fname as Name, lname as Surname, birthdate as BirthDate, address1 as Address, city as City from customer order by Name'. A table below the query definition shows the columns and their visibility status.

Columns	Value column	Visible columns
Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Surname	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
BirthDate	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Address	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
City	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 8 – List-detail Example

## 4 Functionality overview

### 4.1 SETTINGS

In SpagoBI demo the administrator (biadmin/biadmin user) has some portlets; one of them is the SbiSettings portlet which permits him to:

- register and configure each analytical engine inside the platform;
- synchronize SpagoBI roles with the portal roles;
- import/export documents.



If you do not have familiarity with SpagoBI yet, we suggest you to skip all this section (chapter 4.1) using the standard demo settings.



Figure 9 - Settings portlet

#### 4.1.1 ENGINES CONFIGURATION

**Engines** are external applications or internal SpagoBI classes delegated to display the final results of an analytical document. There are different engines to deal with different analytical areas (Report, OLAP, Data Mining and Dashboard). More than one engine could be available for the same area.

The SpagoBI administrator has to mark the attributes that are necessary for the correct use of an engine. By means of a correct configuration the user can both use the same instances of the same engine inside different environments (development, test, production), in order not to invalidate its performances, and to use different and parallel engines inside the same environments.

For the correct use of the engines it is very important to set the proper **driver**: this is a SpagoBI component delegated to configure the analytical document properly communicating to a specific engine. Therefore, they can be seen as an Adapter set between the analytical document and the specific engine.

A list of all the registered engines can be displayed accessing the *Engine Configuration*.

SBI Settings	SBI Administration Context	SBI Development Context	SBI Functionality
--------------	----------------------------	-------------------------	-------------------

ENGINES LIST				
LABEL	NAME	DESCRIPTION		
BIRT	Birt Engine	Birt Engine		
DASHBOARD_INT_ENG	Dashboard Internal Engine	Dashboard Internal Engine		
GEO	Geo Engine	Geo Engine		
OLAP-JP-MD-DEV	Jpivot-Mondrian Dev	Jpivot-Mondrian Olap Engine in Development Environment		
OLAP-JP-MD-REL	Jpivot-Mondrian Rel	Jpivot-Mondrian Olap Engine in Release Environment		
QBE_INT_ENG	Qbe Internal Engine	Qbe Internal Engine		
REP-JASP-DEV	Jasper Report Dev	Jasper Report Engine for static reporting in Development Environment		
REP-JASP-REL	Jasper Report Rel	Jasper Report Engine for static reporting in Release Environment		
WEKA	Weka Engine	Weka Engine		

page 1 of 1

The value of the column LABEL as a string starts with  [Filter All](#)

**Figure 10 – Engines list**

Each engine is depicted by a unique label, a name and a brief description. The administrator can create a new engine, erase an existing one or access their details page in order to change their configuration.

The information required for each engine are:



- **Label:** engine unique identifier;
- **Name:** engine name;
- **Description:** brief engine description (optional);
- **Document type:** the document type this engine will be applied to (Report/On-line analytical processing/Dashboard/Datamart Model/Data mining model/Map);
- **Engine type:** internal/external.

Engines can be internal or external (respect to the SpagoBI web application): in the former case you have to specify:

- **Class name:** name of the engine main class (it must implement the interface *it.eng.spagobi.engines.InternalEngineIFace*).

SBI Settings	SBI Administration Context	SBI Development Context	SBI Functionality
--------------	----------------------------	-------------------------	-------------------

ENGINE DETAILS			
Label	<input type="text" value="QBE_INT_ENG"/>	*	
Name	<input type="text" value="Qbe Internal Engine"/>	*	
Description	<input type="text" value="Qbe Internal Engine"/>		
Document type	<input type="text" value="Datamart Model"/>		
Engine type	<input type="text" value="Internal"/>		
Class	<input type="text" value="it.eng.spagobi.engines.datamart.SpagoBIQbeInternalE"/>	*	

**Figure 11 - Internal engine details**

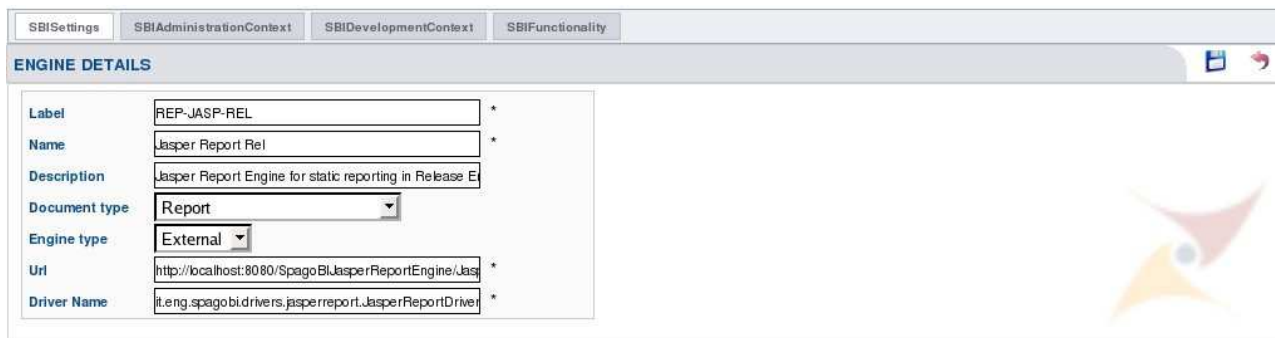
The current SpagoBI version uses the following internal engines:



- *Qbe Internal Engine*: for Datamart models;
- *Dahsboard Internal Engine*: for Dashboards.

It the latter case you have to specify:

- **URL:** location where the engine can be accessed by the server;
- **Driver Name:** class that creates an URL compliant with the specific engine.



The screenshot shows the 'ENGINE DETAILS' form in the SpagoBI Administration Context. The form includes the following fields:

- Label:** REP-JASP-REL
- Name:** Jasper Report Rel
- Description:** Jasper Report Engine for static reporting in Release E
- Document type:** Report
- Engine type:** External
- Url:** http://localhost:8080/SpagoBIJasperReportEngine/Jasp
- Driver Name:** it.eng.spago.bi.drivers.jasperreport.JasperReportDriver

**Figure 12 - External engine details**



The current SpagoBI version uses the following external engines:

- *JasperReport*: for Report analysis;
- *Birt*: for Report analysis;
- *Jpivot*: for OLAP analysis;
- *Weka*: for Data mining models;
- *Geo*: for Maps.











#### 4.1.2 ROLES SYNCHRONIZATION

A list of all SpagoBI's roles can be visualized by accessing to the *Roles Synchronization*. This area allows the administrator to synchronize portal roles with SpagoBI roles. Roles synchronization is made automatically by SpagoBI when the application starts, but it can be also forced by the administrator when the application has started.

Each role is described by two attributes:

- **Name:** the role's name;
- **Description:** the role's description.



Static Settings		Development Environment
ROLES LIST		 
NAME	DESCRIPTION	
/spagobi/admin	the /spagobi/admin group	
/spagobi/dev	the /spagobi/dev group	
/spagobi/share	the /spagobi/share group	
/spagobi/test	the /spagobi/test group	
/spagobi/user	the /spagobi/user group	
//spagobi/pippo	pippo	
page 1 of 1		 
The value of the column		NAME <input type="text"/> starts with <input type="text"/> <a href="#">Filter All</a>

**Figure 13 - Roles list**

The administrator can:

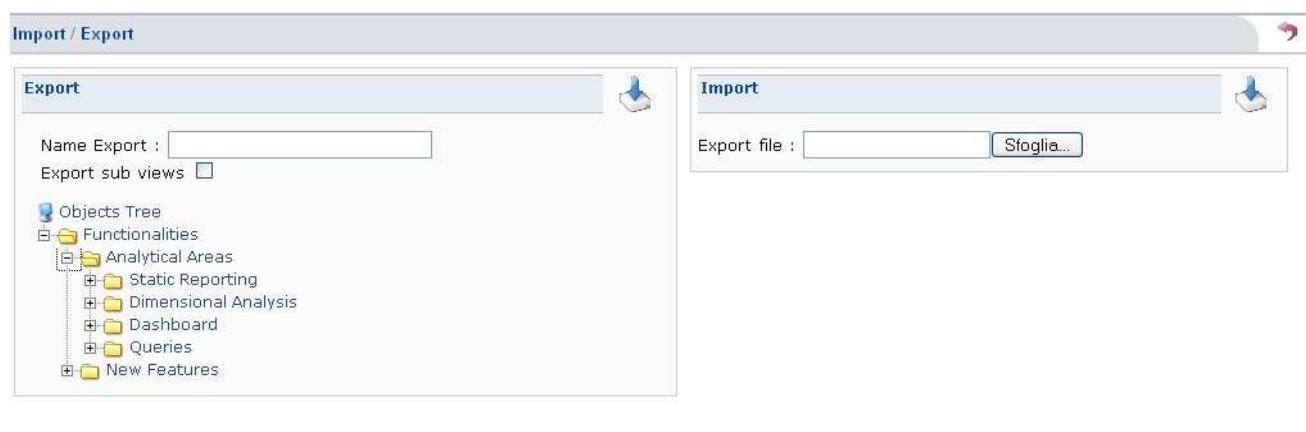
- erase an existing role (if that role is not used by any functionality or parameter use)
- synchronize SpagoBI's roles with portal's roles by clicking on the green two-arrows button in the Roles List page.

The roles list will be reloaded according to the roles erasing or synchronization.

### 4.1.3 IMPORT/EXPORT

The import/export feature allows you to export one or more business objects from a SpagoBI platform and to import them into a different SpagoBI Installation. There can be many different reasons for an export operation but some typical cases can be backups, upgrade to a new platform version and deploy of business documents from a development environment to a production environment.

It's possible to access to the import/export feature from the SpagoBI configuration tools (settings portlet) selecting the 'Import/Export' link. Once entered into the Import / Export section you can see the main page with the export form on the left side and the import one on the right side.



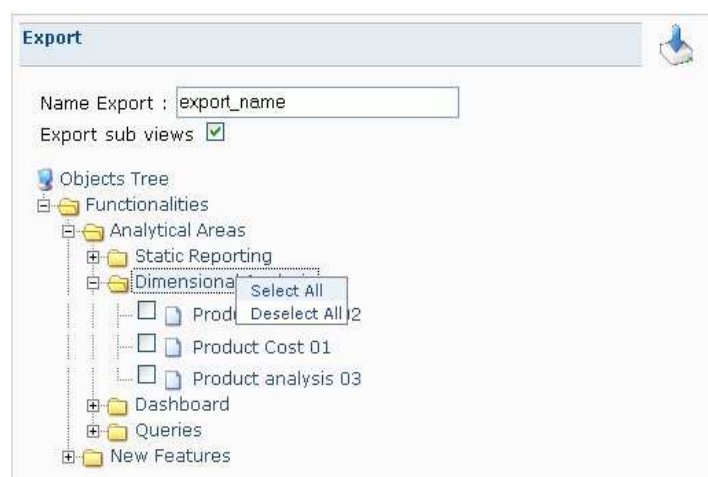
**Figure 14 - Import / Export**

#### 4.1.3.1 Export

The export form allows you to:

- give a name to the export archive, that will be created, setting the 'Name Export' field; (mandatory)
- export the sub views of the exported objects, checking the 'Export sub views' option
- select the business objects to export, opening the tree and checking the checkbox corresponding to each document.

Into the tree you can select the documents one by one, or you can click over the folder row and, within the contextual menu, you can choose to select / deselect all the documents contained.



**Figure 15 - Export documents selection**

Once selected the documents to export, you can click the export image on the right upper corner. The system starts to produce the export archive and an 'operation in progress' message will appear into the form. Once the archive production is terminated a 'download' link will appear. Click on it in order to download the export archive.



**Export**

Operation Complete [Download](#)

Name Export :

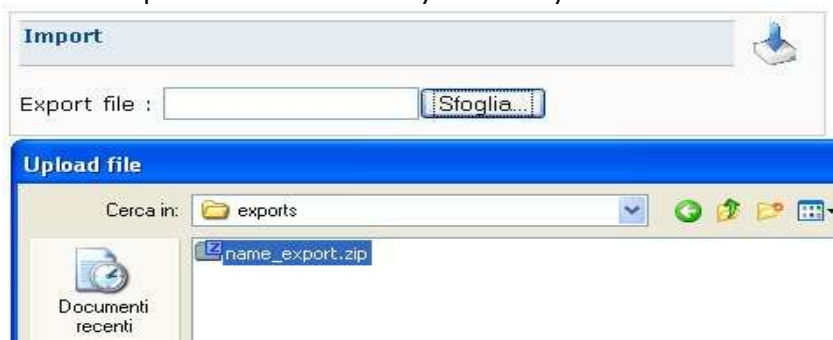
Export sub views ☐

**Figure 16 - Export operation complete**

Once selected the download link you get a zip file, with name equals to the one setted into the export form, which contains all the information useful to import the documents into a different platform.

#### 4.1.3.2 Import

The Import form is the start point of the import procedure. The 'Export File' selection field allows you to choose an exported archive from your file system.



**Import**

Export file :  [Sfoglia...](#)

**Upload file**

Cerca in:

Documenti recenti

name\_export.zip

**Figure 17 - Import archive selection**

Once selected the archive, you can click the import image on the right upper corner to start the import procedure. Since the the roles exported can be different form the ones of the current platform the next step require to map the roles.



**Role Associations**

Exported roles	Roles of the portal in use
/spagobi/admin	<input type="text" value="/spagobi/admin"/>
/spagobi/dev	<input type="text" value="/spagobi/dev"/>
/spagobi/test	<input type="text" value="/spagobi/test"/>
/spagobi/user	<input type="text" value="/spagobi/user"/>
/spagobi	<input type="text" value="/spagobi"/>

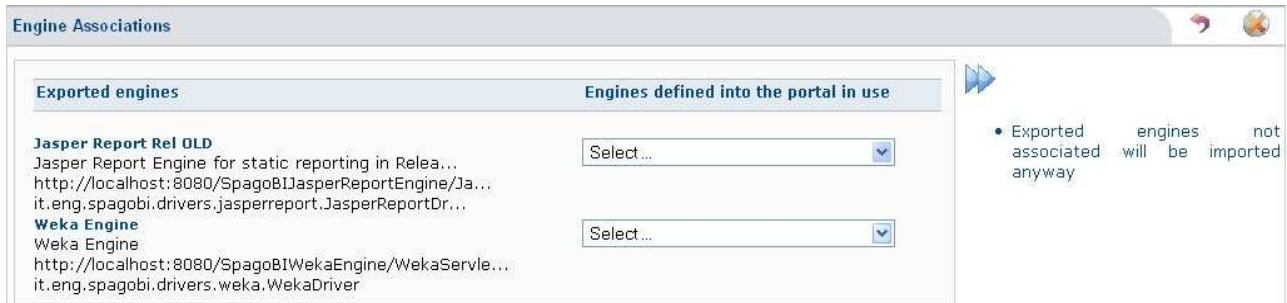
- Exported roles and current roles with the same name are associated automatically
- Exported roles not associated will be imported anyway
- Two or more exported roles cannot be associated to one current portal role

**Figure 18 - Import roles association**

The form shows a the complete list of exported roles (roles associated to the exported documents) and, for each one of them, there's a combo which allows you to select the corresponding role into the import system. If a role with the same name of the exported one already exists, then the combo is disabled and the mapping is automatic. Otherwise, if the

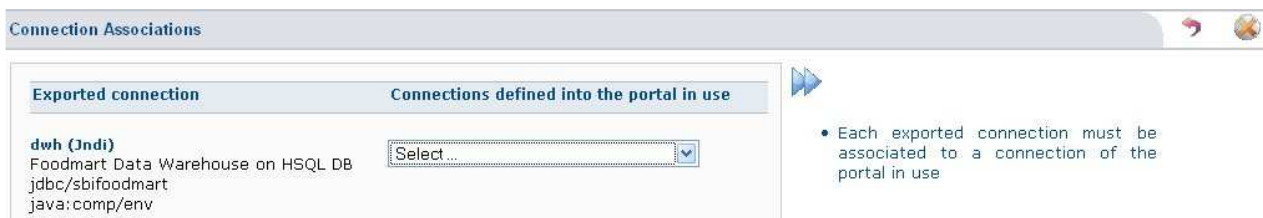
combo is selectable, you can choose the mapping role or you can let it blank, and, in this case, the exported role will be insert into the current system. Remember that each exported role can be associated only to one existing role.

Once complete the role mapping you can select the next button (the right arrows) which will allow you to perform the engine mapping. In fact, as for the roles, the exported engines should be different from the ones of the current system, and so, a mapping operation is necessary.



**Figure 19 - Import engines association**

The form shows the complete list of exported engines (engines associated to the exported documents) and, for each one of them, there's a combo which allows you to select the corresponding engine into the current system. If you don't provide an association the exported engine will be inserted anyway. Once complete the engine mapping you can select the next button (the right arrows), which will allow you to perform the connection mapping. This is the last mapping step and it's necessary because some of the exported list of values were associated to a database connection, which should be different from the ones of the current system



**Figure 20 - Import connections association**

Once again there is a description for each exported connection and a combo which allows you to associated a connection of the current system. This time, however, if you don't provide an association for each connection, the system shows you an error. In fact, it is not possible neither to insert an exported connection, nor to have some imported list of values connecting to an connection witch doesn't exist.. Once complete the connection mapping you can select the next button (the right arrows) which will complete the import operation.

During this latest phase the system checks if some exported objects conflicts with existing objects (for example if they have the same label or the same path). In case some conflicts are found, the sytem shows a complete list of all the conflicts. The user has the possibility to stop the import, in order to resolve the conflicts and perform again the export operation, or continue, pressing the next button. If the user continues, the system doesn't import the metadata which causes the conflict, but replace it with the existing one.


List of values		 <ul style="list-style-type: none"><li>• The system has found that some exported metadata conflict</li><li>• Exported metadata with the same unique identifier of the existing portal metadata cannot be inserted</li><li>• You can stop the import procedure and change the metadata identifiers which cause the conflict</li><li>• If you continue the system will not insert exported metadata that causes the conflict but will use the existing one</li></ul>
Exported list of values	List of values defined into the portal in use	
<b>FOODMART_JNDI_DS</b> Foodmart Jndi Datasource	<b>FOODMART_JNDI_DS</b> Foodmart_Jndi_Ds Foodmart Jndi Datasource	
<b>HIB_HSQL_DIALECT</b> Hibennate Hsqldb Dialect	<b>HIB_HSQL_DIALECT</b> Hib_Dialect_Hsql Hibennate Hsqldb Dialect	
Functionalities		
Exported functionalities	Functionalities defined into the portal in use	
<b>Queries</b> Queries Queries /Functionalities/Analytical Areas/Quer ...	<b>Queries</b> Queries Queries /Functionalities/Analytical Areas/Quer ...	

Figure 21 - Import conflicts

Once finished the import phase the user is redirect to the import / export main page. When displayed after an import operation the main page contains a 'Download log file' link, which allows to download a log file containing all the operation performed by the system.

Import

Export file :

Sfoggia...

Operation Complete [Download Log File](#)

Figure 22 - Import operation complete

## 4.2 ADMINISTRATOR

The SpagoBI demo administrator (biadmin/biadmin user) has also the functionalities and documents administration portlet which permits him to perform the following operations:

- Managing the functional structure that classifies the analytical documents and configuring permissions on the functionalities;
- Maintaining the registered analytical documents.



Notice that the administrator manages the **structural configuration** of the platform



Figure 23 – Administration portlet

#### 4.2.1 FUNCTIONALITIES MANAGEMENT

SpagoBI uses its own file system, named "**Functionalities Tree**", that allows to better organize documents by grouping them by folder: the access to those folders is regulated by roles permissions.

This multi-level hierarchical structure (Figure 24 – Functionalities Tree) can be created and modified exclusively by the administrator in the "Functionalities Management" area.



Notice that it is only possible to add a new folder to the root element, called *Functionalities*, but not to modify it.

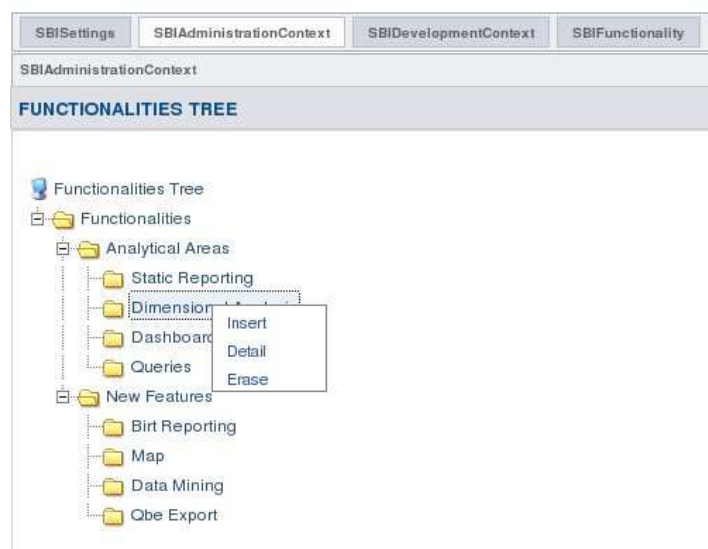


Figure 24 – Functionalities Tree

A list of possible actions can be visualized by clicking on a node of the *Functionalities Tree*. The administrator can **Delete** an existing functionality, if this doesn't contain any sub-nodes. Moreover, he can create a new functionality. By choosing the **Insert** option, he can access a new page where he can fill in all required information. This new element will be child of the selected one. Detailed information regarding an existing functionality can be displayed and modified by selecting the **Detail** option.



SBIAdministrationContext

**FUNCTIONALITY DETAILS**

Label: Static Reporting \*

Name: Static Reporting \*

Description: Static Reporting

Roles	Development	Test	Execution
/spagobi (spagobi root group)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/spagobi/admin (spagobi administrators)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/spagobi/dev (spagobi developers)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/spagobi/test (spagobi testers)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/spagobi/user (spagobi users)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 25 – Functionality details

Each folder is characterized by a name, a unique code and a optional brief description. The list of the *Roles* allows the administrator to choose, for the selected functionality, whether to assign or remove the permissions required for the development, test or execution phase for each role.

