

## SpagoBI QuickStart

### Authors

Luca Scarel  
Grazia Cazzin  
Davide Zerbetto  
Luca Fiscato

## Index

<b>VERSION .....</b>	<b>5</b>
<b>1. DOCUMENT GOAL.....</b>	<b>5</b>
<b>2. REFERENCES .....</b>	<b>6</b>
<b>3. HELP FOR LECTURE.....</b>	<b>6</b>
3.1. STYLISTIC CONVENTIONS.....	6
3.2. SPECIAL SECTIONS .....	6
<b>4. SPAGOBI - CONCEPTUAL OVERVIEW.....</b>	<b>8</b>
<b>5. RECURRING THEMES.....</b>	<b>14</b>
5.1. PORTLET LAYOUT .....	14
5.2. LIST AND DETAILED VIEW .....	14
<b>6. FUNCTIONALITY OVERVIEW.....</b>	<b>16</b>
6.1. SETTINGS.....	16
6.1.1. <i>Engines Configuration</i> .....	16
6.1.2. <i>Roles synchronization</i> .....	18
6.1.3. <i>Import/Export</i> .....	19
6.1.4. <i>Export</i> .....	20
6.1.5. <i>Import</i> .....	21
6.2. ADMINISTRATOR .....	24
6.2.1. <i>Functionalities Management</i> .....	25
6.2.2. <i>Document Configuration</i> .....	27
6.2.3. <i>Schedule a Document Execution</i> .....	30
6.3. DEVELOPER .....	35
6.3.1. <i>Predefined List of Value (LOV)</i> .....	36
6.3.2. <i>Predefined Values Constraints</i> .....	39
6.3.3. <i>Parameters Management</i> .....	41
6.3.4. <i>Document configuration</i> .....	43
6.3.5. <i>Parameters correlation</i> .....	45
6.3.6. <i>Links</i> .....	48
6.3.7. <i>Documents visibility</i> .....	50
6.4. TESTER .....	51
6.5. END-USER.....	54
6.6. EVENTS.....	56
<b>7. GETTING STARTED WITH SPAGOBI.....</b>	<b>57</b>
7.1. INSTALL eXO PORTAL TOMCAT AND SPAGOBI DEMO .....	57
7.2. REPORT .....	57
7.2.1. <i>Create a Report template</i> .....	58
7.2.2. <i>Create a Report template using iReport</i> .....	58
7.2.3. <i>Create a Report template using Birt</i> .....	59
7.2.4. <i>Create a Parameter</i> .....	60
7.2.5. <i>Predefined List of Value (LOV)</i> .....	60
7.2.6. <i>Predefined Values Constraints</i> .....	62
7.2.7. <i>Parameters Management</i> .....	62
7.2.8. <i>Register the Analytical Document (the built report) into the platform</i> .....	63

7.2.9.	<i>Test the Analytical Document</i>	65
7.2.10.	<i>Execute the Analytical Document</i>	65
7.3.	<b>OLAP ANALISYS</b>	65
7.3.1.	<i>Create a template</i>	65
7.3.2.	<i>Create Parameters</i>	66
7.3.3.	<i>Predefined List of Value (LOV)</i>	67
7.3.4.	<i>Predefined Values Constraints</i>	68
7.3.5.	<i>Parameters Management</i>	68
7.3.6.	<i>Register the Analytical Document (the built OLAP) into the platform</i>	69
7.3.7.	<i>Test the Analytical Document</i>	71
7.3.8.	<i>Execute the Analytical Document</i>	71
7.3.9.	<i>Dynamic Template Creation</i>	71
7.3.10.	<i>Profiled data</i>	75
7.4.	<b>DASHBOARD</b>	76
7.4.1.	<i>Create a list of value</i>	77
7.4.2.	<i>Create a template</i>	78
7.4.3.	<i>Register the Analytical Document (the built DASHBOARD) into the platform</i>	78
7.4.4.	<i>Test the Analytical Document</i>	79
7.4.5.	<i>Execute the Analytical Document</i>	79
7.5.	<b>DATA MINING</b>	79
7.5.1.	<i>Create a Data mining process using Weka KnowledgeFlow</i>	80
7.5.2.	<i>Parametrize process flow</i>	81
7.5.3.	<i>Create a Parameter</i>	82
7.5.4.	<i>Predefined List of Value (LOV)</i>	82
7.5.5.	<i>Predefined Values Constraints</i>	84
7.5.6.	<i>Parameters Management</i>	84
7.5.7.	<i>Register the Analytical Document (the built Data-Mining Flow) into the platform</i>	85
7.6.	<b>QUERY BY EXAMPLE</b>	87
7.6.1.	<i>Field Selection</i>	88
7.6.2.	<i>Condition</i>	89
7.6.3.	<i>View Query</i>	90
7.6.4.	<i>Save Query</i>	91
7.6.5.	<i>Execute query</i>	92
7.6.6.	<i>Generate Template</i>	92
7.6.7.	<i>Sub-Query</i>	93
7.6.8.	<i>Create View</i>	95
7.6.9.	<i>Calculated Fields</i>	96
7.6.10.	<i>Document Template and Profiled data</i>	101
7.7.	<b>MAPS</b>	102
7.7.1.	<i>How to get a geography svg map ?</i>	104
7.7.2.	<i>Write a map template file</i>	104
7.7.3.	<i>Define a Map document into SpagoBI</i>	105
7.7.4.	<i>Parameters configuration</i>	105
7.8.	<b>OFFICE DOCUMENT</b>	107
7.9.	<b>BOOKLETS</b>	108
7.9.1.	<i>Creation of a new booklet</i>	108
7.9.2.	<i>Configuration of the booklet template</i>	108
7.9.3.	<i>Create a presentation template</i>	110
7.9.4.	<i>Change booklet workflow process</i>	111
7.9.5.	<i>Execution of a booklet</i>	113
7.10.	<b>ETL PROCESS</b>	115
7.10.1.	<i>Configure Talend Open Studio (TOS) 2.1.0</i>	115
7.10.2.	<i>Deploy a job from TOS to SpagoBITalendEngine</i>	116
7.10.3.	<i>Create SpagoBI document</i>	116
7.10.4.	<i>Parameters configuration</i>	117
7.10.5.	<i>Context run-time selection</i>	119
7.11.	<b>DOCUMENTS CROSS NAVIGATION</b>	120

7.11.1.	SPAGOBIJASPERREPORTENGINE.....	121
7.11.2.	SPAGOBIBIRTREPORTENGINE.....	123
7.11.3.	SPAGOBIQBEENGINE .....	125
7.11.4.	SPAGOBIGEOENGINE .....	125
7.11.5.	SPAGOBIJPIVOTENGINE.....	126
<b>8.</b>	<b>IN MORE DEPTH.....</b>	<b>129</b>
8.1.	PORTAL ADMINISTRATOR AND PORTLETS ORGANIZATION.....	129
8.2.	ANALYTICAL DOCUMENT LIFE-CYCLE.....	129
8.3.	USER ROLES .....	130
8.4.	DOCUMENT ORGANIZATION AND SECURITY POLICY .....	130
8.5.	USER DEFINITION AND ROLES MANAGEMENT.....	131
8.6.	PORTAL DEFINITION.....	131
8.7.	DATA MART (.JAR) DEVELOPMENT FOR QBE FEATURE .....	131
8.7.1.	<i>Brief QBE introduction.....</i>	<i>131</i>
8.7.2.	<i>How to create database model jar file .....</i>	<i>132</i>
<b>9.</b>	<b>GLOSSARY .....</b>	<b>141</b>

### **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		
<b>Version/Release n°:</b>	0.9.1	<b>Data Version/Release:</b>	October, 9 <sup>th</sup> 2006
<b>Update description:</b>	Fifth release - Draft		
<b>Version/Release n°:</b>	0.9.2	<b>Data Version/Release:</b>	January, 24 <sup>nd</sup> 2007
<b>Update description:</b>	Sixth release - Draft		
<b>Version/Release n°:</b>	0.9.3	<b>Data Version/Release:</b>	July, 18 <sup>th</sup> 2007
<b>Update description:</b>	Seventh 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

## 2. 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

## 3. Help for lecture

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

### 3.1. 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.

### 3.2. SPECIAL SECTIONS

---

Note




---

Example




---

Reference to other section




---

In revision phase



---

Future implementation. To be done.



---

Advice for the reading of the section

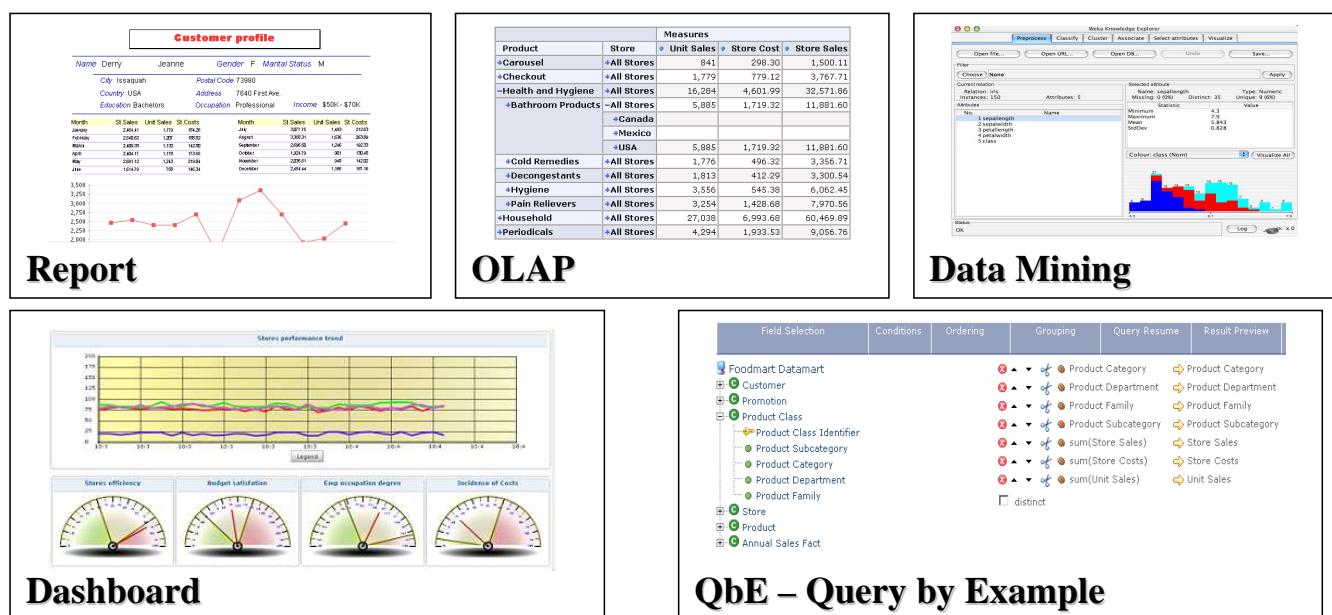


## 4. SpagoBI - Conceptual overview

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, ETL processes.



**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.

### Analytical document

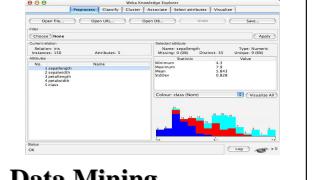
**Report**



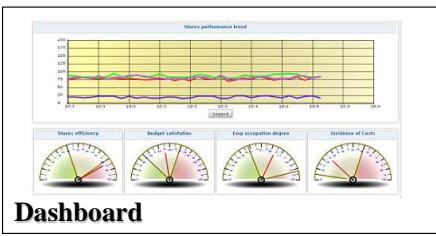
**OLAP**



**Data Mining**



**Dashboard**



**QbE – Query by Example**



**Structural configuration**

**Engine and Driver**

**CMS data:** template, version and location in the functional tree

**Executive configuration**

**Parameters**

**Figure 2 – Analytical documents support**

Regarding to the **structural configuration**, every analytical documents' category refers to a particular (one or many) execution engine: this engine can be internal or external respect to the SpagoBI core; in case it is external, it interacts with the core 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:

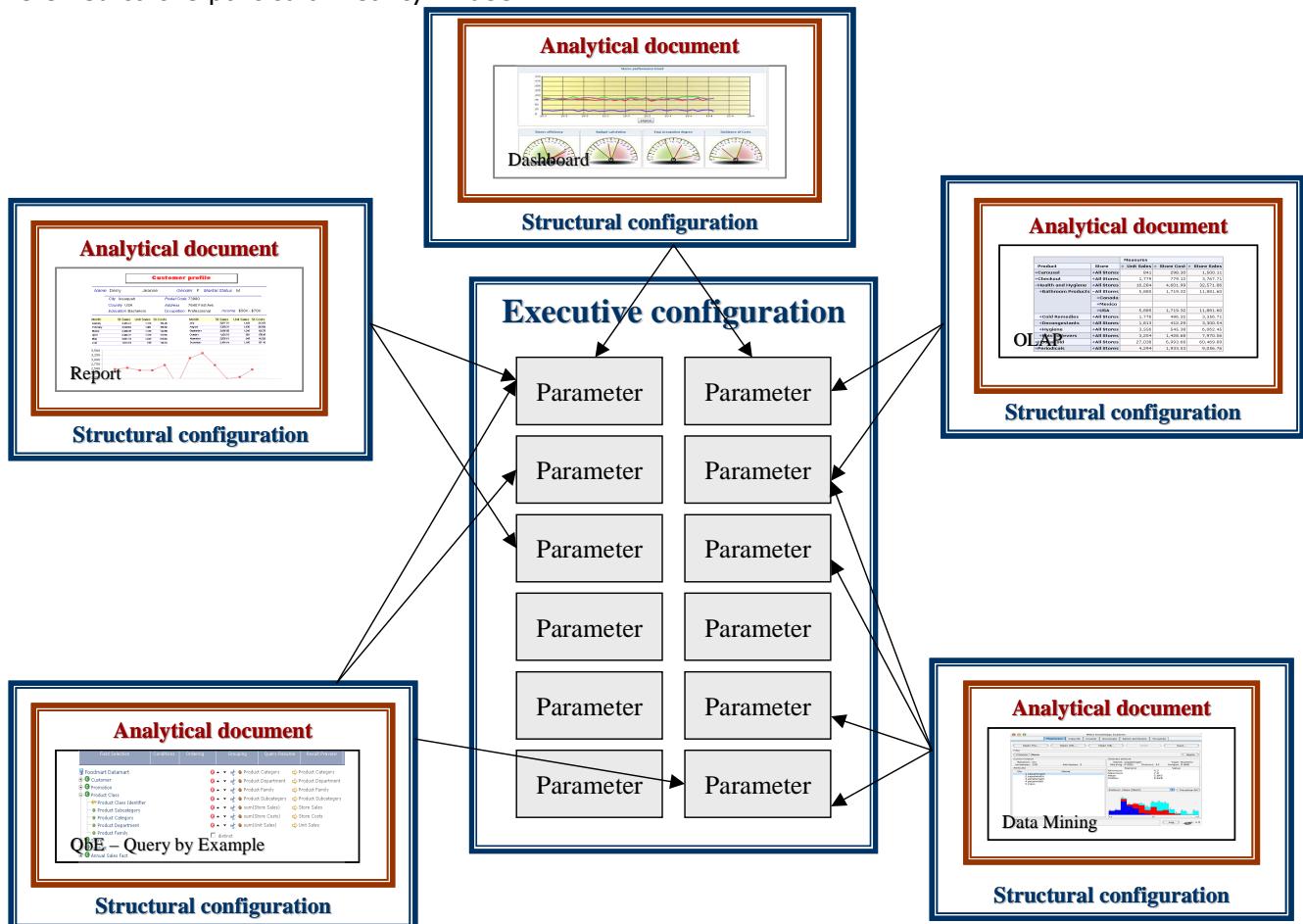
- report on the JasperReport engine and Birt engine;
- OLAP on the Mondrian engine, with Jpivot interface;
- Dashboard on the OpenLaszlo engine;
- free inquiry on a Hibernate implementation;
- data mining model on Weka engine;
- ETL processes designed with Talend.

 The Dashboard implementation by means of OpenLaszlo 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.

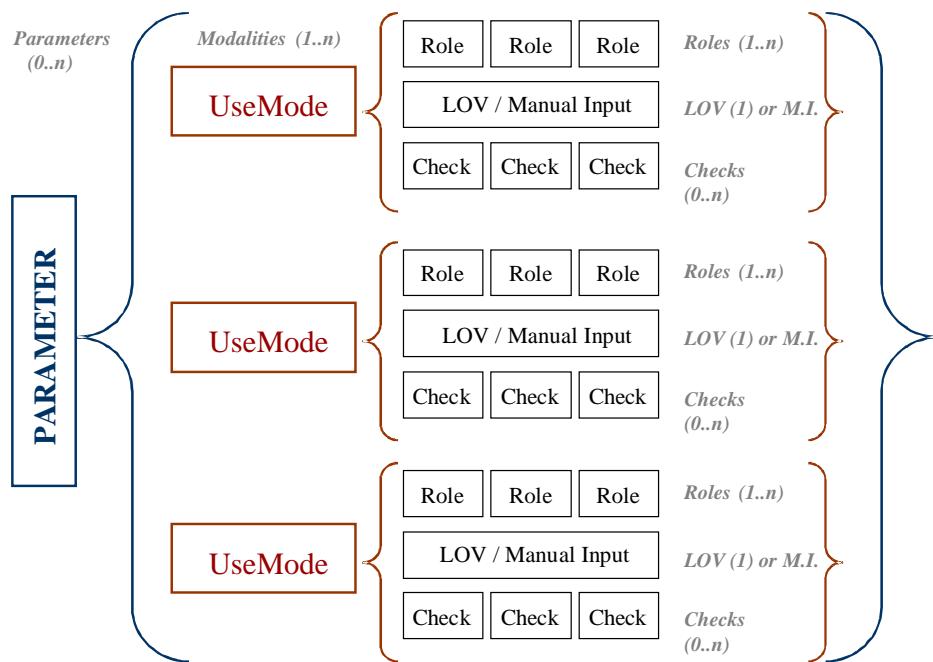
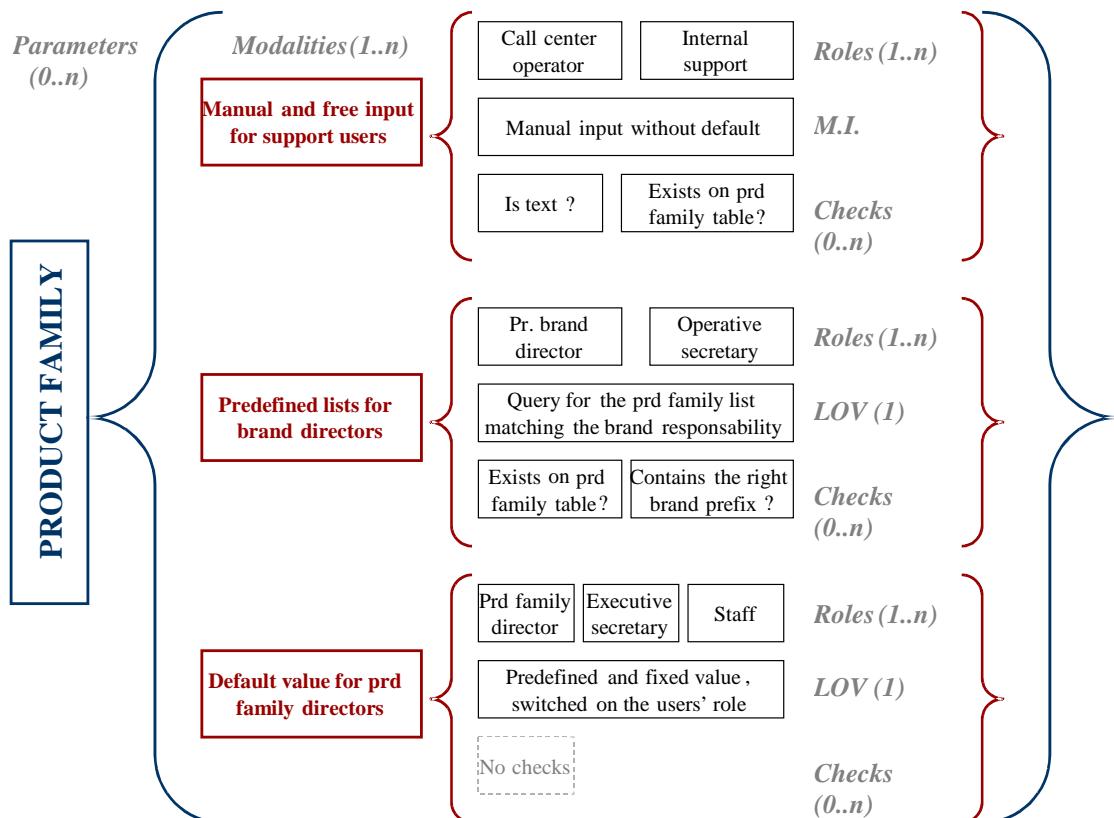


**Figure 3 – Executive configuration**

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.

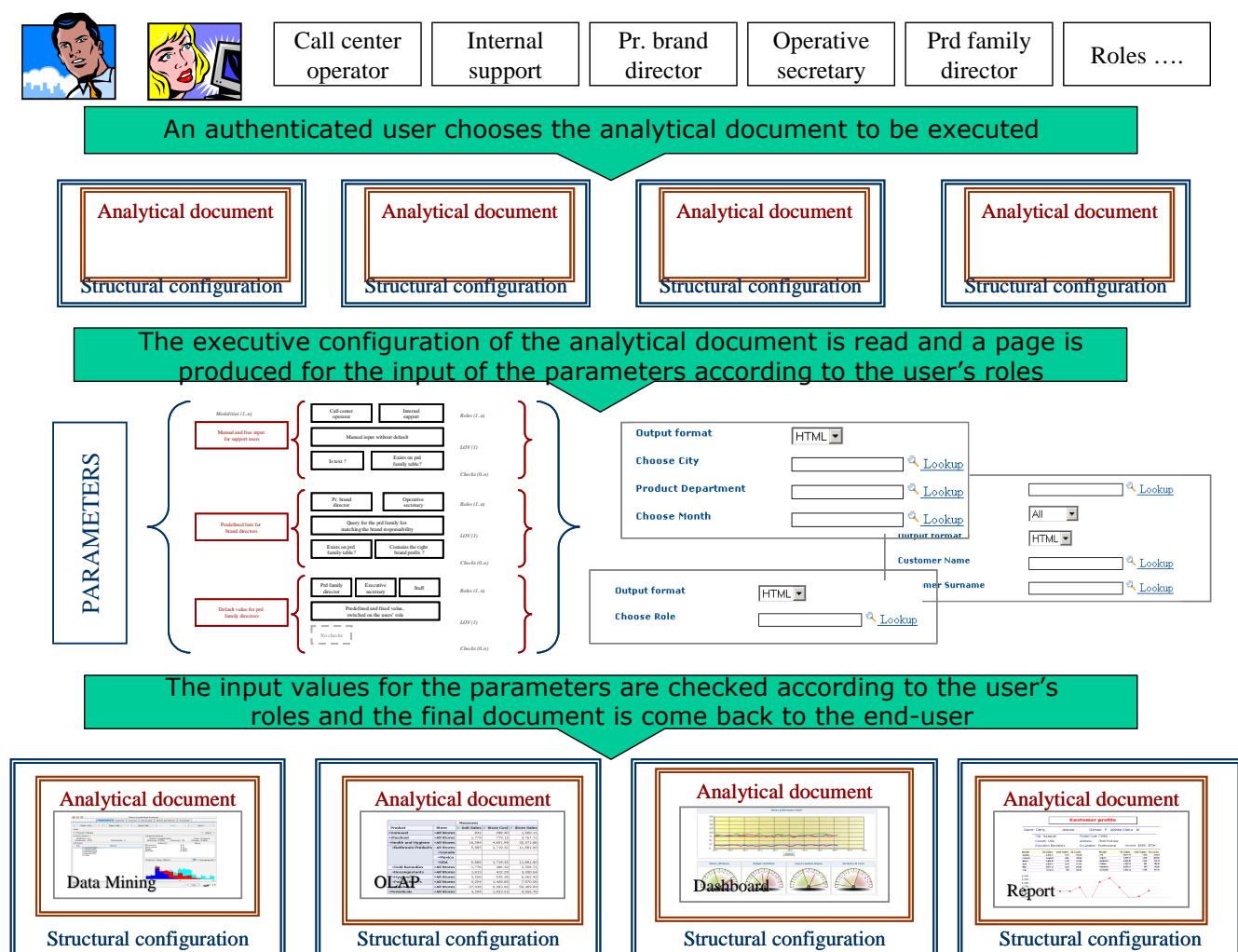

**Figure 4 – Parameter's abstract structure**

**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.

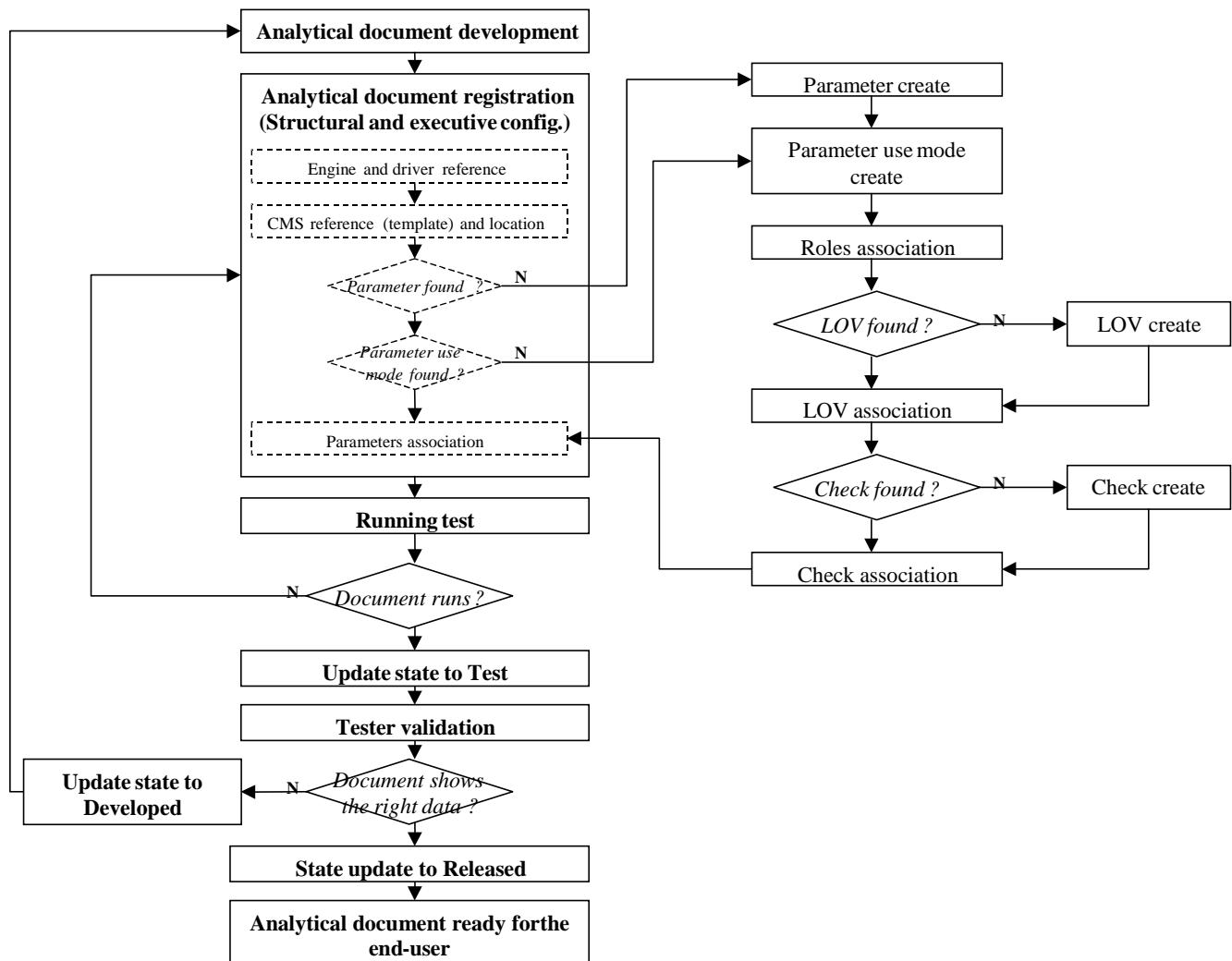
When a user (with its role) runs an analytical document, the structural information are read and then a custom page for the parametric input is produced on the basis of the execution information. At the end the document is produced on the basis of the inserted values (explicitly or implicitly).

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.