For instance, checking the Development and the Test boxes of the “SpagoBI developers group”, all users belonging to that group will only be able to develop and test documents contained into the selected functionality, but not to execute them when they are in Released state.

When we add or modify a functionality, all permissions that we can add or remove to each role follow a well defined policy. When we create a new functionality, by adding a new folder to the tree represented in the Figure 24 – Functionalities Tree, it inherits automatically all father’s permissions. All permissions that the father doesn’t have, *cannot be assigned neither to the children, nor to all the other descendants*.

According to this policy, if we add a child folder “Static Reporting Child 1” to “Static Reporting” folder, whose assigned permissions are represented in the Figure 25 – Functionality details, we have the following situation:

SBIAdministrationContext

**FUNCTIONALITY DETAILS**

Label: Static Reporting Child 1 \*

Name: Static Reporting Child 1 \*

Description:

Roles	Development	Test	Execution
/spagobi (spagobi root group)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/spagobi/admin (spagobi administrators)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/spagobi/dev (spagobi developers)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/spagobi/test (spagobi testers)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/spagobi/user (spagobi users)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 26 - Adding a new folder

As we can see, all Static Reporting functionality's roles are inherited, all the other ones are not assignable. Now, if we don't check any Execution permission for the Static Reporting Child 1, and after we add to it a new child functionality, for this one won't be able to select not only all permissions that we couldn't select for his father, but also the other two permission just revoked to the father itself.

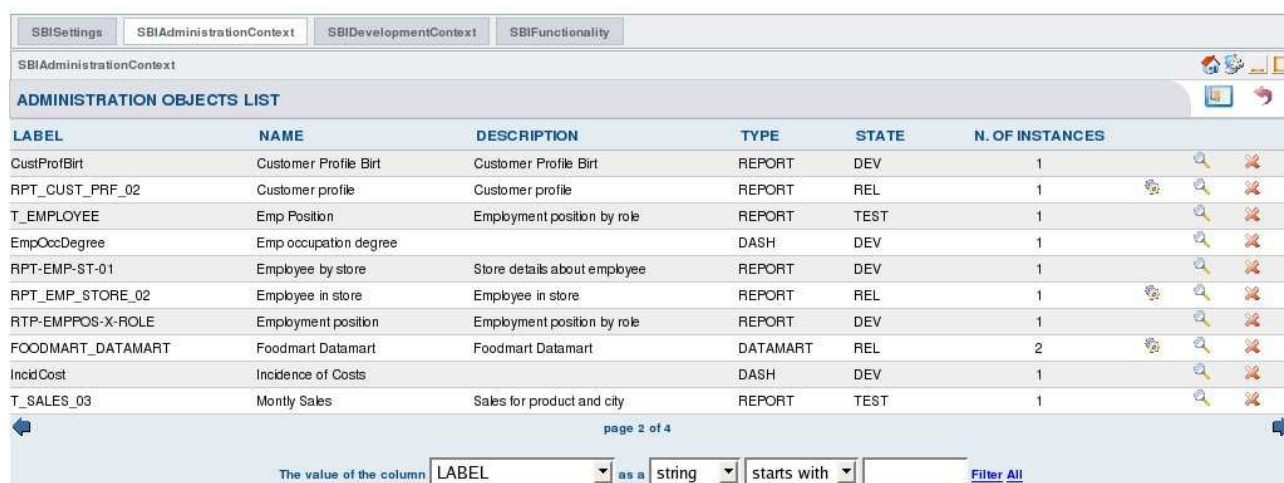
Finally, the permissions management is configured in a way that we can never assign to a child functionality a permission that his father doesn't have, and this behaviour transfers itself from the root to the leaves nodes of the functionalities tree.

## 4.2.2 DOCUMENT CONFIGURATION

A list of all the *Analytical Documents* registered in SpagoBI can be listed accessing to the *Documents Configuration* by the administrator. This area allows the administrator to manage the extraordinary maintenance of the documents.



Notice that the administrator is also a user and therefore he can execute all the released documents belonging to a folder on which he has the *Execution* permission.



The screenshot shows the 'ADMINISTRATION OBJECTS LIST' interface. It features a table with columns: LABEL, NAME, DESCRIPTION, TYPE, STATE, and N. OF INSTANCES. Below the table is a filter section with a dropdown for 'The value of the column' (set to LABEL), a dropdown for 'as a' (set to string), and a dropdown for 'starts with'. There is also a 'Filter All' button. The table contains 12 rows of data, including documents like 'CustProfBirt', 'RPT\_CUST\_PRFF\_02', 'T\_EMPLOYEE', etc.

LABEL	NAME	DESCRIPTION	TYPE	STATE	N. OF INSTANCES
CustProfBirt	Customer Profile Birt	Customer Profile Birt	REPORT	DEV	1
RPT_CUST_PRFF_02	Customer profile	Customer profile	REPORT	REL	1
T_EMPLOYEE	Emp Position	Employment position by role	REPORT	TEST	1
EmpOccDegree	Emp occupation degree		DASH	DEV	1
RPT-EMP-ST-01	Employee by store	Store details about employee	REPORT	DEV	1
RPT_EMP_STORE_02	Employee in store	Employee in store	REPORT	REL	1
RTP-EMPPOS-X-ROLE	Employment position	Employment position by role	REPORT	DEV	1
FOODMART_DATAMART	Foodmart Datamart	Foodmart Datamart	DATAMART	REL	2
IncidCost	Incidence of Costs		DASH	DEV	1
T_SALES_03	Monthly Sales	Sales for product and city	REPORT	TEST	1

Figure 27 – Administration Objects List

Each document is described by a subset of its attributes, as follows:

- **Label:** the document unique identifier;
- **Name:** the document name;
- **Description:** a brief description of the document (optional);
- **Type:** this field shows if the document is a Report, an On-line analytical processing (OLAP), a Data Mining model, a Dashboard, etc.;
- **State:** this information indicates if the document must be developed (*Development*), tested (*Test*) or can be executed (*Released*). Moreover the document can also be *Suspended* if it cannot be executed anymore;
- **N. of instances:** since one document can be referenced in more than one folder, this is the folders number that contain the document.

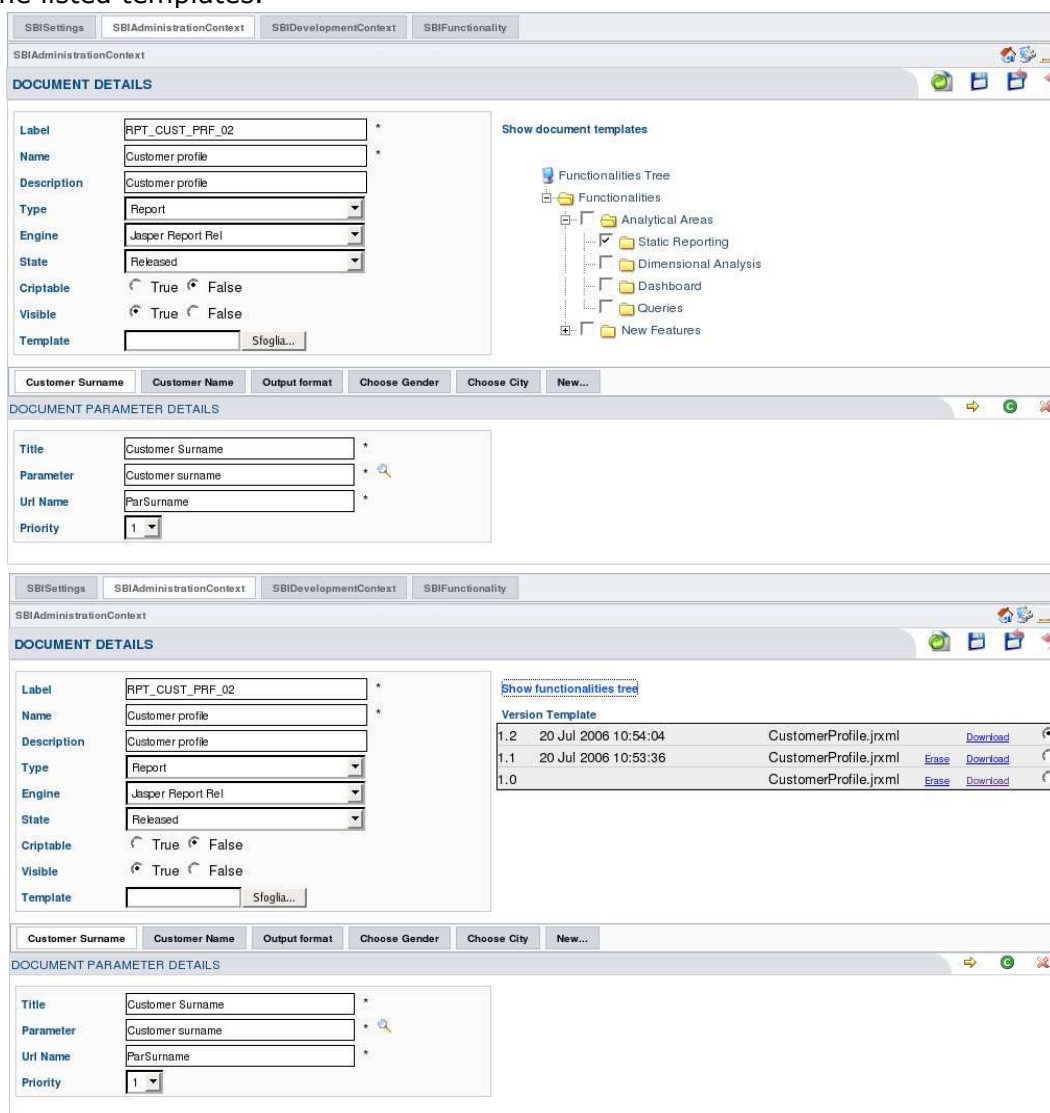
The administrator has the *List View* but he can switch to the *Tree View* in order to have a list of documents grouped by functionality, clicking on the VIEW AS TREE icon.



When he deletes a document from the *List View* he deletes all the document instances he can see, while when he deletes a document from the *Tree View* he deletes only the selected instance.

The complete list of information can be seen in the *Document Details* page:

- **Engine:** the name of a registered engine that has to be used to execute the *Analytical Document*;
- **Criptable:** a flag indicating id the document is criptable;
- **Visible:** a flag indicating if the document is visible when it is in Released state;
- **Template:** a file containing the model of the document to be created with an external application suitable for the specific type of the Analytical Document. On the right hand side of the page, in the "Template Version" table, all templates that have been selected for this document since it was created are listed. For each template, this list specifies the version identifier, the date when this selection occurred first and the name of the file. Through this view, the user will always be able to erase, download or select one of the listed templates.



The screenshot shows the SpagoBI Administration Object Details page. The page is divided into two main sections: 'DOCUMENT DETAILS' and 'DOCUMENT PARAMETER DETAILS'.

**DOCUMENT DETAILS:**

- Label:** RPT\_CUST\_PRFF\_02
- Name:** Customer profile
- Description:** Customer profile
- Type:** Report
- Engine:** Jasper Report Rel
- State:** Released
- Criptable:** True (selected), False
- Visible:** True (selected), False
- Template:** Sfoglia...

**DOCUMENT PARAMETER DETAILS:**

- Title:** Customer Surname
- Parameter:** Customer surname
- Url Name:** ParSurname
- Priority:** 1

**FUNCTIONALITIES TREE:**

- Functionalities
  - Analytical Areas
    - Static Reporting
    - Dimensional Analysis
    - Dashboard
    - Queries
    - New Features

**VERSION TEMPLATE:**

Version	Date	File Name	Actions
1.2	20 Jul 2006 10:54:04	CustomerProfile.jrxml	Download
1.1	20 Jul 2006 10:53:36	CustomerProfile.jrxml	Erase Download
1.0		CustomerProfile.jrxml	Erase Download

**Figure 28 – Administration Object Details**

In the lower part of the page you can see all the parameters applied to the document. In the *DOCUMENT PARAMETER DETAILS* section there are the following information:

- **Title:** the title of the document parameter;
- **Parameter:** the parameter that is applied to this document;
- **Url Name:** the key of the parameter: the URL for the document execution will contain an attribute with key specified by the content of this field and value specified by the parameter value;
- **Priority:** the document parameters are sorted by this number; clicking on the yellow arrows you can make a single step shift for the current document parameter.

You can switch from one parameter to another by clicking on the required element in the tab list.



The meaning of the field Parameter will be clarified in the next paragraph and then in the examples in chapter 5.

## 4.3 DEVELOPER



The developer's role is quite complex. This section explains its functionalities. In the chapter 5 you can find an example showing how the single functionalities work together for the right environment settings.

The developer (bidev/bidev user) main tasks are:

- to define the possible presentation and the preloading way (LOV – list of values) for the parameters;
- to define the validation rules (CHECK) for the input value;
- to create the parameters (PARAMETER) and to set up their behaviour rules associating LOV and CHECK to the user's roles;
- to register and to configure analytical documents, referring to the used parameters.



Figure 29 – Development portlet



Notice that the developer manages the **executive configuration** of the platform and only a few **structural configuration**.



Notice that at the start-up you have to create many LOV, CHECK and PARAMETER, but when the system runs regularly, you have to associate just the parameters already created to the new documents. You have to create new ones only for exceptions.

#### 4.3.1 PREDEFINED LIST OF VALUE (LOV)

From the *Developer Tools* view it is possible to access the list of the *Predefined List of Value* (LOV). It is described by some identifying data (label, name and description) and by its Input Type.

SBIDevelopmentContext

SBIFunctionality

PREDEFINED LIST OF VALUES

LABEL	NAME	DESCRIPTION	INPUT TYPE	USED BY N. PARAMETERS		
CITY_QY_01	City by Query	City List of Values Managed by Query	QUERY	1		
CST_NAME_QY	Customer Name	Customer Name by Query	QUERY	1		
CST_SURNAME_QY	Customer Surname	Customer Surname by Query	QUERY	1		
ColumnsQuery	ColumnsQuery	ColumnsQuery	QUERY	0		
DM_CLUSTERER_VAL	Clustering Algorithms	Clustering Algorithms	FIX_LOV	1		
EMPPOS_QY_01	Employment Position by Query	Employment Position Title Managed by Query	QUERY	1		
FOODMART_JNDI_DS	Foodmart_Jndi_Ds	Foodmart Jndi Datasource	SCRIPT	1		
GENDER_FX_01	Gender by Fixed Value	Gender values by predefined list	FIX_LOV	1		
HIB_HSQL_DIALECT	Hib_Dialect_Hsql	Hibennate Hsqldb Dialect	SCRIPT	1		
LOV_MONTH	Month list	Month list	QUERY	1		

page 1 of 4

The value of the column

LABEL

as a

string

starts with

Filter All

Figure 30 – Predefined List of Values

Accessing the details page, general information are displayed:

- **Label:** LOV unique identifier;
- **Name:** LOV name;
- **Description:** brief description of the LOV (optional);
- **Input Type:** three different types of LOV are admitted allowing:
  - **Query:** the database retrieval of all the selected values;
  - **Fixed LOV:** the organization of an arbitrary value list;
  - **Script:** the registration of a Groovy script delegated to the recovering of all the values to suggest.

SBIDevelopmentContext		SBIFunctionality	
PREDEFINED LIST OF VALUES DETAILS			
Label	CST_NAME_QY *		
Name	Customer Name *		
Description	Customer Name by Query		
Input Type	Query statement		
Wizard Query			
Connection name	Foodmart Data Warehouse on F		
Query definition	<pre>select fname as Name, lname as Surname, birthdate as BirthDate, address1 as Address, city as City from customer order by Name</pre>		
<a href="#">Update fields from query</a>			
Columns	Value column	Visible columns	
Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Surname	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
BirthDate	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Address	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
City	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

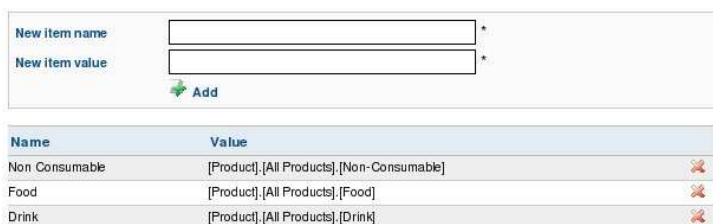
Figure 31 - Predefined List of Values Details

According to the selected typology, in the bottom of the page a wizard is available in order to ease its specific composition.

In the *Query Statement* case (the picture example above) the required information are:

- **Connection Name:** logic identifier of the database source;
- **Query Definition:** the SQL statement;
- **Value Column:** name of the unique column of the dataset containing values that will be return as a result of the parameter;
- **Visible Columns:** name of the columns of the dataset that will be displayed;

In the *Fixed Values* case, you can create a table of pairs (Name,Value) simply by filling the *New item name* and the *New item value* fields and clicking on the ADD icon. In order to erase an existing pair you can use the icon on the corresponding row of the list at the bottom of the window.



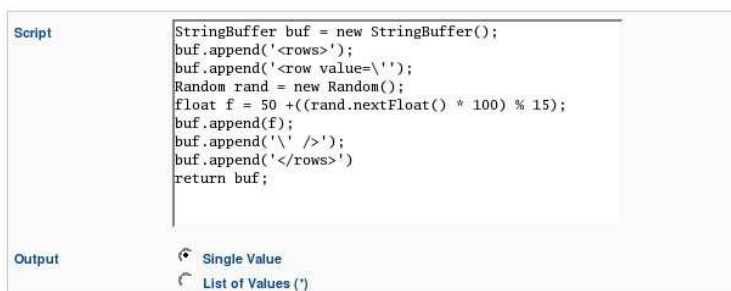
The interface shows two input fields: "New item name" and "New item value", each with a small asterisk icon to its right. Below these fields is a green "Add" button with a plus icon. At the bottom, there is a table with two columns: "Name" and "Value".

Name	Value
Non Consumable	[Product].[All Products].[Non-Consumable]
Food	[Product].[All Products].[Food]
Drink	[Product].[All Products].[Drink]

Each row in the table has a small red 'X' icon on the right side for deletion.

**Figure 32 – Fixed values wizard**

In the *Script* case you have to write the Groovy script to be executed at run time.



The interface is split into two sections. The top section, labeled "Script", contains a text area with the following Groovy code:

```
StringBuffer buf = new StringBuffer();
buf.append('<rows>');
buf.append('<row value=\'');
Random rand = new Random();
float f = 50 + ((rand.nextFloat() * 100) % 15);
buf.append(f);
buf.append('<\' />');
buf.append('</rows>');
return buf;
```

The bottom section, labeled "Output", has two radio buttons: "Single Value" (selected) and "List of Values (\*)".

**Figure 33 – Script wizard**

### 4.3.2 PREDEFINED VALUES CONSTRAINTS

The developer can register some typologies of formal controls applying to the values inserted in the documents activations phase.

CONFIGURABLE CONSTRAINTS			
LABEL	NAME	DESCRIPTION	Check type
CK-CUS-01	Range 10-20	Value between 10-20	RANGE
CK-CUS-02	2 decimals places	2 decimals places	DECIMALS
CK-CUS-03	Max lenght 20	Max lenght 20	MAXLENGTH
page 1 of 1			
The value of the column LABEL as a string starts with Filter All			
PREDEFINED CONSTRAINTS			
LABEL	NAME	DESCRIPTION	Check type
CK-FIX-01	Internet Address	Control if parameter is an Internet Address	INTERNET ADDRESS
CK-FIX-02	Numeric	Control if a parameter is Numeric	NUMERIC
CK-FIX-03	Alfanumeric	Control if a parameter is Alfanumeric	ALFANUMERIC
CK-FIX-04	Letter String	Control if a parameter is a letter string	LETTERSTRING
CK-FIX-05	Mandatory	Control if the parameter is present	MANDATORY
CK-FIX-06	Fiscal Code	Control if parameter is a Fiscal Code	FISCALCODE
CK-FIX-07	E-Mail	Control if parameter is a E-Mail	EMAIL
page 1 of 1			

**Figure 34 - Configurable Constraints and Predefined Constraints**

At the bottom of the displayed page there is a list of *Predefined Constraints* that cannot be modified. On the top of the page it is possible to create a set of *Configurable constraints*, simply clicking on the INSERT icon and entering the *Constraint Details* page. The developer can add new configurable controls, erase the existing ones or access their detailed information in order to modify them.

CONSTRAINT DETAILS	
Label	CK-CUS-02 *
Name	2 decimals places *
Description	2 decimals places
Check type	Decimal
Decimal Places	2

**Figure 35 - Constraint Details**

Every constraint is identified by a unique label, a name and a brief and optional description. In addition to some identification data (label, name and description) it is possible to configure some different control typologies:

- **date**: date format control;
- **regexp**: control through regular expressions;
- **min length**: minimum number of characters for the inserted values;
- **range**: to control a value included into two limits;
- **decimal**: decimal digits control;
- **max length**: maximum number of characters for the inserted values.

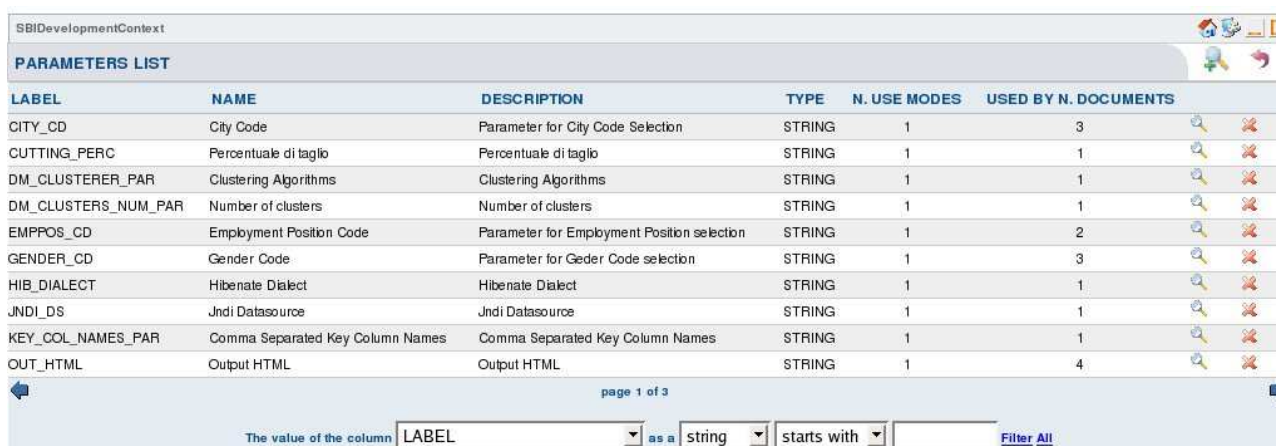
The developer can select a *Check Type* from the list and filling in the required values. Each constraint can have just one *Check Type*.



### 4.3.3 PARAMETERS MANAGEMENT

SpagoBI handles the parameters in term of autonomous entities, each one with its own behaviour based on users' roles. So it is possible to associate to them different presentation (LOV) and validation (CHECK) rules.

The list view allows the developer to add new parameters, to erase the existing ones or to access their detailed information in order to modify them.



The screenshot shows a web application window titled 'SBIDevelopmentContext' with a 'PARAMETERS LIST' tab. It contains a table with the following data:

LABEL	NAME	DESCRIPTION	TYPE	N. USE MODES	USED BY N. DOCUMENTS
CITY_CD	City Code	Parameter for City Code Selection	STRING	1	3
CUTTING_PERC	Percentuale di taglio	Percentuale di taglio	STRING	1	1
DM_CLUSTERER_PAR	Clustering Algorithms	Clustering Algorithms	STRING	1	1
DM_CLUSTERS_NUM_PAR	Number of clusters	Number of clusters	STRING	1	1
EMPROS_CD	Employment Position Code	Parameter for Employment Position selection	STRING	1	2
GENDER_CD	Gender Code	Parameter for Gender Code selection	STRING	1	3
HIB_DIALECT	Hibernate Dialect	Hibernate Dialect	STRING	1	1
JNDI_DS	Jndi Datasource	Jndi Datasource	STRING	1	1
KEY_COL_NAMES_PAR	Comma Separated Key Column Names	Comma Separated Key Column Names	STRING	1	1
OUT_HTML	Output HTML	Output HTML	STRING	1	4

Below the table, there is a pagination bar showing 'page 1 of 3' and a filter bar with the text 'The value of the column LABEL as a string starts with' and a 'Filter All' button.

Figure 36 – Parameters list

A parameter is identified by the following information:

- **Label:** unique identifier of the parameter;
- **Name:** name of the parameter;
- **Description:** brief description of the parameter;
- **Type:** define if this parameter is a date, a number or a string;

To insert a new parameter the developer has to fill the following form:

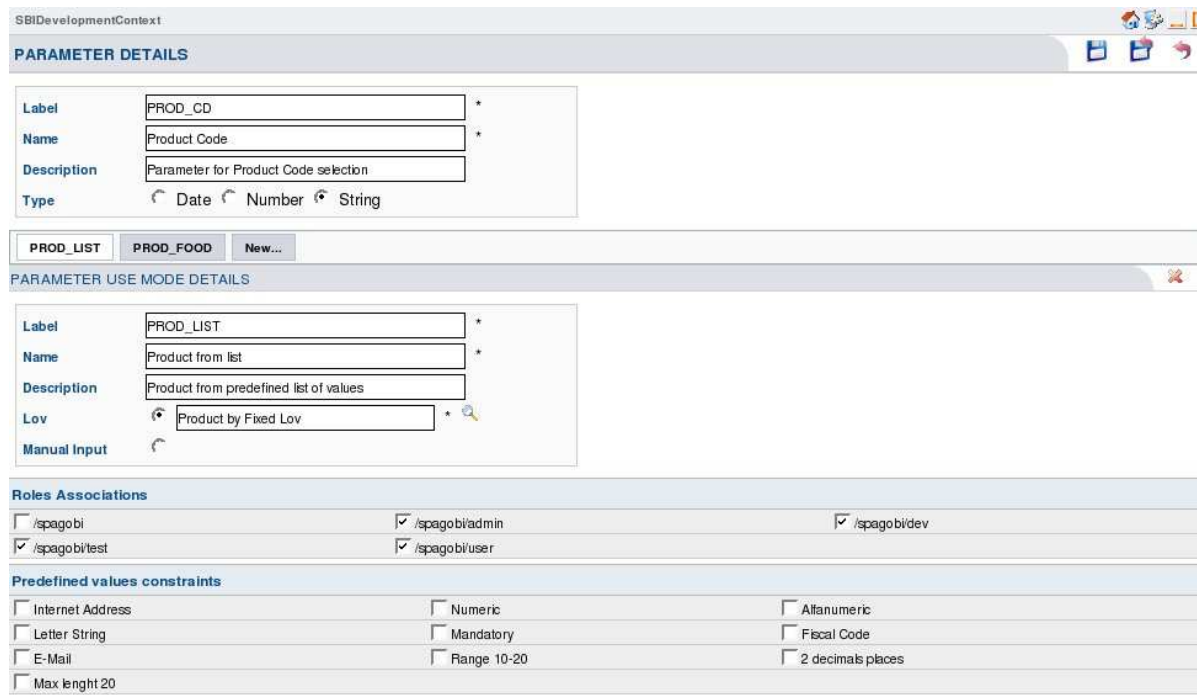


The screenshot shows a web application window titled 'SBIDevelopmentContext' with a 'PARAMETER DETAILS' tab. It contains a form with the following fields:

- Label:** A text input field with an asterisk (\*) indicating it is required.
- Name:** A text input field with an asterisk (\*) indicating it is required.
- Description:** A text input field.
- Type:** A radio button selection with three options: ☒ Date, ☐ Number, and ☐ String.

Figure 37 – New parameter form

When entering instead an existing parameter detail page he can see also the use modes of the parameter (the tab list in the following picture):



SBIDevelopmentContext

**PARAMETER DETAILS**

Label: PROD\_CD \*

Name: Product Code \*

Description: Parameter for Product Code selection

Type: ☐ Date ☐ Number ☒ String

PROD\_LIST | PROD\_FOOD | New...

**PARAMETER USE MODE DETAILS**

Label: PROD\_LIST \*

Name: Product from list \*

Description: Product from predefined list of values

Lov: ☒ Product by Fixed Lov \*

Manual Input: ☐

**Roles Associations**

<input type="checkbox"/> /spago bi	<input checked="" type="checkbox"/> /spago bi/admin	<input checked="" type="checkbox"/> /spago bi/dev
<input checked="" type="checkbox"/> /spago bi/test	<input checked="" type="checkbox"/> /spago bi/user	

**Predefined values constraints**

<input type="checkbox"/> Internet Address	<input type="checkbox"/> Numeric	<input type="checkbox"/> Alphanumeric
<input type="checkbox"/> Letter String	<input type="checkbox"/> Mandatory	<input type="checkbox"/> Fiscal Code
<input type="checkbox"/> E-Mail	<input type="checkbox"/> Range 10-20	<input type="checkbox"/> 2 decimals places
<input type="checkbox"/> Max lenght 20		

**Figure 38 – Parameter details**

As described before, each parameter can manage different user roles simply by assigning a specific way to collect data and specific constraints to validate the final input. This means that each role must be assigned to a specific *Use Mode*.

Each Use Mode requires the following information:

- **Label:** *Use Mode* unique identifiers;
- **Name:** *Use Mode* name;
- **Description:** brief description of the *Use Mode*;
- **Lov:** it is possible to select just one way to collect data from the list of available LOVs. Click on the LOV LOOKUP image and then select a Predefined List of Value from the lookup page (to create a new *List of Values* please refer to the Predefined List of Value paragraph); this is in alternative with the manual input;
- **Manual input:** as an alternative to the lov association, the modality can be manual input type, i.e. the parameter value will be filled manually by the user;
- **Roles Association:** list of the roles associated to this *Use Mode*. A role cannot be associated to more than one *Use Mode*; anyway a role may not be associated to any Parameter *Use Mode*. In the second case a user having just that role will not be able to execute a document using this Parameter;
- **Predefined Values Constraints:** it is possible to assign zero, one or more constraints selecting them from the list of available constraints.

#### 4.3.4 DOCUMENT CONFIGURATION

The user, simply by clicking on *Documents Configuration* from the *Developer Tools* page, can display the *Development Object List*. This page catalogues all the *Analytical Documents* with a *Development* state or with a *Released* state. They have to be contained in a folder for which the user has a role authorized respectively for *Development* and for *Execution*.





Please notice that every new document will have the Development state. For a better comprehension, please refer to the Analytical Document life-cycle section.

Each document is described by a unique label, a name, a description and a type (report, OLAP, etc.).



LABEL	NAME	DESCRIPTION	TYPE	STATE	N. OF INSTANCES
Best5ProdMonthSales	Best 5 Products for Monthly Sales Cols		DASH	DEV	1
TOP-PROD	Best Products	Best Products in Month	REPORT	DEV	1
BestProdBirt	Best Products Birt	Best Products Birt	REPORT	DEV	1
BestProdDailySales	Best Products for Daily Sales Lines		DASH	DEV	1
BestProdDailySaITab	Best Products for Daily Sales Tab		DASH	DEV	1
BestProdMonthSaITab	Best Products for Monthly Sales Tab		DASH	DEV	1
TOP-STORES	Best Stores	Best Stores in Month	REPORT	DEV	1
BudgSatisf	Budget satisfaction		DASH	DEV	1
ClusterDesc	Cluster Description	Cluster Description	REPORT	DEV	1
RPT-CUST-PRF-01	Customer Profile 01	Report about Customer details	REPORT	DEV	1

Figure 39 – Analytical documents list

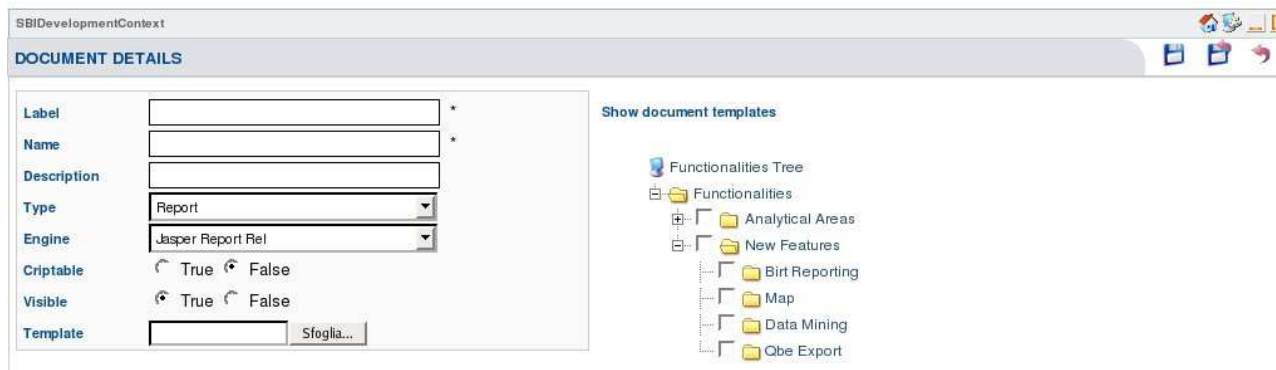
The *Documents Details* page displays a complete list of attributes:

- **Label:** document unique identifier;
- **Name:** document name;
- **Description:** brief description of the document (optional);
- **Type:** document type (report, OLAP, Data mining model, Dashboard, etc.);
- **Engine:** engine that will be used to execute the document. The available engines are registered by the administrator (see Engines Configuration paragraph);
- **Criptable:** flag indicating if the document is criptable;
- **Visible:** flag indicating if the document is visible when it is in Released state;
- **Template:** file containing the model of the document. It has to be created with an external application suitable for the specific type of Analytical Document.

When the developer inserts a new document he has to indicate the **Parent folder** under which the document will be created: this can be set by selecting the corresponding check box on the *Functionality Tree* displayed on the right hand side of the window.



The developer can see only the details of the documents that are in *Development* state, so the **State** field (visible by the administrator) is hidden.



**Figure 40 – Analytical document details (new document)**

Multiple selection of the parent folder is allowed.



When the developer sees an existing document and selects the DETAILS icon, additional elements will be displayed:

- **Detailed information** (Top-left side): already explained data; they are the same view of the document creation phase;
- **Version template list** (Top-right side, clicking on "Show document templates"): every time the developer uploads a new template it will be set as the current default and will be added to this list specifying the version identifier, the date when this selection occurred first and the file name. By means of this view the user will always be able to erase, download or select one of the listed templates;
- **Functionalities tree** (Top-right side, clicking on "Show functionalities tree"): the functionalities tree, where functionalities containing the current document are checked. All the functionalities on which the user has the development permission can be checked and unchecked;
- **Parameters tabs** (bottom side): there is one tab for each parameter associated to the Analytical document. An additional tab (*New ...*) creating a new association is provided.

In every parameter tab (for *Document Parameter Details*) the user is required to fill in the following information:

- **Title**: Document Parameter name;
- **Parameter**: clicking on the PARAMETERS LIST icon a lookup page for the available parameters list will be displayed;
- **URL Name**: parameter name on the document execution URL. This must match the corresponding parameter belonging to the template;
- **Priority**: the document parameters are sorted by this number; clicking on the yellow arrows you can make a single step shift for the current document parameter.

The developer can switch from one parameter to another by clicking on the required element in the tab list and can insert new parameters by clicking on the "New..." tab and fill in the form.

SBIDevelopmentContext

DOCUMENT DETAILS

Label

RPT-CUST-PRF-01

Name

Customer Profile 01

Description

Report about Customer details

Type

Report

Engine

Jasper Report Rel

Criptable

True False

Visible

True False

Template

Sfoglia...

Show document templates

Functionalities Tree

Functionalities

Analytical Areas

Static Reporting

Dimensional Analysis

Dashboard

Queries

New Features

Choose City

Customer surname

Customer name

Choose output format

Choose Gender

New...

DOCUMENT PARAMETER DETAILS

Title

Choose City

Parameter

City Code

Url Name

ParCity

Priority

1

SBIDevelopmentContext

DOCUMENT DETAILS

Label

RPT-CUST-PRF-01

Name

Customer Profile 01

Description

Report about Customer details

Type

Report

Engine

Jasper Report Rel

Criptable

True False

Visible

True False

Template

Sfoglia...

Show functionalities tree

Version Template

1.1

20 Jul 2006 19:05:19

CustomerProfile.jrxml

Download

1.0

CustomerProfile.jrxml

Erase Download

Choose City

Customer surname

Customer name

Choose output format

Choose Gender

New...

DOCUMENT PARAMETER DETAILS

Title

Choose City

Parameter

City Code

Url Name

ParCity

Priority

1

Figure 41 – Analytical document details (existing document)



Notice that during this phase it is not necessary to specify anymore about users' roles because they are completely managed through the visibility rules of the functional tree-view and through the behavioural parameters' description.

#### 4.3.4.1 Parameters correlation

Generally speaking two different parameters are saperated and indipendent entities. In the context of a document two different parameters can be correlated instead: this means that the

collection of the possible values of the former (the dependent parameter) is restricted by the value of the latter (the "father" parameter).

Since in SpagoBI the parameters can have different modalities, the actual correlation is between one parameter modality and the father parameter (no matter its own modalities, since only its final value is considered which is single).

Notice that the father parameter is only one: it is not possible to establish a correlation with more than one father parameter; it is possible instead to establish cascading correlations (one parameter is dependent by a father parameter which depends by a third parameter and so on).

The correlation is not possible obviously if the dependent modality is single value type; it must be a multi-value modality. At the time being only modalities associated to queries can be put in correlation, but in the future may be it will be possible to correlate all modalities that produce a list of values.

In order to configure a correlation, within a document detail page, click on the dependent parameter detail and then click on the "Correlation" button; in the following page you will see a list containing all the modalities that can be put into correlation (i.e. all the modalities of the parameter that are associated to a query).

In the first column of the table you can check the modalities that you want to be correlated, then: since the query associated to a modality has one or more columns, you can set the column to be filtered by the father parameter and the filter type.

At last you have to choose the father parameter: obviously the possible parameters are all the other document parameters.

Notice that if the father parameter is String type the filter will make a literal filtering, if the father parameter is Number type the filter will make a numerical filtering and if the father parameter is Date type the filter will make a calendar filtering: this means that the column to be filtered is assumed to be of the same type (String, Number or Date) of the value of the father parameter.

Example:

The parameter "Customer surname" is correlated with the "Choose City" as in figure.



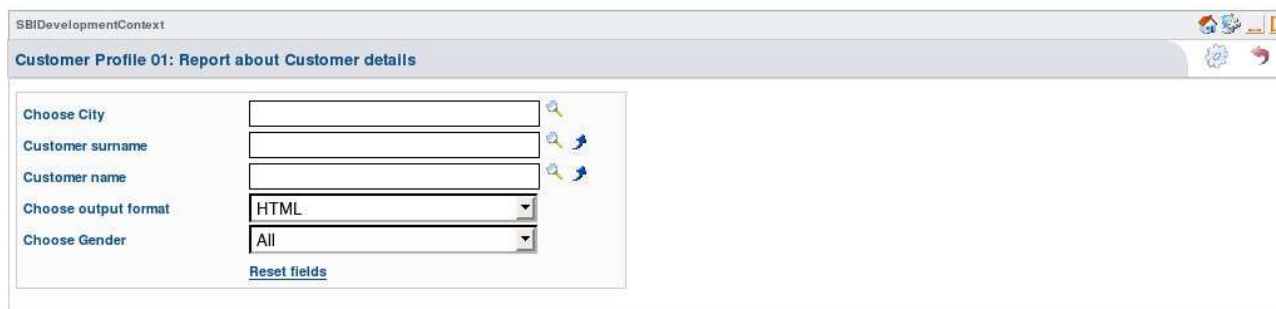
The screenshot shows a web browser window titled 'SBIDevelopmentContext'. Below the title bar is a header 'Correlation of parameter: Customer surname'. The main content area contains a table with the following structure:

Active correlations	Parameter modality	Filter	Filter column	Dependency parameter
<input checked="" type="checkbox"/>	MOD_CST_SURNAME: Modality for customer surname	=	City	Choose City

**Figure 42 - Correlation configuration example**

When the document is executed the possible values of the dependent parameter ("Customer surname") will be the customers living in the city specified by the father parameter ("Choose City").

The page for the document parameters selection remembers you all the existing correlations, as shown in the figure below (the father parameter is showed when the mouse is over the blue arrows).



**Figure 43 - Parameters selection page with correlated parameters**

In case the father parameter was not valued, the dependent parameter ignores the correlation and produces the complete collection of possible values.

#### 4.3.4.2 Links

In SpagoBI some documents can be correlated (linked): the kind of correlation depends on the type of document.

At the time being SpagoBI supports two types of links:

1. subreports (master – detail report documents using JasperReport engine);
2. Data mining model – presentation report document.

#### Subreports

Subreports are reports that can be imported as building blocks into other more complex reports. Subreports allow to modularize the process of report definition. Modularization of reports can be very useful to simplify report template file in particular in the case of reports that have a master-detail structure.

From now on we will refer to a report that uses subreports as a *master report*. The goal of this section is to explain how to insert a master report into SpagoBI environment and make it works. The definition of a master report's template is an advanced JasperReports functionality and goes beyond the scope of this short introduction. More infos on this topic can be found into the JasperReport User Guide.

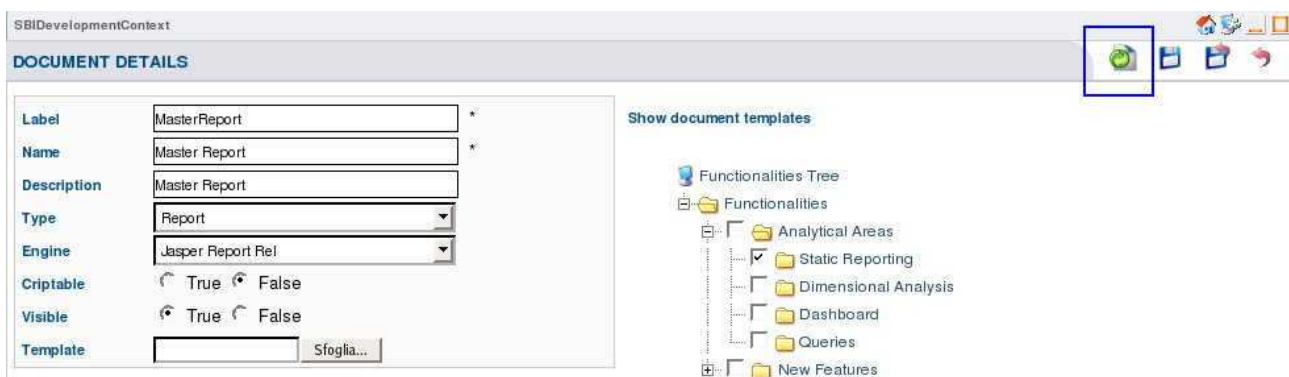
In order to insert a master report into SpagoBI environment you have to insert its subreports in the system first. A subreport it's just a normal report included in some other reports. There is no difference between the insertion procedure of a subreport and the insertion procedure of a normal report just because there is no difference between subreports and normal reports (i.e every report can be used as subreport).

Once all the subreports used by the master report you want to use have been inserted into the system you can also insert the master report. In order to execute the master report at the end of the insertion procedure without any errors it is important to verify that its template file points to subreports included just by name (i.e. *subreport1.jasper*) and not by relative or absolute path (i.e. */myreports/subreport1.jasper*).

```
<subreport isusingCache="true">
  <reportElement
    mode="Opaque"
    x="0"
    y="48"
    width="782"
    height="20"
    forecolor="#000000"
    backcolor="#FFFFFF"
    key="subreport-1"
    stretchType="NoStretch"
    positionType="FixRelativeToTop"
    isPrintRepeatedValues="true"
    isRemoveLineWhenBlank="false"
    isPrintInFirstWholeBand="false"
    isPrintWhenDetailOverflows="false"/>
  <subreportParameter name="Year">
    <subreportParameterExpression><![CDATA[${P{Year}}]></subreportParameterExpression>
  </subreportParameter>
  <subreportParameter name="MonthName">
    <subreportParameterExpression><![CDATA[${F{MONTHNAME}}]></subreportParameterExpression>
  </subreportParameter>
  <connectionExpression><![CDATA[${P{REPORT_CONNECTION}}]></connectionExpression>
  <subreportExpression class="java.lang.String"><![CDATA["subreport.jasper"]]></subreportExpression>
</subreport>
```

**Figure 44 - Subreport definition in master report template**

Once you have checked this you can proceed with the insertion procedure of the master report. Also this procedure is very similar to the standard insertion procedure of a normal report. The only difference is that you have to specify which reports, among already inserted ones, your master report will include. In order to specify this list of subreports you have to click on the subreport button in the document detail page (Figure 45 - Links configuration button) and then check from the list shown the report included by your master report (Figure 46 - Links list).