## 5. Recurring themes



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

### 5.1. 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.

### 5.2. 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.

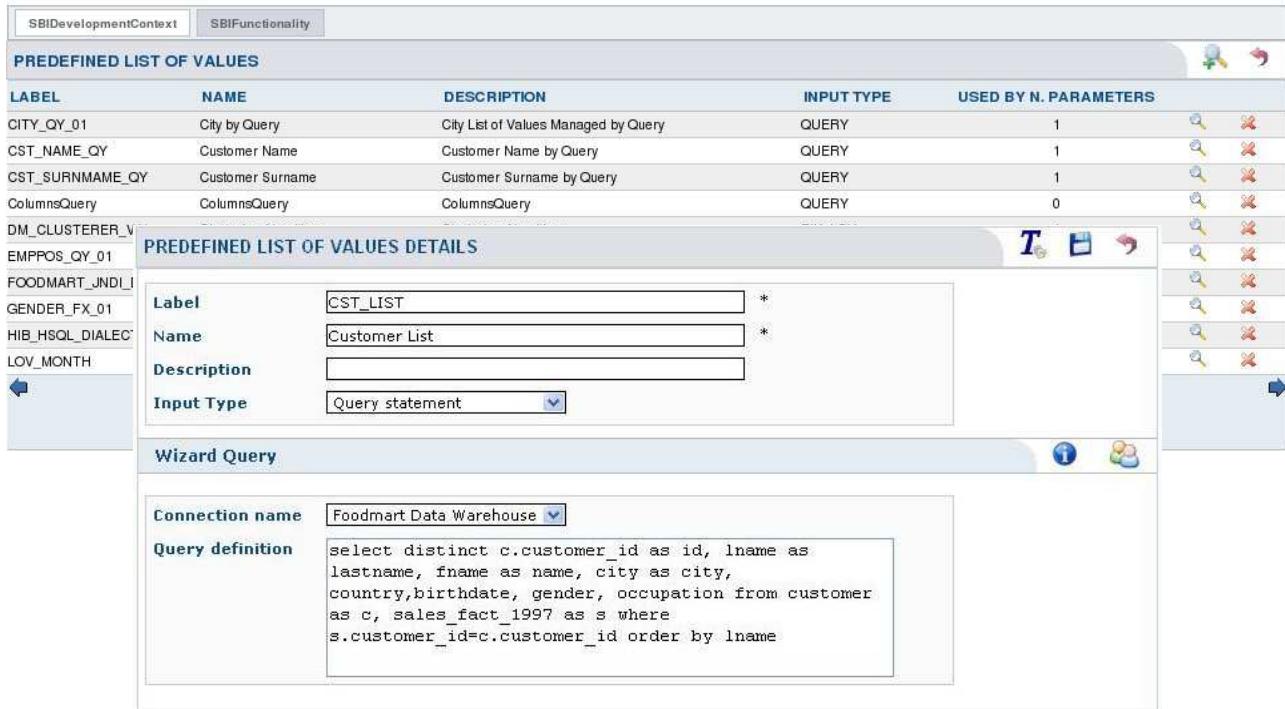
On the right side of every row, some icons drive the operativeness on the single element (row) of the list. The main possible functions are (where admitted):

- 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 shows the SpagoBI interface with two main windows:

- PREDEFINED LIST OF VALUES**: A table listing various predefined lists. One entry is selected, showing its details in the second window.
- PREDEFINED LIST OF VALUES DETAILS**: Displays the details for the selected entry ('CST\_LIST'). Fields include Label (CST\_LIST), Name (Customer List), Description (empty), and Input Type (Query statement). A query definition is also shown: 

```
select distinct c.customer_id as id, lname as lastname, fname as name, city as city, country, birthdate, gender, occupation from customer as c, sales_fact_1997 as s where s.customer_id=c.customer_id order by lname
```

.

**Figure 8 – List-detail Example**

## 6. Functionality overview

### 6.1. 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 6.1) using the standard demo settings.



**Figure 9 - Settings portlet**

#### 6.1.1. 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, Maps, Visual Inquiry, ETL processes 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 external engines it is very important to set their 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*.

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

**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`).

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="button" value="Datamart Model"/>
Engine type	<input type="button" value="Internal"/>
Class	<input type="text" value="it.eng.spagobi.engines.datamart.SpagoBIQbeInternalE"/>

**Figure 11 - Internal engine details**

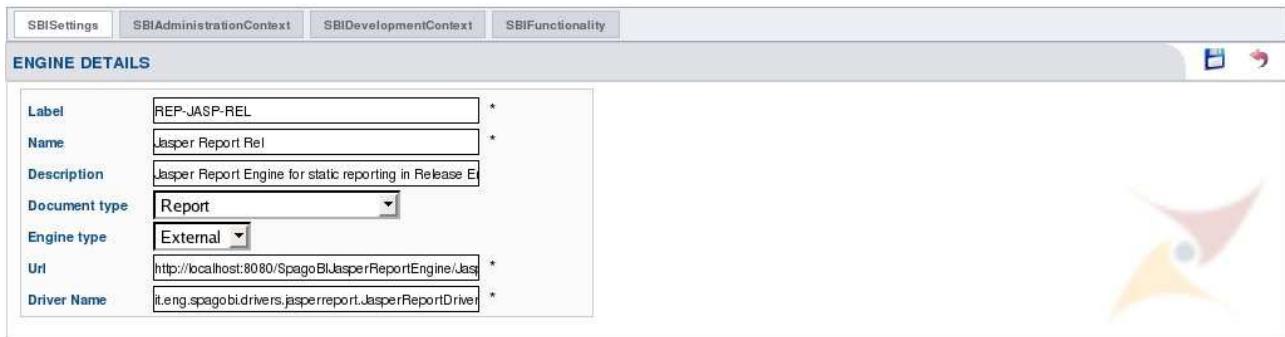
The current SpagoBI version uses the following internal engines:



- *Dashboard Internal Engine*: for Dashboards;
- *Office Document Internal Engine*: for various format static documents;
- *Booklet Internal Engine*: for booklets creation.

If 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 a configuration screen for an external engine. At the top, there are tabs: SBI Settings, SBI Administration Context, SBI Development Context, and SBI Functionality. Below the tabs, the title 'ENGINE DETAILS' is displayed. On the right side of the screen, there is a small logo of a stylized bird or arrow pointing upwards. The configuration form contains 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/SpagoBI/JasperReportEngine/Jas...	*
Driver Name	it.eng.spago.bi.drivers.jasperreport.JasperReportDrive	*

**Figure 12 - External engine details**

The current SpagoBI version uses the following external engines:



- *JasperReport*: for Report analysis;
- *Birt*: for Report analysis;
- *Mondrian-JPivot*: for OLAP analysis;
- *Weka*: for Data mining models;
- *Geo*: for Maps;
- *Talend*: for ETL processes;
- *Qbe*: for Visual Inquiry analysis;
- *JPivotXMLA*: client to a XMLA server for OLAP analysis.

### 6.1.2. 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.

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  starts with

**Figure 13 - Roles list**

The administrator can:

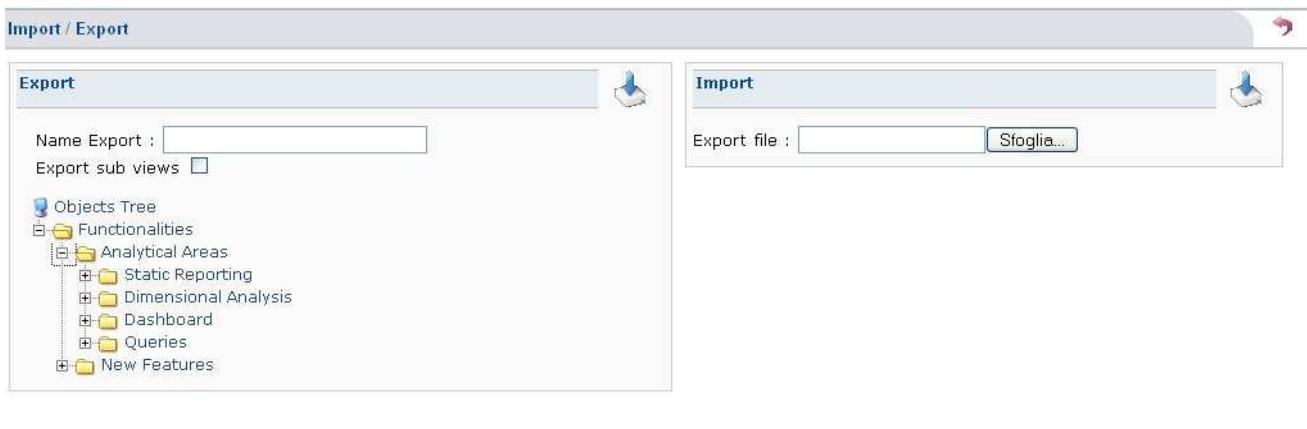
- erase an existing role (if that role is not used by any functionality or parameter use)
- synchronize SpagoBI's roles with portal's roles by clicking on the green two-arrows button in the Roles List page.

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

### 6.1.3. 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**

As best practices, we suggest to:

1. when the server is stopped, edit file importexport.xml in spagobi\WEB-INF\conf\components\importexport folder, set 'exportFolder' attribute in 'EXPORTER' tag and 'tmpFolder' in 'IMPORTER' tag as short absolute paths, for example:

```
<EXPORTER class="it.eng.spagobi.importexport.ExportManager"
          exportFolder="/temp/exports" />
<IMPORTER class="it.eng.spagobi.importexport.ImportManager"
          tmpFolder="/temp/import"/>
```

in order to avoid problems related to long paths (make sure that the user who starts the server has the write permission on those folders). Then you can restart the server;

2. during export operation, we suggest to exclude documents with no templates.

SpagoBI import procedure imports roles, parameters, engines, folders and documents with their templates.

When importing into a clean new environment, the procedure is not able to perform all the operations that must be physically performed on external engines: the following operations are not performed by the procedure and must be manually performed:

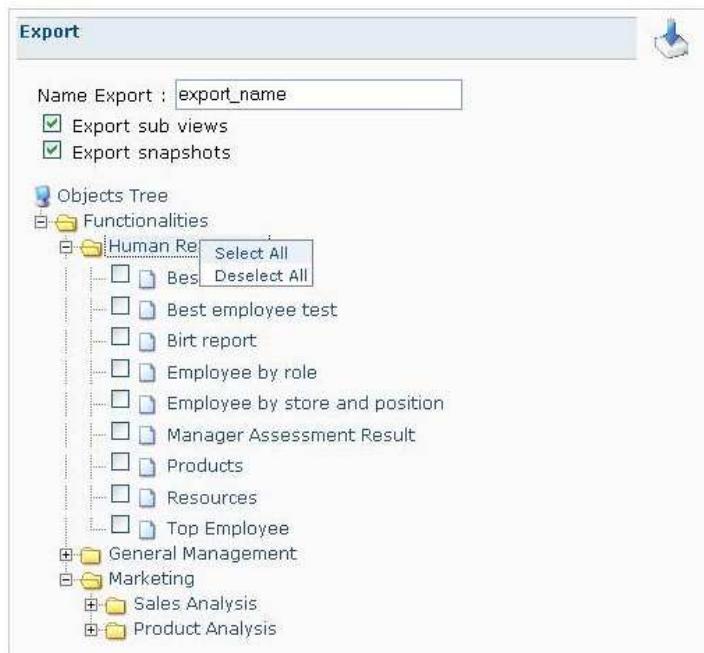
- importing Qbe objects that must be deployed on SpagoBIQbeEngine:
  - datamart jar files (Hibernate jar)
  - subobjects on SpagoBIQbeEngine side, that are files with extension ".qbe"
  - formulas definition files "formula.xml"
- importing Talend jobs that must be deployed on SpagoBITalendEngine
- importing images referenced by report engines
- importing schema definition files for SpagoBIJPivotEngine
- importing maps for SpagoBIGeoEngine

#### 6.1.4. 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
- export the snapshots of the exported objects, checking the 'Export snapshots' 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.

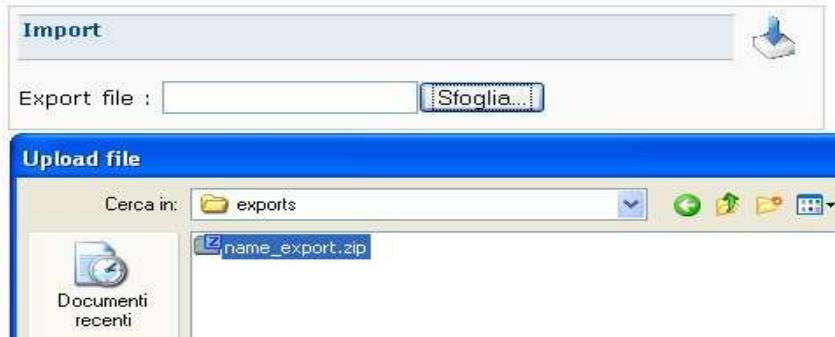


**Figure 16 - Export operation complete**

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

#### 6.1.5. 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.


**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 roles exported can be different from the ones of the current platform the next step require to map the roles.



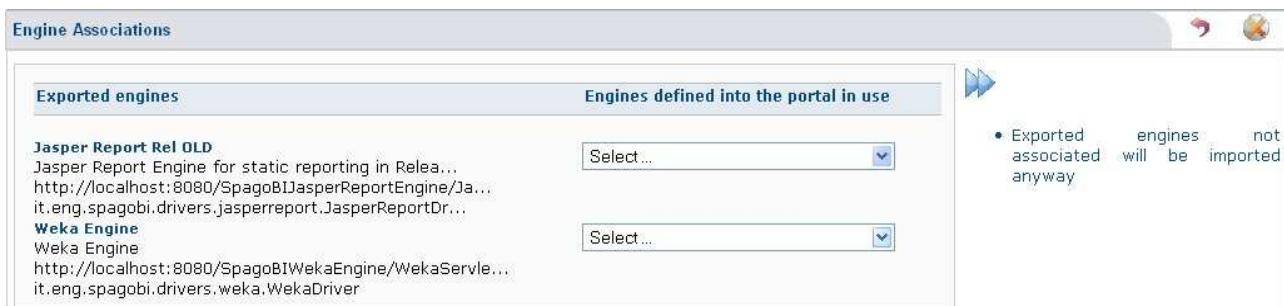
The screenshot shows a 'Role Associations' form. It has two columns: 'Exported roles' and 'Roles of the portal in use'. Under 'Exported roles', there is a list of roles: /spagobi/admin, /spagobi/dev, /spagobi/test, /spagobi/user, and /spagobi. Each role has a corresponding dropdown menu in the 'Roles of the portal in use' column. To the right of the form, there is a list of instructions:

- 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 current 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 inserted 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.

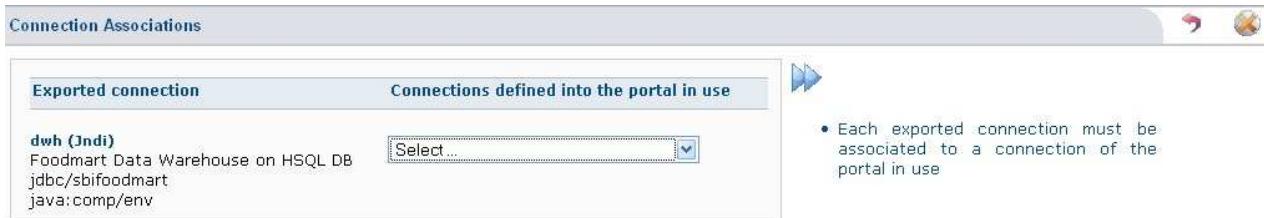


The screenshot shows an 'Engine Associations' form. It has two columns: 'Exported engines' and 'Engines defined into the portal in use'. Under 'Exported engines', there are two entries: 'Jasper Report Rel OLD' and 'Weka Engine'. Each entry has a corresponding dropdown menu in the 'Engines defined into the portal in use' column. To the right of the form, there is a list of instructions:

- Exported engines not associated will be imported anyway

**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 completed 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



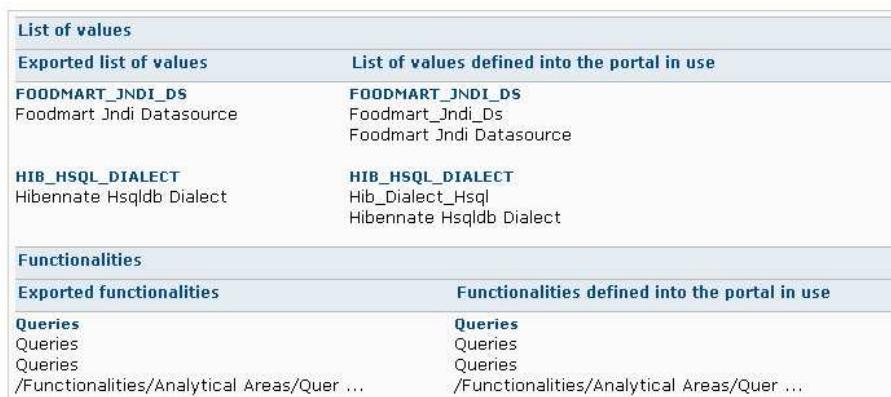
Exported connection	Connections defined into the portal in use
dwh (Jndi) Foodmart Data Warehouse on HSQL DB jdbc:sbifoodmart java:comp/env	Select ...

Each exported connection must be associated to a connection of the portal in use

**Figure 20 - Import connections association**

Once again there is a description for each exported connection and a combo which allows you to associate 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 a connection that 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 system 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.



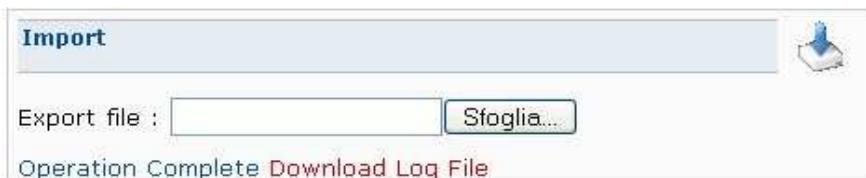
List of values	
Exported list of values	List of values defined into the portal in use
FOODMART_JNDI_DS Foodmart Jndi Datasource	FOODMART_JNDI_DS Foodmart_Jndi_Ds Foodmart Jndi Datasource
HIB_HSQL_DIALECT Hibernate HsqlDb Dialect	HIB_HSQL_DIALECT Hib_Dialect_Hsql Hibernate HsqlDb Dialect

Functionalities	
Exported functionalities	Functionalities defined into the portal in use
Queries Queries Queries /Functionalities/Analytical Areas/Quer ...	Queries Queries Queries /Functionalities/Analytical Areas/Quer ...

The system has found that some exported metadata conflict  
• Exported metadata with the same unique identifier of the existing portal metadata cannot be inserted  
• You can stop the import procedure and change the metadata identifiers which cause the conflict  
• If you continue the system will not insert exported metadata that causes the conflict but will use the existing one

**Figure 21 - Import conflicts**

Once finished the import phase the user is redirected 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.



**Figure 22 - Import operation complete**

Besides, when you import documents into a SpagoBI installation that is newer than the source installation (example: the source installation is SpagoBI 1.9.1 and the target installation is SpagoBI 1.9.3), there might be some operations that the procedure is not able to automatically perform: when those operations are required, a manual task is defined. A manual task is a collection of steps that the user must manually do in order to complete the import operation. Those steps are explained in an archive file produced as output by the procedure (a part from the always present log file). You have to download the archive file and to follow the instructions.

## 6.2. 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;
- Schedule documents execution.



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



The screenshot shows the 'ADMINISTRATION TOOLS' section of the SpagoBI administration portlet. It includes three main items:

- Functionalities Management (represented by a folder icon)
- Documents Configuration (represented by a document icon)
- Schedule a Document Execution (represented by an alarm clock icon)

**Figure 23 – Administration portlet**

### 6.2.1. FUNCTIONALITIES MANAGEMENT

SpagoBI uses its own file system, named "**Functionalities Tree**", that allows to better organize documents by grouping them in folders: 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. He can change folders order using "Move up" or "Move down" actions. 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.

SBI Settings	SBI Administration Context	SBI Development Context	SBI Functionality																								
SBI Administration Context																											
<b>FUNCTIONALITY DETAILS</b>																											
Label: Static Reporting Name: Static Reporting Description: Static Reporting		<input type="button" value="New"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Save"/>																									
<table border="1"> <thead> <tr> <th>Roles</th> <th>Development</th> <th>Test</th> <th>Execution</th> </tr> </thead> <tbody> <tr> <td>/spagobi (spagobi root group)</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> <tr> <td>/spagobi/admin (spagobi administrators)</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> <tr> <td>/spagobi/dev (spagobi developers)</td> <td><input checked="" type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/> </td> </tr> <tr> <td>/spagobi/test (spagobi testers)</td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> <tr> <td>/spagobi/user (spagobi users)</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> </tbody> </table>				Roles	Development	Test	Execution	/spagobi (spagobi root group)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" 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 type="checkbox"/>	<input checked="" type="checkbox"/>
Roles	Development	Test	Execution																								
/spagobi (spagobi root group)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" 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 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:

SBI Settings	SBI Administration Context	SBI Development Context	SBI Functionality																								
SBI Administration Context																											
<b>FUNCTIONALITY DETAILS</b>																											
Label: Static Reporting Child 1 Name: Static Reporting Child 1 Description:		<input type="button" value="New"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Save"/>																									
<table border="1"> <thead> <tr> <th>Roles</th> <th>Development</th> <th>Test</th> <th>Execution</th> </tr> </thead> <tbody> <tr> <td>/spagobi (spagobi root group)</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> <tr> <td>/spagobi/admin (spagobi administrators)</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> <tr> <td>/spagobi/dev (spagobi developers)</td> <td><input checked="" type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/> </td> </tr> <tr> <td>/spagobi/test (spagobi testers)</td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> <tr> <td>/spagobi/user (spagobi users)</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/> </td> </tr> </tbody> </table>				Roles	Development	Test	Execution	/spagobi (spagobi root group)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" 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 type="checkbox"/>	<input checked="" type="checkbox"/>
Roles	Development	Test	Execution																								
/spagobi (spagobi root group)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" 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 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.

### **6.2.2. 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.

ADMINISTRATION OBJECTS LIST						
LABEL	NAME	DESCRIPTION	TYPE	STATE	N. OF INSTANCES	
CustProfBirt	Customer Profile Birt	Customer Profile Birt	REPORT	DEV	1	  
RPT_CUST_PRF_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-EMPPRO-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	Montly Sales	Sales for product and city	REPORT	TEST	1	  

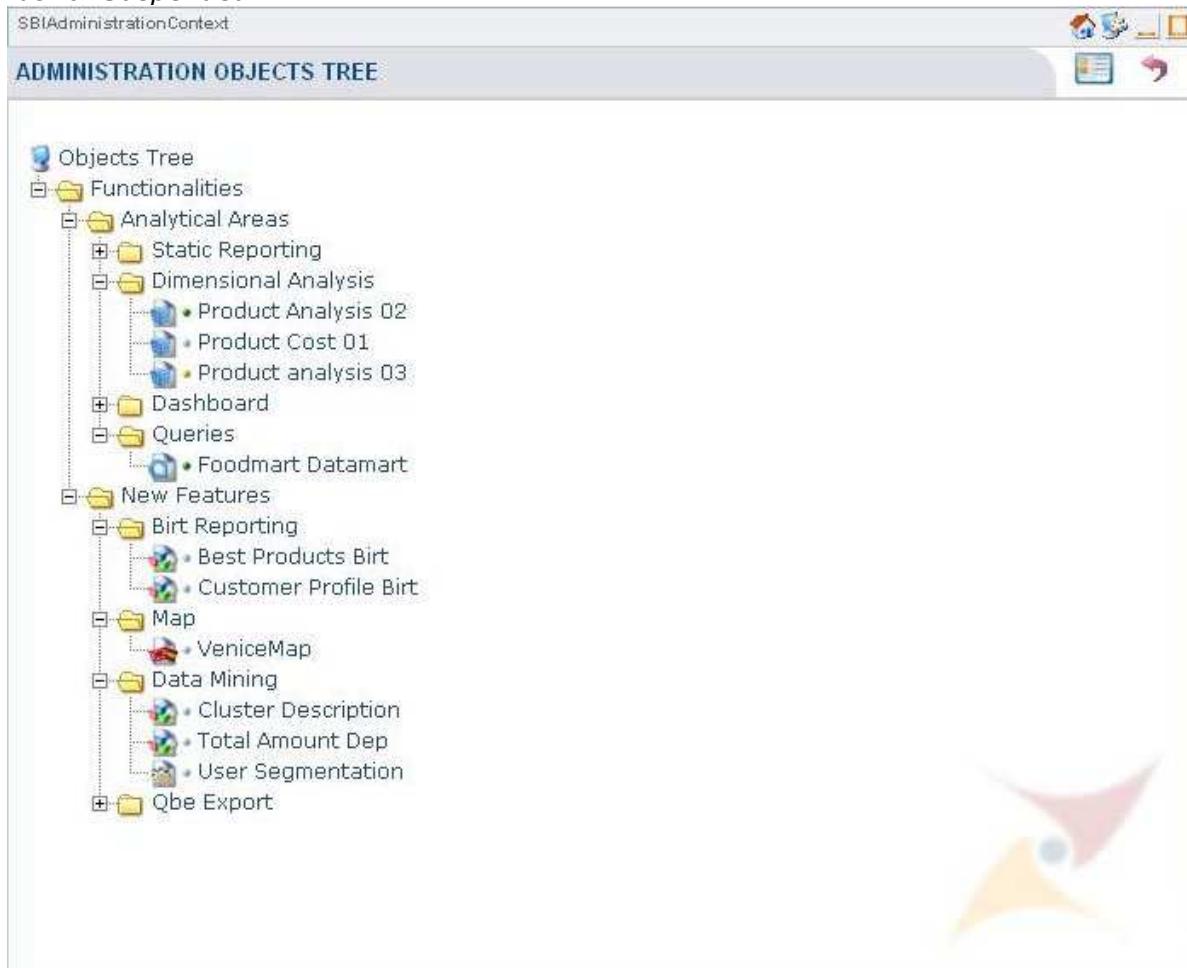
**Figure 27 – Administration Objects List**

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. In the tree, each

document has an icon, that represents the type, and a led, that represents the state: blue led stands for *Development*, yellow led stands for *Test*, green led stands for *Released*, red led stands for *Suspended*.



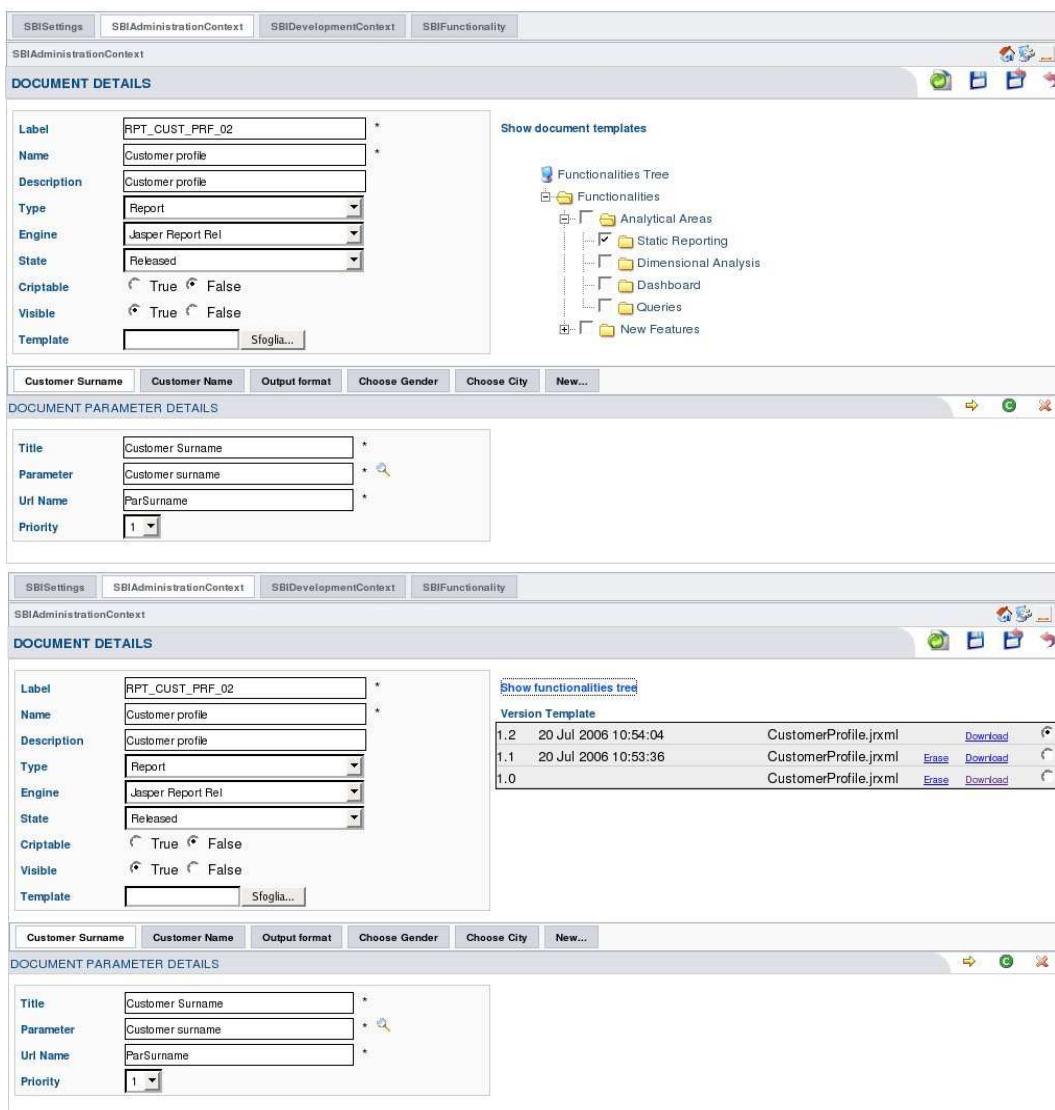
**Figure 28 - Administration Objects Tree**

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 if 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.

## SpagoBI QuickStart



The screenshot displays the SpagoBI Administration Object Details interface. It consists of two main sections: DOCUMENT DETAILS and DOCUMENT PARAMETER DETAILS.

**DOCUMENT DETAILS:**

- Label:** RPT\_CUST\_PRF\_02
- Name:** Customer profile
- Description:** Customer profile
- Type:** Report
- Engine:** Jasper Report Rel
- State:** Released
- Criptable:** True
- Visible:** True
- Template:** Sfoglia...

**DOCUMENT PARAMETER DETAILS:**

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

**Functionalities Tree:**

- Functionnalities Tree
  - Analytical Areas
    - Static Reporting
    - Dimensional Analysis
  - Dashboard
  - Queries
  - New Features

**Version Template:**

Version	Date	Template	Download	Erase
1.2	20 Jul 2006 10:54:04	CustomerProfile.jrxml	<a href="#">Download</a>	<a href="#">Erase</a>
1.1	20 Jul 2006 10:53:36	CustomerProfile.jrxml	<a href="#">Download</a>	<a href="#">Erase</a>
1.0		CustomerProfile.jrxml	<a href="#">Download</a>	<a href="#">Erase</a>

**Figure 29 – 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.

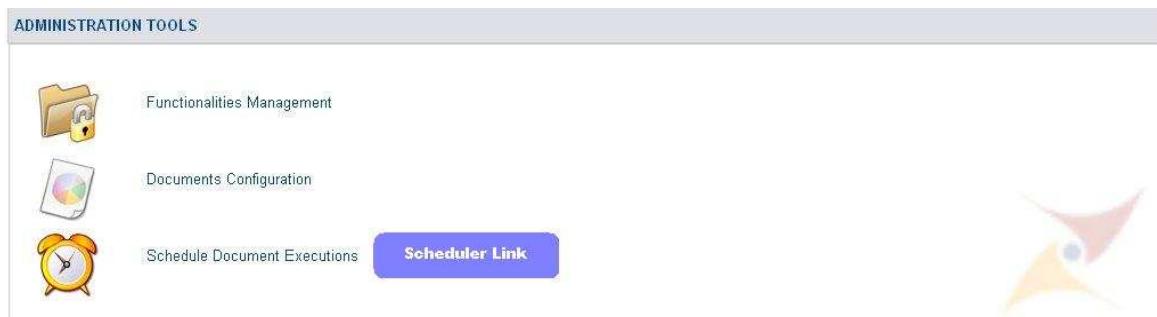
---

### 6.2.3. SCHEDULE A DOCUMENT EXECUTION

An important functionality that a platform of business intelligence can offer is the ability to execute determined tasks in an automatic way and in a predetermined moment. Some typical example uses of this functionality can be the execution of ETL or Data Mining process or the production of reports during the night.

This functionality is generally called Scheduler.

The SpagoBI platform provides a scheduler that gives to the administrator users the ability to schedule the execution of every kind of business documents. This functionality is accessible from the 'Administration Context portlet'.



**Figure 30 - Administration context portlet**

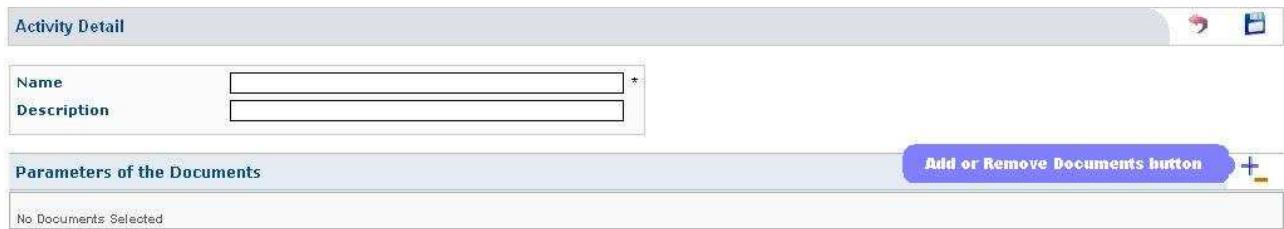
SpagoBI divides the definition of the scheduled task into two phases, the first one is useful to define what will be scheduled (the Activity) whereas the second one is necessary to decide when the activity will be executed (The Schedule). These two concepts are independent and this means, for instance, that a user can schedule the same activity more times.

Selecting the scheduler link you enter into the 'Activity' list which summarizes all the activities already defined. You can insert a new activity, change / erase the existing ones or view the list of schedules associated.

Activity List			New Activity Button
Name	Description	Number of Schedules	
MyActivity	MyActivity Description	0	
page 1 of 1			

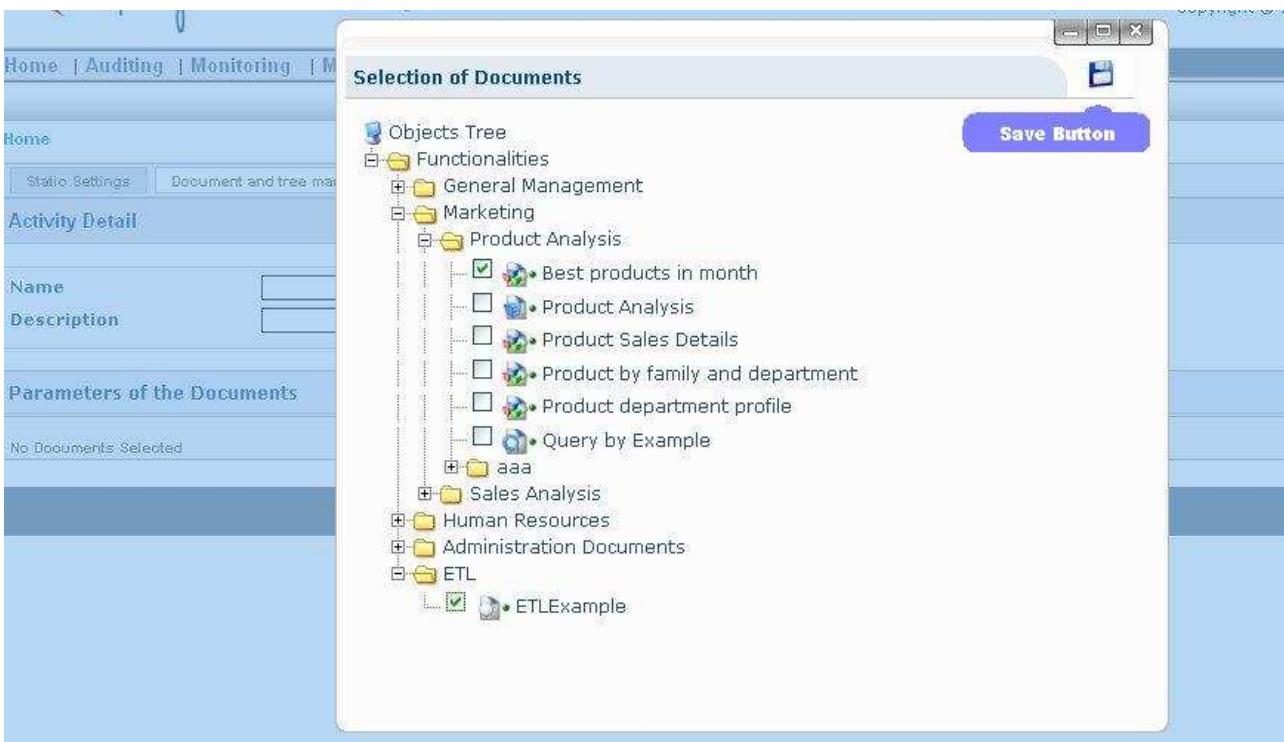
**Figure 31 - Activities list**

Selecting the 'New Activity' or 'Detail' button you enter into the detail page of an activity. An activity is a collection of business documents and their relative execution parameter values. In fact, since the scheduled execution is an automatic task, the parameters values must be filled at the creation time. The collection of documents is identified by a name and a description.



**Figure 32 -Activity detail**

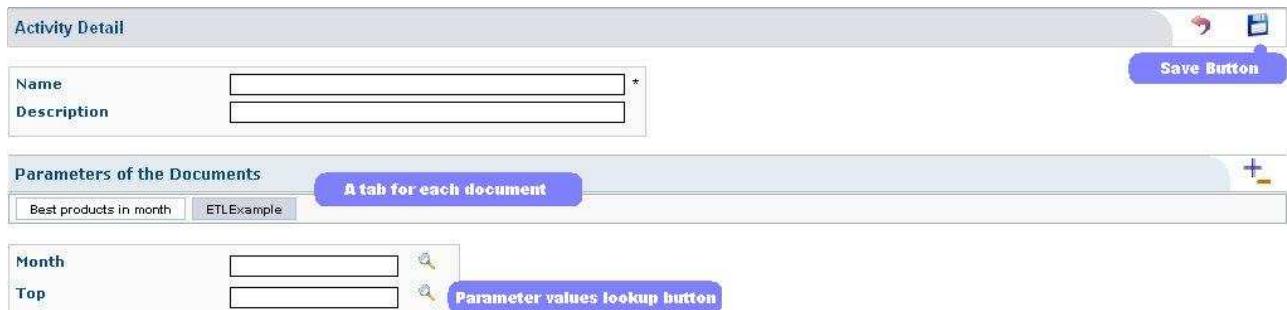
In order to add one or more documents to the activity you must select the 'Add or Remove Documents' button. This will make appear a pop-up containing the tree of documents in released state.



**Figure 33 - Documents selection**

You can choose one or more documents from the tree by selecting the relative check and then pressing the 'Save' button.

Once finished the selection the pop-up will disappear and the section 'Parameters of the documents' of the detail page will show a new tab for each new document.



**Activity Detail**

Name  \*

Description

Save Button

Parameters of the Documents

A tab for each document

Best products in month    ETLExample

Month  

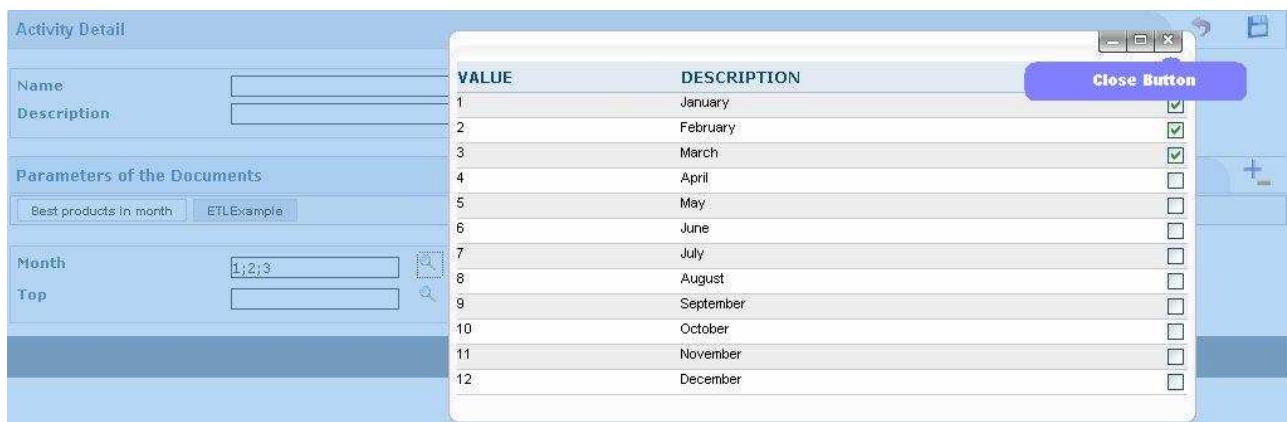
Top  

Parameter values lookup button

**Figure 34 - Documents launch configuration**

Each tab contains the form useful to fill the parameters values of the document associated. For each parameters there's a row with the parameter url name, an editable input field which allow to insert the value, and a button that open a pop-up containing the set of possible values. Since the values shown are calculated based on the parameter use associated to the user, and since a user can have more than one parameter use (based on his roles), sometimes could happen that near the button appear the selection of the role which will be used for the values calculation. The values pop-up allows to choose one or more values.

When you close the pop-up the selected values will fill the input field associated to the parameter. Notice that if you select more values, into the input field they are separated by a ';' character.



VALUE	DESCRIPTION	Close Button
1	January	<input checked="" type="checkbox"/>
2	February	<input checked="" type="checkbox"/>
3	March	<input checked="" type="checkbox"/>
4	April	<input type="checkbox"/>
5	May	<input type="checkbox"/>
6	June	<input type="checkbox"/>
7	July	<input type="checkbox"/>
8	August	<input type="checkbox"/>
9	September	<input type="checkbox"/>
10	October	<input type="checkbox"/>
11	November	<input type="checkbox"/>
12	December	<input type="checkbox"/>

**Figure 35 - Parameter values selection**

Once added all the documents and relative parameters values you can save them pressing the 'Save' button in the right-upper corner and then you will return to the activity list.

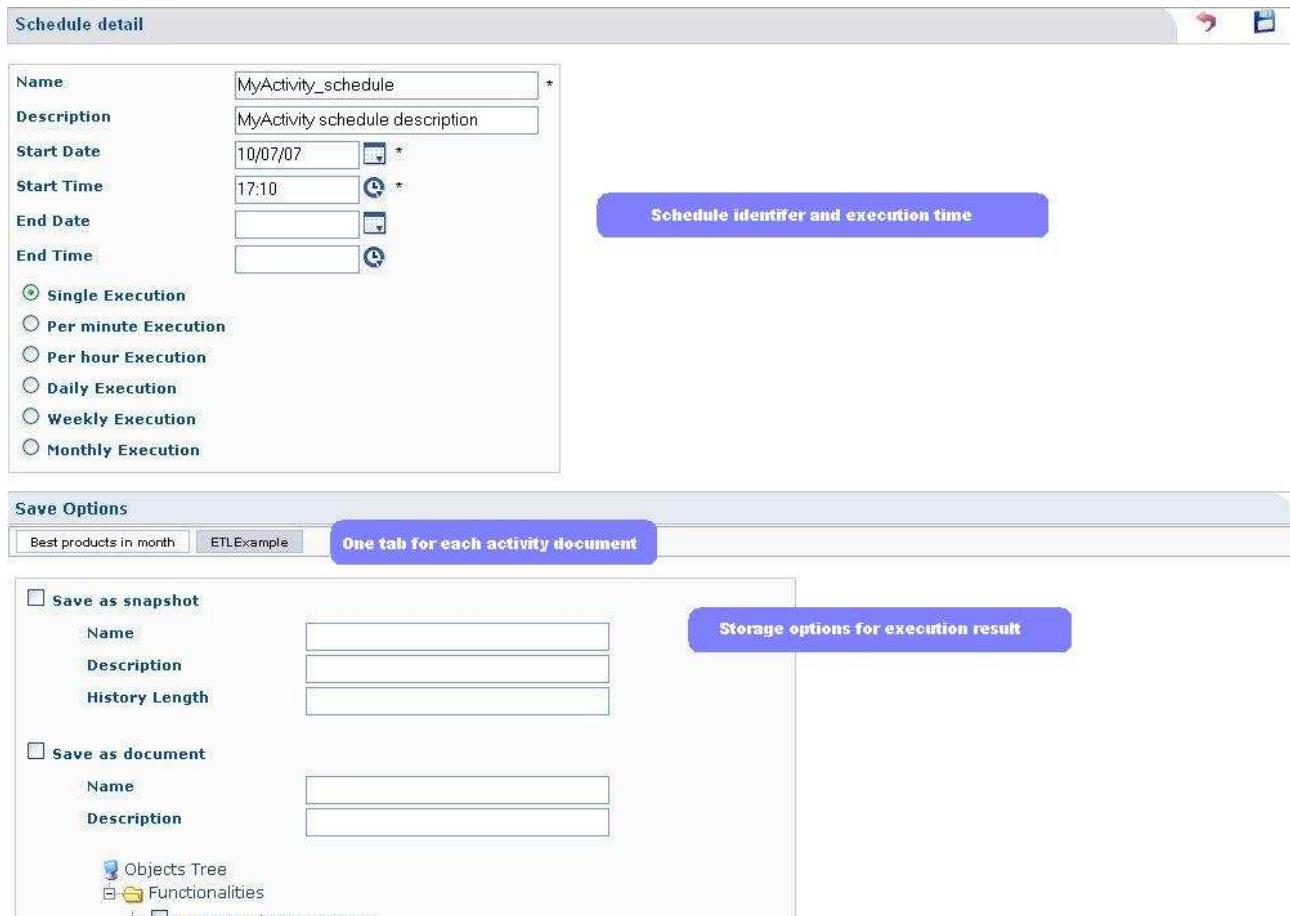
Once the activity is recorded it is necessary to define 'when' it will be executed. To do this you must select the 'Schedule List' button of the relative activity list row. This operation will open a new list showing all the schedules associated to the activity. As for the other lists there's the possibility to insert a new schedule or to change / erase the existing ones. In addition for each schedule there's also a 'Run Now' button that allows to execute the activity immediately without waiting for its scheduled time.



Schedules of the activity		New Schedule Button			
Name	Description	Start Date/Time	End Date/Time	Detail Button	Run now Button
MyActivity_schedule	MyActivity schedule description	Tue Jul 10 14:05:00 CEST 2007			
page 1 of 1					

**Figure 36 - Activity scheduling**

Selecting the 'New Schedule' or 'Detail' button you enter into the detail page of an activity schedule. A schedule is identified by a name and a description and it defines 'when' an activity must be executed. In addition the schedule maintains also information about the storing of the activity output.



The screenshot shows the 'Schedule detail' page. At the top left, there's a 'Save' icon and a 'Print' icon. Below the header, there are several input fields:

- Name:** MyActivity\_schedule \*
- Description:** MyActivity schedule description
- Start Date:** 10/07/07 \*
- Start Time:** 17:10 \*
- End Date:** [empty]
- End Time:** [empty]

A blue callout box labeled 'Schedule identifier and execution time' points to the Name, Description, Start Date, Start Time, End Date, and End Time fields.

Below these fields is a section for repetition policy:

- Single Execution
- Per minute Execution
- Per hour Execution
- Daily Execution
- Weekly Execution
- Monthly Execution

At the bottom of the page, there's a 'Save Options' section with tabs:

- Best products in month
- ETLExample
- One tab for each activity document** (highlighted)

A blue callout box labeled 'Storage options for execution result' points to the 'One tab for each activity document' tab. Within this tab, there are two sections:

- Save as snapshot:** Contains fields for Name, Description, and History Length.
- Save as document:** Contains fields for Name and Description, along with a tree view for Objects Tree, Functionalities, General Management, and so on.

**Figure 37 - Schedule details**

Looking at the detail page of a schedule should be clear that the first part is useful to set the identifier, the schedule time and the possible repetition policy, while the second one, which is called 'Save Options', contains information about activity output storage.

Notice that the save option part contains a tab for each document of the related activity, this is because different documents usually require different storage output configurations or no storage at all.

In the first part of the page the user can insert:

- the name of the schedule (a default name is suggested automatically)
- the description of the schedule
- the initial date after which the schedule will be activated
- the initial time, relative to the initial date, after which the schedule will be activated
- the final date after which the active schedule will be deactivated
- the final time, relative to the final date, after which the schedule will be deactivated.

After these fields there are the repetition options. You can choose only one repetition policy and based on the kind of policy chosen the system opens dynamically a form for its configuration.

Schedule detail

Name	MyActivity_schedule *
Description	MyActivity schedule description
Start Date	10/07/07 <input type="button" value="..."/> *
Start Time	17:10 <input type="button" value="..."/> *
End Date	<input type="button" value="..."/>
End Time	<input type="button" value="..."/>
<input type="radio"/> Single Execution <input type="radio"/> Per minute Execution <input type="radio"/> Per hour Execution <input type="radio"/> Daily Execution <input type="radio"/> Weekly Execution <input checked="" type="radio"/> Monthly Execution	
<input type="radio"/> Every n months <input type="button" value="1"/> * <input checked="" type="radio"/> In month <input type="checkbox"/> JAN <input type="checkbox"/> FEB <input type="checkbox"/> MAR <input type="checkbox"/> APR <input checked="" type="checkbox"/> MAY <input type="checkbox"/> JUN <input type="checkbox"/> JUL <input checked="" type="checkbox"/> AUG <input type="checkbox"/> SEP <input type="checkbox"/> OCT <input type="checkbox"/> NOV <input type="checkbox"/> DIC <input type="radio"/> The day <input type="button" value="1"/> <input checked="" type="radio"/> The week In day	
Opened dinamically at user selection	
<input type="radio"/> First <input type="radio"/> Second <input type="radio"/> Third <input type="radio"/> Fourth <input type="radio"/> Last <input type="checkbox"/> SUN <input checked="" type="checkbox"/> MON <input type="checkbox"/> TUE <input type="checkbox"/> WED <input type="checkbox"/> THU <input checked="" type="checkbox"/> FRI <input type="checkbox"/> SAT	

**Figure 38 - Repetition policies**

The possible policies are:

- Single Execution: the activity is executed only one time at the initial date and time inserted by the user
- Per minute execution: the activity is executed every n minutes starting from the initial date and time.
- Per hour execution: the activity is executed every n hours starting from the initial date and time.
- Daily Execution: the activity is executed every n days starting from the initial date and time.
- Weekly Execution: the activity is executed every week at the initial time of the selected days. If no day is selected the Monday will be chosen as default.
- Monthly Execution: the user can choose to execute the activity every n month after the month of the initial date or to execute it only at the selected months. In both cases the user can choose to run the activity in a definite day, to run it all the selected week-day of the month or finally to run it at the selected week-day of the selected week of the month. In this last case there's a limitation: for this release, if you choose the execution week, you cannot choose more than one day-week.

All the policies imply the repetition of the activity till the final date and time, if the user has inserted it, or forever.

Into the second part of the page ('Save Options') you find a tab for each document associated to the activity and each tab contains the configuration for the storage of the document execution output. Obviously if a document hasn't a meaningful output, like for example an ETL process, this part can be skipped.

**Save Options**

Best products in month	ETLExample												
<input checked="" type="checkbox"/> <b>Save as snapshot</b> <table border="1"> <tr> <td>Name</td> <td>snapshot name</td> </tr> <tr> <td>Description</td> <td>snapshot description</td> </tr> <tr> <td>History Length</td> <td>5</td> </tr> </table> <input checked="" type="checkbox"/> <b>Save as document</b> <table border="1"> <tr> <td>Name</td> <td></td> </tr> <tr> <td>Description</td> <td></td> </tr> </table>  <ul style="list-style-type: none"> <li>Functionalities             <ul style="list-style-type: none"> <li>General Management</li> <li>Marketing</li> <li>Human Resources</li> <li>Administration Documents</li> <li>ETL</li> </ul> </li> </ul> <input checked="" type="checkbox"/> <b>Send mail</b> <table border="1"> <tr> <td>Mail to</td> <td>spagobi@eng.it</td> </tr> </table>		Name	snapshot name	Description	snapshot description	History Length	5	Name		Description		Mail to	spagobi@eng.it
Name	snapshot name												
Description	snapshot description												
History Length	5												
Name													
Description													
Mail to	spagobi@eng.it												

**Figure 39 - Save options**

The possible configurations are:

- Save as snapshot: The output will be saved as sub-document of the main one. When the users will execute the main document they system will show them also the list of snapshots available. In this way the user can choose to view the result of a previous execution or to run it again. This can be useful for example for a report that takes a long time to produce the output, in fact, you can schedule it during the night and in the morning the users can see the result without wasting time. If you choose this options you need to specify the name and description of the snapshot and the history length. The history length is the maximum number of document snapshot that the system maintain before overwrite the oldest one.
- Save as document: The output will be saved as a new document (Office Document kind). If you choose this option you need to insert the name and description of the new document and the positions into the business document tree where it will be recorded. Remember that the new document will be inserted in development state.
- Send mail: the output will be sent by email as an attachment. If you choose this option you need to insert the recipient mail addresses. You can specify more than one mail addresses, separating them with a comma. See "How to configure mail service?" in "How To" document in order to see how to configure this mail service.

Once configured the schedule you can save it pressing the 'Save' button in the right-upper corner which will bring you back to the activity schedule list.

## 6.3. 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 40 – 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.

### 6.3.1. 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.

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
EMPOS_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	Hibernate Hsqldb Dialect	SCRIPT	1
LOV_MONTH	Month list	Month list	QUERY	1

page 1 of 4

The value of the column **LABEL**

**Figure 41 – 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:** four 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;
  - **Java class:** a Java class that implements the interface `it.eng.spago.bi.javaClassLows.IJavaClassLov` with the following methods:
    - `getValues(IEngUserProfile profile)` that returns an xml envelope as a String: the xml must have the following syntax

```
<ROWS>
  <ROW ... />
  ...
</ROWS>
```

    - `getNamesOfProfileAttributeRequired()` that returns the list of the profile attribute names required for its calculation.

**PREDEFINED LIST OF VALUES DETAILS**

Label	<input type="text" value="CST_LIST"/> *
Name	<input type="text" value="Customer List"/> *
Description	<input type="text"/>
Input Type	<input type="button" value="Query statement"/>

**Wizard Query**

Connection name	<input type="button" value="Foodmart Data Warehouse"/>
Query definition	<pre>select distinct c.customer_id as id, lname as lastname, fname as name, city as city, country,birthdate, gender, occupation from customer as c, sales_fact_1997 as s where s.customer_id=c.customer_id order by lname</pre>

**Figure 42 - 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.

Then you have to click on the "Test before save" button: you will see the query result on a page similar to the figure below:

Predefined List of values details - Test results							
Columns		Value column	Description column	Visible columns			
ID		<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>			
LASTNAME		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
NAME		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
CITY		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
COUNTRY		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
BIRTHDATE		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
GENDER		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
OCCUPATION		<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>			
<hr/>							
ID	LASTNAME	NAME	CITY	COUNTRY	BIRTHDATE	GENDER	OCCUPATION
220	Ace	Dorothy	Torrance	USA	16-01-1955	F	Management
303	Albright	Anna	Fremont	USA	22-02-1944	M	Skilled Manual
36	Arnold	Donna	Oregon City	USA	22-11-1915	M	Skilled Manual
216	Baird	Dorothy	Olympia	USA	21-07-1924	F	Professional
20	Baker	Beverly	Spring Valley	USA	16-04-1974	F	Manual
174	Bayol	James	San Carlos	USA	23-06-1953	F	Management
41	Bechard	Howard	W. Linn	USA	19-04-1919	M	Professional
26	Bernard	Jose	Burbank	USA	25-08-1954	M	Skilled Manual
62	Borden	Mary	Santa Monica	USA	25-06-1931	M	Skilled Manual
22	Borges	Laurie	Bellingham	USA	07-10-1947	F	Skilled Manual

**Figure 43 - Query test**

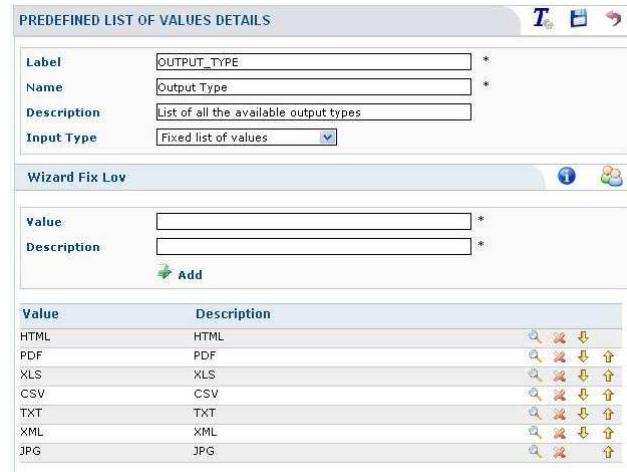
You can use list navigation arrows and filter to scroll the entire list.

Then you have to select the column to be the value column (i.e. parameter values come from this column), the description column (i.e. parameter descriptions come from this column) and the visible columns (i.e. the columns that you will see when you click on the parameter lookup button on the document execution parameters form).

At last click on the "Save" button.

In the *Fixed Values* case, you can create a table of pairs (Name, Description) simply by filling the *Value* and *Description* form 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.

Then you have to click on the "Test before save" button, choose the values column, description column and visible columns as seen for query lovs.

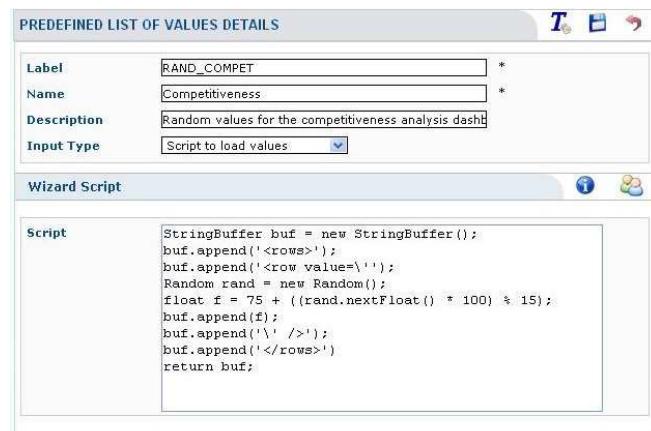


The screenshot shows the 'PREDEFINED LIST OF VALUES DETAILS' dialog. It includes fields for Label (OUTPUT\_TYPE), Name (Output Type), Description (List of all the available output types), and Input Type (Fixed list of values). Below this is a 'Wizard Fix Lov' section with 'Value' and 'Description' fields, and an 'Add' button. At the bottom is a table with columns 'Value' and 'Description', containing rows for HTML, PDF, XLS, CSV, TXT, XML, and JPG, each with edit and delete icons.

**Figure 44 – Fixed values wizard**

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

Then you have to click on the "Test before save" button, choose the values column, description column and visible columns as seen for query lovs.



The screenshot shows the 'PREDEFINED LIST OF VALUES DETAILS' dialog for a script-based list. It includes fields for Label (RAND\_COMPET), Name (Competitiveness), Description (Random values for the competitiveness analysis dashb), and Input Type (Script to load values). Below this is a 'Wizard Script' section with a code editor containing Groovy script code:

```

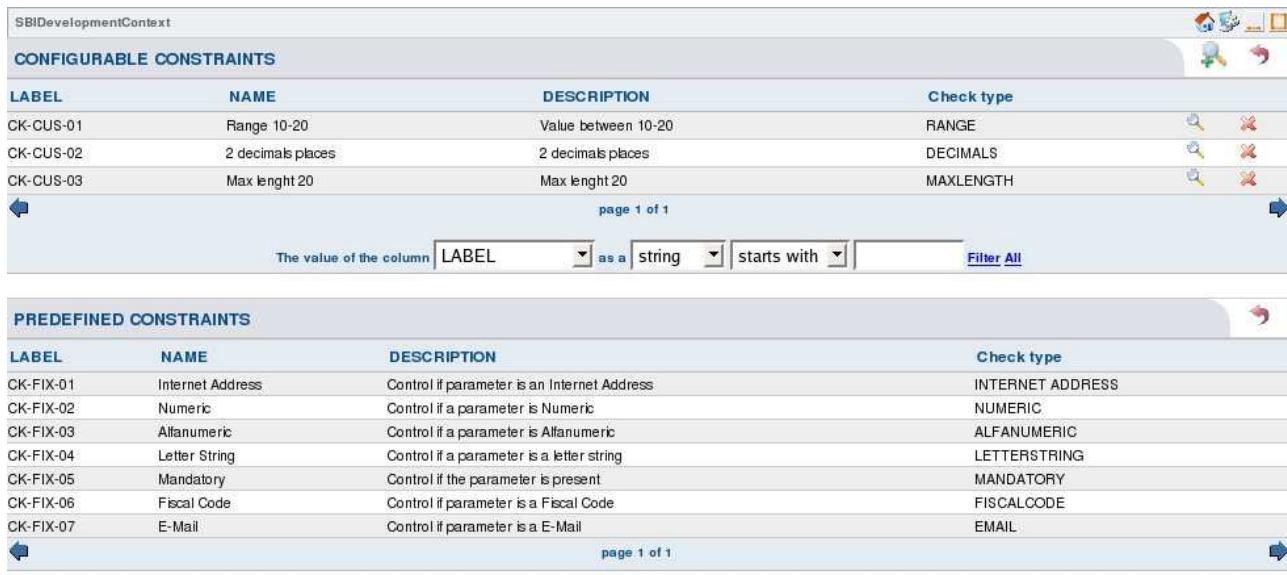
StringBuffer buf = new StringBuffer();
buf.append('<rows>');
buf.append('<row value=\'');
Random rand = new Random();
float f = 75 + ((rand.nextFloat() * 100) % 15);
buf.append(f);
buf.append('\' />');
buf.append('</rows>');
return buf;

```

**Figure 45 – Script wizard**

### 6.3.2. PREDEFINED VALUES CONSTRAINTS

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



The screenshot shows two tables of constraints. The top table is titled "CONFIGURABLE CONSTRAINTS" and contains three rows:

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

The bottom table is titled "PREDEFINED CONSTRAINTS" and contains seven rows:

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	Alphanumeric	Control if a parameter is Alphanumeric	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

**Figure 46 - 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.



The screenshot shows the "CONSTRAINT DETAILS" dialog box. It contains fields for Label (CK-CUS-02), Name (2 decimals places), Description (2 decimals places), Check type (Decimal), and Decimal Places (2).