**Figure 45 - Links configuration button**



SBIDevelopmentContext

Check-List

Label	Name	Description	
SALES-X-CITY	Sales by city 01	Product sales for city	<input type="checkbox"/>
RPT_EMPPOS_FX	Single Employee	Single Employee	<input type="checkbox"/>
StorePerfTrendLines	Store Performance Trend Lines		<input type="checkbox"/>
StoreEffSmall	Stores efficiency		<input type="checkbox"/>
StoreEff	Stores efficiency		<input type="checkbox"/>
SubReport	SubReport	SubReport	<input checked="" type="checkbox"/>
TotalAmountDep	Total Amount Dep	Total Amount Department	<input type="checkbox"/>
Dmp_user_seg	User Segmentation	User Segmentation	<input type="checkbox"/>
VeniceMap	VeniceMap	VeniceMap	<input type="checkbox"/>
colquery	colquery	colquery	<input type="checkbox"/>

page 3 of 3

Figure 46 - Links list

## Data mining model – presentation report document

A data mining model in SpagoBI is actually an asynchronous process that produces some results. The presentation of the results is delegated to another report document (one or more) that must be executed only when data mining process is completed.

The result presentation report must be configured as a link on the data mining document configuration page. The user has to click on the "Links" button and then check the report document.

### 4.3.4.3 Documents visibility

Checking visible properties into the document detail page (Figure 47 - Document visibility field) to false make the document not visible in the object tree view when the document itself passes from the developing state to the testing or releasing state. This option is useful when a document is used always as a building block in the definition of more complex documents and it has no sense to show it by its own (i.e. Subreport).

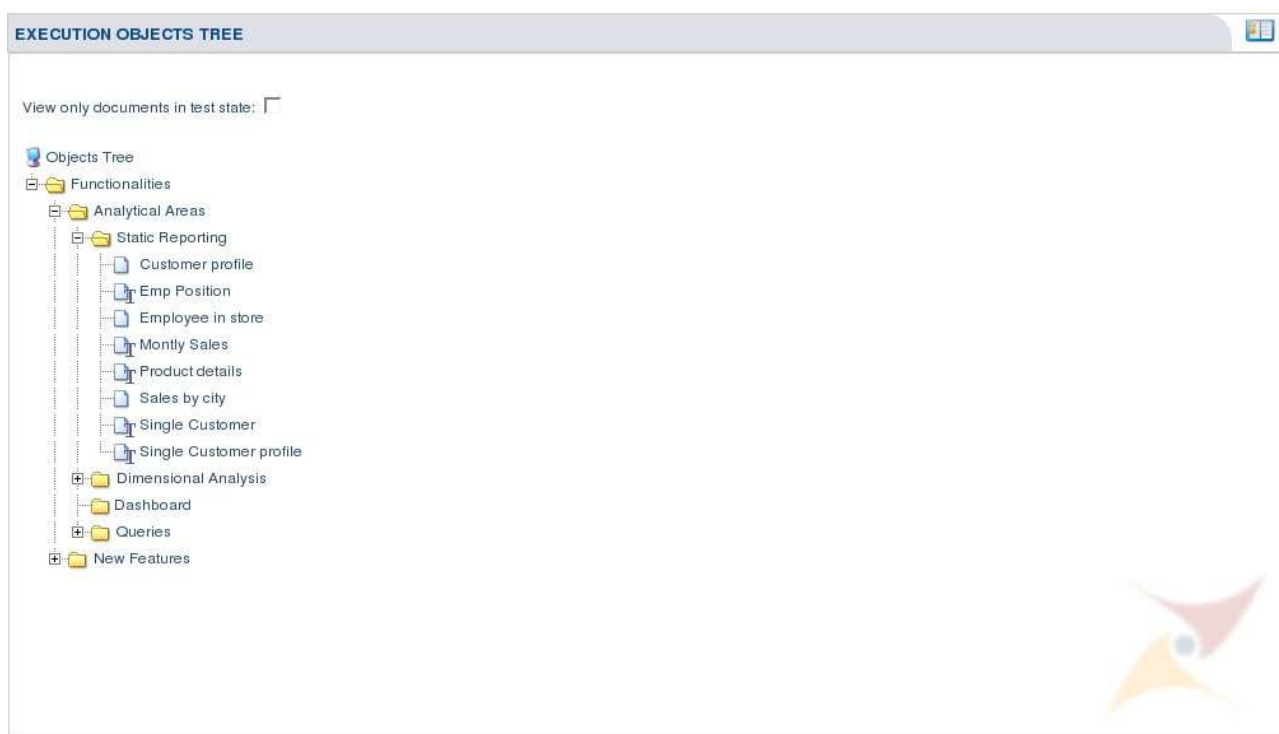
Label	MasterReport *
Name	Master Report *
Description	Master Report
Type	Report
Engine	Jasper Report Rel
Criptable	<input type="radio"/> True <input checked="" type="radio"/> False
Visible	<input checked="" type="radio"/> True <input type="radio"/> False
Template	<input type="text"/> Sfoglia...

Figure 47 - Document visibility field

## 4.4 TESTER

The tester (bittest/bittest user) main tasks are:

- validating the produced Analytical Document to simulate all its predefined roles;
- updating the Document state to release the documents that becomes available for the end-user.



**Figure 48 – Tester’s documents tree**

This type of user must verify the formal correctness of the registered documents and check if the documents in a test state works correctly and if they fulfil the requirements.

By means of the functional tree-view, this page lists all documents having *Test* or *Released* as current state (a document in *Test* state shows a “T” in its icon). They belong to a folder for which the user has at least one role with a required permission respectively for Test and Execution.



For a better understanding of the Security Policy please refer to the Document organization and to the Security Policy paragraph.

The Tester can only execute a specific document by selecting it from the *Functionalities Tree* or from the correspondent list view.



EXECUTION OBJECTS LIST					
LABEL	NAME	DESCRIPTION	TYPE	STATE	N. OF INSTANCES
RPT_CUST_PRJ_02	Customer profile	Customer profile	REPORT	REL	1
T_EMPLOYEE	Emp Position	Employment position by role	REPORT	TEST	1
RPT_EMP_STORE_02	Employee in store	Employee in store	REPORT	REL	1
FOODMART_DATAMART	Foodmart Datamart	Foodmart Datamart	DATAMART	REL	1
T_SALES_03	Monthly Sales	Sales for product and city	REPORT	TEST	1
OLAP-PROD-02	Product Analysis 02	Online analysis about product sales	OLAP	REL	1
OLAP-PROD-03	Product analysis 03	Online analysis about product sales	OLAP	TEST	1
T_PROD_PRJ	Product details	Product profile form name	REPORT	TEST	1
RPT_SALES_CITY_02	Sales by city	Month sales for product and city	REPORT	REL	1
RPT_CUST_PRJ_FX	Single Customer	Single Customer	REPORT	TEST	1

page 1 of 2

The value of the column LABEL as a string starts with Filter All

Figure 49 – Tester's documents list

If the selected document is in a *Test* state with a parameter associated to different *Use Modes*, the user has to choose a role from the *Role* combo box in the *Select Role For Execution* page.

SELECT ROLE FOR EXECUTION

Role : /spagobi/admin

Figure 50 – Role selection page

Notice that the list includes all the system roles available for the document and not only the ones belonging to the *Tester*. This means that the *Tester* will be able to test the different behaviour of the *Analytical Document* in relation to every user's role. No matter whether he owns the role or not. Instead, if the document is in *Released* state, the list includes only his own roles (in case he is associated with more than one role).

By clicking on the **EXECUTION** icon, the *Analytical Document* will be run and, if necessary, a page for the input parameter will be displayed allowing the user to insert the required information (from and according to the *Analytical Document* configuration).

Employee in store: Employee in store

Choose store:

Output Format: HTML

Position title:

[Reset fields](#)

Figure 51 – Parameters page

The page is under a revision phase.



When a parameter is defined as a *Query* type, the corresponding input field becomes a *Lookup* table where you can choose your value.

Stores List					
store_name	store_type	store_city	store_state	store_country	
HQ	HeadQuarters	Alameda	CA	USA	✓
Store 1	Supermarket	Acapulco	Guerrero	Mexico	✓
Store 10	Supermarket	Orizaba	Veracruz	Mexico	✓
Store 11	Supermarket	Portland	OR	USA	✓
Store 12	Deluxe Supermarket	Hidalgo	Zacatecas	Mexico	✓
Store 13	Deluxe Supermarket	Salem	OR	USA	✓
Store 14	Small Grocery	San Francisco	CA	USA	✓
Store 15	Supermarket	Seattle	WA	USA	✓
Store 16	Supermarket	Spokane	WA	USA	✓
Store 17	Deluxe Supermarket	Tacoma	WA	USA	✓

page 1 of 3

The value of the column  as a  starts with  [Filter All](#)

**Figure 52 – Lookup for parameter input value**

The list pages can be turn over using the two arrows on the bottom row. The desired value can be selected by clicking on the corresponding icon. Moreover, to ease the user to find the required data, it is possible to filter the list.

When the parameter selection is completed the user can execute the document by clicking on the icon.

Customer profile: Customer profile				
Customer profile				
Name	Albright	Anna	Gender	M
City	Fremont	Postal Code	66160	
Country	USA	Address	5927 Mt. Hood Circle	
Education	High School	Occupation	Skilled Manual	Income \$30K - \$50K

Month	St.Sales	Unit Sales	St.Costs
March	21,76	10	5,24
April	3,78	1	1,66
May	53,15	29	14,33
June	18,34	8	5,00
July	18,23	8	7,11
August	39,53	22	13,62

Month	St.Sales	Unit Sales	St.Costs
September	16,06	5	5,64
October	9,22	3	3,25
December	68,96	29	17,16

**Figure 53 – Final execution**

Now the test user can update the document state to *Release*, if all the required tests worked correctly; otherwise to *Development*.

Moreover, he can click on the BACK icon in order to execute a different test using a different role.

## 4.5 END-USER

The end-user (biuser/biuser user) works with the **Analytical Portal** made by the Business Intelligence designer and developer.

The on-line demo is an example of an analytical portal whose purpose is to let you see the basic elements for the building of your analytical portal.

For example, from the menu of the Demo portal, you can access many sections:

- **Home:** many portlets which combine punctual views of the performance indicators with synthetic reports.
- **Navigation:** a free navigation on the functional tree of the documents.
- **Instanced Reports:** a page divided into subsections for the presentation of single reports already instanced (not parametrics).
- **Parametric Reports:** a page divided into subsections for the parametric activation of the reports with default values.
- **Olap:** a multidimensional analysis model.
- **Dashboard:** a synthetic, static and dynamic presentation of historical and current series with especially interests.
- **Dynamic Dashboard:** a synthetic and dynamic presentation of the performance indicators values to be monitored at fixed time intervals.
- **QbE** (Query by Example): the module for a free and visual inquiry of the predefined data items.
- **Manual:** the user manual.

Even if the navigation portlet can run all the visible documents, the portal can be composed by several pages and sections: every portlet addresses a specific document, for a free composition of the informative scene and for an immediate view of the particular business context.

Every user can use the *Released* document according to his role's visibility. When the user owns different roles and the documents have different behaviours, the role for which the document has to be executed is required.

The end user has different freedom degrees of movement and of personalization of the analysis. The modules which allow the greater freedom degree are:

- **QbE:** the user can produce and save in the repository its own interrogations;
- **OLAP:** the user can freely reorient his data model saving his more interesting view.

---

To be continued.



## 5 Getting started with SpagoBI

This chapter explains, using some examples, the single steps allowing you to enrich the demo portal with new analytical documents, by means of the SpagoBI functionalities.

First of all you have to install ExoTomcat and the SpagoBI DEMO.  
How to handle the different categories of analytical documents follows.

---

This chapter is in revision phase.



---

### 5.1 INSTALL EXOTOMCAT AND SPAGOBIDEMO

Download ExoTomcat 1.1.3 from :

[http://forge.objectweb.org/project/showfiles.php?group\\_id=151&release\\_id=1378](http://forge.objectweb.org/project/showfiles.php?group_id=151&release_id=1378)

Unzip the file exoPortal-tomcat-1.1.3.zip but do NOT run the application, because this would cause the failure of the following SpagoBI installation.

Download SpagoBI Demo 1.8 from:

[http://forge.objectweb.org/project/showfiles.php?group\\_id=204&release\\_id=1407](http://forge.objectweb.org/project/showfiles.php?group_id=204&release_id=1407)

Unzip the file.

At the command shell navigate to the directory that contains the jar file obtained and type:

```
java -jar SpagoBIDemoInstaller.jar
```

Follow the instructions to complete the installation process. Notice that it will be required to specify the directory where you have previously unzipped ExoTomcat 1.1.3.

Once completed SpagoBI installation, at the command shell it is necessary to navigate to the temp/data/databases directory of Exo-Tomcat and type start.bat for Windows or ./start.sh for Unix: remember to check if you have the required permission for execution.

Then, at the command shell navigate to the /bin directory of Exo-Tomcat and type startup.bat for Windows or ./startup.sh for Unix.

You can read the log file by typing:

```
tail -f ../logs/catalina.out
```

### 5.2 REPORT

The main steps to manage a report are:

1. Create a Report Template
2. Create Parameters
  - a. Create Lists of Value

- b. Create Constraints
3. Register the Analytical Document (the built report) into the platform
  - a. Add Template
  - b. Assign Parameters
4. Test the Analytical Document
5. Execute the Analytical Document



Notice that JasperReport is the first report engine chosen, but it is not the only one allowed. Similarly iReport is the first chosen interface for JasperReport engine but other solutions also exists and the developer can use what he prefers in order to produce the report template.

## 5.2.1 CREATE A REPORT TEMPLATE

### 5.2.1.1 Create a Report template using iReport

1. If you don't have iReport 1.2.5 you can download it from:  
[http://sourceforge.net/project/showfiles.php?group\\_id=64348](http://sourceforge.net/project/showfiles.php?group_id=64348)
2. Unzip the downloaded file.
3. Before starting iReport it is necessary to copy the file *hsqldb1\_8\_0\_2.jar*, that can be found in the *common/lib* directory of Exo-Tomcat, in the */lib* directory of iReport. Furthermore delete the *hsqldb1\_61.jar* file.
4. Now it's possible to run iReport.
5. In order to create a simple report example, select **New Document** from the **File** menu. Type *SpagoBI\_Example* as Report name and click on **OK**.  
From the **Datasource** menu select **Connection/Datasources**.  
Click on **New** and fill in the following information:
  - **Name:** SpagoBI\_foodmart
  - **JDBC Driver:** org.hsqldb.jdbcDriver
  - **JDBC URL:** jdbc:hsqldb:hsqldb://localhost/foodmart
  - **Username:** sa

Select **Save**.

6. Open the **Report query** window from the **Datasource** menu and in the **Report SQL query** text field enter the following example query:

```
select FIRST_NAME, LAST_NAME
from EMPLOYEE e, POSITION p
where p.POSITION_ID = e.POSITION_ID
and p.POSITION_TITLE='$P{EmployeePosition}'
```

This simple query will visualize the First Name and the Last Name of every employee whose position title is equal to the value of the parameter *EmployeePosition*.

7. From the **Project Browser** on the left hand side of the window, expands **Object library**.

Right-click on the Project Browser and click on **Add** and then on **Parameter**.

Type *EmployeePosition* in the **Parameter name** field, selecting the **Is for prompting** check box and then select **OK**.

8. It is now possible to create the report layout.

Select the **"T"** icon from the top toolbar and insert a new text field in the **Detail** area of the report. Double-clicking on the new field and in the **Static Text**, type *First Name*.

Exit the properties dialog box and create another text field positioning in the **Detail** area below the first one.

Double-click on the **Fields** element in the **Project Browser** and drag the *FIRST\_NAME* field next to the corresponding static text field just created. Repeat the same action for the *LAST\_NAME* field.

This report will simply display a list of all employees obtained by the query.

9. Now it is possible to **execute the report** by selecting the corresponding command from the **Build** menu.

Before displaying the final result, the application will ask you to enter the *EmployeePosition* parameter value. You can type: "President".

### 5.2.1.2 Create a Report template using Birt

1. If you don't have Birt 2.0.1 you can download it from:

<http://download3.eclipse.org/birt/downloads/>

We suggest to download the all-in-one package you find under "Report Designer Full Eclipse Install" section.

2. Before starting eclipse it is suggested (but it is not mandatory) to download the Eclipse Birt plugin you find in

[http://forge.objectweb.org/project/showfiles.php?group\\_id=204](http://forge.objectweb.org/project/showfiles.php?group_id=204)

This Eclipse-Birt plugin will permit SpagoBIBirtReportEngine to use a given JDBC connection instead of retrieving one by the ODA framework; in the former case the connection can be set as a document parameter and can be changed at runtime (changing the parameter value) without editing the document template; in the latter case the parameter for the connection to be used will be completely useless and the only way to change connection will be editing the document template, changing connection configuration and uploading the modified template.

Unzip the file *EclipseBirtPlugins-bin-1.9RC.zip* inside the Eclipse home directory;

3. Then start the Foodmart HSQL db and copy the database driver *hsqldb1\_8\_0\_2.jar* inside the following directory:

*ECLIPSE-HOME\plugins\org.eclipse.birt.report.data.oda.jdbc\_2.0.1\drivers.*

4. Now you can start Eclipse-Birt.

5. In order to create a simple report example, select New Project → Business Intelligence and Reporting Tools → Report Project from the File menu. Then select New Report from the same menu. Type *SpagoBI\_Example* as Report name and click "Finish".



6. From the Data Explorer view right click on Data Sources and edit a new Data Source. If you have installed the Eclipse Birt plugin as described at point 2, choose "External (or standard) JDBC Data Source" as data source type. Please consider that this plugin will permit to use only one JDBC connection so you cannot use this plugin for more than one data sources. If you need to use more than one data sources or if you haven't installed the Eclipse Birt plugin, you can configure them as standard "JDBC Data Source" (ODA framework will be activated). Enter "foodmart" as Data Source name and click "Next". Then fill in the following information:

- **JDBC Driver:** org.hsqldb.jdbcDriver (if you don't see this driver from the list go to "Manage drivers" and add the driver for HSQL selecting the jar *hsqldb1\_8\_0\_2.jar* copied at point 3)
- **JDBC URL:** jdbc:hsqldb:hsq://localhost/foodmart
- **Username:** sa

Click "Finish".

7. Then right-click on Data Sets and add a new Data Set with the following information:

- **Data Set name:** Employees
- **Data Source:** Foodmart
- **Data Set Type:** SQL Select Query

8. In the Edit Data Set window, inside **Query** view, insert the following example query:

```
select FIRST_NAME, LAST_NAME
from EMPLOYEE e, POSITION p
where p.POSITION_ID = e.POSITION_ID
and p.POSITION_TITLE= ?
```

Then go to **Parameters** view and add this parameter:

- **name:** EmployeePosPar
- **Data Type:** String
- **Direction:** input
- **Default Value:** 'Store Manager'

This simple query will visualize the First Name and the Last Name of every employee whose position title is equal to the value of the parameter *EmployeePosPar*.

9. Using drag-and-drop from Palette view to report layout, add a table with 2 columns and 1 detail row: using drag-and-drop from the Employees data set put FIRST\_NAME on the detail row of the first column and LAST\_NAME on the detail row of the second column.

10. In Data Explorer View right-click on "Report Parameters" and add a new Parameter:

- **name:** EmployeePosition
- **Data Type:** String
- **Display type:** Text box
- **Default Value:** Store Manager

Then, select the table you added at point 9., click on "Binding" in the table Property Editor and substitute the default value of the data set parameter EmployeePosPar in the Expression Builder window with:  
params["EmployeePosition"]

11. Now it is possible to see the report preview clicking on "Preview" button; to change the parameter value click on "Show Report Parameters" button.



## 5.2.2 CREATE A PARAMETER

Connect to the home page of SpagoBI portal (<http://localhost:8080/sbiportal>) and log on using both "bidev" as username and password. This user is a *Developer* for the SpagoBI Demo and therefore you will access the *Developer Tools* page.

To **create a new Parameter** the following steps are required:

1. Predefined List of Value (LOV);
2. Predefined Values Constraints;
3. Parameters Management.

### 5.2.2.1 Predefined List of Value (LOV)

1. The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools*.
2. Click on the INSERT icon to add the new desired element.
3. In the *Predefined List of Values Details* page fill in the following information:
  - **Label:** Report - LOV QUERY
  - **Name:** Report - LOV QUERY
  - **Description:**
  - **Input Type:** Query statement

Then, in the *Wizard Query* fill in as follows:

- **Connection Name:** Foodmart Data Warehouse
  - **Query Definition:** select POSITION\_TITLE from POSITION  
(then click on "Update fields from query")
  - **Visible Columns:** POSITION\_TITLE
  - **Value Column:** POSITION\_TITLE
4. Once completed the data entry, click on the Save icon saving the information and exit to the *Predefined List of Values* page.
  5. Now click again on the INSERT icon to create a second LOV.
  6. In the *Predefined List of Values Details* page fill in the following information:
    - **Label:** Report - LOV FIX\_LOV
    - **Name:** Report - LOV FIX\_LOV
    - **Description:**
    - **Input Type:** Fixed list of values

Then, in the *Wizard Fix Lov* add the following pairs:

- **New item name:** HQ Information System  
**New item value:** HQ Information System  
(click on the ADD icon)
- **New item name:** HQ Marketing  
**New item value:** HQ Marketing  
(click on the ADD icon)

- **New item name:** HQ Human Resources  
**New item value:** HQ Human Resources  
(click on the ADD icon)
- **New item name:** HQ Finance and Accounting  
**New item value:** HQ Finance and Accounting  
(click on the ADD icon)

Notice that usually *Name* is the field that allows the comprehension of the *Value* field.

7. When the data entry is completed, click on the SAVE icon saving the information and exit to the *Developer Tools* page.