**Figure 47 - 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*.

### 6.3.3. 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 them to 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.

PARAMETERS LIST						
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	 
EMPPOS_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	 

**Figure 48 – 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;
- **Functional**: define if the parameter is functional or not: a functional parameter is a business logic parameter (examples: city, customer, year of analysis ...) while a non functional parameter is independent from the business logic (examples: output format, database connection ...).

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

PARAMETER DETAILS	
Label	<input type="text"/> *
Name	<input type="text"/> *
Description	<input type="text"/>
Type	<input checked="" type="radio"/> Date <input type="radio"/> Number <input type="radio"/> String
Functional	<input checked="" type="checkbox"/>

**Figure 49 – 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):

SBI Development Context:

<b>Label</b>	PROD_CD	*
<b>Name</b>	Product Code	*
<b>Description</b>	Parameter for Product Code selection	
<b>Type</b>	<input type="radio"/> Date <input type="radio"/> Number <input checked="" type="radio"/> String	
<b>Functional</b>	<input checked="" type="checkbox"/>	

**PARAMETER USE MODE DETAILS**

<b>Label</b>	PROD_LIST	*
<b>Name</b>	Product from list	*
<b>Description</b>	Product from predefined list of values	
<b>Lov</b>	<input checked="" type="radio"/> Product by Fixed Lov	*
	<input type="button" value="List values selection"/>	
<b>Manual Input</b>	<input type="radio"/>	

**Roles Associations**

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

**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		

**Figure 50 – 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; then you have to choose the modality of visualization of the parameter between: lookup list values selection (using this selection type the user will be able to choose only one value for the parameter), lookup checklist values selection (the user will be able to choose more than one parameter values<sup>(1)</sup>) or combobox (one parameter value);
- **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.

<sup>(1)</sup> Multi values parameters are useful only using SpagoBI JasperReportEngine (see Report); all other engines will consider only the first value.

### 6.3.4. 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.).

SBIDevelopmentContext					
DEVELOPMENT OBJECTS LIST					
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
BestProdDailySalTab	Best Products for Daily Sales Tab		DASH	DEV	1
BestProdMonthSalTab	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 51 – Analytical documents list**

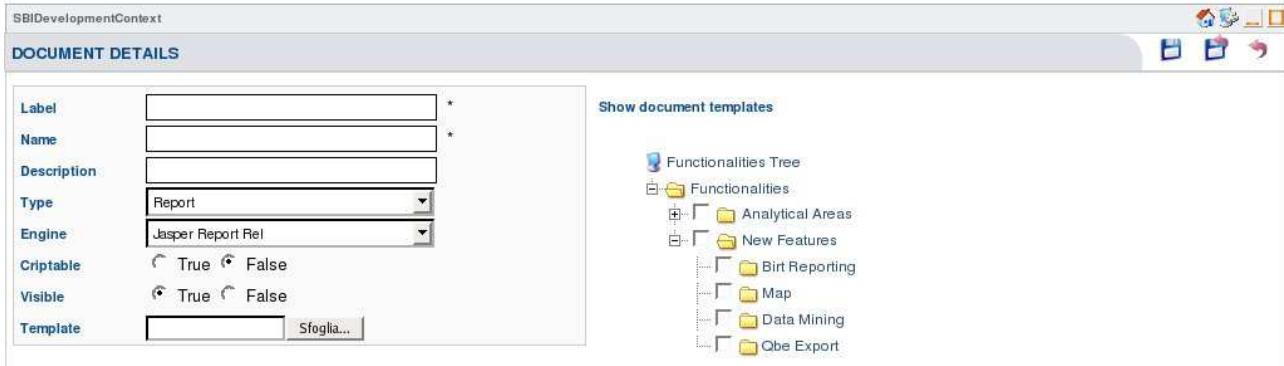
The *Documents Details* page displays a complete list of attributes:

- **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 52 – Analytical document details (new document)**

Multiple selection of the parent folder is allowed.



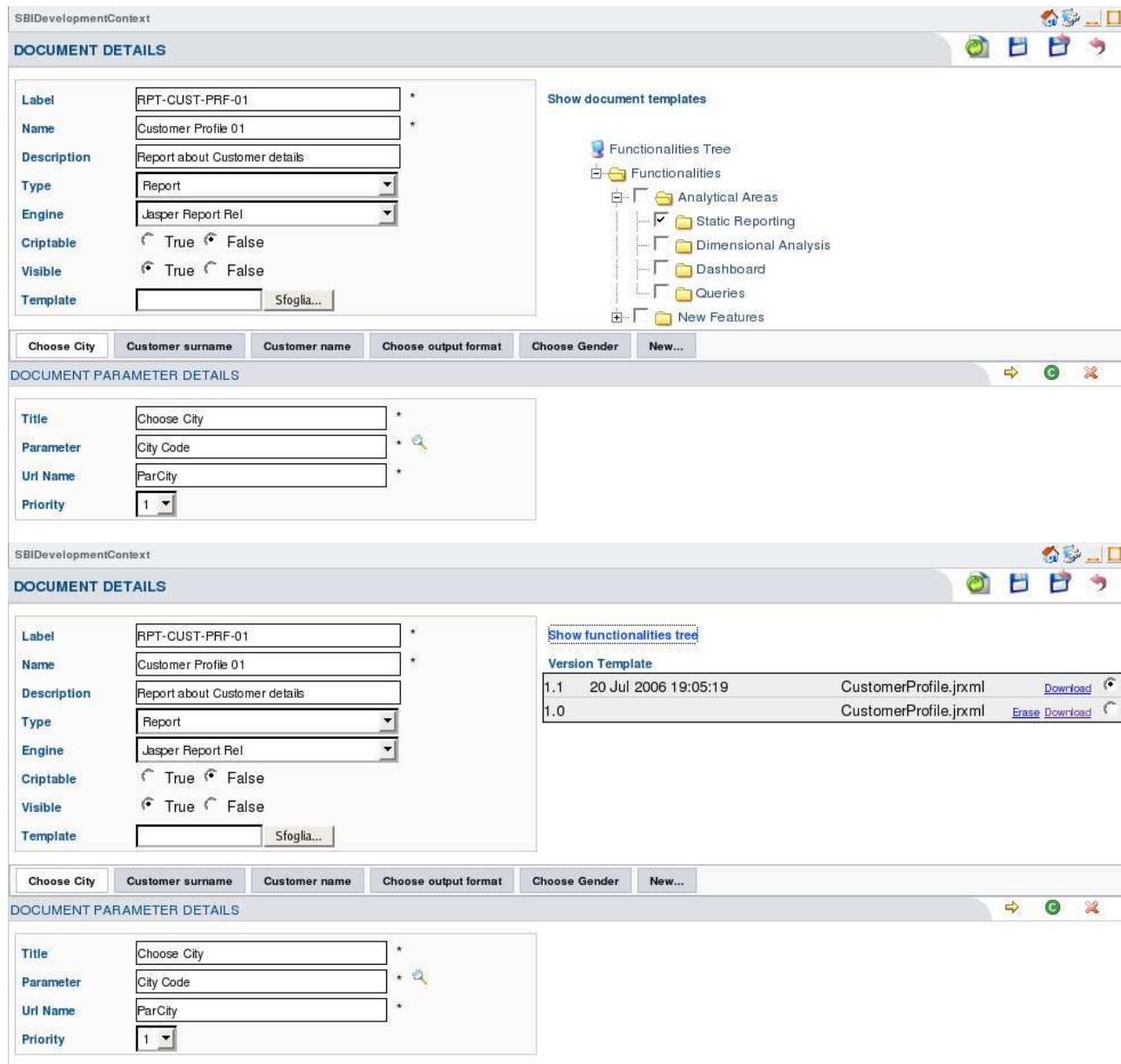
When the developer sees an existing document and selects the DETAILS icon, additional elements will be displayed:

- **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 template 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.

**SpagoBI QuickStart**


The screenshot displays three windows of the SpagoBI QuickStart interface, illustrating the configuration of an analytical document.

**Top Window (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:** [Browse] Sfoglia...

**Right Panel (Show document templates):**

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

**Middle Window (DOCUMENT PARAMETER DETAILS):**

- Title:** Choose City
- Parameter:** City Code
- Url Name:** ParCity
- Priority:** 1

**Bottom Window (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:** [Browse] Sfoglia...

**Right Panel (Show functionalities tree):**

- Version Template
 

1.1	20 Jul 2006 19:05:19	CustomerProfile.jrxml	<a href="#">Download</a>
1.0		CustomerProfile.jrxml	<a href="#">Erase Download</a>

**Figure 53 – 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.

### 6.3.5. PARAMETERS CORRELATION

The parameters can be considered as a filter for the data, in fact, based on the parameter value, the data returned or processed by business document is 'sliced and diced'. For the

SpagoBI platform two different parameters are separated and independent entities and for this reason they can be reused more times for different business documents.

However, sometimes, into the context of a document, two different parameters can be correlated: this means that the collection of the possible values of one parameter (the dependent) will be restricted by the platform, based on the value of the other parameter (the "father").

As an example consider two geographical parameters 'continent' and 'state', if you associate both to a business document probably you want that if a user choose a continent then the state parameter shows only the states of the continent selected.

For this reason the correlation between parameters must be configured for each business document which need it, so, for each parameter tab into the biobject detail page, there's a bottom which allows to enter into the dependency configuration.



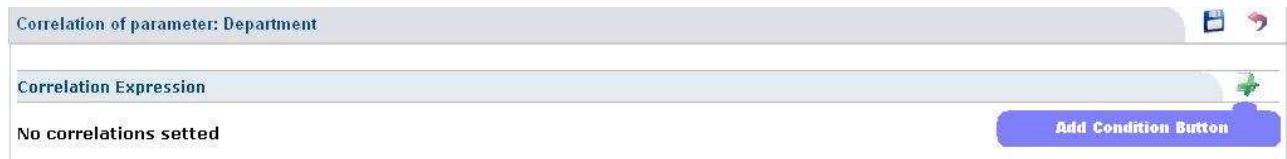
**DOCUMENT PARAMETER DETAILS**

**Title:** Department \* **Parameter:** Products department \* **Url Name:** ParDept \* **Priority:** 2

**Correlation Button**

**Parameters part of the BIObject detail page**

**Figure 54 - Document parameter details**



**Correlation of parameter: Department**

**Correlation Expression**

No correlations setted **Add Condition Button**

**Figure 55 - Parameter correlation page**

A parameter can depend from one or more parameters of the same business document. Each dependency is expressed by a condition which is a part of the 'Correlation Expression'. This expression is a composition of the conditions built using logic operators (AND / OR) and brackets.

Once entered into the parameter dependency detail page you can see the list of simple conditions that compose the correlation expression. As for every SpagoBI list is possible to insert new conditions or to delete / modify the existing ones. To add a new condition select the 'Add Condition' button. This will make appear a pop-up useful to configure the condition.



**Condition**

The parameter Department depends from the parameter **Family**

Filter Operator (Operator used to compare a column value of the parameter Department and the value of the parameter on which it depends) =

List of parameter uses associated to the parameter Department

is Active	Parameter Use Name	Columns generated by the parameter use
<input checked="" type="checkbox"/>	USE_PRF: USE_PRF	PRODUCT_DEPAR
<input checked="" type="checkbox"/>	USE_ALL: USE_ALL	PRODUCT_DEPAR PRODUCT_DEPARTMENT PRODUCT_FAMILY NUM_OF_CATEGORY SALES UNIT_SALES

**Save Button**

**Figure 56 - Condition configuration**

Before explaining the condition configuration it is better to analyze how the current parameter is related to the 'father' one. As you know the set of possible values for a parameter is calculated using a lov. A lov doesn't return a list of single values but a list of result records (it can be seen like a table). Moreover you have to consider that a parameter has different 'parameter uses' and, as a consequence, different lovs, based on the user roles.

This means that the table which represent the possible values for a parameter can be completely different based on the user roles. So it's important to notice that to insert a dependency you need to specify for each 'parameter use' of the parameter which column of the lov result will be correlated.

Taking this consideration into account it's possible to describe the dependency condition pop-up.

The first thing that the pop-up ask you is the 'father' parameter whose value will be used to filter the list of possible values for the current parameter. Notice that the set of possible parameters includes all the other parameters of the same business document.

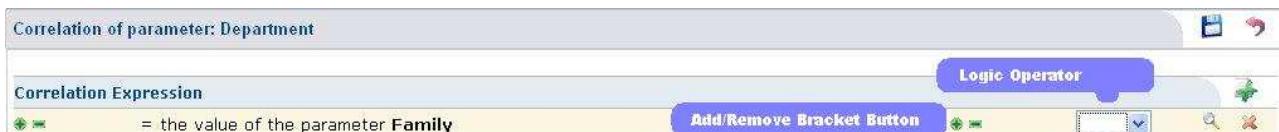
Then you need to specify the filter operator that will be used to compare the value of the father parameter and a value of the current one. The last pop-up part is the list of the 'parameter uses' and for each one of them you must choose (from the related select form) which column will be used to recover the value of the current parameter.

For each 'parameter use' there's also a check that allows to activate or deactivate the dependency.

Considering the example dependency of the 'state' parameter from the 'continent' one we can suppose that the 'state' parameter has two parameter uses, one that return the list of all the states in the world and the other that returns the list of Europe and Asia states.

Both the result lists contain a columns 'is\_in\_continent' which contains the continent to which the state belongs. In this case, to insert the dependency, you should configure that, for both 'parameter use' of the state parameter, the column 'is\_in\_continent' is related to the father parameter 'continent' using the filter operator 'equal'.

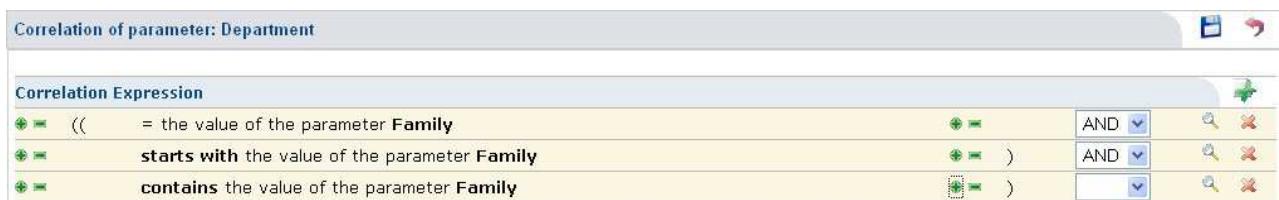
Once configured the dependency you can press the 'Save' button, then the pop-up will be closed and a new row will be shown into the correlation expression list.



**Figure 57 - New correlation defined**

You can add more than one row into the correlation expression list repeating the process described above. In that case you have to compose the final expression linking the single conditions using logical operators and brackets. Each row, in fact, has buttons useful to insert/remove brackets before or after the condition and a select field to choose the logic operator that tie the current condition and the next one. Supposing that A, B and C are single conditions you can compose an expression like ((A and B) or C).

A typical example is when you want that the value of a date parameter D is major than a date parameter D1 and minor than a date parameter D2 (date range).



**Figure 58 - Conditions expression**

At the end you have to notice that:

- it is possible to establish cascading correlations (one parameter is dependent by a father parameter which depends by a third parameter, and so on).
- 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.
- If the father parameter is multi-values, the dependent parameter value satisfies the dependency if the filtering condition is satisfied with at least one father parameter value. Since the filter operators "<", ">", "<=", ">=" have no sense in case of multi-values father parameter, in such cases the dependency is ignored.

### 6.3.6. 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 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*).

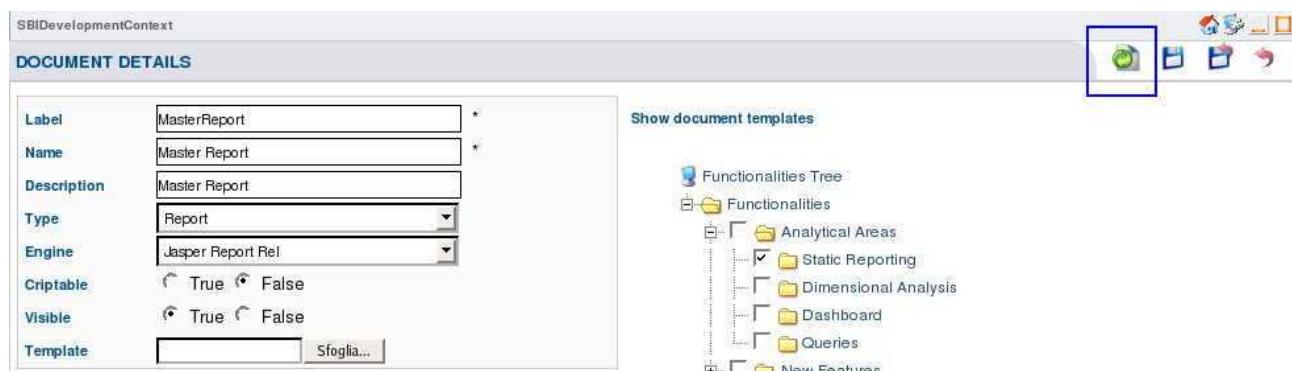
```

<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 59 - 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 links button in the document detail page (Figure 60 - Links configuration button) and than check from the list shown the report included by your master report (Figure 61 - Links list).



**Figure 60 - Links configuration button**

SBI Development Context		
Check-List		
Label	Name	Description
RPT_PROD_PRF_02	Product profile 02	Product details about sales and data
RPT_SALES_CITY_FX	Product subcategory sales	Product subcategory sales
ProdProfic	Products profitability	
RPT_SALES_CITY_02	Sales by city	Month sales for product and city
SALES-X-CITY	Sales by city 01	Product sales for city
RPT_EMPOPOS_FX	Single Employee	Single Employee
StorePerfTrendLines	Store Performance Trend Lines	
StoreEffSmall	Stores efficiency	
StoreEff	Stores efficiency	
SubReport	SubReport	SubReport

page 3 of 4

[View only checked elements](#)

**Figure 61 - Links list**

In this page there is also the possibility to see only the checked documents by clicking on the checkbox below.

### Data mining model or ETL process – presentation report document

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

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

#### 6.3.7. DOCUMENTS VISIBILITY

Checking visible properties into the document detail page (Figure 62 - Document visibility field) to false make the document not visible in the object tree view when the document itself passes from the develop state to the test or released 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).

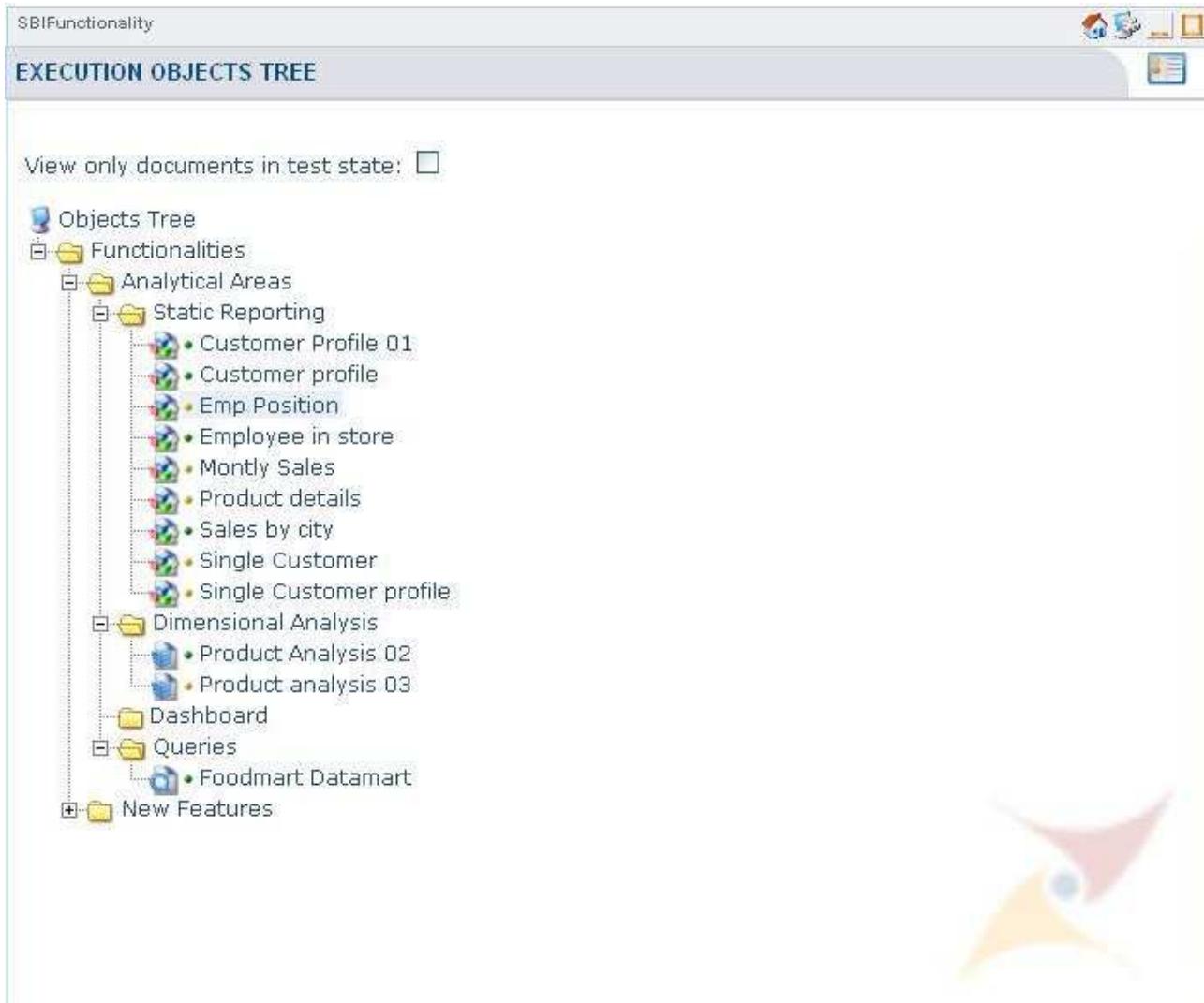
Label	MasterReport	*
Name	Master Report	*
Description	Master Report	
Type	Report	
Engine	Jasper Report Rel	
Criptable	<input type="radio"/> True <input checked="" type="radio"/> False	
Visible	<input checked="" type="radio"/> True <input type="radio"/> False	
Template	<input type="button" value=""/>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

**Figure 62 - Document visibility field**

## 6.4. TESTER

The tester (bitest/bitest 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 63 – 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 has a yellow led, a document in *Released* state has a green led). 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_PRF_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	Monty 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_PRF	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_PRF_FX	Single Customer	Single Customer	REPORT	TEST	1

**Figure 64 – 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.



**Figure 65 – 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).



**Figure 66 – Parameters page**

When a parameter is defined to have a *List* presentation, the corresponding input field becomes a *Lookup* table where you can choose its 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 67 – 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 execution icon.

Customer profile: Customer profile

### Customer profile

Name	Albright	Gender	M	Marital Status	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
March	21,76	10
April	3,78	1
May	53,15	29
June	18,34	8
July	18,23	8
August	39,53	22
		St.Costs
		5,24
		1,66
		14,33
		5,00
		7,11
		13,62

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

**Figure 68 – Final execution**

Now the test user can update the document state to *Released*, 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.

## 6.5. END-USER

The end-user (staff\_gen/staff\_gen is an user example) 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.

In order to limit user freedom degrees, document can be configured to show to the user only a portion of the datamart (see OLAP and Qbe profiled data mechanism).

When executing a document, the end-user can edit some notes directly in the execution page: these notes are correlated to the document functional parameters values, so executions with different parameters values will have different notes. The notes can be disabled/abled by portlet preferences in the execution portlet.



The screenshot shows the SpagoBI Notes editor interface. The title bar reads "Employee in store: Employee in store". The left pane contains a toolbar with icons for back, forward, search, and other functions. The right pane displays a table titled "Employee for store" with the following data:

Surname	Name	Gen	Mng.Role	Position	Ed.Lev	Salary
Binali	Brian	M	Middle Management	HQ Finance and	High School	5.000
Buker	Terri	F	Middle Management	HQ Finance and	Graduate Degree	5.000
De Carlo	Lauretta	F	Middle Management	HQ Finance and	High School	6.500
Lozada	Concepcion	F	Middle Management	HQ Finance and	Partial College	5.000
Osborn	Audrey	F	Middle Management	HQ Finance and	Bachelors	5.000
Sims	Rose	F	Middle Management	HQ Finance and	High School	6.600
Staton	Ernest	M	Middle Management	HQ Finance and	Graduate Degree	6.800
Williams	Mary	F	Middle Management	HQ Finance and	Partial College	7.200

**Figure 69 - Notes editor**

## 6.6. EVENTS

In SpagoBI an "event" is the registration of an action that is performed by a user; an event is correlated to some portal roles with a logic that depends on the type of action and it is notified to all the users having at least one of those roles. The images below show a list of events and an event detail.

At the time being only SpagoBIWekaEngine (the engine for data-mining processes integrating Weka) and SpagoBITalendEngine (the engine for ETL processes) can register events: the engine registers when a process is started and when it is completed (as they are asynchronous processes). These events are notified to the users having at least a role that can execute the data-mining/ETL document. In the detail page of the end process event there is a link to the document that will show the process results.

The events management and registration is under a phase of greater integration through.

Events list			
Event id	Event date	User	Event description
142	29-09-2006	biadmin	Started execution of Weka data mining process. Exe...
143	29-09-2006	biadmin	Execution of Weka flow successfully terminated! Ex...
page 1 of 1			
The value of the column <input type="text" value="Event id"/> as a <input type="text" value="string"/> starts with <input type="text"/> <input type="button" value="Filter All"/>			

**Figure 70 - Events list**

Event details	
<b>Identifier</b>	143
<b>Date</b>	2006-09-29 13:49:46.886
<b>User</b>	biadmin
<b>Document</b>	Dmp_user_seg: User Segmentation [User Segmentation]
<b>Description</b>	Execution of Weka flow successfully terminated! Execution parameters: <ul style="list-style-type: none"><li>• keys = banana</li><li>• versionColumnName = version</li><li>• clusterNum = 20</li><li>• version = test7</li><li>• clusterer = weka.clusterers.SimpleKMeans</li><li>• writeMode = DROP_INSERT</li><li>• versioning = TRUE</li></ul> The execution start event id is 142. <a href="#">ClusterDesc: Cluster Description [Cluster Description]</a>
<b>Related documents</b>	

**Figure 71 - Event detail**

## 7. 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.



### 7.1. INSTALL EXO PORTAL TOMCAT AND SPAGOBI DEMO

Download eXo Portal 1.1.4 Tomcat from :

[http://forge.objectweb.org/project/download.php?group\\_id=151&file\\_id=8171](http://forge.objectweb.org/project/download.php?group_id=151&file_id=8171)

Unzip the file exo-portal-1.1.4-tomcat.zip but do NOT run the application, because this would cause the failure of the following SpagoBI installation.

Download SpagoBI Installer 1.9.3 from:

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

Unzip the file and launch the following file:

*StartSpagoBIInstaller.bat* (Windows users)

*StartSpagoBIInstaller.sh* (Linux users)

Follow the instructions to complete the installation process: select to install all the engines and the examples. Notice that it will be required to specify the directory where you have previously unzipped eXo Portal 1.1.4 Tomcat.

Once completed SpagoBI installation, go to the home directory of eXo Portal 1.1.4 Tomcat and launch

*startSpagoBI.bat* (Windows users)

*startSpagoBI.sh* (Linux users)

### 7.2. 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.

---

### 7.2.1. CREATE A REPORT TEMPLATE

#### 7.2.2. CREATE A REPORT TEMPLATE USING IREPORT

1. If you don't have iReport 1.3.2 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 older *hsqldb-1.7.1.jar* library.
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:hsq://localhost:9002/foodmart
- **Username:** sa



Note that HSQLDB server must be running on port 9002: this happen if you installed demo examples (chapter 7.1) without modifying default HSQLDB port.

---

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”.

### 7.2.3. CREATE A REPORT TEMPLATE USING BIRT

1. If you don't have Birt 2.2.0 you can download it from:  
<http://europa-mirror1.eclipse.org/birt/downloads/>  
We suggest to download the all-in-one package you find under “Report Designer Release Build 2.2.0” section.
2. Then copy the database driver hsqldb1\_8\_0\_2.jar inside the following directory:  
ECLIPSE-HOME\plugins\org.eclipse.birt.report.data.oda.jdbc\_2.2.0.vxxxxxx\drivers.
3. Now you can start Eclipse-Birt.
4. In order to create a simple report example, select File → New → Project → Business Intelligence and Reporting Tools → Report Project from the File menu. Then select File → New → Report from the same menu. Type ‘SpagoBI\_Example’ as Report name and click “Finish”.
5. From the Data Explorer view right click on Data Sources and edit a new Data Source: select ‘JDBC Data Source’ as Data Source type and enter “foodmart” as Data Source name and click “Next”. Then fill in the following information:
  - **Driver Class:** 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)
  - **Database URL:** jdbc:hsqldb:hsq://localhost:9002/foodmart
  - **Username:** sa
  - **JNDI URL:** java:comp/env/jdbc/sbifoodmart

Click “Finish”.

6. 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
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. Then 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= ?
```

Click "Finish". Then go to **Parameters** view and add this parameter:

- **name:** EmployeePosPar
- **Data Type:** String
- **Direction:** input
- **Default Value:** 'Store Manager'
- **Linked to report parameter:** 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*.

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

#### 7.2.4. 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.

#### 7.2.5. 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 "Test before save" button)
- **Visible Columns:** POSITION\_TITLE
- **Value Column:** POSITION\_TITLE
- **Description 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:

- **Value:** HQ Information System  
**Description:** HQ Information System  
(click on the ADD icon)
- **Value:** HQ Marketing  
**Description:** HQ Marketing  
(click on the ADD icon)
- **Value:** HQ Human Resources  
**Description:** HQ Human Resources  
(click on the ADD icon)
- **Value:** HQ Finance and Accounting  
**Description:** HQ Finance and Accounting  
(click on the ADD icon)

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

Then click on "Test before save" button and confirm

- **Visible Columns:** DESCRIPTION
- **Value Column:** VALUE
- **Description Column:** DESCRIPTION

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

### 7.2.6. 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 .

### 7.2.7. 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
  - **Functional:** yes (check it)

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**
- **Presentation:** List values selection

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**
- **Presentation:** ComboBox

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*.

#### **7.2.8. 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: check a folder in the tree you see on the right.
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.
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 SELECT 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

### 7.2.9. TEST THE ANALYTICAL DOCUMENT

---

Work in progress.



---

### 7.2.10. EXECUTE THE ANALYTICAL DOCUMENT

---

Work in progress.



---

## 7.3. OLAP ANALYSIS

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

---

### 7.3.1. 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;
- <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 *as* attribute refers to the parameter name contained in the query, while the *name* 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>
  <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], "what
      willste?").children} ON rows
      from Sales where ([Time].[1997])
    <parameter name='prdCd' as='ProductMember' />
  </MDXquery>
</olap>
```

Save the file as *productSales.olap*.

Please note that the example above uses a parameter with the proper MDX sintax. You can also insert parameters in a MDX query using SpagoBI sintax \${parameter\_name}: in this case the parameter value will literally substitute the parameter declaration before query execution. The example above could be:

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

Using the SpagoBI sintax you can insert more than one parameter in the same dimension in a very easy way (example: [Time].[\$year].[\${month}]).

### **7.3.2. CREATE PARAMETERS**

Connect to the home page of SpagoBI demo portal (<http://localhost:8080/sbiportal>) 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.

### 7.3.3. 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

Insert the following value-description pairs:

- **Description:** Non Consumable  
**Value:** [Product].[All Products].[Non-Consumable]  
(click on the ADD icon)
- **Description:** Food  
**Value:** [Product].[All Products].[Food]  
(click on the ADD icon)
- **Description:** Drink  
**Value:** [Product].[All Products].[Drink]  
(click on the ADD icon)

Then click on "Test before save" button and confirm

- **Visible Columns:** DESCRIPTION
- **Value Column:** VALUE
- **Description Column:** DESCRIPTION

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

Insert the following value-description pairs:

- **Description:** Non Consumable  
**Value:** [Product].[All Products].[Non-Consumable]  
(click on the ADD icon)
- **Description:** Food  
**Value:** [Product].[All Products].[Food]  
(click on the ADD icon)

Then click on "Test before save" button and confirm

- **Visible Columns:** DESCRIPTION
- **Value Column:** VALUE
- **Description Column:** DESCRIPTION

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

#### **7.3.4. 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
  - **Description**
4. Select *MAXLENGTH* 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.

#### **7.3.5. 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
  - **Functional:** yes (check it)

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**
- **Presentation**: ComboBox

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**
- **Presentation**: ComboBox

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*.

### 7.3.6. 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
  - **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: check a folder in the tree on the right.
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 (*name* 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

### **7.3.7. TEST THE ANALYTICAL DOCUMENT**

Work in progress.



### **7.3.8. EXECUTE THE ANALYTICAL DOCUMENT**

Work in progress.



### **7.3.9. DYNAMIC TEMPLATE CREATION**

Since SpagoBI version 1.9.2, it is possible to build the template of an OLAP document dynamically, in alternative of what you read in chapter 7.3.1.

Let's see an example. Configure a new document:

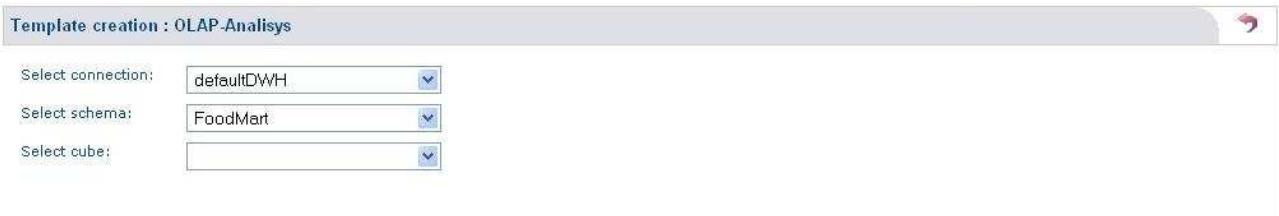
**Figure 72 - New OLAP document**

Then click on the Save button without inserting any template file. Then click on the template build button.

Now you have to specify the connection to be used and the schema definition available (available connections and schema are specified in exo-home/webapps/SpagoBI/pivotEngine/WEB-INF/classes/engine-config.xml).

**Figure 73 - Connection and schema selection**

Then you have to specify the analisys cube.



**Figure 74 - Cube selection**

After you chose the cube, you can use the olap cube navigator to specify the dimensions and measures to be displayed.

	Measures		
Product	Unit Sales	Store Cost	Store Sales
+Drink	24.597	4.505,83	48.836,21

**Figure 75 - Cube navigation**

Then you can save the template (without parameters): click on the save template button and fill the form with the template name, then click on the "Save" button.

	Measures		
Product	Unit Sales	Store Cost	Store Sales
+Drink	24.597	4.505,83	48.836,21

**Figure 76 - Saving template without parameters**

If you want to insert also one or more parameter click on the "Add parameters" button: you will see the MDX query you have obtained using the cube navigator.

Template creation : OLAP-Analysys


**Type parameters in the query below:**

```
select {[Measures].[Unit Sales], [Measures].[Store Cost], [Measures].[Store Sales]} ON COLUMNS,
 {[Product].[All Products].[Drink]} ON ROWS
 from [Sales]
```

**Type name and url name for each parameter inserted above:**

Insert parameter name:



Insert parameter url name:

**Figure 77 - Add parameters page**

Now you can modify manually the query in order to insert a parameter. You can use both Mondrian or SpagoBI parameter syntax. In this example we use SpagoBI syntax, so change [Product].[All Products].[Drink]

with

[Product].[All Products].[\$ProductFamily}]

in the MDX query and fill the form below as shown in figure:

Template creation : OLAP-Analysys


**Type parameters in the query below:**

```
select {[Measures].[Unit Sales], [Measures].[Store Cost], [Measures].[Store Sales]} ON COLUMNS,
 {[Product].[All Products].[$ProductFamily]} ON ROWS
 from [Sales]
```

**Type name and url name for each parameter inserted above:**

Insert parameter name:



Insert parameter url name:

**Figure 78 - Parameter definition**

Then click on the "Add parameter" button (the one with the "+" symbol) and the parameter will be registered.

Template creation : OLAP-Analysys

**Type parameters in the query below:**

```
select {[Measures].[Unit Sales], [Measures].[Store Cost], [Measures].[Store Sales]} ON COLUMNS,
    {[Product].[All Products].[$ProductFamily]} ON ROWS
from [Sales]
```

**Type name and url name for each parameter inserted above:**

Insert parameter name:	<input type="text"/>	+
Insert parameter url name:	<input type="text"/>	

**Defined parameters:**

Parameter name	Parameter url name
ProductFamily	ParProductFamily

**Figure 79 - Parameter inserted**

Then you can save the new template by clicking on the "Save template" button, filling the form with the name of the template and then clicking the "Save" button.

At last you have to configure the parameter as a document parameter: supposing you have previously defined a fixed list of values with values "Food", "Drink" and "Non-Consumable" (remember that we used the SpagoBI parameter syntax) and associated this lov to a parameter named "Product family", configure a document parameter as in the figure below. Note that SpagoBI parameter url name and MDX query parameter url name must be the same.

**DOCUMENT DETAILS**

Label	<input type="text" value="OLAP-Analysys"/> *
Name	<input type="text" value="OLAP-Analysys"/> *
Description	<input type="text"/>
Type	<input type="button" value="On-line analytical processing"/>
Engine	<input type="button" value="Jpivot-Mondrian"/>
Criptable	<input type="radio"/> True <input checked="" type="radio"/> False
Visible	<input checked="" type="radio"/> True <input type="radio"/> False
Template	<input type="button"/> Sfoglia... <input type="button"/>
Template build	

**Show document templates**

- Functionalities Tree
- Functionalities
  - General Management
  - Marketing
    - Product Analysis
    - Sales Analysis
    - Human Resources

**Product family** **New...**

**DOCUMENT PARAMETER DETAILS**

Title	<input type="text" value="Product family"/> *
Parameter	<input type="text" value="Products family"/> *
Url Name	<input type="text" value="ParProductFamily"/> *
Priority	<input type="button" value="1"/>

**Figure 80 - Document parameter definition**

Note that you can go into the template build page to modify the query (you will have to specify the connection to be used, while schema and cube will remain the same). If you inserted parameters with SpagoBI syntax, the system will execute the last Mondrian query (the query obtained before adding parameters); use the Jpivot interface to modify the query and then re-define the parameters as seen before.

### **7.3.10. PROFILED DATA**

An olap document is a way to inquiry your business data freely and dinamically. This means that normally the user start his analysis from a predefined view and then he can completely change the inquiry interacting with the platform. This freedom to surfing through business data could be dangerous because sometimes you want that different users cannot analyze some parts of the datamart.

Think for instance to a regional director and to a state director who use the same olap document, in this case you probably want that the first one can see only the data of his region while the second one can see the data of the entire state. To satisfy this requirement there is need of a profiled access to the data and SpagoBI platform manages it using the attributes of the user profile.

In order to configure a profiled access for an olap document you need to change its template. Unfortunately this template configuration cannot be done using the 'Dynamic template creation' procedure described above (at least for this release) and so you have to change the template manually. What you have to do is to add an xml envelope, like the one below, into the root tag <olap> of the template.

```
<DATA-ACCESS>
  <GRANTED-DIMENSIONS>
    <DIMENSION name="Product" grantSource="ProfileAttributes" >
      <RULES access="custom"
        topLevel="[Product].[Product Family]"
        bottomLevel="[Product].[Product Family].[Product Department]" >
        <MEMBERS>
          <MEMBER name="[Product].[All Products].[$family]" access="all" />
          <MEMBER .....
        </MEMBERS>
      </RULES>
    </DIMENSION>
    <DIMENSION .....
  </GRANTED-DIMENSIONS>
</DATA-ACCESS>
```

The idea at the base of this xml is the possibility to restrict the access to one or more cube dimensions. Each dimension can have three level access:

- none: the dimension won't be available to the olap document
- custom: only some members of the dimension will be available
- all: complete access to the dimension

Only if you decide to apply the custom access you have to configure the dimension members access. To do this you need to consider that dimensions are structured as hierarchy of levels. For this reason a member is expressed using a hierarchical dot name. A member hierarchical name identifies the member itself and all its children. Inside the name of the member you can place an expression like \${region}. This expression means that you want to replace it with the value of the user profile attribute 'region'. Example: if the user has a profile attribute region with value 'North-West' and member name is [Region].[All Regions].[\$region] the final and real member that will be used is [Region].[All Regions].[North-West].

Each member can have two level access:

- none: the member and all its children won't be available
- all: the member, its ancestors and its children will be available

Remember that if you configure the access to the members, all the members not defined won't be available.

In addition there's the possibility to define the lower and higher level boundaries for the dimension members. In that case only the members inside the boundaries will be considered.

Inside one dimension you can define the access for one or more members.

After this considerations the description of the xml added should be more clear.

The first two level tags are fixed and you cannot change them. Inside the GRANTED-DIMENSIONS tag you can insert one or more DIMENSION tags, all with the same structure.

- DIMENSION.name = the name of cube dimension to filter (mondrian cube dimension name)
- DIMENSION.grantSource = this attribute identifies which information the platform recovers from the user profile. You should not change this attribute but, just to know it, there's also an old but still supported modality, called ProfileFunctionalities, which is based only on the eXo portal security environment. If you want more information about this modality take a look to the section 'How to configure data filtering using SpagoBIJpivotEngine ?' of the 'How to' document (SpagoBI documentation).
- DIMENSION.RULES.access: the access level of the dimension (none, custom, all)
- DIMENSION.RULES.topLevel: the higher boundary of the dimension level. All ancestor members won't be considered. Note that the level is expressed using a dot hierarchical notation. Each part of the notation is the name of a dimension level.
- DIMENSION.RULES.bottomLevel: the lower boundary of the dimension level. All child members won't be considered. Note that the level is expressed using a dot hierarchical notation. Each part of the notation is the name of a dimension level.
- DIMENSION.RULES.MEMBERS.MEMBER.name: the name of the member containing the expression to substitute with the value of the profile attribute. The profile expression can be placed in every position of the member name and it is also possible to insert more than one expression.
- DIMENSION.RULES.MEMBERS.MEMBER.access: the access level for a member (none, all)

Once configured the template, each time a user will exec the document, the platform will substitute the profile attribute values to the relative placeholder in member name and then send all the information to the olap engine.

The olap engine, once received the filter information, will prevent any access to the members and dimensions not authorized.

Considering the above example and supposing that the family profile attribute of the user contains the value 'Drink', the engine will allow to the user to access only to:

- the ancestors of the Product dimension member [Product].[All Products].[Drink]
- the children of the Product dimension member [Product].[All Products].[Drink] till the level [Product Department]. It is not possible to go more in deep
- all the member of the other dimensions (not Product).

## 7.4. 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 OpenLaszlo.

In this document we will not explain how to build a SWF movie with OpenLaszlo; we invite you to visit the OpenLaszlo 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

#### **7.4.1. 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.

Then click on "Test before save" button and confirm

- **Visible Columns:** value
- **Value Column:** value
- **Description Column:** 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.

#### 7.4.2. 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.

---

#### 7.4.3. 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 previous paragraph.
6. Moreover, you have to indicate the parent folder of the document: check a folder on the tree on the right.
7. 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

#### **7.4.4. TEST THE ANALYTICAL DOCUMENT**

---

Work in progress.



---

#### **7.4.5. EXECUTE THE ANALYTICAL DOCUMENT**

---

Work in progress.



---

### **7.5. 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

### 7.5.1. 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.

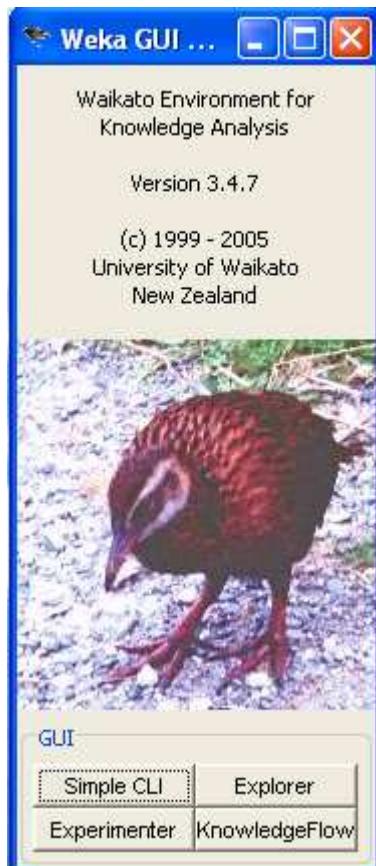


Figure 81 - Weka GUI Chooser window

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 filed (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.

## 7.5.2. 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 happen because Weka applies constraints to some properties' value format. Using parameters as value of that properties violates the applied constraints and cause errors (usually default values are used instead the inserted ones). In these cases you have to type your parameter directly inside the generated *kfm* 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 *kfm* 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 possible now to replace this line with the following parametrized one ...

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

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

### 7.5.3. 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.

### 7.5.4. 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:** returnValue('hsqlfoodmart')

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

Then click on "Test before save" button and confirm

- **Visible Columns:** value
- **Value Column:** value
- **Description Column:** value

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:

- **Description:** Simple K-Means  
**Value:** weka.clusterers.SimpleKMeans  
(click on the ADD icon)
- **Description:** Farthest First  
**Value:** weka.clusterers.FarthestFirst

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

Then click on "Test before save" button and confirm

- **Visible Columns:** DESCRIPTION
- **Value Column:** VALUE
- **Description Column:** DESCRIPTION

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

### 7.5.5. PREDEFINED VALUES CONSTRAINTS

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

### 7.5.6. 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
  - **Functional:** no (don't check it)

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 HSQL\_FM\_CONN LOV from lookup list
- **Presentation:** anything, since the lov is single-value (parameter will not be shown)

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
  - **Functional:** yes (check it)

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
- **Presentation:** ComboBox

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
- **Functional:** yes (check it)

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
- **Manual 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.

### 7.5.7. 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: check a folder in the tree you see on the right.

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.

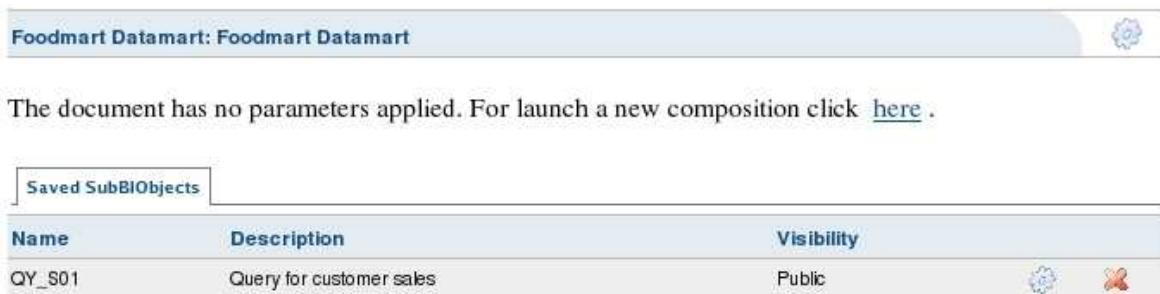
12. Click on the icon EXECUTE which corresponds to the new *DATA\_MINING* analytica document.

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. You can see the events relevant to the process start and process end into the Events Monitor portlet (see chapter "6.6 Events").
18. Logout

## 7.6. 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.

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



The screenshot shows a web-based application window titled "Foodmart Datamart: Foodmart Datamart". At the top, there is a button with a gear icon. Below the title, a message says "The document has no parameters applied. For launch a new composition click [here](#)." A table titled "Saved SubBIO Objects" is displayed, containing one row:

Name	Description	Visibility
QY_S01	Query for customer sales	Public

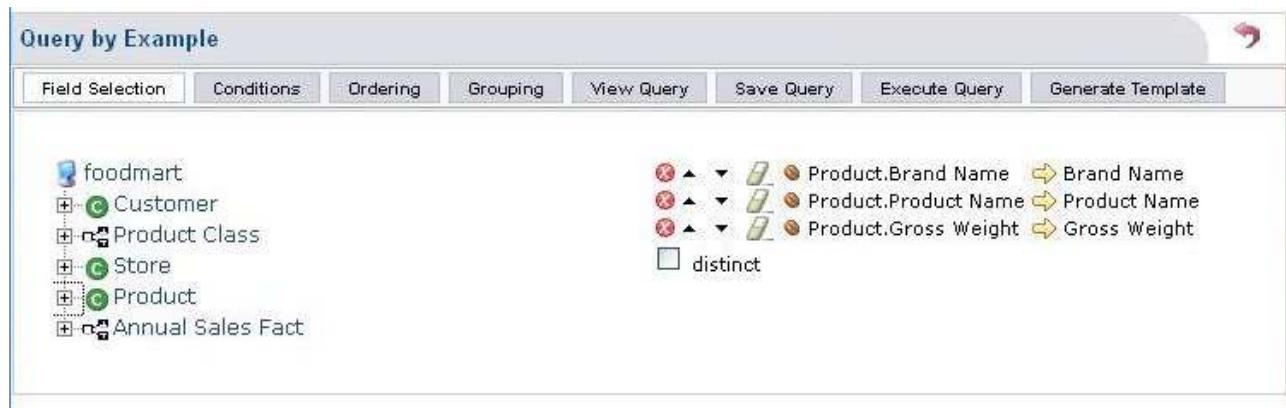
**Figure 82 – QbE: Query list**

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 eight steps that will ease the user to create a new query:

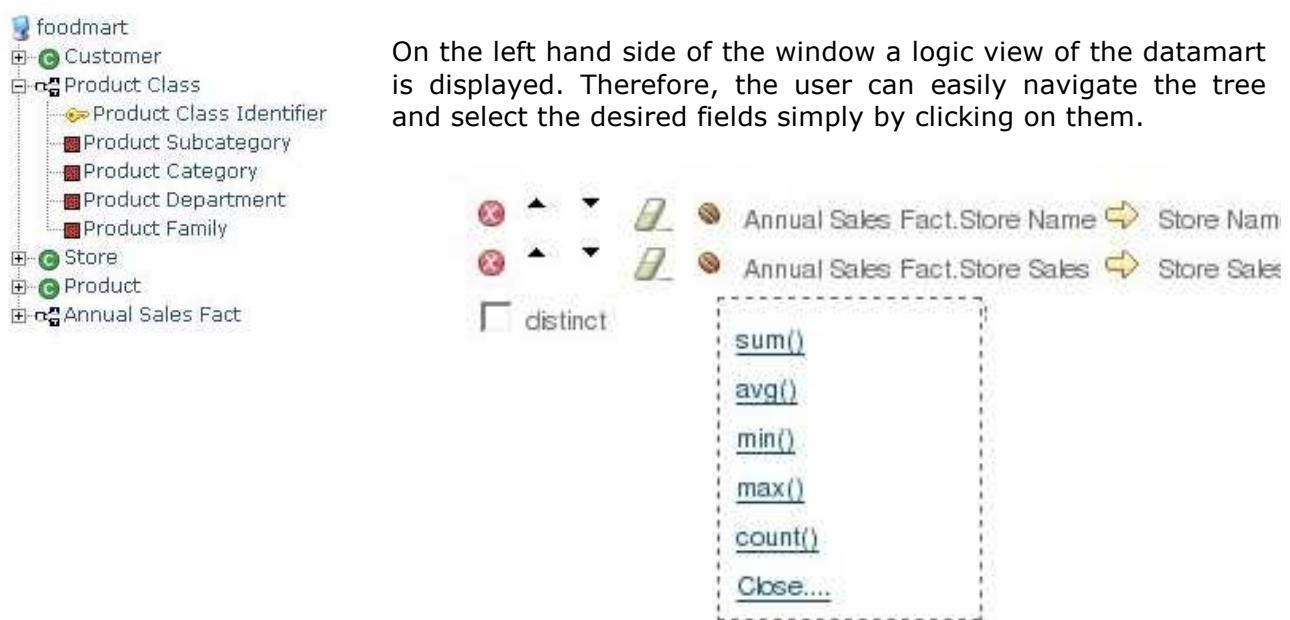
- **Select Fields** : to choose the **select** fields;
- **Conditions**: to set the **where** conditions;
- **Ordering**: to select the fields for the **order by**;
- **Grouping**: to select the fields for the **group by**

- 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 83 – QbE: Field Selection**

### 7.6.1. 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.

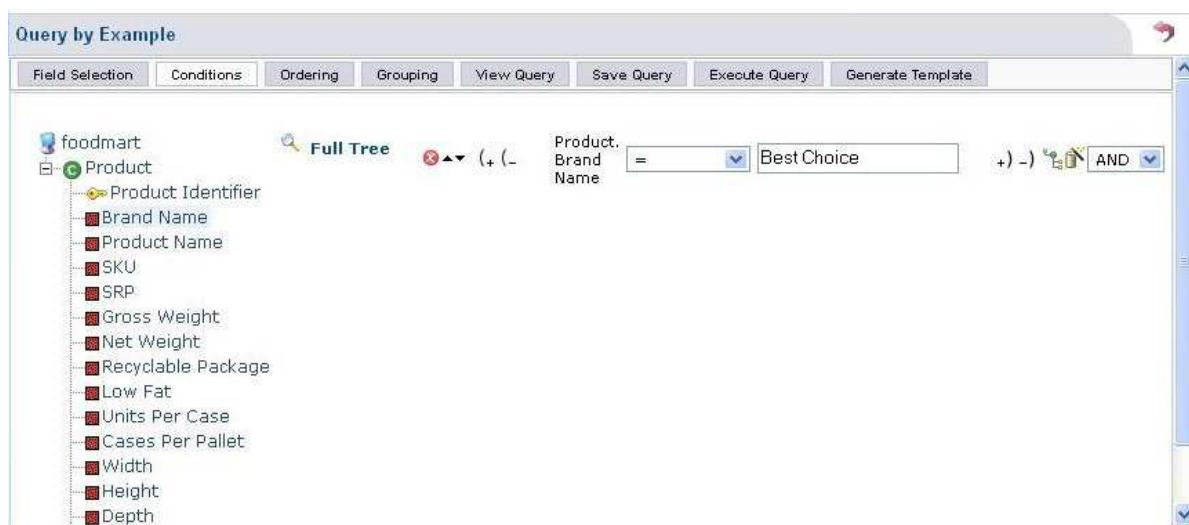
**Figure 84 – QbE: Field Selection detail**

Once selected a field, it appears on the right hand side.  
The following option 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.

## 7.6.2. CONDITION



**Figure 85 – 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.

The user can also use brackets in order to produce complex condition expressions.



**Figure 86 - 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 selected 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.

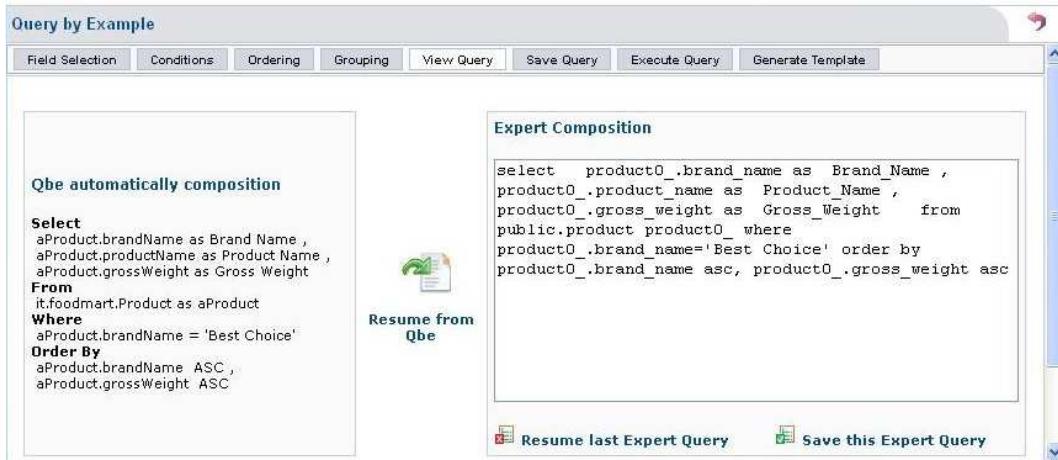
If the user tries to access one of these pages without having already selected at least a field in **Field Selection** page, the Qbe displays an error message

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

- To delete
- To sort

### 7.6.3. VIEW QUERY

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



**Qbe automatically composition**

```
select
  aProduct.brandName as Brand Name ,
  aProduct.productName as Product Name ,
  aProduct.grossWeight as Gross Weight
from
  it.foodmart.Product as aProduct
where
  aProduct.brandName = 'Best Choice'
order by
  aProduct.brandName ASC ,
  aProduct.grossWeight ASC
```

**Expert Composition**