### 5.2.2.2 Predefined Values Constraints

1. The *Predefined Values Constraints* page can be accessed by selecting the corresponding link from the *Developer tools*. It is divided into two parts: on the top side a list of *Predefined Constraints* is displayed; on the bottom the *Configurable Constraints*.
2. Click on the INSERT icon to access the *Constraint Details* page and create a new constraint.
3. Insert the following information:
  - **Label:** Report - Constraint
  - **Name:** Report - Constraint
  - **Description:**
4. Select *MAXLENGTH* as *Check Type* and type 23 in the corresponding text field.
5. When completed, click on the SAVE icon saving the information and exit to the previous page; then select the BACK icon to go to the *Developer Tools* page .

### 5.2.2.3 Parameters Management

When the required LOV and CHECK are created, a new *Parameter* can be created too.

1. Enter the Parameter List page by selecting Parameters Management from the Developer Tools page.
2. Click on the INSERT icon and open the *Parameter Details* page. Insert the following information:
  - **Label:** Report - Parameter
  - **Name:** Report - Parameter
  - **Description:**
  - **Type:** String

Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** Report - Use Mode 1
- **Name:** Report - Use Mode 1
- **Description.**

In the *Role Association* table, select */spagobi/admin* and */spagobi/dev*.

The Administrator or Developer executing a document associated to this parameter will use this specific *Use Mode*.

Then click on LOV LOOKUP icon; in the *Lov lookup list* page now displayed select *Report - LOV QUERY* from the table listing all *Predefined List of Values* (the filter will help you to find it soon).

From the *Predefined Values Constraints* table don't select any constraints.

6. Now click on the SAVE icon.

It can be useful to add another Use Mode in order to understand the roles management performed by *Parameters*.

7. Click on *New...* tab and a new (empty) *Parameter Use Mode Details* section will appear.

8. Fill in the following information:

- **Label:** Report - Use Mode 2
- **Name:** Report - Use Mode 2
- **Description**

Then click on LOV LOOKUP icon; in the *Lov lookup list* page now displayed select *Report - LOV FIX\_LOV* from the table.

Select the *Report - Constraint* from the *Predefined Values Constraints* list.

Notice that in the *Role Association* table the */spagobi/admin* and */spagobi/dev* cannot be selected. In fact, each role can be matched at most to one *Use Mode*.

Check the */spagobi/biuser*.

9. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The *Report - Parameter* will be now displayed in the list. Notice that the *Number of Use Modes* should be 2.

Then click again on the BACK icon to reach the *Developer Tools*.

### 5.2.3 REGISTER THE ANALYTICAL DOCUMENT (THE BUILT REPORT) INTO THE PLATFORM

1. From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.

2. To create a new *Analytical Document* it is necessary to select the INSERT icon.

3. In new *Document Details* page you have to fill in the following information:

- **Label:** Report - Document
- **Name:** Report - Document
- **Description**
- **Type:** Report
- **Engine:** "Jasper Report Dev" if you created the template using iReport, or "Birt Engine" if you created the template with Birt

- **Template:** click on the *browse* button to select the report template created in the Create a Report template using iReport or in the Create a Report template using Birt paragraph.
4. Moreover, it is necessary to indicate which is the parent folder of the document by selecting the check box corresponding to *Static Reporting*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page.
  5. To save and exit from this page click on the SAVE AND RETURN icon.
  6. The *Development Objects List* will be updated with a new row containing the document just created.
  7. Now you can access the *Document Details* page simply by clicking on the DETAILS icon on the row of the new document. This page will list the general detailed information of the document. On the right side of the page a new table listing the just added template is displayed.
  8. In order to set *Document Parameters* it is necessary to fill the *DOCUMENT PARAMETER DETAILS* form below the 'NEW ...' tab: insert the following information:
    - **Title:** Report - Doc Param
    - **URL Name:** EmployeePosition

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select *Report - Parameter* from the table.

Notice that the URL Name must match the name of the parameter created in the report template example in the Create a Report template or Create a Report template using Birt paragraph.
  9. When this operation is completed, save information by clicking on the SAVE icon.
  10. It is possible to create a standard parameter in order to choose the output format of the document. Click on the 'NEW ...' tab and add the following information:
    - **Title:** Choose output format
    - **Parameters:** Output Type
    - **URL Name:** param\_output\_format



Notice that the Parameter is already present in the SpagoBI DEMO. Moreover the same parameter is used by many of the existing Analytical Document listed in this DEMO. This is an example of how to use the same resource for different documents.

Furthermore, notice that the URL Name refers to a predefined parameter for the *Jasper Report* and *Birt* engines.

11. When this operation is completed, save information by clicking on the SAVE AND RETURN icon. The new parameter will be displayed in the list.
12. Go back to the *Development Object List* by clicking on the BACK icon.

Now, all the parameters have been correctly created and configured on the *Analytical Document*. So you can execute it.

13. Click on the EXECUTE icon corresponding to the new *Report - Document*.
14. In the new page you will have to enter a value for the *Report - Doc Param*.
15. Click on the DETAILS icon and choose one of the predefined values in the list by clicking on the corresponding DETAILS icon. Select *HQ Information System*.
16. Click on the EXECUTE icon to execute the document.
17. Now it is possible to Update State by clicking on the corresponding icon at the bottom of the window. Notice that when the *Document Example* is updated, it will not be listed in the *Development Object List* anymore.
18. Logout

#### 5.2.4 TEST THE ANALYTICAL DOCUMENT

---

Work in progress.



#### 5.2.5 EXECUTE THE ANALYTICAL DOCUMENT

---

Work in progress.



### 5.3 OLAP ANALISYS

This following example is designed to quick introduce new users to create a new OLAP in SpagoBI.

The main steps to manage a report are:

1. Create a Template
2. Create Parameters
  - a. Create Lists of Value
  - b. Create Constraints
3. Register the Analytical Document (the built OLAP) into the platform
  - a. Add Template
  - b. Assign Parameters
4. Test the Analytical Document
5. Execute the Analytical Document

### 5.3.1 CREATE A TEMPLATE

In order to create a proper template for an OLAP document, it is only necessary to realize an XML file containing the following elements:

- `<olap>` root element;
- `<connection>` logical name that will be interpreted by the engine to identify the proper connection ;
- `<cube>` the *reference* attribute of this element identifies the XML file that describes the datamart cube using the Mondrian syntax.
- `<MDXquery>` the text of this element is a query executed on the datamart, written in the MDX syntax.
- `<parameter>` this element contained in the `<MDXquery>` identifies a parameter for the query. The *name* attribute refers to the parameter name contained in the query, while the *as* attribute identifies the alias that will be used by the SpagoBI document.

For this example it is necessary to create an XML file containing the following text:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<olap>
  <connection name='defaultDWH' />
  <cube reference='/WEB-INF/queries/FoodMart.xml' />
  <MDXquery>
    select
      {[Measures].[Unit Sales], [Measures].[Store Cost], [Measures].[Store Sales]} on
columns,
      {Parameter("ProductMember", [Product], [Product].[All Products].[Food], "wat
willste?").children} ON rows
    from Sales where ([Time].[1997])
  <parameter name='prdCd' as='ProductMember' />
</MDXquery>
</olap>
```

Save the file as *productSales.olap*.

### 5.3.2 CREATE PARAMETERS

Connect to the home page of SpagoBI portal (<http://localhost:8080/portal>) and log on using "bidev" both as username and password. This user is a *Developer* for the SpagoBI Demo and therefore you will access the *Developer Tools* page .

To **create a new Parameter** the following steps are required:

1. Predefined List of Value (LOV);
2. Predefined Values Constraints;
3. Parameters Management.

#### 5.3.2.1 Predefined List of Value (LOV)

1. The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools*.
2. Click on the INSERT icon to add the new desired element.

3. In the *Predefined List of Values Details* page fill in the following information:

- **Label:** OLAP - LOV FIX\_LOV 1
- **Name:** OLAP - LOV FIX\_LOV 1
- **Description**
- **Input Type:** Fixed list of values
- **New item name:** Non Consumable  
**New item value:** [Product].[All Products].[Non-Consumable]  
(click on the ADD icon)
- **New item name:** Food  
**New item value:** [Product].[All Products].[Food]  
(click on the ADD icon)
- **New item name:** Drink  
**New item value:** [Product].[All Products].[Drink]  
(click on the ADD icon)

4. When the data entry is completed, click on the SAVE icon to save the information and exit to the *Predefined List of Values* page.

5. Now click again on the INSERT icon to create a second LOV.

6. In the *Predefined List of Values Details* page fill in the following information:

- **Label:** OLAP - LOV FIX\_LOV 2
- **Name:** OLAP - LOV FIX\_LOV 2
- **Description:**
- **Input Type:** Fixed list of values
- **New item name:** Non Consumable  
**New item value:** [Product].[All Products].[Non-Consumable]  
(click on the ADD icon)
- **New item name:** Food  
**New item value:** [Product].[All Products].[Food]  
(click on the ADD icon)

7. When the data entry is completed, click on the SAVE icon to save the information and exit to the *Developer Tools* page.

### 5.3.2.2 Predefined Values Constraints

1. The *Predefined Values Constraints* page can be accessed by selecting the corresponding link from the *Developer tools*. It is divided into two parts: on the top side a list of *Configurable Constraints* is displayed; on the bottom the *Predefined Constraints*.

2. Click on the INSERT icon to access the *Constraint Details* page and create a new constraint.

3. Insert the following information:

- **Label:** OLAP - Constraint
- **Name:** OLAP - Constraint
- **Desription**

4. Select *MAXLENGHT* as *Check Type* and type *35* in the corresponding text field.



5. When completed, click on the SAVE icon to save the information and exit to the previous page. Then select the BACK icon to go to the *Developer Tools* page.

### 5.3.2.3 Parameters Management

When the required LOV and constraints are created, a new *Parameter* can be created too.

1. Enter the *Parameter List* page by selecting *Parameters Management* from the *Developer Tools* page.
2. Click on the INSERT icon and open the *Parameter Details* page.
3. Insert the following information:
  - **Label:** OLAP - Parameter
  - **Name:** OLAP - Parameter
  - **Description**
  - **Type:** String
4. Click on the SAVE icon. An empty *Parameter Use Mode Details* section will be displayed. Enter the following information:
  - **Label :** OLAP - Use Mode 1
  - **Name:** OLAP - Use Mode 1
  - **Description**

In the *Role Association* table, select */spagobi/admin* and */spagobi/dev*.

The Administrator or the Developer executing a document associated to this parameter, will use this specific *Use Mode*.

Then select *OLAP - LOV FIX\_LOV 1* from the lookup page listing all Predefined List of Values.

From the *Predefined Values Constraints* table don't select any constraints.

5. Now click on the SAVE icon.

It can be useful to add another Use Mode in order to understand the roles management performed by *Parameters*.

6. Click on the 'NEW ...' tab and add the following information:
  - **Label:** OLAP - Use Mode 2
  - **Name:** OLAP - Use Mode 2
  - **Description**

Then select *OLAP - LOV FIX\_LOV 2* from the lookup page listing all Predefined List of Values.

This time select the *OLAP - Constraint* from the *Predefined Values Constraints* list.

Notice that in the *Role Association* table the */spagobi/admin* and */spagobi/dev* cannot be selected. In fact each role can be matched at most to one *Use Mode*.

This time check the */spagobi/biuser*.

7. Click on the SAVE AND RETURN icon to go back to the *Parameters List*. The *OLAP Parameter Example* will be now displayed in the list. Notice that the *Number of Use Modes* should be 2.
8. Then click again on the BACK icon to reach the *Developer Tools*.

### 5.3.3 REGISTER THE ANALYTICAL DOCUMENT (THE BUILT OLAP) INTO THE PLATFORM

1. From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.
2. Select the icon to create a new *Analytical Document*.
3. In new *Document Details* page you will be required to fill in the following information:
  - **Label:** OLAP - Document
  - **Name:** OLAP - Document
  - **Description:**
  - **Type:** On-line analytical processing
  - **Engine:** Jpivot-Mondrian Dev
  - **Template:** click on the *browse* button to select the template created in the paragraph.
4. Moreover, you have to indicate the parent folder of the document selecting the check box corresponding to *Dimensional Analysis*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page.
5. To save and exit from this page click on the SAVE AND RETURN icon.
6. The *Development Objects List* will be updated with a new row containing the document just created.
7. Now you can access the *Document Details* page simply by clicking on the DETAILS icon on the row of the new document. This page will list the general detailed information of the document. On the right side of the page a new table listing the just added template is displayed.
8. In order to set *Document Parameters* it is necessary to fill the *DOCUMENT PARAMETER DETAILS* form below the 'New ...' tab: insert the following information:
  - **Title:** OLAP - Doc Param
  - **URL Name:** prdCd

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select *OLAP - Parameter* from the table.

Notice that the URL Name must match the alias (as attribute) of the parameter created in the template example created in the paragraph.

9. Save and go back to the *Development Object List* by clicking on the SAVE AND RETURN icon.

Now that all the parameters have been correctly created and configured on the *Analytical Document*, you can execute it.

10. Click on the icon EXECUTE which corresponds to the new *OLAP - Document*.
11. In the new page you will be required to enter a value for the *OLAP - Doc Param*.
12. Select *Food* from the combo-box.
13. Click on the EXECUTE icon to execute the document.
14. Now it is possible to Update State by clicking on the corresponding icon at the bottom of the window. Notice that once updated the *OLAP - Document* will not be listed in the *Development Object List* anymore.
15. Logout

### 5.3.4 TEST THE ANALYTICAL DOCUMENT

---

Work in progress.



### 5.3.5 EXECUTE THE ANALYTICAL DOCUMENT

---

Work in progress.



## 5.4 DASHBOARD

This following example is designed to quick introduce new users to create a new DASHBOARD in SpagoBI starting from a SWF movie already created with OpenLazslo.

In this document we will not explain how to build a SWF movie with OpenLazslo; we invite you to visit the OpenLazslo home page [www.openlaszlo.org](http://www.openlaszlo.org) for more information.

SpagoBI is released with some SWF movies; you can find them in the folder *webapps/spagobi/dashboards* inside Exo-Tomcat main directory. For this example we will consider *rot.lzx.swf*. For the source code and documentation about SpagoBI movies download the package SpagoBI Components from [http://forge.objectweb.org/project/showfiles.php?group\\_id=204&release\\_id=1407](http://forge.objectweb.org/project/showfiles.php?group_id=204&release_id=1407).

A Dashboard element must be fed with the values to be displayed: these values are supplied by a predefined List of Value.

The main steps to manage a dashboard are:

1. Create the Lists of Value for movie feeding
2. Create a Template for movie configuration
3. Register the Analytical Document (the built DASHBOARD) into the platform
  - a. Add Template
4. Test the Analytical Document
5. Execute the Analytical Document

#### 5.4.1 CREATE A LIST OF VALUE

1. The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools*.
2. Click on the INSERT icon to add the new desired element.
3. In the *Predefined List of Values Details* page fill in the following information:

- **Label:** DASHBOARD - SCRIPT
- **Name:** DASHBOARD - SCRIPT
- **Description**
- **Input Type:** Script
- **Script:**

```
StringBuffer buf = new StringBuffer();
buf.append('<rows>');
buf.append('<row value=');
Random rand = new Random();
float f = 80 + ((rand.nextFloat() * 100) % 15);
buf.append(f);
buf.append('\ />');
buf.append('</rows>')
return buf;
```
- **Output:** Single Value

The above script will only return a random value.

4. When the data entry is completed, click on the SAVE icon to save the information and exit to the *Predefined List of Values* page.

#### 5.4.2 CREATE A TEMPLATE

In order to create a proper template for a DASHBOARD document, it is only necessary to realize an XML file containing the movie configuration: create an XML with the following content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

```
<DASHBOARD movie="dashboards/rot.lzx.swf" displayTitleBar="true">
  <DIMENSION width="240" height="150" />
  <CONF>
    <PARAMETER name="minValue" value="0" />
    <PARAMETER name="maxValue" value="200" />
    <PARAMETER name="lowValue" value="50" />
    <PARAMETER name="highValue" value="120" />
  </CONF>
  <DATA url="DashboardService">
    <PARAMETER name="dataname" value="DASHBOARD - SCRIPT" />
  </DATA>
</DASHBOARD>
```

Save the file as DASH\_EXAMPLE.xml.



Notice the following: the *movie* attribute of the *DASHBOARD* tag refers to the SWF file in SpagoBI context; *DIMENSION* and *CONF* tags refers to the movie configuration; *DATA* tag defines the Servlet and the List of Value for the movie values feeding.

### 5.4.3 REGISTER THE ANALYTICAL DOCUMENT (THE BUILT DASHBOARD) INTO THE PLATFORM

1. From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.
2. Select the icon to create a new *Analytical Document*.
3. In new *Document Details* page you will be required to fill in the following information:
  - **Label:** DASHBOARD - Document
  - **Name:** DASHBOARD - Document
  - **Description:**
  - **Type:** Dashboard
  - **Template:** click on the *browse* button to select the template created in the previews paragraph.
5. Moreover, you have to indicate the parent folder of the document selecting the check box corresponding to *Dashboard*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page.
6. To save and exit from this page click on the SAVE AND RETURN icon.

Now that the document has been correctly created and configured on the *Analytical Document*, you can execute it.

7. Click on the icon EXECUTE which corresponds to the new *DASHBOARD - Document*.
8. Now it is possible to Update State by clicking on the corresponding icon at the top of the window. Notice that once updated the *DASHBOARD - Document* will not be listed in the *Development Object List* anymore.

9. Logout

#### 5.4.4 TEST THE ANALYTICAL DOCUMENT

---

Work in progress.



#### 5.4.5 EXECUTE THE ANALYTICAL DOCUMENT

---

Work in progress.



### 5.5 DATA MINING

The main steps to manage a data mining process are:

1. Design a data mining process using Weka KnowledgeFlow
2. Save the process flow in the kflm format
3. Parametrize the process flow
4. Create Parameters
  - a. Create Lists of Value
  - b. Create Constraints
5. Register the Analytical Document (the built data mining process) into the platform
  - a. Add Template
  - b. Assign Parameters
6. Test the Analytical Document
7. Execute the Analytical Document

#### 5.5.1 CREATE A DATA MINING PROCESS USING WEKA KNOWLEDGEFLOW

1. If you don't have yet Weka-3-4 you can download it from:

<http://www.cs.waikato.ac.nz/ml/weka/>

2. Unzip the downloaded file and install the program
3. Add your JDBC driver library to the classpath or simply copy it into the weka installation directory
4. Run weka
5. In order to create a simple data mining process example, select **KnowledgeFlow** button from the **Weka GUI Chooser** window.

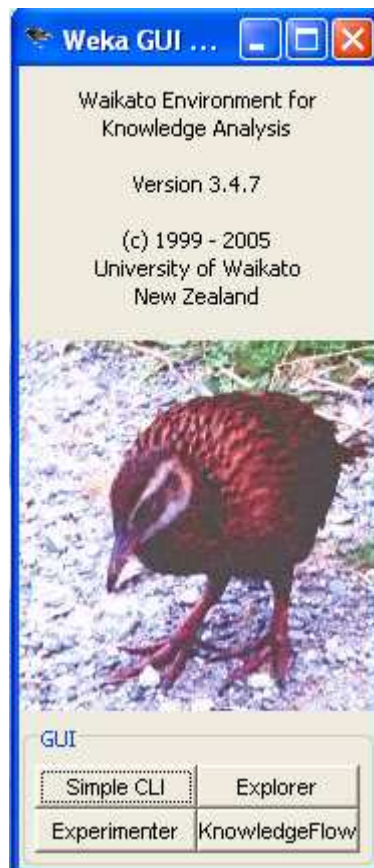


Figure 54 - Weka GUI Chooser window

6. Choose **Database Loader** component from **DataSource Tab** and add it to the flow (for the moment the SpagoBI Weka Engine supports only this kind of DataSource).
7. Right click on Database Loader component and choose **configure option** from the pop-up menu.
8. Set up the query parameter as follow ...

```
SELECT * FROM cust_to_dept
```

The table `cust_to_dept` have an instance for each customer and a column for each store department recording the total amount of money spent by the specified customer in the last year in that department (i.e. [`cust_id`, `dep1`, `dep2`, ..., `depN`]). Others parameters in this configuration window can be ignored because the connection to the database is setted by the engine according to its configuration settings



9. Choose **Database Saver** component from **DataSink Tab** and add it to the flow (for the moment the SpagoBI Weka Engine supports only this kind of DataSink).
10. Right click on Database Saver component and choose **configure option** from the pop-up menu.
11. Set up the **Table Name parameter** as follow ...

```
cust_to_dept_clustered
```

This table is used to store the outcomes of the mining process. It associates each customer to a cluster (i.e. [cust\_id, cluster]). Others parameters in this configuration window can be ignored because the connection to the database is setted by the engine according to its configuration settings

12. Now you can add to the flow a mining algorithm. Choose, for example, **Add Cluster** component from **Filters Tab**. This component adds a new nominal attribute representing the cluster assigned to each instance of the input dataset by the specified clustering algorithm.
13. Right click on Add Cluster component and choose **configure option** from the pop-up menu.
14. Choose the clustering algorithm to be used by the Add Cluster component. Than choose fields of input instances to ignore during clustering process. In this example we choose Simple-KMeans as clustering algorithms and we choose to ignore the first field (i.e. the customer id).
15. Right click on DatabaseLoader component and choose **dataset option** from the pop-up menu in order to connect this component to the Add Cluster component. The dataset connection is just a pipe that push the source component output to the destination component input. In this case we pass the whole dataset read from the database by the DatabaseLoader component to the Add Cluster component. **Add Cluster** component process the input datasource ignoring the first field and append to each instance a cluster attribute.
16. Because the output dataset of the Add Cluster component (i.e. [cust\_id, dep1, dep2, ..., depN, cluster]) doesn't fit with our destination table cust\_to\_dept\_clustered (i.e [cust\_id, cluster]) we can't connect directly the Add Cluster component to the DatabaseSaver component. We have to add **Remove** component between these two component in order to filter out all the unnecessary fields. Choose the **Remove** component from **Filters Tab** and add it to the process. Configure it in order to filter out all the fields in input dataset except the first (customer id) end the last (cluster). Connect the Add Cluster component to the Remove component and than the Remove component to the Database Saver component.
17. Save your newly generated data mining process to a file in *kflm* format.

## 5.5.2 PARAMETRIZE PROCESS FLOW

It is possible to parametrize the mining process just using parameters as value of some flow component configuration properties. A parameter is just a string enclosed in the special characters `$P{` and `}`. The enclosed string is the name given to the parameter.

All added parameters must than be associated with SpagoBI parameters when the document is inserted into the system in order to execute it.

Weka doesn't support parameters. For this reason it isn't always possible to add a parameter directly from the GUI Knowledge Flow Editor. This happens because Weka applies constraints to some properties' value format. Using parameters as value of that properties violates the applied constraints and causes errors (usually default values are used instead of the inserted ones). In these cases you have to type your parameter directly inside the generated *kfml* file. For example if you want to parametrize the clustering algorithm used in the previously defined flow you can do that directly from the GUI interface because Weka is very severe about the format of this property. So you have to choose an algorithm, save the flow, open the *kfml* file with a text editor, look for the name of the algorithm inserted from the GUI interface and replace it with a parameter. Searching for SimpleKMeans in the flow defined above we find this line...