```
select product0_.brand_name as Brand_Name ,
product0_.product_name as Product_Name ,
product0_.gross_weight as Gross_Weight   from
public.product product0_ where
product0_.brand_name='Best Choice' order by
product0_.brand_name asc, product0_.gross_weight asc
```

**Resume from Qbe**

**Resume last Expert Query**   **Save this Expert Query**

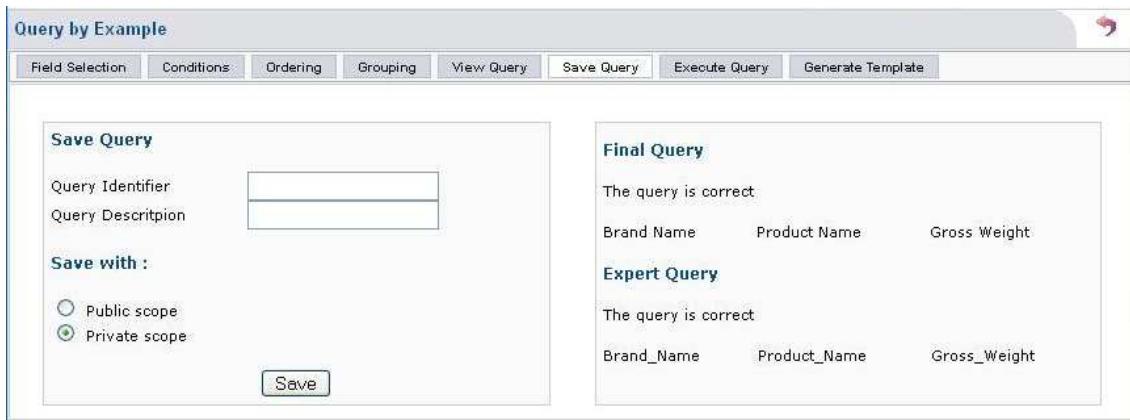
**Figure 87 – QbE: View Query**

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.

#### 7.6.4. SAVE QUERY



**Save Query**

Query Identifier:

Query Description:

Save with :

Public scope  
 Private scope

**Save**

**Final Query**

The query is correct:

Brand Name	Product Name	Gross Weight
Brand_Name	Product_Name	Gross_Weight

**Expert Query**

The query is correct:

Brand_Name	Product_Name	Gross_Weight
Brand_Name	Product_Name	Gross_Weight

**Figure 88 – 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.

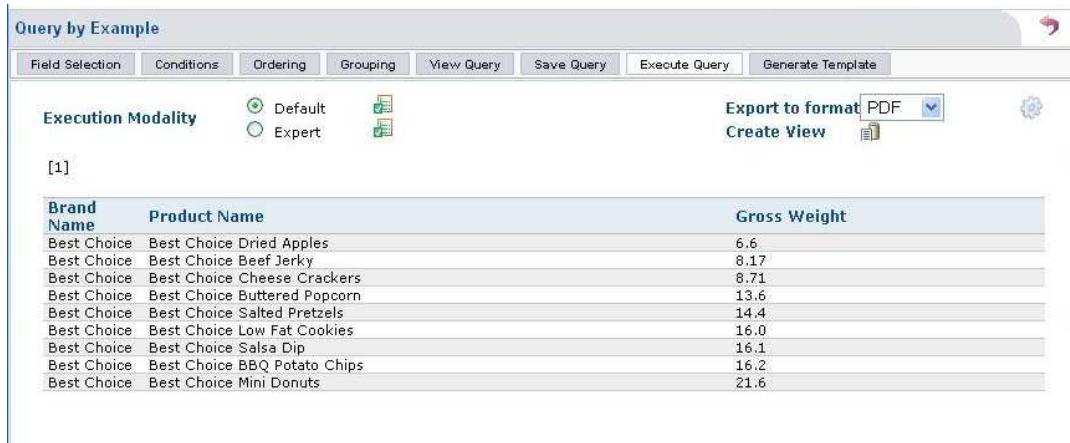
On the other side of the page, the user can fill in the form required to save the query.

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

### 7.6.5. EXECUTE QUERY



The screenshot shows the 'Query by Example' interface. At the top, there are tabs for 'Field Selection', 'Conditions', 'Ordering', 'Grouping', 'View Query', 'Save Query', 'Execute Query', and 'Generate Template'. Below these are buttons for 'Execution Modality' (Default selected, Expert), 'Export to format' (PDF selected), and 'Create View'. A table below displays product data:

Brand Name	Product Name	Gross 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 89 – QbE: Execute query**

The **Execute query** page first tries to execute the automatically composed query.

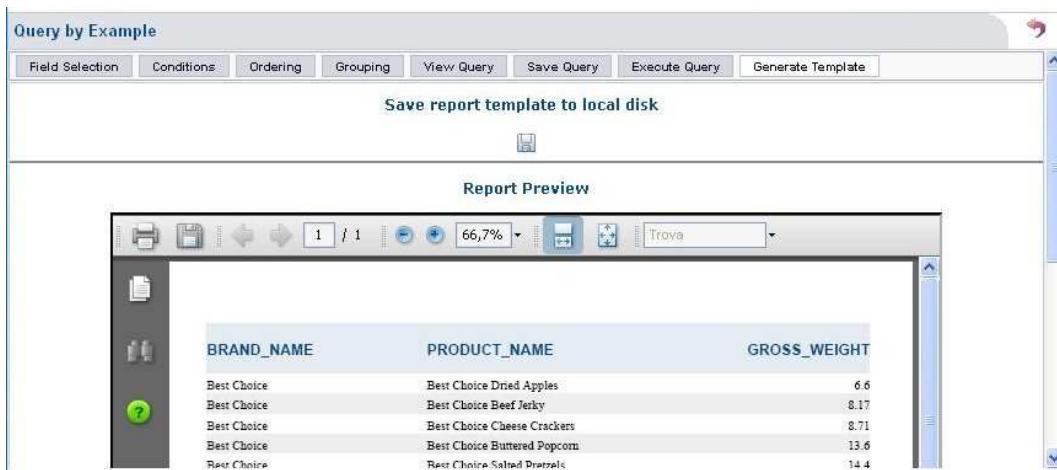
If the query works correctly, a table containing the result set is displayed. Otherwise, a text area containing a description of the error occurred will be displayed.

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-configurated plugin viewer for the target exportation format the exportation result can be visualized into the browser window before saving.

### 7.6.6. 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.



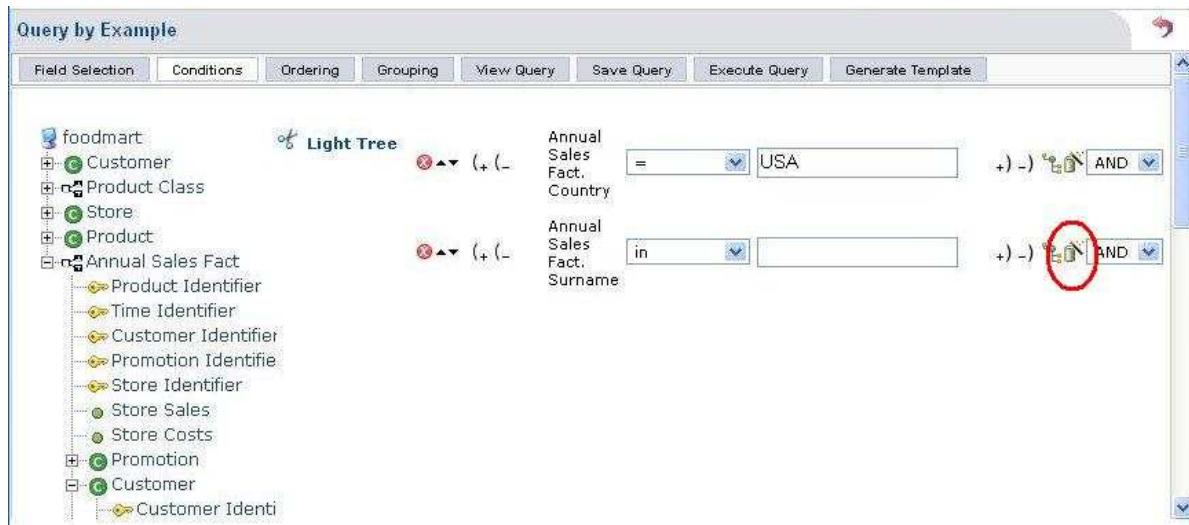
The screenshot shows the SpagoBI QbE interface. At the top, there's a toolbar with buttons for Field Selection, Conditions, Ordering, Grouping, View Query, Save Query, Execute Query, and Generate Template. Below the toolbar, a message says "Save report template to local disk". Underneath that is a "Report Preview" section. The preview window has a toolbar at the top with icons for print, save, and zoom (66,7%). It displays a table with three columns: BRAND\_NAME, PRODUCT\_NAME, and GROSS\_WEIGHT. The data in the table is:

BRAND_NAME	PRODUCT_NAME	GROSS_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 Salty Pretzels	14.4

**Figure 90 – QbE: Generate Template**

### 7.6.7. SUB-QUERY

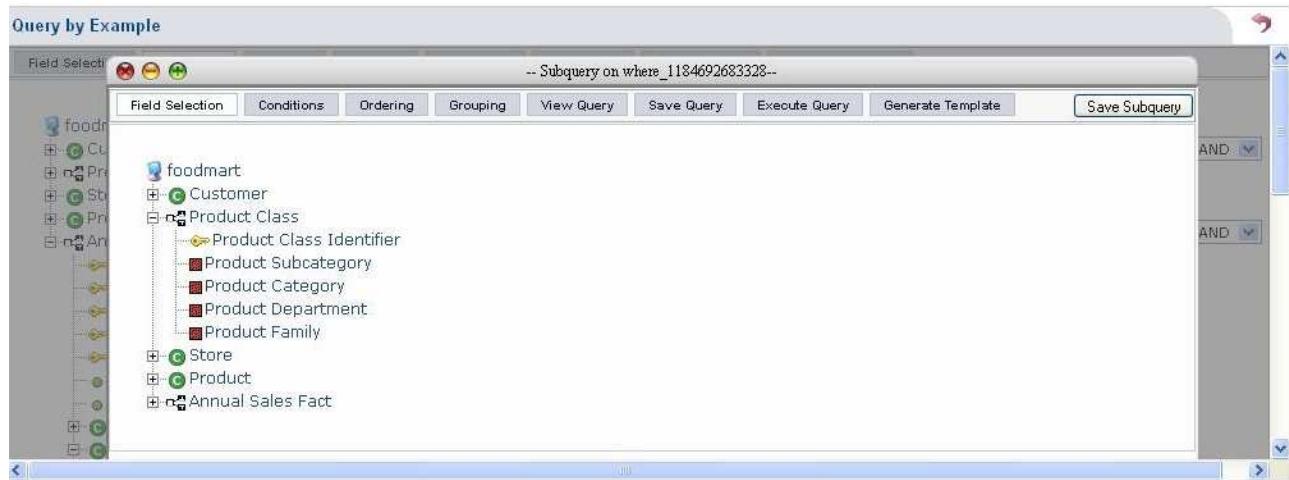
Qbe can handle subquery. The result of a subquery can be used as value in the right term of a where clause. To associate a new subquery to a where clause just click the proper button situated on the right of the choosen clause in the conditions page.



The screenshot shows the SpagoBI QbE interface on the "Conditions" tab. On the left is a "Light Tree" sidebar with nodes like foodmart, Customer, Product Class, Store, Product, and Annual Sales Fact. The "Annual Sales Fact" node is expanded, showing sub-nodes like Product Identifier, Time Identifier, Customer Identifier, Promotion Identifier, Store Identifier, Store Sales, Store Costs, Promotion, Customer, and Customer Identifier. In the main area, there are two clauses under "Conditions". The first clause has a condition "Annual Sales Fact. Country = USA" with an "AND" button to its right. The second clause has a condition "Annual Sales Fact. Surname in [ ]" with an "AND" button to its right. The "AND" button for the second clause is circled in red.

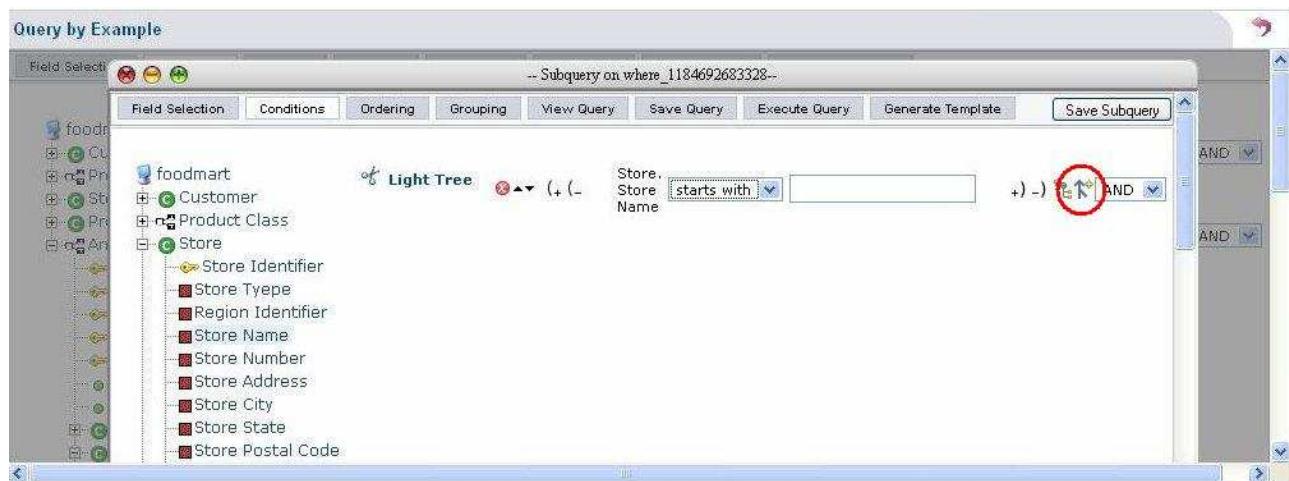
**Figure 91 - Sub-query button**

A new Qbe window will be popped up. You can now build the subquery in the same way you build a standard query using all the qbe facilites.



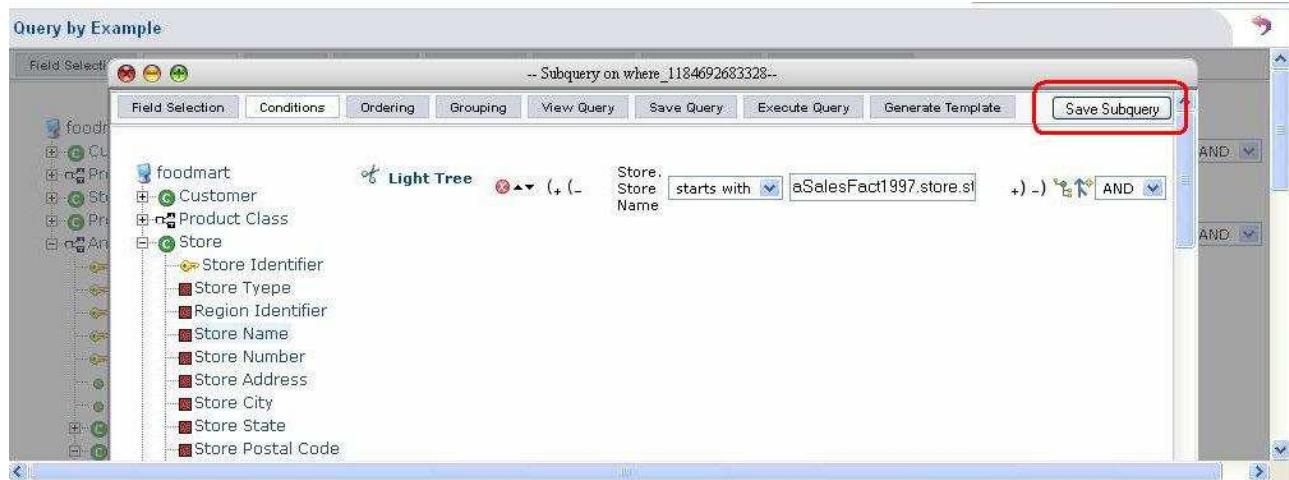
**Figure 92 - Sub-query fields selection window**

There are anyway two important things to keep in mind when you build a sub query. First, only one field is allowed in the selection clause. This should be the field holding the value or values you want to check against in the selected where clause of the parent query. Second, in the where clauses of a subquery you can reference field appearing in the parent clause through the 'join with parent' button, situated on the right side of each where clause.



**Figure 93 - "Join with parent" button**

When you make a reference to fields appearing in the parent query it is not possible to execute, export or save the subquery separated from his parent query. Once you have finished to define the subquery you can save the subquery clicking on the "Save Subquery" button.

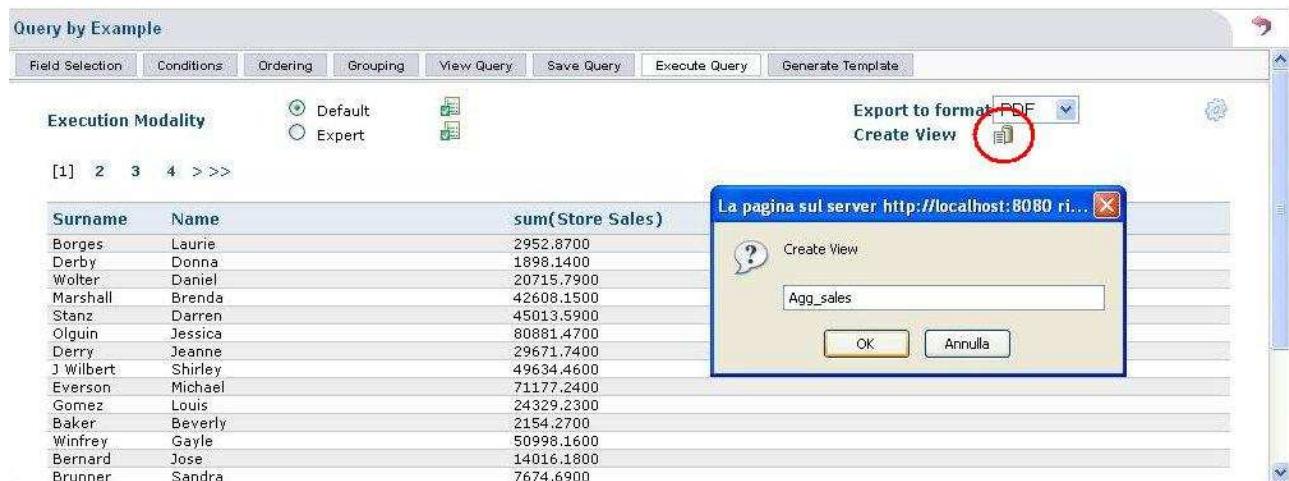


**Figure 94 - Sub-query window save button**

A saved subquery than can be modified or just displayed at any time using the same button you used to create it.

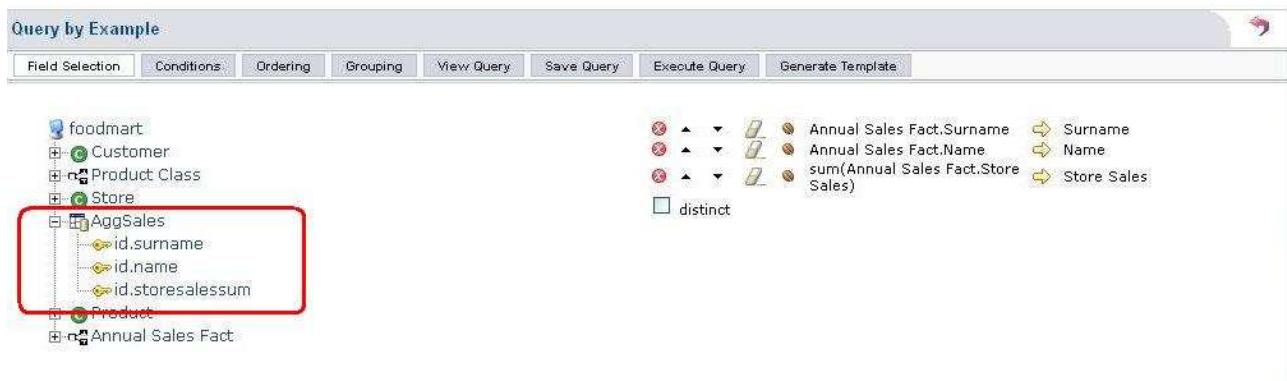
### 7.6.8. CREATE VIEW

Each query built with Qbe can be saved in the database as view using the 'create view' button situated in the up right corner of the 'execute query' page.



**Figure 95 - Create a view**

The newly saved view are treated by Qbe as a table. Saved views can so be used as a shortcut to define more complex query.



**Figure 96 - Created view**

### 7.6.9. CALCULATED FIELDS

SpagoBIQbeEngine uses Groovy scripts to generate calculated fields; those scripts must be contained inside the folder specified by the attribute 'dir' of the tag QBE-SCRIPT-DIR in file SpagoBIQbeEngine/WEB-INF/conf/qbe.xml.

The scripts must return an object that will be rendered through the `toString()` method; they can return a String that can be also HTML code, so they can generate a link:

```
return "<a href=\"...\">....</a>"
```

Scripts can be modified at runtime, without restarting the server.

To call groovy scripts in your datamart, you have to put an xml file named 'formula.xml' inside your datamart directory. As an example: suppose you have `<QBE-MART_DIR dir="tmp/datamarts"/>` and `<QBE-SCRIPT-DIR dir="tmp/groovy"/>`, you must have the following directories structure:

SpagoBIQbeEngine/

```
-----tmp/
-----datamarts/
-----foodmart/
-----datamart.jar
-----formula.xml
-----groovy/
-----customerInk.groovy
```

where `customerInk.groovy` is the file containing the groovy script (different datamarts can use the same script).

This is an example of formula.xml that defines 3 calculated fields:

```
<FORMULAS>
<FORMULA id="cField1"
  onEntity="it.foodmart.Customer"
  mode="manual"
  mappings="customerId->input1"
  resultingFieldName="Link Detail"
  script="customerInk.groovy"
  inExport="true"/>
<FORMULA id="cField2"
```

```

onEntity="it.foodmart.Customer"
mode="manual"
mappings="Iname->input1,fname->input2"
resultingFieldName="Link Name"
script="customername.groovy"
inExport="false"/>
<FORMULA id="cField3"
onEntity="it.foodmart.SalesFact1997"
mode="auto"
mappings="customer.customerId->input1"
resultingFieldName="Link Detail"
script="customerlnk.groovy"
inExport="false"/>
</FORMULAS>
```

The 'onEntity' attribute specifies the entity which the calculated field is based on.

The 'resultingFieldName' attribute specifies the name of the calculated field that will be showed in the query execution.

The 'script' attribute specifies the groovy script file to be executed.

The 'mappings' attribute specifies the associations between the properties of the entity to be passed to the groovy script and the script input variables (example based on formula with id="cField2": property 'Iname' of entity it.foodmart.Customer will be passed to the script as a variable named 'input1' while the property 'fname' will be passed as another variable named 'input2'). These script variables can be used in the script with the following syntax: \${variable name} (see below the script example).

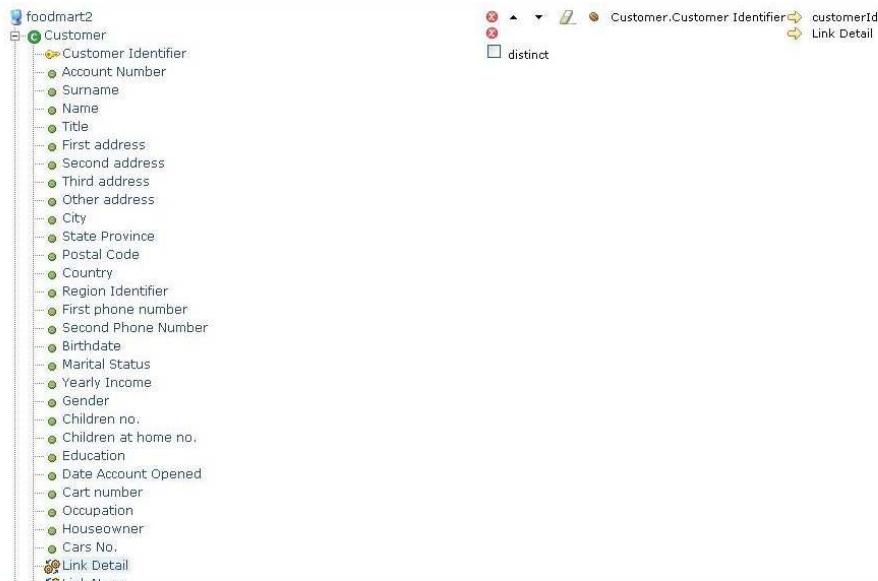
The 'mode' attribute can be 'manual' or 'auto'.

'manual' mode means that you will find the calculated field as a new property of the base entity in the entities selection tree.



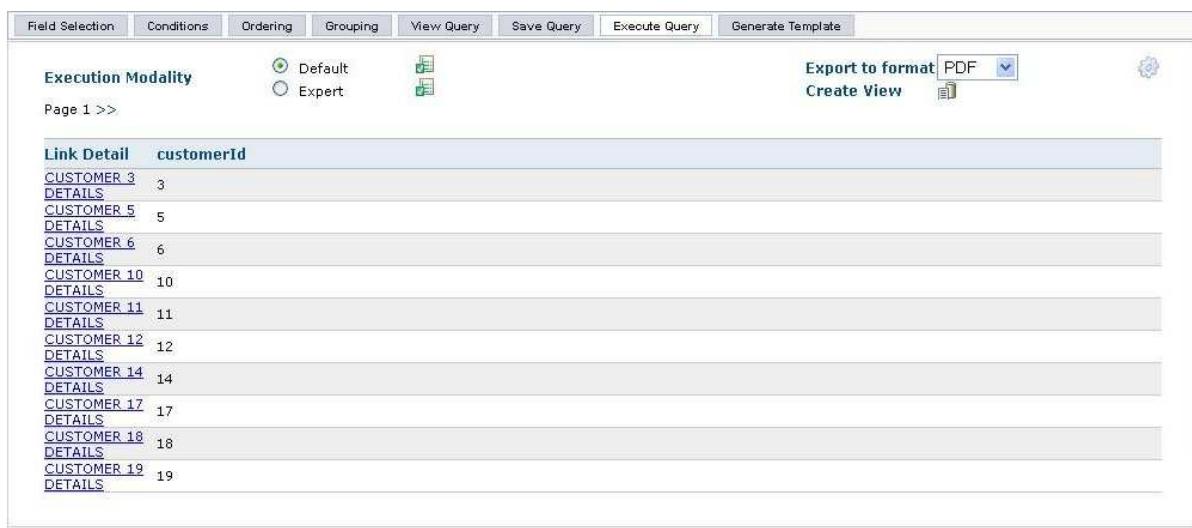
**Figure 97 - Calculated fields, mode manual**

When you select the calculated field all the properties that are needed to calculate this field will be automatically selected also (example based on formula with id="cField1": if you select "Link Detail" the property it.foodmart.Customer.customerId will be automatically selected also).



**Figure 98 - "Manual" mode calculated field selection**

Then, if you execute the query, you will see the calculated field.



**Figure 99 - "Manual" mode calculated field execution**

'auto' mode means that the calculated field will be always calculated when you select the base entity (the entities selection tree will not show the calculated field if you use this modality); you have also to select on the entities selection tree all the fields which the calculated field depends on,



The screenshot shows the SpagoBI Field Selection interface. At the top, there are tabs: Field Selection, Conditions, Ordering, Grouping, View Query, Save Query, Execute Query, and Generate Template. The 'Field Selection' tab is active.

The left pane displays a hierarchical tree of fields from the 'foodmart2' database, including Customer, Promotion, Product Class, Store, and various tables like 'Annual Sales Fact' and 'Product'. The right pane shows the selected fields for the calculated field 'Customer Identifier'. The selected fields are: 'Annual Sales Fact.Store Name' (with a yellow arrow pointing to 'Store Name'), 'Annual Sales Fact.Customer Identifier' (with a yellow arrow pointing to 'Customer Identifier'), and a checked 'distinct' checkbox.

**Figure 100 - "Auto" mode calculated field, all fields selected**



The screenshot shows the SpagoBI query results interface. At the top, there are tabs: Field Selection, Conditions, Ordering, Grouping, View Query, Save Query, Execute Query, and Generate Template. The 'Execute Query' tab is active.

On the left, there are buttons for 'Execution Modality' (Default or Expert), 'Export to format' (PDF, XML, CSV, etc.), and 'Create View'. Below these are buttons for 'Page 1 >>' and a gear icon.

The main area displays a table with three columns: 'Link Detail', 'Store Name', and 'Customer Identifier'. The table contains 14 rows of data, each with a blue link labeled 'CUSTOMER\_xxx' followed by the store name and customer identifier. The data is as follows:

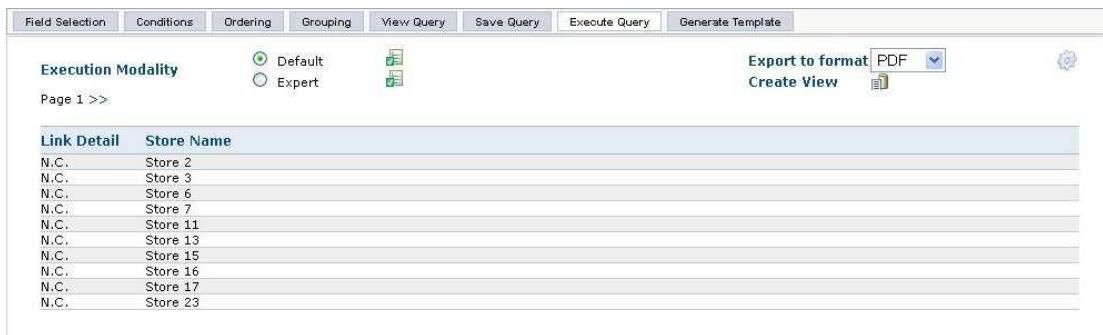
Link Detail	Store Name	Customer Identifier
CUSTOMER_22	Store 2	22
DETAILS		
CUSTOMER_86	Store 3	86
DETAILS		
CUSTOMER_18	Store 6	18
DETAILS		
CUSTOMER_17	Store 7	17
DETAILS		
CUSTOMER_10	Store 11	10
DETAILS		
CUSTOMER_39	Store 13	39
DETAILS		
CUSTOMER_3	Store 15	3
DETAILS		
CUSTOMER_134	Store 16	134
DETAILS		
CUSTOMER_61	Store 17	61
DETAILS		
CUSTOMER_73	Store 23	73
DETAILS		

**Figure 101 - "Auto" mode calculated field**

if you don't the calculated field cannot be calculated and a 'N.C.' will be displayed on the relevant column of the query result.



The screenshot shows the SpagoBI Field Selection dialog. At the top, there are tabs: Field Selection, Conditions, Ordering, Grouping, View Query, Save Query, Execute Query, and Generate Template. Below the tabs, there is a tree view of available fields under the schema 'foodmart2'. A specific node, 'Annual Sales Fact.Store Name', is highlighted with a yellow background and has a checkmark next to it. To its right, there is a small interface for selecting options like 'distinct'. Other nodes visible in the tree include Customer, Promotion, Product Class, Store, and various fact tables like Annual Sales Fact, Product, and Store.

**Figure 102 - "Auto" mode calculated field, not all needed fields are selected**


Link Detail	Store Name
N.C.	Store 2
N.C.	Store 3
N.C.	Store 6
N.C.	Store 7
N.C.	Store 11
N.C.	Store 13
N.C.	Store 15
N.C.	Store 16
N.C.	Store 17
N.C.	Store 23

**Figure 103 - "Auto" mode calculated field, missing needed fields in the selection clause**

The 'inExport' attribute specifies if the calculated fields must be present also in the exported document.

Look at the following groovy script file example:

```
def detailLink(pValue) {
    return "<a href=\"http://.../CUST_DETAIL_ACTION?id=${pValue}\">DETAIL ${pValue}</a>"
}

def detailInExport(pValue) {
    return "PDF [ ${pValue} ]"
}

if ("${qbe_mode}" == "exec"){
    return detailLink("${input1}");
}else{
    return detailInExport("${input1}");
}
```

you can see how to use variables. You can see also that there is another implicit variable named 'qbe\_mode', that specifies if the query is being normally executing or if it is being exported.

## 7.6.10. DOCUMENT TEMPLATE AND PROFILED DATA

See "How to deploy a new Datamart into SpagoBIQbeEngine" in "How to" document in order to understand how to deploy the Hibernate jar file into the engine.

SpagoBI document template is an xml file that contains only a reference to the datamart deployed into the engine and the connection parameters, as the following example:

```
<QBE>
    <DATASOURCE name="hsq1Foodmart"
        dialect="org.hibernate.dialect.HSQLDialect" />
    <DATAMART name="foodmart"/>
</QBE>
```

Datasource name is the name of the connection defined in SpagoBIQbeEngine/WEB-INF/conf/data-access.xml while dialect attribute specifies the hibernate dialect to be used.

Since SpagoBI 1.9.3, SpagoBIQbeEngine permits to control user data visibility on datamart through the profiled data access modality.

It is configurable on document template, no other change in configuration files are needed.

The following is an example of template suitable for SpagoBIQbeEngine 1.9.3 using profiled data access mechanism:

```
<QBE>
    <DATASOURCE name="hsq1Foodmart"
        dialect="org.hibernate.dialect.HSQLDialect" />
    <DATAMART name="foodmart"/>
    <MODALITY name='ProductManagerAnalysis'>
        <TABLE name='it.foodmart.ProductClass' accessible='true'>
            <FILTERS>
                <FILTER>
                    F{productFamily} = 'P{productFamily}'
                </FILTER>
            </FILTERS>
        </TABLE>
        <TABLE name='it.foodmart.Customer' accessible='true'>
            <FIELDS>
                <FIELD name='fname' accessible='false' />
                <FIELD name='lname' accessible='false' />
                <FIELD name='fullname' accessible='false' />
                <FIELD name='address1' accessible='false' />
                <FIELD name='address2' accessible='false' />
                <FIELD name='address3' accessible='false' />
                <FIELD name='address4' accessible='false' />
                <FIELD name='phone1' accessible='false' />
                <FIELD name='phone2' accessible='false' />
            </FIELDS>
        </TABLE>
        <TABLE name='it.foodmart.Store' accessible='false'>
        </TABLE>
    </MODALITY>
    <FUNCTIONALITIES>
```

```
<FUNCTIONALITY name="expertQuery">
  <PARAMETER name="editable" value="TRUE" />
</FUNCTIONALITY>
</FUNCTIONALITIES>
</QBE>
```

As you can see xml envelopes DATASOURCE and DATAMART are unchanged.

Xml envelope MODALITY is used to define the profiled view you want to have on the datamart. You can specify for each table if it is accessible or not (<TABLE name='it.foodmart.Customer' accessible='true'>).

You can also specify for each field if it is accessible or not (<FIELD name='fname' accessible='false' />).

Note that if the accessible property of one table or one field is not specified it is assumed accessible by default.

In this way you have only to specify which tables and fields are not visible.

You can also specify one ore more filters on tables data.

For example the following filter defined on 'it.foodmart.ProductClass' table:

```
<FILTERS>
  <FILTER>
    F{productFamily} = 'P{productFamily}'
  </FILTER>
</FILTERS>
```

filters out all products in which 'productFamily' field is not equal to P{productFamily} where P{productFamily} is a runtime parameter defined on the SpagoBI document detail page.

Filters on a table are applied on each query that use data contained in that table: filter conditions are put on the query where clause but the user will not be able to see (and modify) them.

Note that, since P{productFamily} depends on SpagoBI document parameter, its actual value is controlled by SpagoBI behavioural model, in other words it depends on the user execution role and/or his profile attributes.

Last, but not least, the xml envelope FUNCTIONALITIES allows to disable the expert query editor. This is important if you want a strong data-access control: since (at the moment) filters conditions are visible (and editable) in expert modality, if the user were free to manually edit the query he could modify or delete them.

## **7.7. 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 an 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 execs the query in order to extract the data associated to svg geographic elements.
- reads from the template the svg map logical name and gets the map from its catalogue
- 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:**

<b>Ids</b>	<b>Population</b>
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>
```

### **7.7.1. 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 postgis database usign the postgis tools for svg convertn
- Extracting data from an opengis database and converting it into svg with a custom program.

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

### **7.7.2. WRITE A MAP TEMPLATE FILE**

The template file is divided into some main parts

```
<?xml version="1.0" encoding="UTF-8"?>
<MAP name="name of the map">
    <DATAMART_PROVIDER />
    <CONFIGURATION />
</MAP>
```

The tag <MAP/> is the root of the template and contains the "name" attribute. The value of this attribute must be the logical name of a map contained into the catalogue of the SpagoBI Geo Engine. To learn how to know which maps are contained into the Engine catalogue or to add a new map you can look at the SpagoBI 'How to' document (question 'How to deploy a new Map into SpagoBIGeoEngine').

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

```
<DATAMART_PROVIDER connection_name=" name of a connection defined "
    query="select id, value from ... where ..."
    column_id="id"
    column_value="value"/>
```

You need to configure four attributes:

- connection\_name: the name of one of the database connection pools configured into the data\_access.xml file (SPAGOBIGEO\_HOME/WEB-INF/conf/data\_access.xml). The connection pool is used to get the connection towards the database.
- Query: the query which recovers the data associated to the svg element ids
- column\_id: the name of the column returned by the query which contains the ids related to the svg element ids.
- column\_value: 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 style 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="legent 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' an 'title' which allow you to set the x/y position, the dimensions and title of the legend.

### **7.7.3. 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.

### **7.7.4. PARAMETERS CONFIGURATION**

As the other kind of business documents also the Map can be parametric.

To add a parameter to a Map document you need to change the query defined into the template and to configure the parameter into the map SpagoBI document detail page.

Supposing that you want to add a parameter 'product' to your map document, the first thing to do is to change the query contained into the template file (DATAMART\_PROVIDER.query attribute) adding the expression \${productParameter} everywhere you want.

Then you need to add the parameter to the Map document (inside the document detail page) remembering that the 'Url name' must be 'productParameter'.

Notice that:

- The name 'productParameter' is only an example, you can assign it what you want, the important thing is that the name must be equal to the value of the url name of the document parameter.
- The expression in the query will be replaced by the value of the parameter chosen by the user.

Typically you place this expression in a where condition, like '... where product = \${productParameter} ...', but you are free to use it in another way.

## 7.8. OFFICE DOCUMENT

An "Office document" is simply a document where the template is a file that can be of various known formats: .doc, .xls, .pdf, .txt, .html ... This file can include notes, comments, calculations, images and so on.

An internal engine (Office document Internal Engine) will try to show the file inside the browser page (see the example below with a .pdf file): in case the browser has not the plugin for the correct visualization, the file will be downloaded.

These documents are considered static and parameters are unuseful for them.

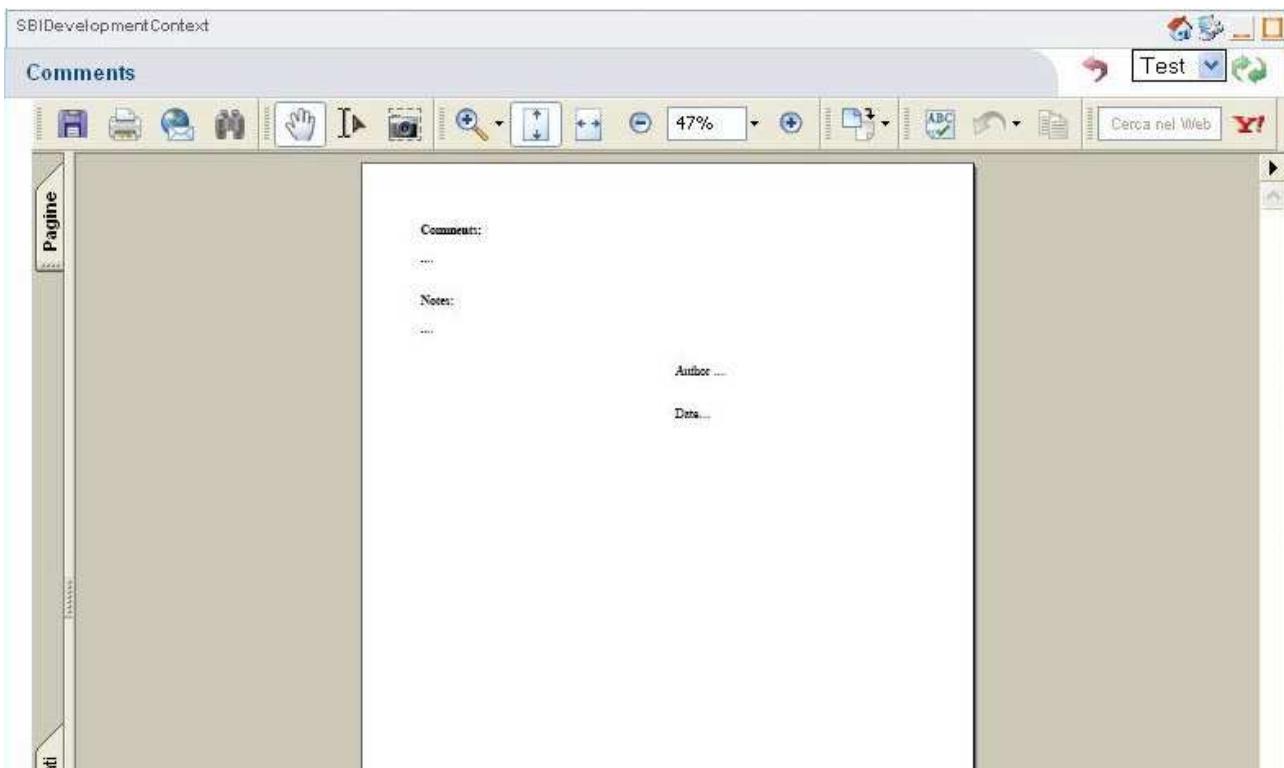


Figure 104 - Office document example

## 7.9. BOOKLETS

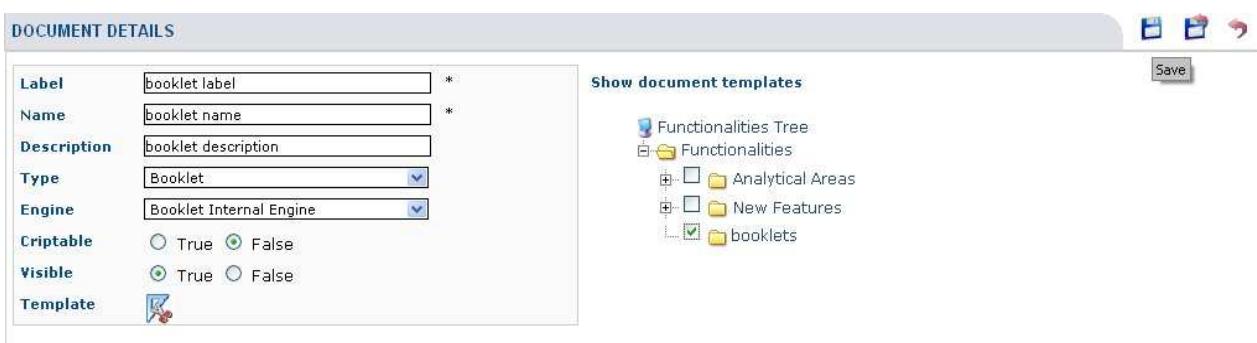
A booklet is collection of report snapshots and relative text notes inserted by SpagoBI users. The final product of the booklet execution is a presentation document (a power point file) which contains, into different slides, the report snapshots and notes. A booklet is a kind of SpagoBI document, but it is a little bit different from the others for two reasons:

- The template of the document is not a loaded file but it is created using a graphical interface accessible from the detail page of the booklet document.
- The execution of the booklet is not a unique request to an engine but it is a process, led by a workflow engine, which involves some spagobi users and automatic mechanisms, in some different activities.

A booklet is a way to share opinion on business documents and produce different presentations of your business areas.

### 7.9.1. CREATION OF A NEW BOOKLET

Since a booklet is a particular kind of SpagoBI document you can create it in the same way as for the other document types (from the list of business documents press the 'insert' button). Once entered the empty detail page of the document choose, select the 'Booklet' option as document type. The system will automatically change the engine to a default internal engine and the file upload form will be replaced by a 'Generate Template' button (if you press the button before save the document it won't work). Then fill the other fields and save the document.



DOCUMENT DETAILS	
Label	booklet label *
Name	booklet name *
Description	booklet description
Type	Booklet
Engine	Booklet Internal Engine
Criptable	<input type="radio"/> True <input checked="" type="radio"/> False
Visible	<input checked="" type="radio"/> True <input type="radio"/> False
Template	<a href="#">Template</a>

Show document templates

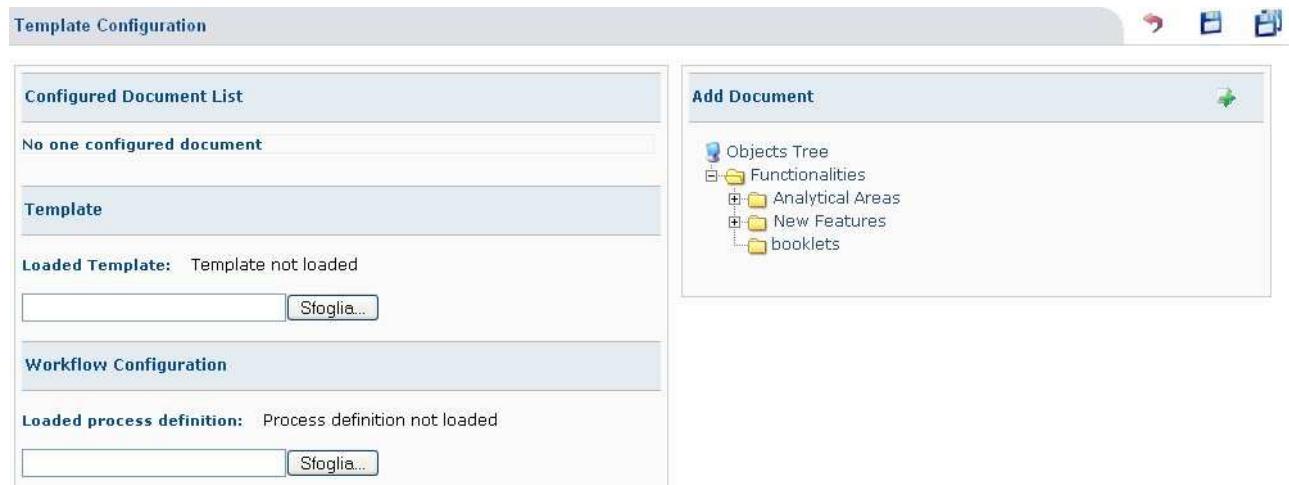
Functionalities Tree

- Functionalities
- Analytical Areas
- New Features
- booklets

**Figure 105 Booklet Document detail page**

### 7.9.2. CONFIGURATION OF THE BOOKLET TEMPLATE

The booklet document doesn't allow you to load a file as execution template but give you the possibility to configure a template using a graphical interface (GUI). Using this GUI you can change the current version of the template or you can save a particular configuration as a new version. In the document detail page, as for the other document types, you can see the version template list and set the current one. To access the graphical interface press the button near the 'template' label in the booklet document detail page.



**Figure 106 Booklet Template configuration page**

The template GUI allows you to:

- Add to the booklet a SpagoBI document. The snapshot of the added document will be inserted in the final ppt presentation. Each document added will be added into the Configured Document List in the left side of the gui. This release allow you to add only jasper reports (document associated to the SpagoBI JasperReportEngine).
- Load the power point template which will be the base for the final document. The template is a power point file (you can create it using the free 'open office' suite) which contains some place holders that will be replaced by the SpagoBI document snapshots. Once loaded the template file, its name will appear near the 'Loaded Template' label together with a button that allow you to download it.
- Add the workflow process definition file. For this version the process definition file must be a jbpm file. Since the booklet module requires a fixed structure for the process definition, in order to change your workflow process, don't create a new definition file but recover the one contained into the BookletModule binary distribution, and then change the allowed parts as described in the section below. Once loaded the workflow process definition it's name will appear near the 'Loaded process definition' label together with a button that allow you to download it.

To add a new document to the booklet you have to choose it from the 'Add Document' tree in the right side of the gui (to choose the document simply check its checkbox), and the press the 'Add Document' button in the right upper corner of the tree box. This operation will get you into the configuration page for the added document.



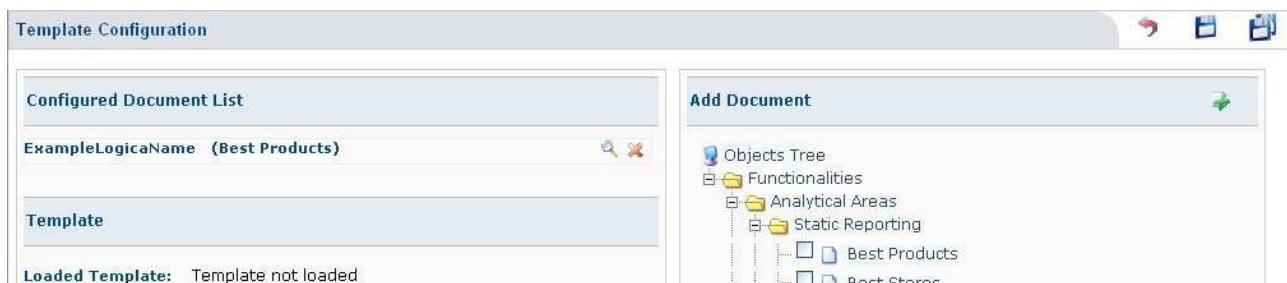
The screenshot shows the 'Template Configuration' interface. It has two main sections: 'Document Information' and 'Document Parameters'. In 'Document Information', there is a table with columns 'Name', 'Description', and 'Label'. The 'Label' column contains 'Best Products', 'Best Products in Month', and 'TOP-PROD'. Below this is a 'Logical Name' input field. In 'Document Parameters', there are three input fields: 'Month:' (empty), 'Output:' (filled with 'JPGBASE64'), and 'Number tops:' (empty).

**Figure 107 Configuration page for a document added to the booklet**

The configuration page contains:

- some summary information about the document chosen (label, name, description)
- An input text which must be filled with a logical name for the document. The logical name is mandatory and must be unique within the booklet. It will be used to identify the place holder, into the presentation template, which has to be replaced by the snapshot of the document
- A parameters section where are listed all the parameters of the document together with an input text, useful to specify an execution value. The only parameter value that cannot be changed is the one relative to the output format (it is filled with default value). Remember that the values entered will be used to produce the snapshot of the document which will be inserted into the final presentation.

Once finished the configuration you can save it using the 'save' button in the right upper corner. Then you will return into the booklet template page and the document configured will be shown into the 'Configured document' list (from the list it's possible to erase or modify each document added).



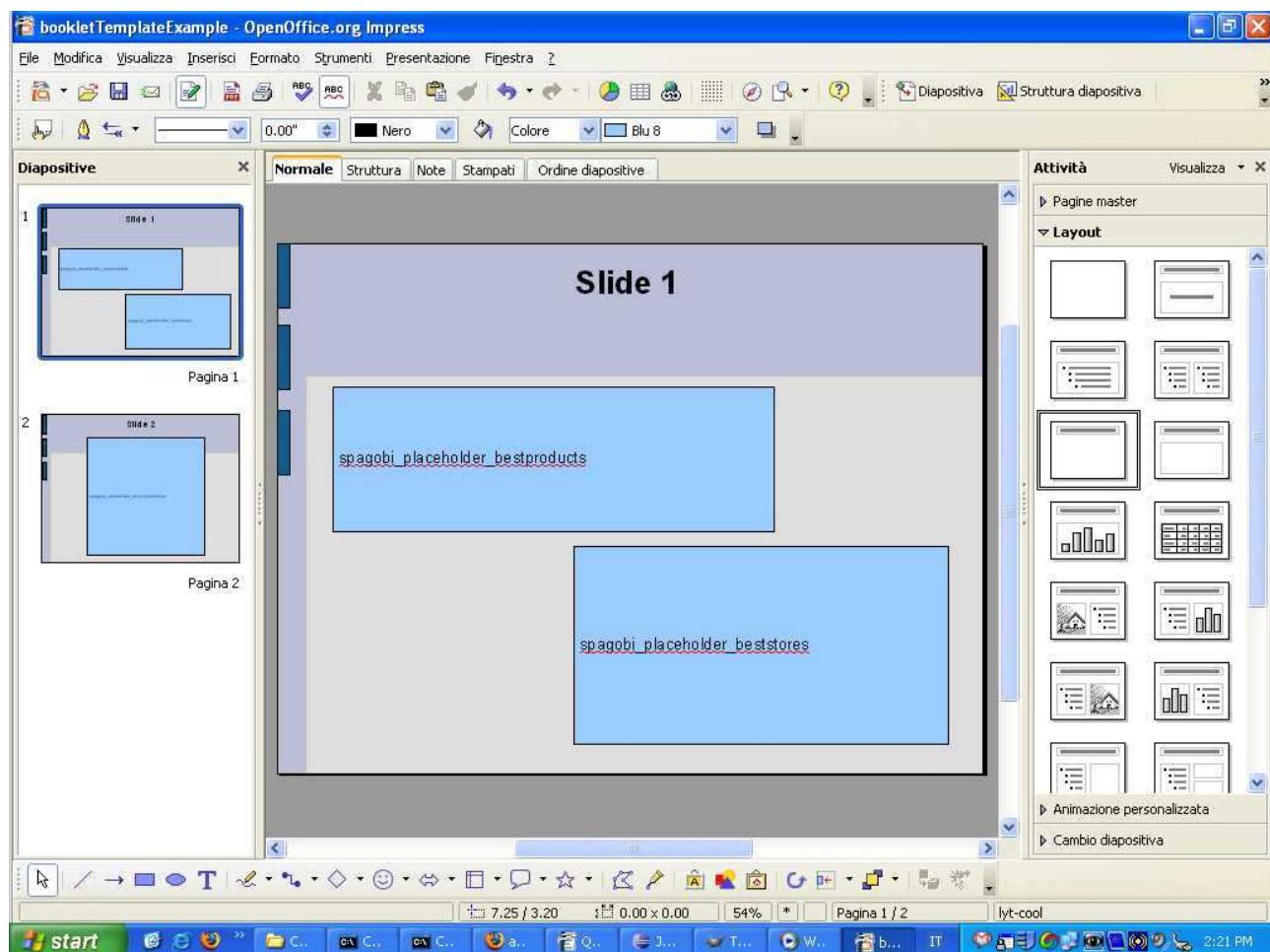
The screenshot shows the 'Booklet Configured document list' interface. On the left is a 'Configured Document List' panel containing 'ExampleLogicaName (Best Products)' with edit and delete icons. Below it is a 'Template' panel with the message 'Loaded Template: Template not loaded'. On the right is an 'Add Document' panel with a tree view of objects: 'Objects Tree', 'Functionnalities', 'Analytical Areas', 'Static Reporting', 'Best Products', and 'Best Stores'.

**Figure 108 Booklet Configured document list**

### 7.9.3. CREATE A PRESENTATION TEMPLATE

A presentation template is a power point document (remember that you can create it using the free office solution 'open office') which will be the base for the final booklet presentation. You can build your power point template as you like, choosing the number of slides, layout, colors, graphical elements ..., but you have to remember two things:

- When you have decided the position of one document snapshot you have to design in this position a rectangle containing a text string like this: `spagobi_placeholder_logicalNameOfTheDocument`. The final part of the string should be replaced with the logical name of the configured document (look at the previous section). The snapshot of the document will fill exactly the area of the rectangle so remember to choose the right dimensions.
- The comments inserted by the SpagoBI users will be inserted into the notes part of each ppt slide. Configuring the workflow process you have to limit the notes insertion for a particular slide to a SpagoBI group. Only the users belonging to the group assigned to the slide can write notes about the content. This behaviour means that if, in the same slide, you put more than one document snapshots, you have to be sure that each document is visible to the role who view the content of the slide



**Figure 109 Presentation template**

#### 7.9.4. CHANGE BOOKLET WORKFLOW PROCESS

When you execute a booklet document a new workflow process is started. The process has automatic tasks and others performed by the users. The automatic tasks, which analyze the configuration and produce the metadata, cannot be changed. The user task definition can be changed, added or removed, based on some rules listed below. For the current release the workflow process definition must be written using the jbpm syntax. You can find an example

workflow definition inside the booklet module binary distribution. In order to develop your booklet, download the file and change it.

The only points where you can change the workflow definition are:

- the swimlane definition: a swimlane is a role of the workflow. Each task in the process definition is assigned to a swimlane. You can add or remove swimlane definitions but each one of them must have the xml structure reported below (if you add a swimlane place it below the others). You can change only the name attribute giving it as value the name of a SpagoBI role. Swimlane are the bridge between workflow roles and SpagoBI roles.

```
<swimlane name="the name of a spagobi role">
    <assignment class="it.eng.spagobi.booklets.assignmenthandlers.SpagoBISwimlaneAssignment">
        </assignment>
</swimlane>
```

- The 'Add notes' tasks between the fork node and the join node. The xml definition of the task is reported below. The number of this task kind should be equal to the number of presentation template slides containing place holders for SpagoBI documents. In case you have to add some 'add Notes' task definitions, copy one of the existing one and paste it below the last one. Then you have to add the reference to the task into the foregoing fork node. For each 'add note' you can change the name, the swimlane assignment and the value of the spagobi\_booklet\_pageindex variable (this variable is the index of the presentation document slide where the notes will be added). Remember that when you change the swimlane name, you are giving the permission to the SpagoBI users who belong to the homonym SpagoBI group, to see the document of the slides and to insert new notes.

```
<task-node name="give it a name">
    <task name="add_note_task1" swimlane="name of a previous defined swimlane">
        <event type="task-create">
            <script>
                <expression>
                    taskInstance.setVariableLocally("spago_handler", "AddNoteHandler");
                    taskInstance.setVariableLocally("spagobi_booklet_pageindex", " ** ");
                </expression>
            </script>
        </event>
    </task>
    <transition name="" to="join1"></transition>
</task-node>

** index of the ppt document slide where the notes will be pasted
```

Inside the workflow definition file you find two different activities performed by the user. The first one is the 'Insert notes' activity which is described with the following xml

- The final "Validation Document" task. The xml definition of the task is reported below. This task is the validation of the final presentation document produced. You can only change the assignment to the swimlane.

```
<task-node name="ValidateFinalDocument">
    <task name="validate_document_task" swimlane="name of a previous defined swimlane">
        <event type="task-create">
            <script>
                <expression>
                    taskInstance.setVariableLocally("spago_handler", "ValidateFinalDocumentHandler");
                </expression>
            </script>
        </event>
    </task>
    <transition name="" to="end1"></transition>
</task-node>
```

### **7.9.5. EXECUTION OF A BOOKLET**

Once configured, a booklet is ready to be executed. As for the other SpagoBI document types you can run it from the Objects tree/list. As a first step of the execution the system shows to the user the list of final presentation previously generated. The user can download or delete the previous version from the list or he can start a new presentation version generation.



Presentation name	Creation date	Version name	Approved
MyBooklet	Mon Sep 11 19:03:38 CEST 2006	1.2	false
MyBooklet	Tue Sep 12 10:44:30 CEST 2006	1.3	false
MyBooklet	Thu Sep 28 18:27:36 CEST 2006	1.15	true

**Figure 110 Booklet execution (presentation version list)**

Once started a new presentation generation the system initializes the workflow engine, starts the process and prepares the initial metadata. If something goes wrong you will see an error message and you need to contact your system administrator to discover the cause. If the startup phase ends without error you should see a message like this:



Collaboration Process Started correctly

**Figure 111 Booklet workflow process started correctly**

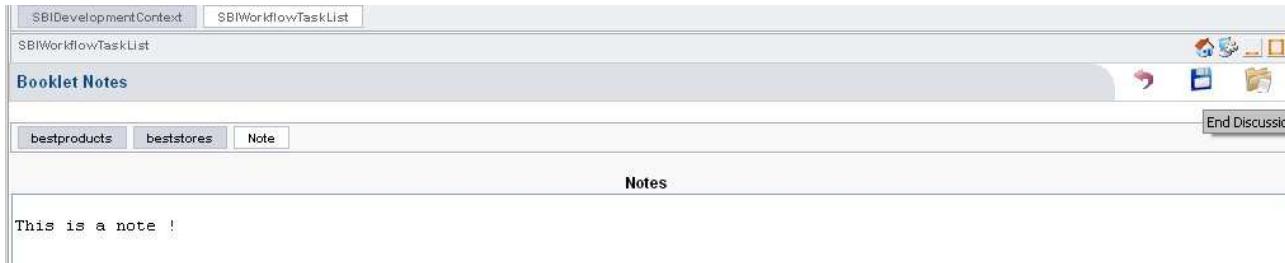
One started, the workflow process executes all the automatic tasks and then assigns to the SpagoBI users the 'Add notes' activities (based on the assignment of the workflow process definition). Each activity is assigned to all the SpagoBI users belonging to the role associated to the task definition (swimlane name). Until all the assigned 'add note' activities aren't performed the workflow process doesn't continue. In order to know when they have to execute a workflow task the SpagoBI users need to use the 'Worklist' Portlet . This portlet every time it is refreshed checks if new workflow activities has been assigned to the current user and present them into a list. The user can accept the activity pressing the button on the left of each row. The first user that accept the activity will complete it, all the others will be no more able to accept it.



Activity name	Activity description	Activity priority	Process name
add_note_task1		3	BookletWorkflowProcess

**Figure 112 Workflow activities list of the users**

Once a user has accepted an 'add note' activity he enters into the add notes editor. The add note editor contains all the snapshot of the document, assigned to the slide associated to the activity, and a text area for the notes insertion (You can switch from snapshots to text area using the tab in the upper part of the editor). The user can insert different notes and save them. At the end of the note insertion, to complete the activity, the user must press the 'End Discussion' button in the right upper corner.