```
<object class="java.lang.String" name="1">weka.clusterers.SimpleKMeans -N 15 -S 10</object>
```

that means that we are going to use SimpleKMeans as clustering algorithm to produce 15 clusters using 10 as the initial seed. It is now possible to replace this line with the following parametrized one ...

```
<object class="java.lang.String" name="1">$P{clusterer} -N $P{clusterNum} -S 10</object>
```

Pay attention when you add parameters to the *kfml* file by hand because then this file cannot be reloaded correctly into the GUI Knowledge Flow Editor. If you want to modify it by hand be sure to have made a copy of the original file before.

### 5.5.3 CREATE A PARAMETER

Connect to the home page of SpagoBI portal (<http://localhost:8080/portal>) and log on using both "bidev" as username and password. This user is a *Developer* for the SpagoBI Demo and therefore you will access the *Developer Tools* page.

To **create a new Parameter** the following steps are required:

1. Predefined List of Value (LOV);
2. Predefined Values Constraints;
3. Parameters Management.

#### 5.5.3.1 Predefined List of Value (LOV)

1. The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools*.

2. Click on the INSERT icon to add the new desired element.

3. In the *Predefined List of Values Details* page fill in the following information:

- **Label:** HSQL\_FM\_CONN
- **Name:** Connection to foodmart on Hsql db

- **Description:** Connection to foodmart on Hsql db
- **Input Type:** Script

Then, in the *Wizard Script* fill in as follows:

- **Script:** 'hsqlfoodmart'
- **Output:** Single Value

where *hsqlfoodmart* is the name given to the hsql connection in the engine configuration file(see engine configuration)

4. Once completed the data entry, click on the Save icon and exit to the *Predefined List of Values* page.
5. Now click again on the INSERT icon to create a second LOV.
6. In the *Predefined List of Values Details* page fill in the following information:

- **Label:** DM\_CLUSTERER\_VAL
- **Name:** Clustering Algorithms
- **Description:** Clustering Algorithms
- **Input Type:** Fixed list of values

Then, in the *Wizard Fix Lov* add the following pairs:

- **New item name:** Simple K-Means  
**New item value:** weka.clusterers.SimpleKMeans  
(click on the ADD icon)
- **New item name:** Farthest First  
**New item value:** weka.clusterers.FarthestFirst

Notice that usually *Name* is the field that allows the comprehension of the *Value* field.

7. When the data entry is completed, click on the Save icon saving the information and exit to the Developer Tools page.

### 5.5.3.2 Predefined Values Constraints

Add constraints if you need them. To add constraints follow the same steps described in session 5.2.2.2.

### 5.5.3.3 Parameters Management

When the required LOV and CHECK are created, a new *Parameter* can be created too.

1. Enter the Parameter List page by selecting Parameters Management from the Developer Tools page.

2. Click on the INSERT icon and open the *Parameter Details* page. Insert the following information:

- **Label:** HSQL\_FM\_CONN\_PAR
- **Name:** Connection to foodmart on Hsql db
- **Description:** Connection to foodmart on Hsql db
- **Type:** String

Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** DEFAULT
- **Name:** Default Modality
- **Description:** Default Modality
- **Lov:** select and lookup for HSQL\_FM\_CONN LOV

In the *Role Association* table, select all roles in order to associate this *Use Mode* to all users of the system during document execution.

4. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The HSQL\_FM\_CONN\_PAR will be now displayed in the list.

5. Now click again on the INSERT icon and open the *Parameter Details* page. Insert the following information:

- **Label: DM\_CLUSTERER\_PAR**
- **Name: Clustering Algorithms**
- **Description: Clustering Algorithms**
- **Type: String**

6. Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** DEFAULT
- **Name:** Default Modality
- **Description:** Default Modality
- **Lov:** select and lookup for DM\_CLUSTERER\_VAL LOV

In the *Role Association* table, select all roles in order to associate this *Use Mode* to all users of the system during document execution.

7. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The DM\_CLUSTERER\_PAR will be now displayed in the list.

8. Now click again on the INSERT icon and open the *Parameter Details* page. Insert the following information:

- **Label: DM\_CLUSTERS\_NUM\_PAR**
- **Name: Number of clusters**
- **Description: Number of clusters**
- **Type: String**

9. Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** DEFAULT
- **Name:** Default Modality
- **Description:** Default Modality
- **Manula Input:** select

In the *Role Association* table, select all roles in order to associate this *Use Mode* to all users of the system during document execution.

10. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The DM\_CLUSTERS\_NUM\_PAR will be now displayed in the list.

#### 5.5.4 REGISTER THE ANALYTICAL DOCUMENT (THE BUILT DATA-MINING FLOW) INTO THE PLATFORM

1. From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.
2. Select the icon to create a new *Analytical Document*.
3. In new *Document Details* page you will be required to fill in the following information:
  - **Label:** USER\_SEG\_DMP
  - **Name:** User segmentation process
  - **Description:** User segmentation process
  - **Type:** Data-Mining Model
  - **Engine:** Weka Dev
  - **Template:** click on the *browse* button to select the template created in the previous paragraph.
4. Moreover, you have to indicate the parent folder of the document selecting the check box corresponding to *Dimensional Analysis*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page
5. To save and exit from this page click on the SAVE AND RETURN icon.
6. The *Development Objects List* will be updated with a new row containing the document just created.
7. Now you can access the *Document Details* page simply by clicking on the DETAILS icon on the row of the new document. This page will list the general detailed information of the document. On the right side of the page a new table listing the just added template is displayed.
8. In order to set *Document Parameters* it is necessary to fill the *DOCUMENT PARAMETER DETAILS* form below the 'NEW ...' tab: insert the following informations:
  - **Title:** Connection
  - **URL Name:** connectionName

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select HSQL\_FM\_CONN\_PAR from the table.

9. Select again the 'NEW ...' tab and insert the following informations:

- **Title:** Clusterer
- **URL Name:** clusterer

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select DM\_CLUSTERER\_PAR from the table.

Notice that the URL Name must match the name of the parameter created in the flow example created in the previous paragraph.

10. Select again the 'NEW ...' tab and insert the following informations:

- **Title:** ClusterNumber
- **URL Name:** clusterNum

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select DM\_CLUSTERS\_NUM\_PAR from the table.

Notice that the URL Name must match the name of the parameter created in the flow example created in the previous paragraph.

11. Save and go back to the *Development Object List* by clicking on the SAVE AND RETURN icon. Now that all the parameters have been correctly created and configured on the *Analytical Document*, you can execute it.

12. Click on the icon EXECUTE which corresponds to the new *DATA\_MINING* analytica document.

13. In the new page you will be required to enter a value for the *DM\_CLUSTERER\_PAR* and *DM\_CLUSTERS\_NUM\_PAR* parameters.

14. Select *Simple-KMeans* from the combo-box.

15. Insert the number of clusters you want to generate into the input box

16. Click on the EXECUTE icon to execute the document.

17. Data Mining Flow processing can take quite a long time depending on the dimensions of your dataset. For this reason weka engine process all the request in background. This mean that when you execute a data mining analytical document you don't have to wait until the engine finished processing it but you can continue with your work

18. Logout

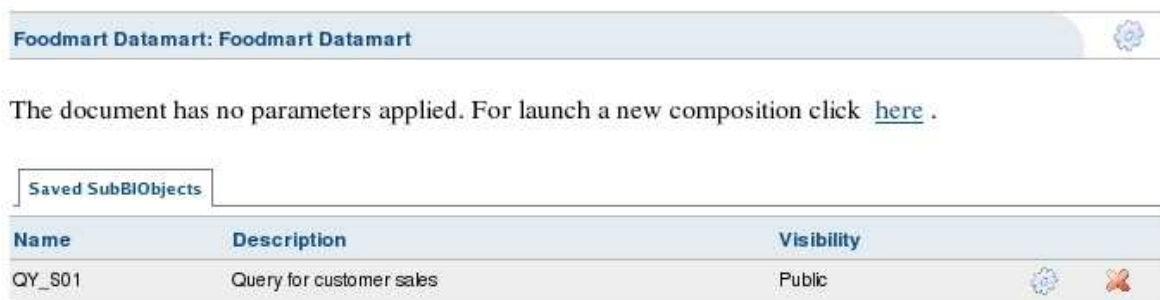
## 5.6 QUERY BY EXAMPLE

Query By Example is a SpagoBI tool realized in order to ease the user to create simple queries through a graphical interface.



Notice that this feature is in a RC (release candidate) state.

By clicking on the relative menu item, a list of the QbE queries is displayed.



The screenshot shows a web interface for 'Foodmart Datamart: Foodmart Datamart'. It includes a message: 'The document has no parameters applied. For launch a new composition click [here](#).' Below this is a table titled 'Saved SubBIObjects'.

Name	Description	Visibility
QY_S01	Query for customer sales	Public

**Figure 55 – QbE: Query list**

On the lower side of the window the user can choose to modify an existing query, by clicking on the corresponding EXECUTE icon, or to create a new one by selecting the *here* link at the top of the window.

The composition process is scheduled into eighth steps that will ease the user to create a new query:

- **Select Fields :** to choose the **select** fields;
- **Conditions:** to set the **where** conditions;
- **Ordering:** to select the fields for the **order by**;
- **Grouping:** to select the fields for the **group by**
- **View Query:** to display the query realized following the wizard or to write a proper query using HQL/SQL language;
- **Save Query:** to verify if the query is formally correct and to save it;
- **Execute Preview:** to display the results obtained by the query realized and eventually export them.
- **Generate Template:** to automatically generate a JasperReports report template based on the query.



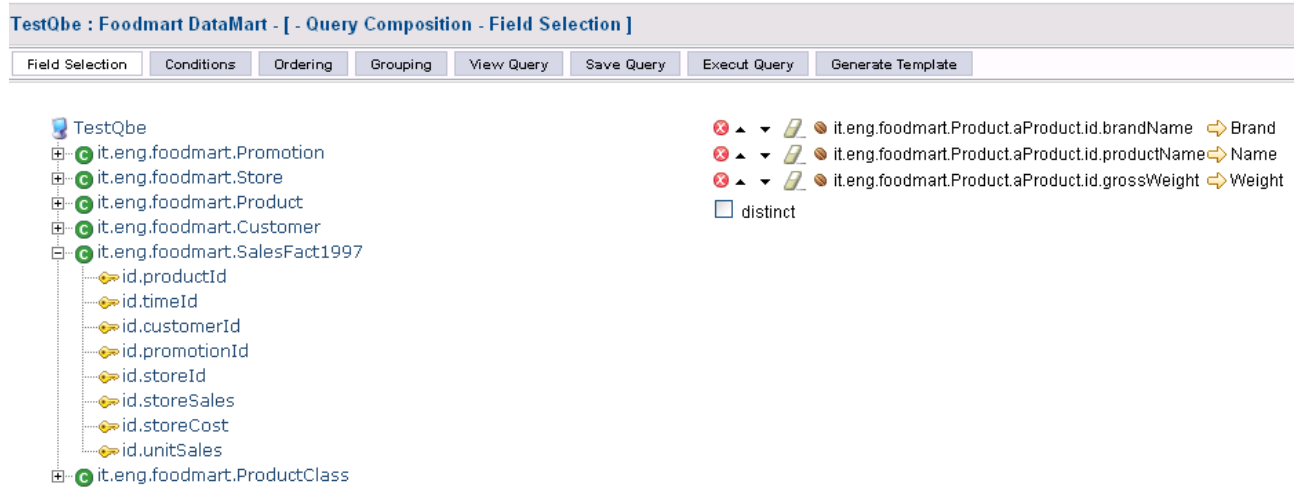


Figure 56 – QbE: Field Selection

## 5.6.1 FIELD SELECTION








On the left hand side of the window a logic view of the datamart is displayed. Therefore, the user can easily navigate the tree and select the desired fields simply by clicking on them.



Figure 57 – QbE: Field Selection detail

Once selected a field, it appears on the right hand side.  
The following options are available:

-  : to erase the corresponding selection;
-  : to move up or down the field in the list, to set the order of the columns of the result table;

-  : to apply a group operator on a field, by selecting the proper one from the list;
-  : to remove a group operator;
-  : to edit an alias name that will be displayed as header of the column corresponding to the selected field in the result table.

Moreover, the **distinct** option can be set by clicking on the corresponding check below the selected fields.

## 5.6.2 CONDITION



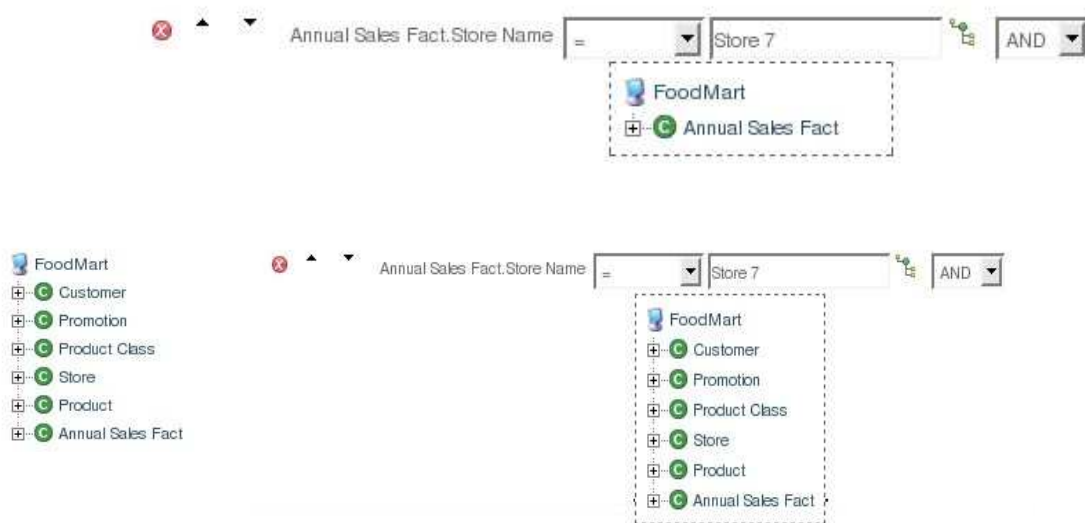
**Figure 58 – QbE: Condition**

On the left hand side of the window the logic view of the datamart model is displayed.



Notice that to help the user, the Qbe will display only the clauses that contain a **select field**. This is called **Light Tree**. To display the complete datamart tree, the user can simply click on the FULL TREE icon.

Once selected a field, on the right hand side of the page, the user can complete the where condition selecting the proper operator, adding the right condition in the text area and choosing the logic operator that will be set before the following condition.



**Figure 59 – QbE: Where condition**



Notice that the right part of the where condition can also be a field: in this case, the user can simply click on the TREE icon and select the desired field.

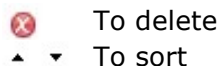
Notice that the datamart tree displayed is the same shown to choose the left condition.

Both the **Ordering** and the **Grouping** page display on the left hand side a list of the select fields of the query.

By clicking on the name of one of them, the user can set respectively the **order by** and the **group by** fields.

If the user tries to access one of these pages without having already selected at least a field in **Field Selection** page, the Qbe displays an error message

Each selection can be erased by means of the relative icons:



### 5.6.3 VIEW QUERY

In the **View Query** the query realized through the wizard procedure is displayed on the left hand side of the page.

**TestQbe : Foodmart DataMart - [ - Query Composition - Resume Query ]**

Field Selection   Conditions   Ordering   Grouping   View Query   Save Query   Execut Query   Generate Template

**Qbe automatically composition**

**Select**  
aProduct.id.brandName as Brand ,  
aProduct.id.productName as Name ,  
aProduct.id.grossWeight as Weight  
**From**  
it.eng.foodmart.Product as aProduct  
**Where**  
aProduct.id.brandName = 'Best Choice'  
**Order By**  
aProduct.id.brandName ,  
aProduct.id.grossWeight

**Expert Composition**

```
select aProduct.id.brandName,
aProduct.id.productName, aProduct.id.grossWeight
from it.eng.foodmart.Product as aProduct order by
aProduct.id.brandName, aProduct.id.grossWeight
```

[Resume from Qbe](#)

[Resume last Expert Query](#)   [Save this Expert Query](#)

**Execution Modality**   ☒ Default   ☐ Expert

**Figure 60 – QbE: View Query**

The user can also try to write his own query in the text area on the right hand side of the page. This is called **Expert Composition**.

The **Resume From Query** icon can be used to copy the query automatically realized by the Qbe in the text area in order to modify it.

The user can also **save the expert query** realized and **resume the last expert query** saved simply by clicking on the corresponding icons.

The two radio buttons on the bottom line of the page must be used to set if the default query to be executed is the one realized in the Qbe Automatically Composition or in the Expert Composition.

## 5.6.4 SAVE QUERY

**Save Query**

Query Identifier

Query Description

Save with :

☐ Public scope  
☒ Private scope

**Final Query**

The query is correct

**Expert Query**

The query is correct

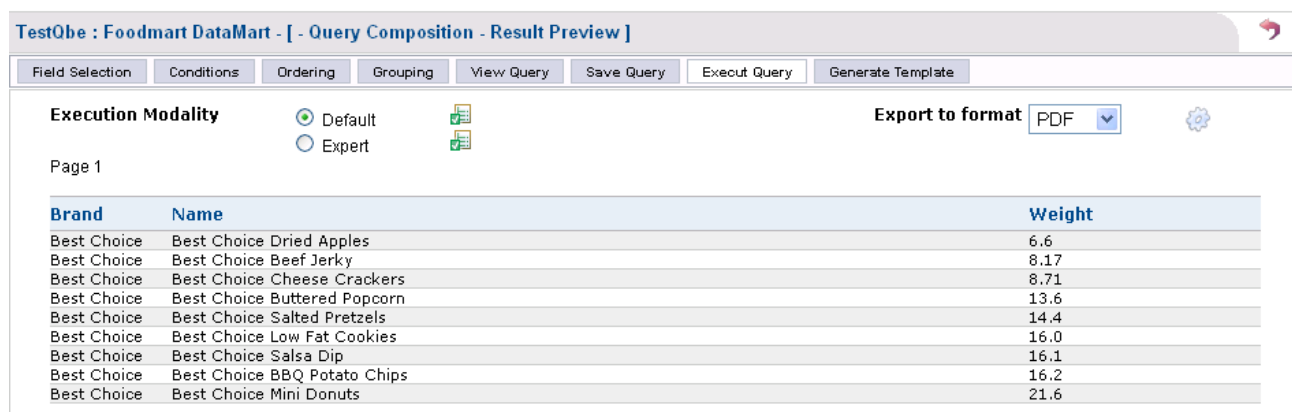
**Figure 61 – QbE: Save Query**

Once selected the **Save Query** tab, the Qbe will try to execute both the automatically composed and the expert query and will display on the right hand side a message to indicate if each query works properly or not.

On the other side of the page, the user can fill in the form required to save the query. The user can also specify the scope of the query:

- Public scope: the query will have the same visibility as the main Data Mart object
- Private scope: the query will be visible only to the user

## 5.6.5 EXECUTE QUERY



Brand	Name	Weight
Best Choice	Best Choice Dried Apples	6.6
Best Choice	Best Choice Beef Jerky	8.17
Best Choice	Best Choice Cheese Crackers	8.71
Best Choice	Best Choice Buttered Popcorn	13.6
Best Choice	Best Choice Salted Pretzels	14.4
Best Choice	Best Choice Low Fat Cookies	16.0
Best Choice	Best Choice Salsa Dip	16.1
Best Choice	Best Choice BBQ Potato Chips	16.2
Best Choice	Best Choice Mini Donuts	21.6

Figure 62 – QbE: Execute query

The **Execute query** page first tries to execute the expert or the automatically composed query, depending on the selection performed in the **View Query** page.

If the query works correctly, a table containing the result set is displayed. Otherwise, a text area containing a description of the error occurred will be displayed.

It is possible to pass from the expert query results to the automatically composed query results or vice-versa using the *execution modality checkbox* situated in the upper left corner of the page.

It is also possible to export results of the query in other formats using the *export options* situated in the upper right corner of the page. The exportation result can be saved to local disk. If the browser have a pre-configured plugin viewer for the target exportation format the exportation result can be visualized into the browser window before saving.

## 5.6.6 GENERATE TEMPLATE

The **Generate Template** page generates a JasperReports report template (i.e. a jrxml file) based on the SQL query produced from the HQL query. An example of a report built upon this template is shown in the bottom of the page. The generated template can be saved to local disk, to be used as a base for further modifications, using the save button situated at the top of the page.

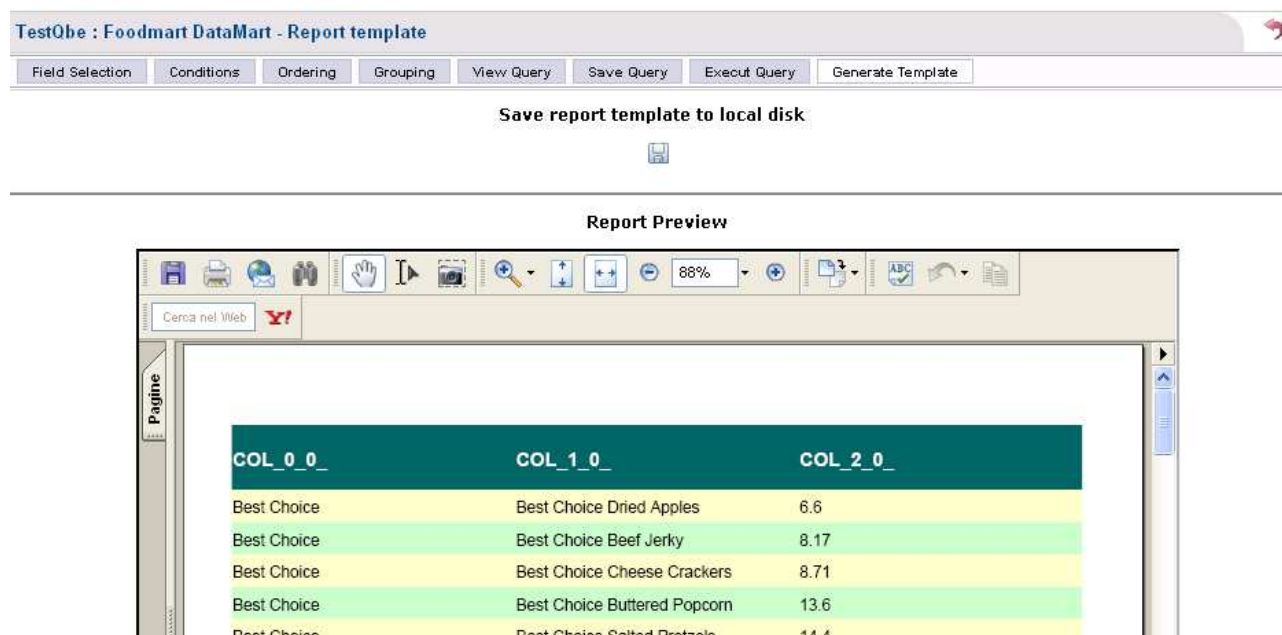


Figure 63 – QbE: Generate Template

## 5.7 MAPS

A Map business object is a visual representation of some spatial data (typically a geographic map) whose elements are enriched with some graphical information (typically colors) based on the value of some attribute related to the spatial element. Example: a geographic map of the United States with the states filled with different colors based on the population of each one.

If you want to develop a map business object using SpagoBI Geo Engine you need:

- an svg file which contains geographic elements (each one identified by a unique id).
- A database which contains information associated to the ids of the svg map
- A map template file (described below), which contains:
  - o the information useful to link the map with the database
  - o the styles to apply to svg geographic elements based on the attribute values stored into the database

When the user executes a map document, the system:

- loads the map template file
- reads the database connection configuration
- connects to the database and executes the query in order to extract the data associated to the svg geographic elements.
- reads from the template the svg map location and gets the map
- scans the map and for each svg element id:
  - o get from the result of the query the attribute value related to the id
  - o reads from the template file the style associated to the value of the attribute
  - o Sets the style, extracted from the map template file, to the svg element

Let's make an example:

#### svg file:

```
<svg>
  <g attrib:name="Europe">
    <path id="1" attrib:name="Italy" d="M0 0 ...." />
    <path id="2" attrib:name="Spain" d="M10 10 ...." />
    <path id="3" attrib:name="Germany" d="M20 20 ...." />
    ...
  </g>
</svg>
```

#### result returned by the database query:

<i>Ids</i>	<i>Population</i>
1	60000000
2	40000000
3	80000000

#### Template file:

```
<LEVELS>
  <LEVEL threshold="0" style="fill:blue;>
    <TEXT description="less than 50000000" />
  </LEVEL>
  <LEVEL threshold="50000000" style="fill:red;>
    <TEXT description="from 50000000 to 70000000" />
  </LEVEL>
  <LEVEL threshold="70000000" style="fill:green;>
    <TEXT description="from 70000000" />
  </LEVEL>
</LEVELS>
```

#### The final svg produced:

```
<svg>
  <g attrib:name="Europe">
    <path id="1" attrib:name="Italy" d="M0 0 ...." style="fill:red; />
    <path id="2" attrib:name="Spain" d="M10 10 ...." style="fill:blue;" />
    <path id="3" attrib:name="Germany" d="M20 20 ...." style="fill:green; />
    ...
  </g>
</svg>
```

### 5.7.1 HOW TO GET A GEOGRAPHY SVG MAP ?

If you don't have a geographic svg map you can generate it in different ways, for example:



- converting a shapefile into an svg map usign some tools that you can download from internet
- Using some gis/map tools, like for example GeoServer o MapServer.
- Converting data contained into a postgres database usign the postgres tools for svg conversion
- Extracting data from an opengis database and converting it into svg with a a custom program.

Once you have the svg map be sure that the geometry element ids correspond to the ids contained into your database.

### 5.7.2 WRITE A MAP TEMPLATE FILE

The template file is divided into some main parts

```
<?xml version="1.0" encoding="UTF-8"?>
<MAPS>
  <MAP name="name of the map" enabled="TRUE" type="svg">
    <MAP_PROVIDER />
    <DATAMART_PROVIDER />
    <CONFIGURATION />
  </MAP>
</MAPS>
```

The 'Map Provider' element is useful to configure how the svg map will be retrived:

```
<MAP_PROVIDER class_name="it.eng.geo.map.DefaultMapProvider"
  uri="url to your a svg map"
  xml_parser_impl="it.eng.geo.document.XercesXMLDocument" />
```

The only attribute to change is 'uri' which must contains the url of your svg map. Typically the map file is placed into a folder of SpagoBi geo engine application, which is visible.

The 'Datamart Provider' element is useful to configure the data retrival from your database.

```
<DATAMART_PROVIDER class_name="it.eng.geo.datamart.DefaultDatamartProvider"
  registered_pool_name="spagobigeco"
  query="select id, value from ... where ..."
  element_id="id"
  element_name="value" />
```

You need to configure four attributes:

- registered\_pool: the name of the database connection pool configured into the data\_access.xml file (SPAGOBIGEO\_HOME/WEB-INF/conf/data\_access.xml). The connection pool is used to ge the connection towards the database.
- Query: the query which recovers the data associated to the svd element ids
- element\_id: the name of the column returned by the query which contains the ids related to the svg element ids.
- element\_name: the name of the column returned by the query which contains the value related to each svn element id. The value will be used to access the configuration tag of the map template file and extract the style to apply to the svg element.

The 'Configuration' element contains the styles, that have to be associate to the svg tags, based on the database attribute value. The styles are organized following a level hierarchy. Each style is associate to a range of values which is defined by two levels, the first one defines the low boundary while the second one defines the up boundary. When an svg element is analyzed the engine gets its attribute value from the database and using it recover the sytle associated to the low boundary level defined into the map template.

```
<CONFIGURATION>
  <LEGEND x="legend x position" y="legend y position"
    width="legend width" height="legend height" style="legend style">
    <TITLE description="legend title" style="legend style"/>
    <LEVELS>
      <LEVEL threshold="0" style="style of the level">
        <TEXT description="less than 100" style="style of the description" />
      </LEVEL>
      <LEVEL threshold="100" style="style of the level">
        <TEXT description="more than 100" style="style of the description" />
      </LEVEL>
      ...
    </LEVELS>
  </LEGEND>
</CONFIGURATION>
```

Each level tag has two attributes:

- threshold: defines the low boundary for the current level and the up boundary for the previous level.
- Style: the style associate to the level

The level tag contains a 'Text' tag where you can set the description of the level

It's possible to configure a legend of the map (which is based on the level definition). You can configure it filling the attributes of the tags 'legend' and 'title' which allow you to set the x/y position, the dimensions, the style and title of the legend.

### 5.7.3 DEFINE A MAP DOCUMENT INTO SPAGOBI

In order to configure a map document into SpagoBI you need to:

- be sure that you have a engine pointing to the SpagoBI Geo Engine
- define a new business object of type 'Map'.
- Setting into the new document the geo engine
- load the template map file as a template for the map document
- be sure that the Adobe Svg viewer plugin is installed on your browser

If your geo engine is well configured you can exec the document and you get a svg map.

## 6 In more depth

### 6.1 PORTAL ADMINISTRATOR AND PORTLETS ORGANIZATION

Portlets are autonomous and independent application windows. They are freely usable inside portal contexts, supporting the JSR 168 specification, by means of a simple configuration. No development is necessary.


Every function in SpagoBI runs in portlets included into a corporate portal or into a particular Business Intelligence environment.

The portlet organization into the portal is realized by the Portal Administrator.

SpagoBI releases specialized portlets according to the different user typologies (administrator, developer, tester, end-user).

Each user is assigned to a specific typology by the Portal Administrator.

---

 For a better understanding of the user typologies refer to the analytical Document life-cycle section.

---

### 6.2 ANALYTICAL DOCUMENT LIFE-CYCLE

Every SpagoBI document usually follows a three steps life-cycle:

1. **Development:** this is the proper state of every document that has to be developed, corrected, modified or improved, and, therefore, it is the initial state of every new document;
2. **Test:** it is the state of a document which has to be tested in order to check if it works correctly returning the requested result for each possible configuration;
3. **Released:** this is the state of a document that has been properly developed and tested and can be employed by the final user.

Moreover, a 4<sup>th</sup> state (**Suspended**) can be assigned to a document that will not be used anymore.

Referring to this life-cycle, SpagoBI users can have a specific function which is assigned by the portal administrator.

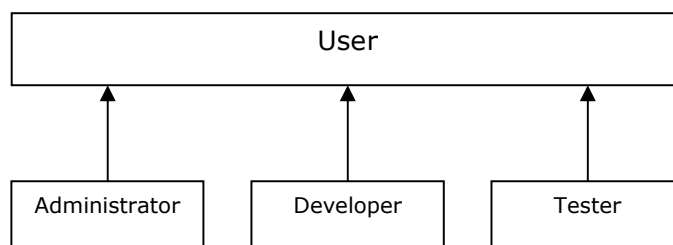
Users can be classified in 4 different typologies:

1. **Administrator:** he deals with configuration and security aspects.
2. **Developer:** this type of user can create or modify documents;
3. **Tester:** he takes the responsibility to verify the formal correctness of the registered documents and if they fulfil the requirements.
4. **User:** he can use all the business objects in a 'released' state, according to his role and with the modalities previously defined in the parameters configuration.

The *User* is characterized by his functional roles, which regulates:

- the analytical documents visibility;
- the visibility of the data shown by documents;
- the behaviour rules of their parameters and the filters.

It is very important to notice that administrators, developers and testers are also users and, therefore, they can act as specialised users with additional functions.



**Errore.**

**Figure 64 - User type hierarchy**

Every user will access a specialized main page that will contain specific tools.

When completed his own phase, a Developer can update the document state to Test, while a Tester, referring to test results, can change it to Development or to Released. The administrator is the only one who can modify a document state without any constraints allowing extraordinary maintenance of the documents. Notice that the simple user cannot modify the document state.

Finally, it is important to observe that in order to develop, test or execute a particular document, it is necessary to have specific rights which can only be assigned by the administrator. For a better understanding of the Security Policy please refer to next paragraph.

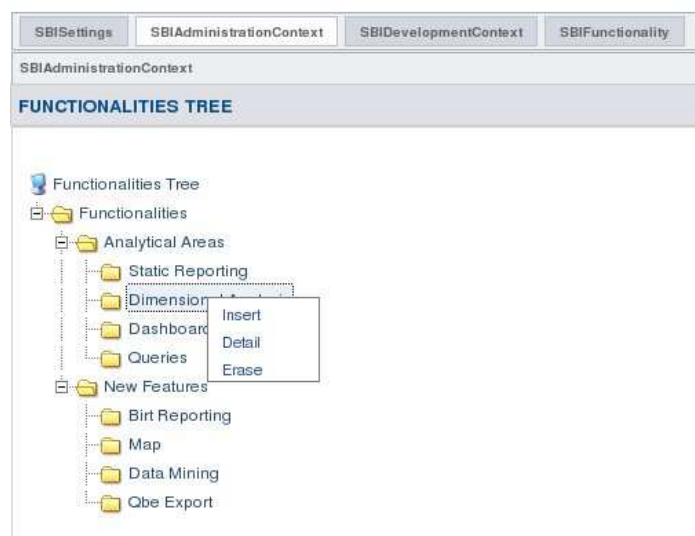
## 6.3 USER ROLES

Every user is characterized by one or more functional roles. SpagoBI manages users by their functional roles in order to regulate:

- the analytical documents visibility;
- the visibility of the data shown by documents;
- the behaviour rules of their parameters and the filters.

## 6.4 DOCUMENT ORGANIZATION AND SECURITY POLICY

SpagoBI sorts documents in a "Functionalities Tree" which is a File System that can be modified only by an administrator user.



**Figure 65 - Functionalities Tree**

This allows to better organize documents, grouping them by folders, and to realize a Security Policy. In fact, a user can develop, test or execute a document only if he has at least one role belonging to the corresponding permissions on the folder containing it. Only an administrator user can set these authorizations for each role and each folder.

For instance, in order to develop a document it is necessary:

- 1) to be defined as *Developer* by the portal administrator;
- 2) to have at least a role that belongs the *Development* rights on the folder that contains the document.

To execute a document it is required to:

- 1) to have at least a role that belongs the *Execute* rights on the folder that contains the document.

## 6.5 USER DEFINITION AND ROLES MANAGEMENT

Work in progress.



## 6.6 PORTAL DEFINITION

Work in progress.



## 6.7 DATA MART (.JAR) DEVELOPMENT FOR QBE FEATURE

### 6.7.1 BRIEF QBE INTRODUCTION

The Query By Example (QBE) is a one of the SpagoBI tools which offers a visual mode for data inquiry. It is based on hibernate technology and it presents a database schema as a group of related visual objects. Each database schema is modelled by a set of java classes and hibernate mapping files, all packaged in a jar file.

In order to insert a new database model and to inquire it with SpagoBI it's necessary to define a new **datamart** object. For this kind of object the template is the jar containing the hibernate class and mapping files.

Since QBE is based on hibernate it needs also to know the type of the target database and some connection informations (i.e. driver name, username, password, ecc...). All these database properties can be defined statically in the hibernate main configuration file (i.e. hibernate.cfg.xml) or passed as SpagoBI parameters when the datamart is inserted into the system. In the latter case the parameters to set are:

- one parameter with url\_name equals to '**JNDI\_DS**' which must be filled with the jndi name of a datasource for the target database. Qbe will use it to get the connection for the inquiry
- one parameter with url\_name equals to '**DIALECT**' which must be filled with the complete hibernate dialect class name suitable for the target database. Qbe will use it to choose the right sql dialect for the inquiry

### 6.7.2 HOW TO CREATE DATABASE MODEL JAR FILE

Since a database model jar is composed by a set of hibernate mapping files and relative java classes it's possible to create it manually, only using java and xml editors. However, this operation can be too long and error prone, so, it's better to use some automatic tools like 'HibernateTools-3.1'.

To get the 'HibernateTools-3.1' working we advise to follow these steps:

- download eclipse 3.1 sdk. It's strongly recommended to install the tools over a new and clean installation of eclipse 3.1. It's possible to download the 3.1 version from the url <http://download.eclipse.org/eclipse/downloads/index.php>
- install the eclipse platform: simply unzip the zipped downloaded file everywhere you want; a new folder named eclipse will be created (ECLIPSE-HOME)
- download JbossIDE 1.5 final tools file(which contains the hibernate tools). It's possible to download it selecting the JBossIDE-1.5-ALL.zip file from the link [http://sourceforge.net/project/showfiles.php?group\\_id=22866&package\\_id=72248&release\\_id=382687](http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=72248&release_id=382687)
- install the JBoss IDE tools over the eclipse platform: unzip the JBossIDE-1.5-ALL.zip file; a new folder named eclipse will be created (TOOLS-HOME); copy all the contents of TOOLS-HOME/plugins and TOOLS-HOME/features into respectively ECLIPSE-HOME/plugins and ECLIPSE-HOME/features folders.
- Start the eclipse platform
- to test if the hibernate tools are working open the view 'Navigator' and then into the navigator box press the right button of the mouse and select 'New/others' menu. Into the new window you should be able to see an hibernate folder(Figure 66 - Eclipse Navigator and Figure 67 - Hibernate options).

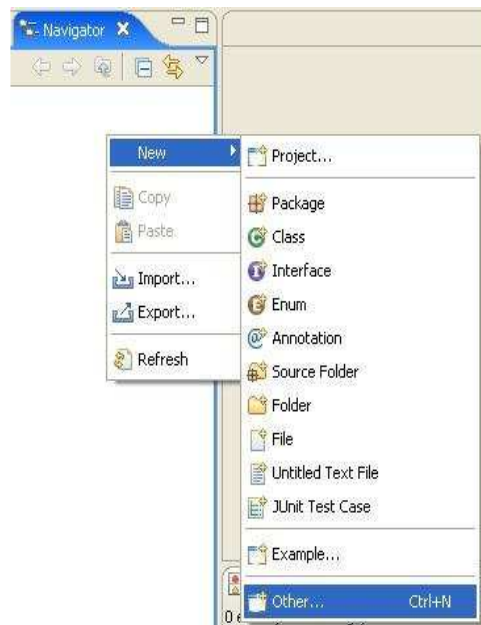


Figure 66 - Eclipse Navigator

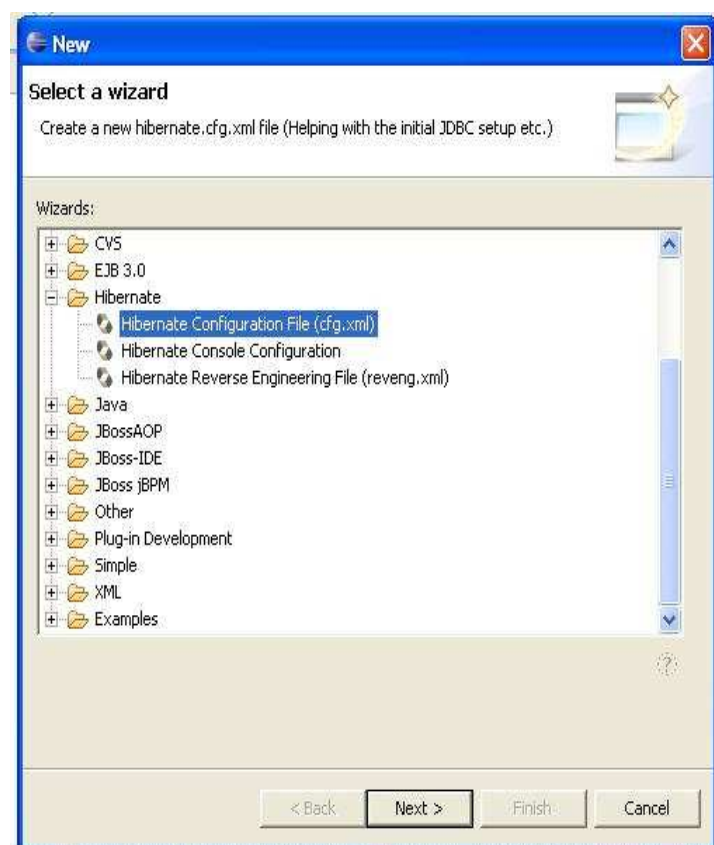


Figure 67 - Hibernate options



Once installed and tested the hibernate tools create a new java project (give it a name and maintain the default properties). Look at Figure 68 - New Java Project and Figure 69 - Default properties of a Java project.

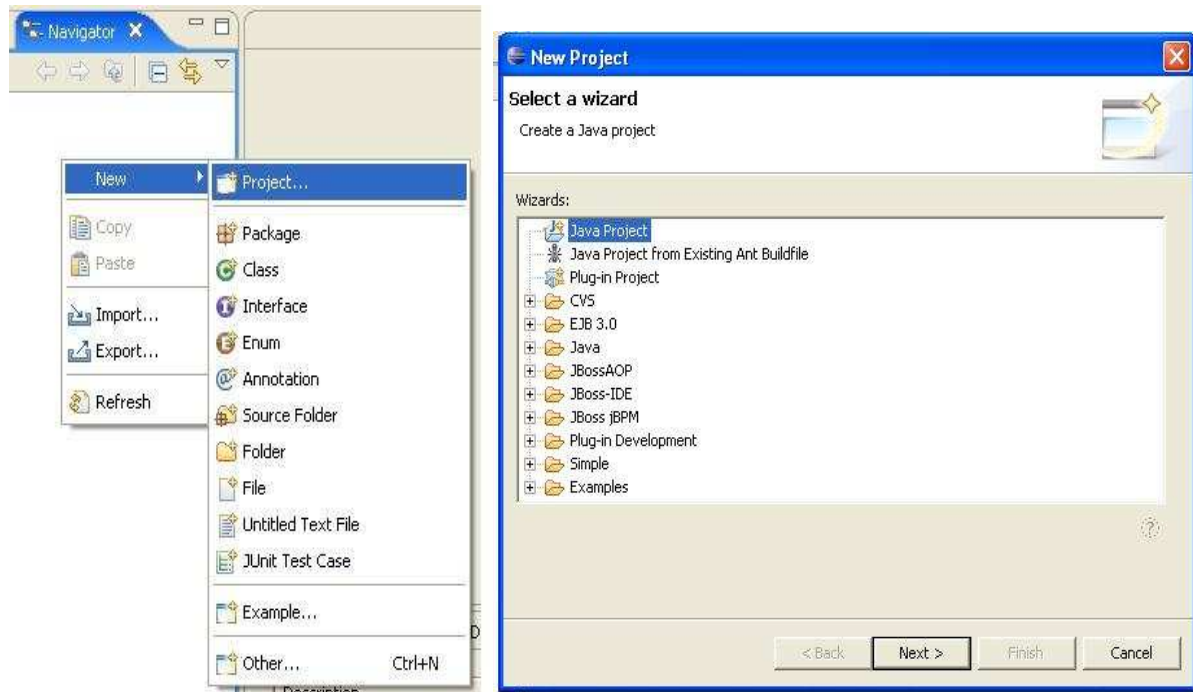


Figure 68 - New Java Project

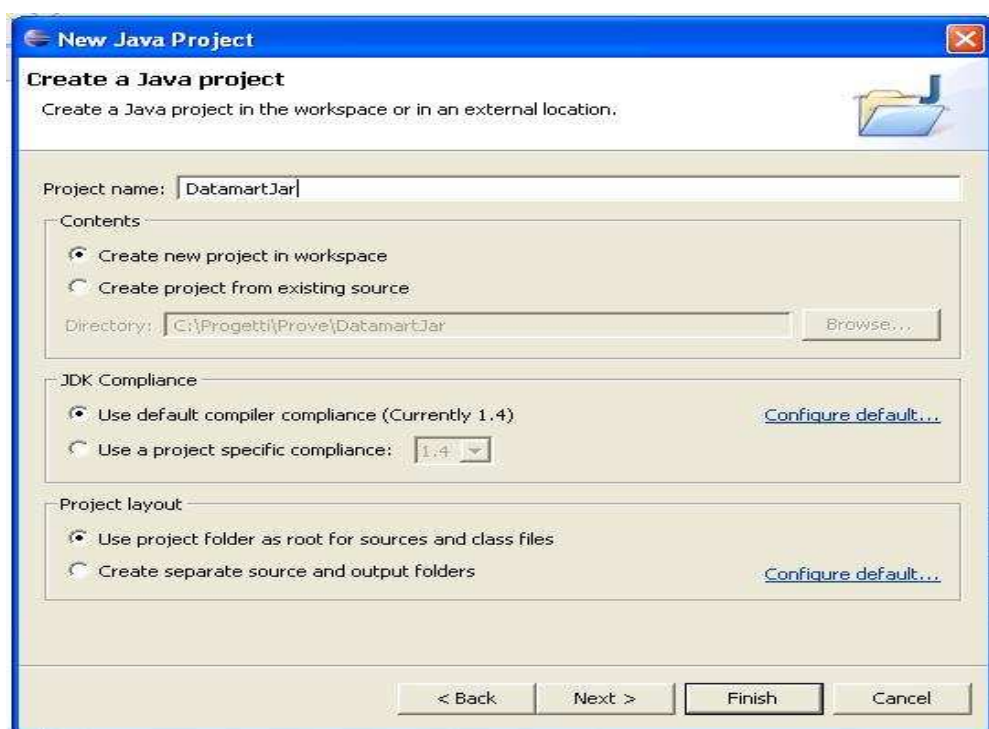
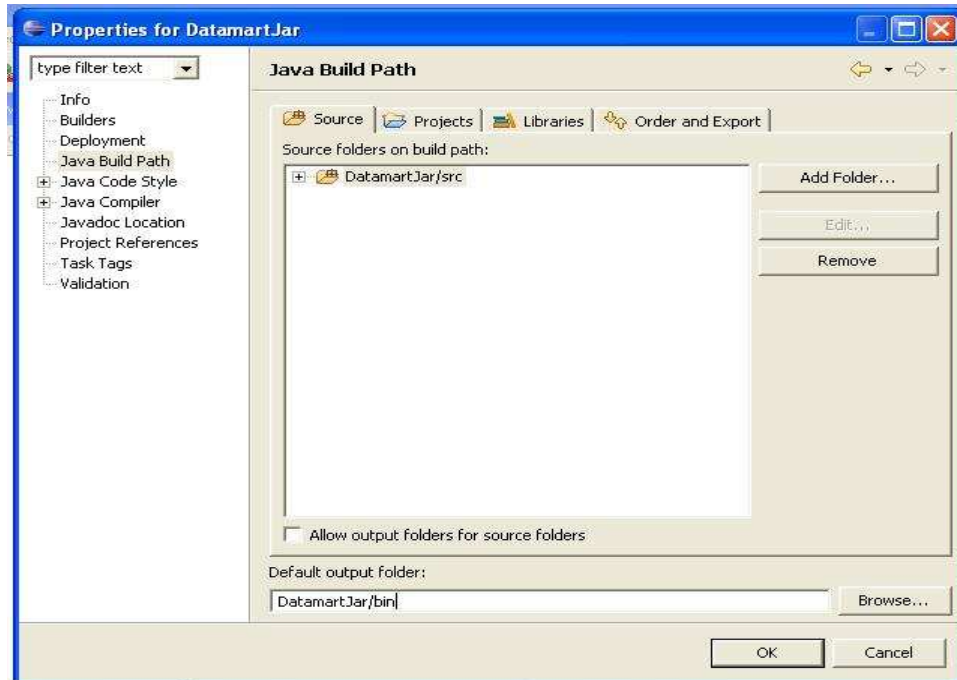


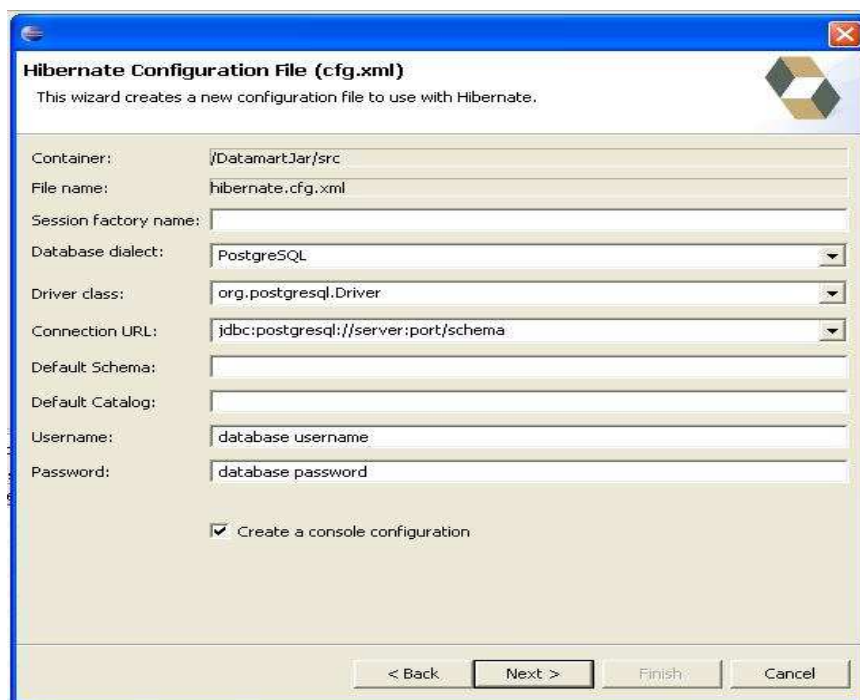
Figure 69 - Default properties of a Java project

Create two project subfolders called 'src' and 'bin'; enter the project properties and select the 'Java Build Path' menu. Into the properties window set the src folder as source folder and the bin folder as output folder (Figure 70 - Java project build path options).

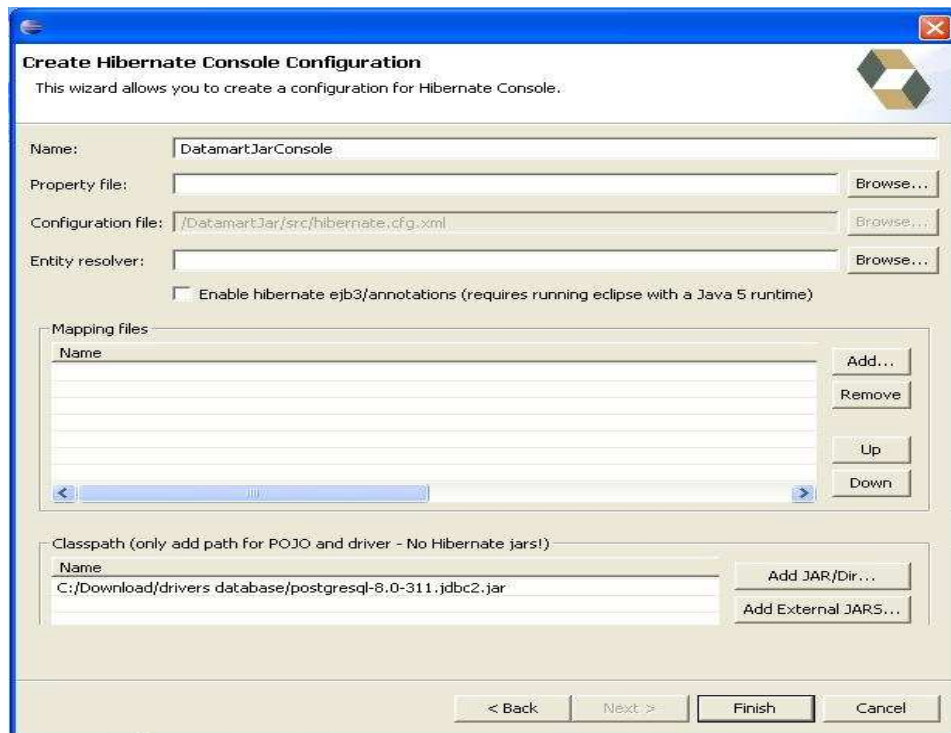


**Figure 70 - Java project build path options**

create a new Hibernate cfg file into the src folder of the project (right click on the project and then select the 'new/other/hibernate/hibernate cfg' file menu item). Into the properties window choose the right dialect for your database, insert the driver class name, the connection string, user and password. Remember to check the 'create console configuration' option and then press the next button (Figure 71 - New Hibernate Configuration File and Figure 72 - New Hibernate Configuration File).

**Figure 71 - New Hibernate Configuration File****Figure 72 – New Hibernate Configuration File**

Give a name to the console, add the jar of the database driver into the classpath and then press the finish button (Figure 73 - New Hibernate Console Configuration)



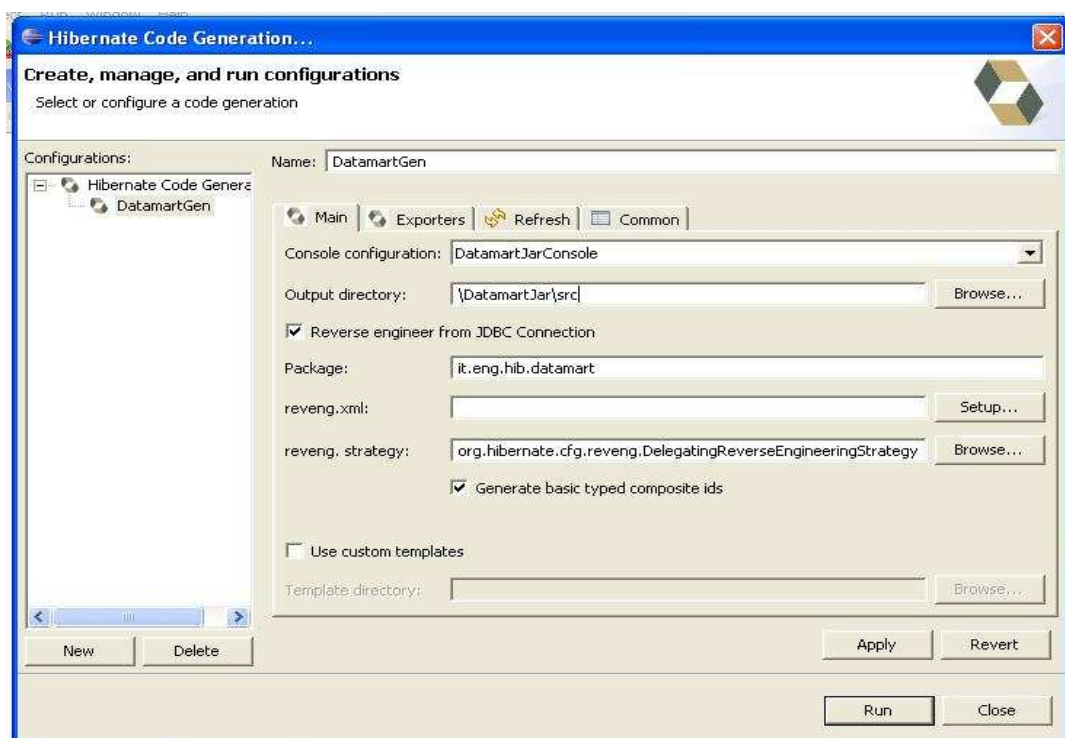
**Figure 73 - New Hibernate Console Configuration**

Exec the hibernate code generation tool (Figure 74 - Hibernate code generation)



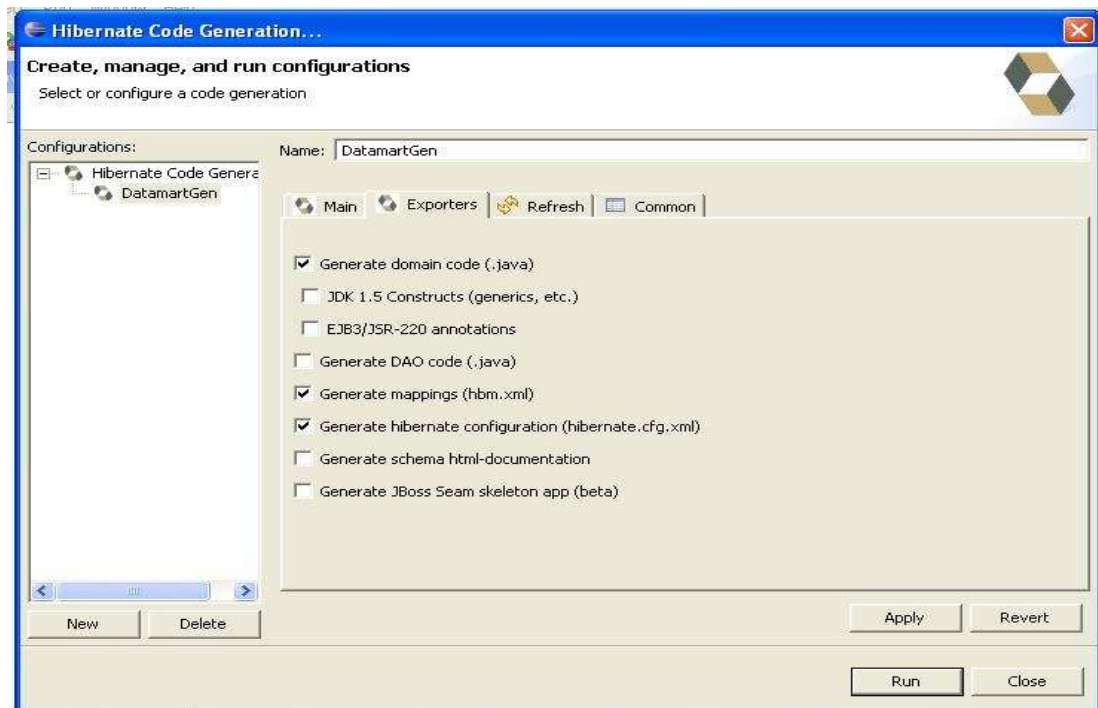
**Figure 74 - Hibernate code generation**

In the main tab give a name to the generation, select as output folder the src folder of the project, check the 'reverse engineer from jdbc connection' option, give a package name for the generated java classes and fill the 'reveng strategy' field with a '**org.hibernate.cfg.reveng.DelegatingReverseEngineeringStrategy**' value (Figure 75 - Hibernate code generation main options).



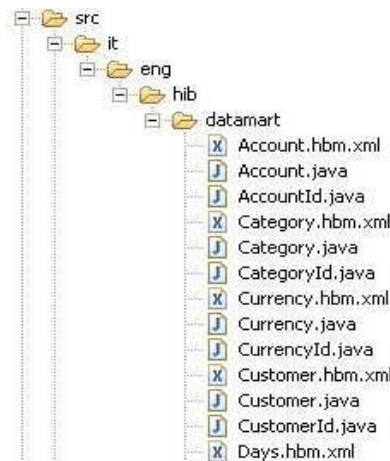
**Figure 75 - Hibernate code generation main options**

If you want to filter the tables to process for the generation click on the 'setup' button and create a new reverse engineering file. Into the new window select the 'refresh' button and wait until the schema of the database is showed into the left box. From the database schema select the table you want to include and then press the include button. All the tables in the right box will be processed by the tool. Press the 'Exporters' Tab and check the three options as in the Figure 76 - Hibernate code generation exporters options. The third option is not compulsory. Just check it to generate the hibernate main configuration file used than by the engine to automatically configure the connection to the database. Doing that you haven't to configure by hand the **JNDI\_DS** and **DIALECT** parameter (see above).



**Figure 76 - Hibernate code generation exporters options**

At the end Press the 'Apply' button and then the 'Run' Button. After the end of the execution into the src folder of the project there will be a set of java and hibernate mapping files (contained into the right package folder structure). See Figure 77 - Java classes and Hibernate mapping files.



**Figure 77 - Java classes and Hibernate mapping files**

If the database is well-defined with all the appropriate primary and foreign keys there's no need to change the code produced, but, if some problem occurs, or you are an hibernate expert and you want to personalize the code, it obviously possible to edit each file and make the changes.



By default the qbe tools will show the objects into the graphical interface using the native names of the database table, adding to it the complete package name of the hibernate classes. This behaviour can be changed adding to the source directory a 'label.properties' file. This file must contain a set of association between the native object name and the label to show into the graphical interface. An example of this file is listed below:

```
class.it.foodmart.Promotion=Promotion
class.it.foodmart.Store=Store
class.it.foodmart.Product=Product
field.customerId=Customer Identifier
field.accountNum=Account Number
field.lname=Surname
```

The part on the left side of the equal sign is the complete name of the hibernate object (with a prefix class for the tables and field for table fields). The QBE tool will use the right part of each row as a graphical label for the correspondent object. The file 'label.properties' is the default one and if you need to define your labels in only one language it's enough, but, if you want to translate each label in different languages, you have to define one properties file for each language. Each file must be named with the following rule:

label\_<<ISO country code>>.properties

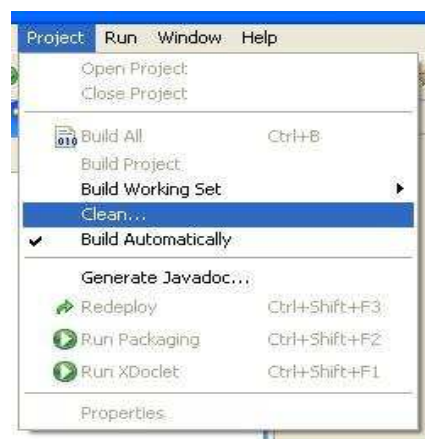
So the files for english / italian labels will be named:

label\_en.properties

label\_it.properties

Qbe will automatically choose the correct file based on the portal or browser language. (in case there isn't a suitable file it will use the default one).

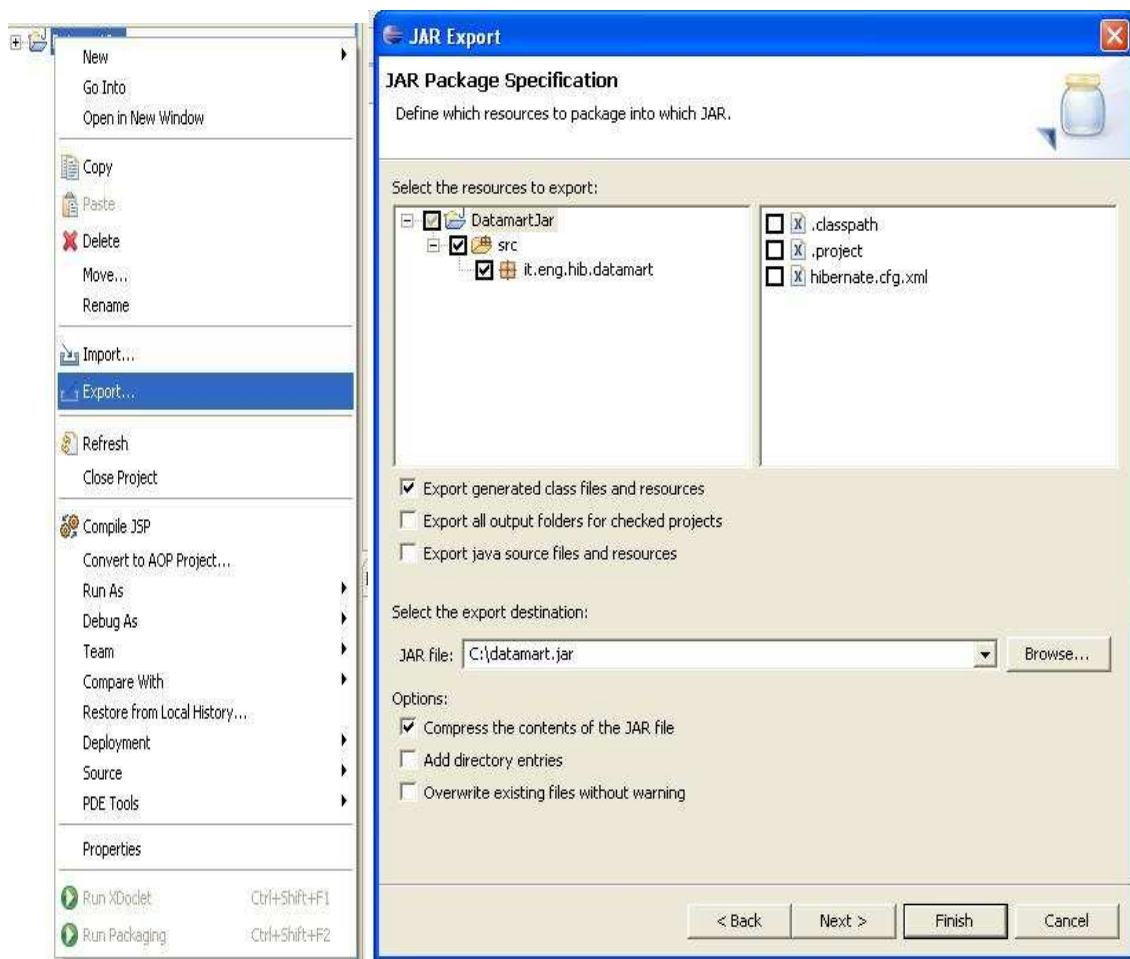
To create the jar file first compile the project selecting the Projects/clean menu (Figure 78 - Compile project)



**Figure 78 - Compile project**

and then export the project as a jar file. Into the export window properties select the src folder of the project, give a destination for the produced jar and then press the 'next' button, 'next' button another time and 'finish' button at the end (Figure 79 - Export jar file).



**Figure 79 - Export jar file**

The datamart jar model can be taken from the destination directory and loaded as template into a SpagoBI datamart Object.

## 7 Glossary

---

Work in progress.



---

Analytical document  
Portal  
Analytical portal  
Parameter  
LOV (list of values)  
User role  
Report  
OLAP  
Data Mining  
Dashboard  
Scorecard