**Figure 113 Note Editor**

Once all the 'add note' activities have been performed the execution continue producing the final presentation document. The presentation produces must be validated. The validation is the last user activity. The users, who can execute it, will see into the task list a new activity and accepting it they will enter into the validation form.



**Figure 114 Approve presentation form**

Using the validation form the users can download the presentation produced and analyze it. To complete the activity the users must press the save button approving or not the presentation (by choosing the right 'Approved' option). Once saved, the presentation is added to the booklet presentation list (shown when a user execs a booklet), so, if the user execs again the booklet, into the presentation list he will see the new version just created. All the presentations, approved or not, will be saved into the booklet presentation list, but only the approved ones can be deployed, into the SpagoBI BIOObject tree, as a document visible to the SpagoBI users. In order to deploy the presentation into the SpagoBI tree the user must press the deploy button, visible only for approved presentations. Once pressed the deploy button the user enter the deploy form where he can choose the label, name, description and tree branch of the presentation document.



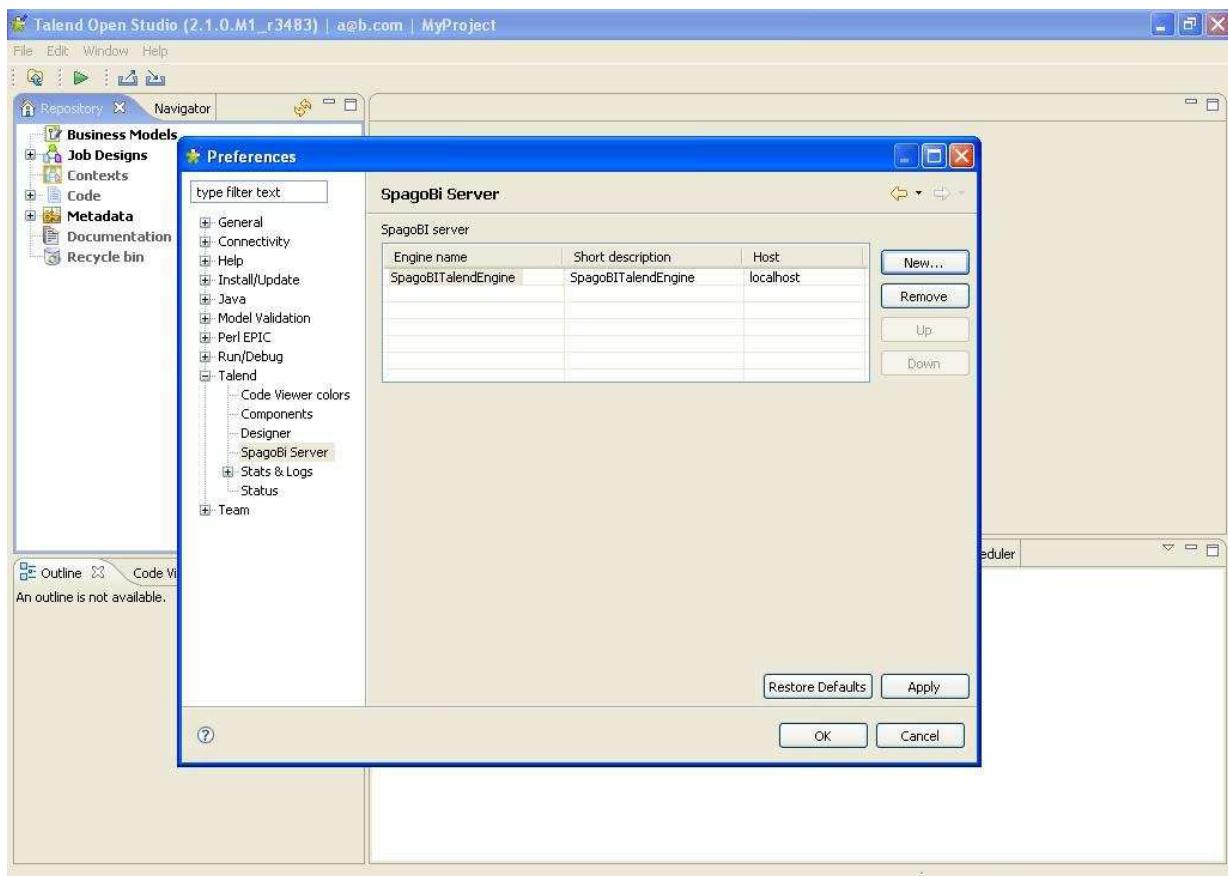
**Figure 115 Deploy booklet presentation into SpagoBI objects tree**

## 7.10. ETL PROCESS

### 7.10.1. CONFIGURE TALEND OPEN STUDIO (TOS) 2.1.0

Open Talend Open Studio (you can download it from <http://www.talend.com/download.php>) and create a new Java project "MyProject". Then

1. Select "Preference" from menu "Windows"
2. Explode 'Talend' node from the preference windows (left part)
3. Select 'SpagoBI Servers' subnode
4. Click on button 'New...' to add a new connection to a SpagoBITalendEngine instance
5. Set properly all the connection parameters, in particular:
  - Host: the host where SpagoBITalendEngine is installed
  - Port: the port the server with SpagoBITalendEngine installed is listening to
  - Login: "spagobi"
  - Password: "spagobi"
  - Application Context: SpagoBITalendEngine web application context name



**Figure 116 - SpagoBITalendEngine configuration on Talend Open Studio**

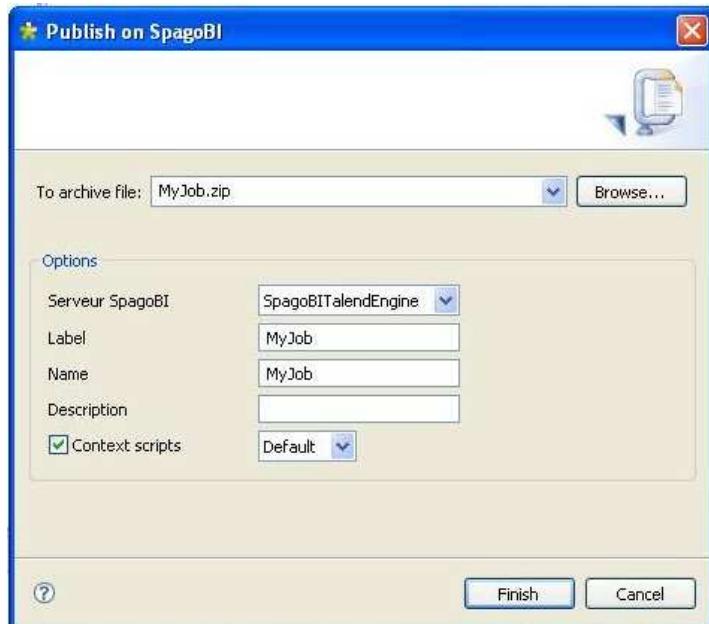
### **7.10.2. DEPLOY A JOB FROM TOS TO SPAGOBI TALEND ENGINE**

Using Talend Open Studio create a simple job "MyJob" that displays a message "Hello World!!": to do this, simply drag & drop the tMsgBox component (inside "Misc" folder) from the palette into the job designer and save the job.

Then:

1. Right click on the job and select the voice 'Publish on SpagoBI' from the contextual menu
2. Select from the combobox the logical name of the connection to SpagoBITalendEngine instance you have defined in the Preference window

Obviously the server with SpagoBITalendEngine must be running.



**Figure 117 - Publish on SpagoBI**

Control if there are the following files in  
SpagoBITalendEngine\RuntimeRepository\java\MyProject\MyJob:

- . myjob.jar file
- . myjob\_run.bat file
- . myjob\_run.sh file

If there is also a folder myproject\myjob\contexts, this folder must inserted inside myjob.jar file and then deleted (see job examples jar files).

### **7.10.3. CREATE SPAGOBI DOCUMENT**

Create a new analytical 'ETL process' type document.

Use as template file an xml file like this one ....

```
<etl>
<job project=<PROJECT_NAME>
      jobName=<JOB_NAME>
      context=<CONTEXT_NAME>
      language=<EXPORTATION_LANGUAGE> />
</etl>
```

where:

<PROJECT\_NAME> is the Talend project name: "MyProject"

<JOB\_NAME> is the Talend job name: "MyJob"

<CONTEXT\_NAME> is the job context to be considered for job execution: "Default"

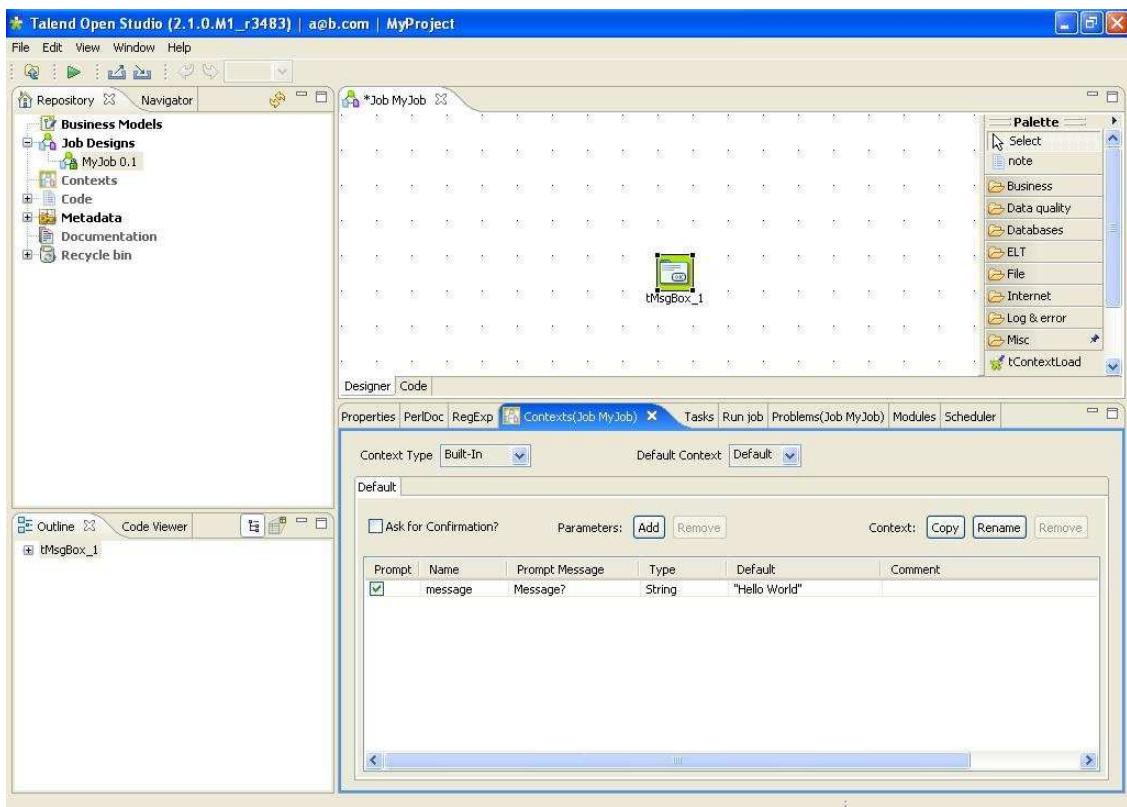
<EXPORTATION\_LANGUAGE> the job language: "java"

Save it and execute it: a "Hello Word" message box will appear (probably in background).

#### **7.10.4. PARAMETERS CONFIGURATION**

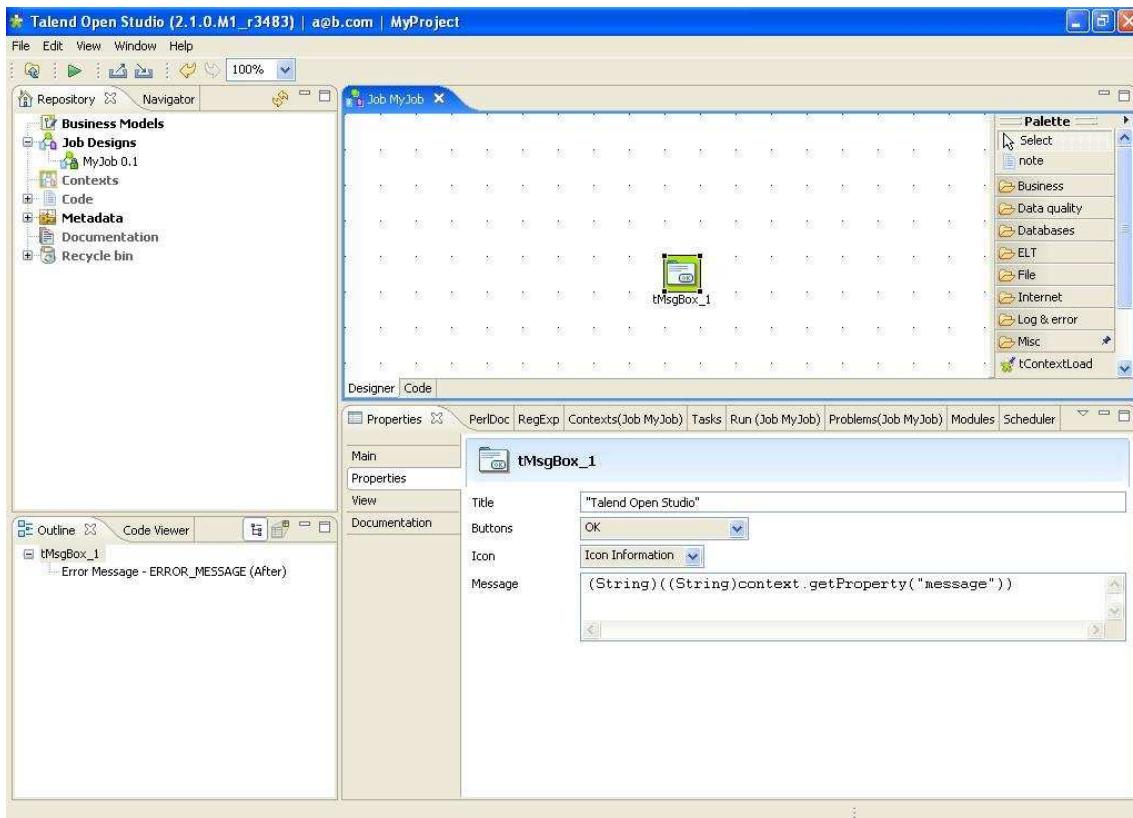
You can define parameters on ETL document: their url names must be equal to a job context parameter.

Considering the example above, suppose we want that the message string in the box to be a parameter: on Talend Open Studio define a context parameter with name "message"



**Figure 118 - Context parameter definition in Talend Open Studio**

Then you have to make a reference to this parameter on message property of the tMsgBox component:



**Figure 119 - Context parameter reference**

Then deploy on SpagoBI (overwrite the previous job and remember to control the jar file structure).

On SpagoBI platform:

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:** FREE\_MESSAGE
  - **Name:** Free message
  - **Description:**
  - **Type:** String
  - **Functional:** no (don't check it)

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
- **Manul Input:** select this modality

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

3. Click on the SAVE AND RETURN icon to go back to the *Parameter List*.
4. Enter the document (previously defined) detail page and define a new parameter:

- **Title:** Message
- **Parameter:** from the lookup list, select the FREE\_MESSAGE parameter
- **URL Name:** message

Note that parameter url name is equal to the job context parameter name.

5. Execute the document: a form will ask you the value of the "Message" parameter, you can edit a free message that will be displayed by the message box.

#### **7.10.5. CONTEXT RUN-TIME SELECTION**

A Job defined with Talend Open Studio could have one or more contexts. You can configure the ETL document in order to retrieve the name of the context to be used at run-time from a parameter. This parameter must have url name "context" and its suitable value are the names of the job contexts. For example, suppose the job has two contexts "Development" and "Production", those are suitable parameter values.

If the context parameter is not defined, the default context is considered.

## 7.11. DOCUMENTS CROSS NAVIGATION

Since SpagoBI 1.9.3, documents can be designed in order to permit you to click on their result thus going to another document execution with dynamic parameters transfer and control: this is 'documents cross navigation'.

As an example of documents cross navigation, suppose that a report shows the best 5 products sales (this is the source document), then you could click on a product and see a map (this is the target document) that shows you the geographical sales distribution for that product; than you could go back to the report and click on another product to see its geographical sales distribution an so on.

Source and target document are SpagoBI documents with their own parameters.

Target document can be in turn a source document, i.e. you can drill through to a third document: in the example above, you could click on a region of the map to see the sales report for that region.

You can drill through more than one document execution and you can come back to the previous ones (and then drill through again...).

Drill through mechanism works thanks to the SpagoBIDrillServlet servlet that is installed on each external engine, therefore a document (that produces a visible output) executed by an external engine (SpagoBIJasperReportEngine, SpagoBIJPivotEngine, SpagoBIQbeEngine, SpagoBIGeoEngine, SpagoBIBirtReportEngine) can be a source document; a document executed by an internal engine, at the moment, cannot be a source document.

There is no restrictions on target document.

What you have to do for a cross navigation is to configure the source document in order to obtain a link in its final result: the link url must points to:

```
http://<server>:<port>/<engine context>/SpagoBIDrillServlet
```

with the following parameters:

```
DOCUMENT_LABEL=<label of the target document>&
DocumentParameterUrlName_1=value_1&
DocumentParameterUrlName_2=value_2&
...

```

where:

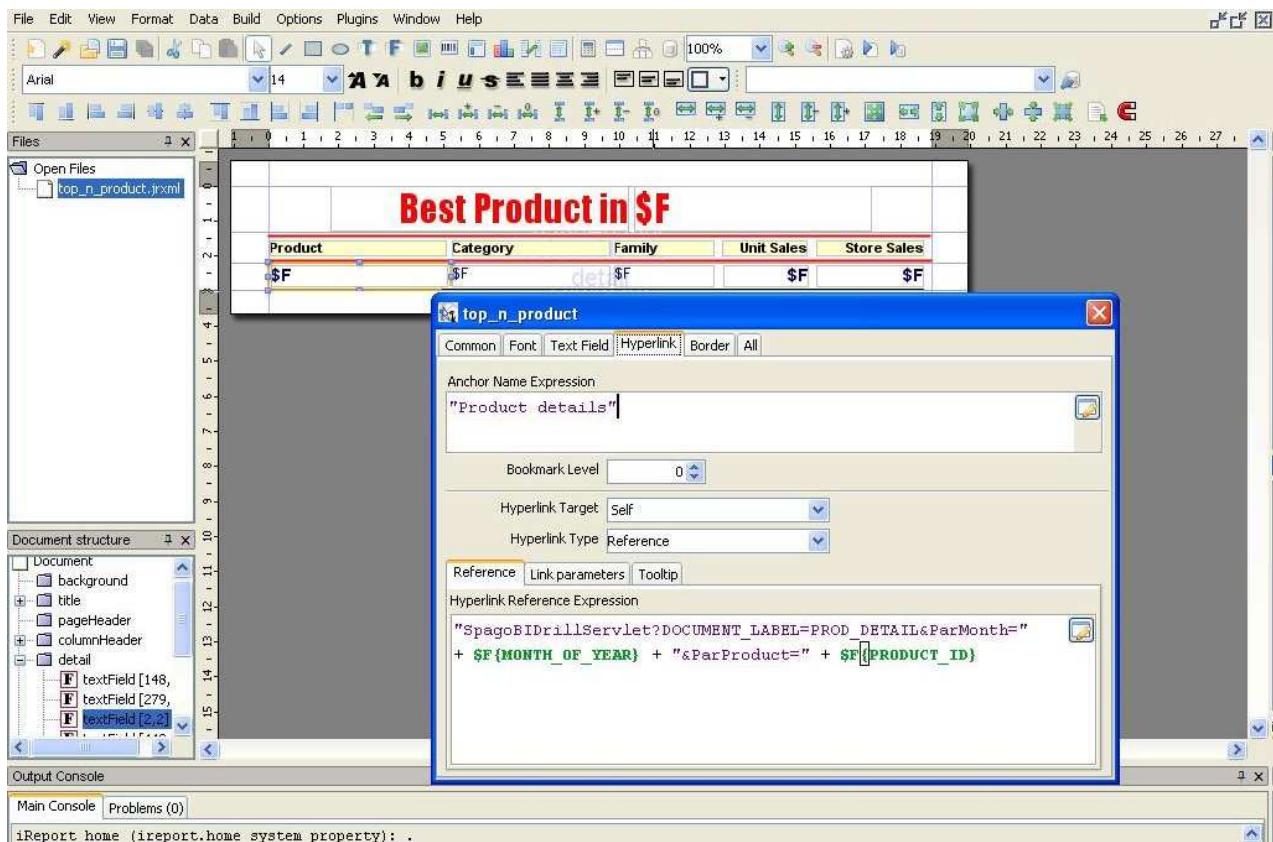
- DOCUMENT\_LABEL is the label of the target document (it is mandatory);
- DocumentParameterUrlName\_x is a target document parameter url name (it is optional);
- value\_x is the value of the parameter (optional).

If the produced link does not contain a valid value for all the target document parameters, a form will ask you to fill in the missing ones.

Let's see some examples for SpagoBI engines. We suggest to read the example for SpagoBIJasperReportEngine as first as it is the most exhaustive.

### 7.11.1. SPAGOBIJASPERREPORTENGINE

Open the report with iReport, suppose you have a text detail field, edit its properties and configure an hyperlink as in the figure below:



**Figure 120 - Hyperlink configuration on iReport**

Note that hyperlink type is set to "Reference" and hyperlink target to "Self", and that the servlet url can be relative, so you don't need to know neither the server name, nor the server port nor the SpagoBIJasperReportEngine application context at design-time.

Note also the link parameters: the target document is "PROD\_DETAIL", and its parameters ("ParMonth" and "ParProduct") values are the local field values ("MONTH\_OF\_YEAR" and "PRODUCT\_ID"): these are dynamic parameters, but you could also put in the link a static parameter, for example "...&param\_output\_format=HTML&...".

When you execute the document, the product name is an hyperlink: in the figure below you can see the link url:

Best products in month

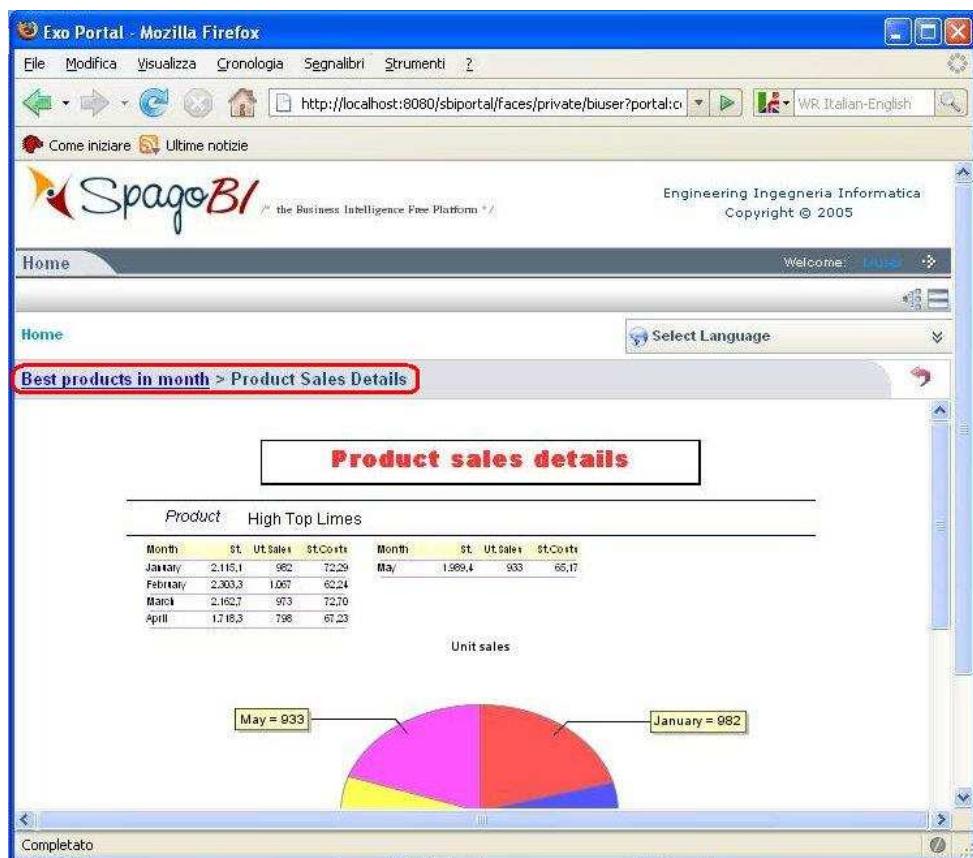
## Best Product in May

Product	Category	Family	Unit Sales	Store Sales
<b>High Top Summer</b>	Vegetables	Food	1.596	3.483,84
<b>High Top Limes</b>	Fruit	Food	933	1.989,46
<b>Club Jack Cheese</b>	Dairy	Food	609	1.512,50
<b>Best Choice Dried</b>	Snack Foods	Food	574	1.270,85
<b>Blue Label Chicken</b>	Canned Soup	Food	638	1.266,58

[http://localhost:8080/SpagoBIJasperReportEngine/SpagoBIDrillServlet?DOCUMENT\\_LABEL=PROD\\_DETAIL&ParMonth=5&ParProduct=193](http://localhost:8080/SpagoBIJasperReportEngine/SpagoBIDrillServlet?DOCUMENT_LABEL=PROD_DETAIL&ParMonth=5&ParProduct=193)

**Figure 121 – Jasper Report hyperlink for cross navigation**

When you click on the link of "High Top Limes", the target document ("PROD\_DETAIL" in this example) is executed and shows "High Top Limes" product sales details:



The screenshot shows a Mozilla Firefox browser window. The address bar contains the URL: [http://localhost:8080/sbiportal/faces/private/biuser?portal:contentId=1&contentLabel=PROD\\_DETAIL&ParMonth=5&ParProduct=193](http://localhost:8080/sbiportal/faces/private/biuser?portal:contentId=1&contentLabel=PROD_DETAIL&ParMonth=5&ParProduct=193). The page title is "Exo Portal - Mozilla Firefox". The main content area displays a "Product sales details" report for "High Top Limes". The report includes a table of monthly sales data and a pie chart showing the distribution of sales units for May, January, and February.

Month	St.	Ut.Sales	St.Costs	Month	St.	Ut.Sales	St.Costs
January	2.115,1	982	72,29	May	1.989,4	933	65,17
February	2.303,3	1.067	62,24				
March	2.162,7	973	72,70				
April	1.718,3	798	67,23				

Unit sales

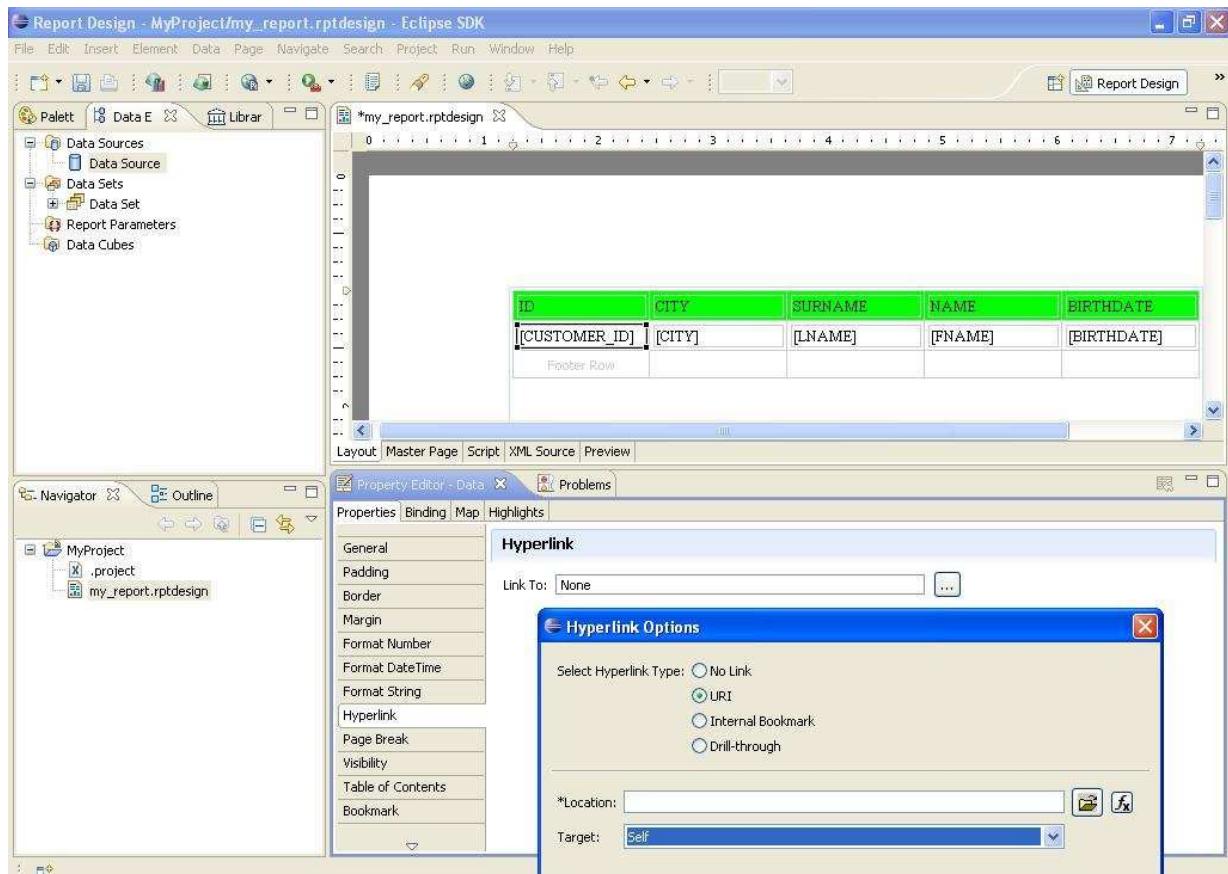
May = 933      January = 982

**Figure 122 – Cross navigation example**

You can click on "Best products in month" to return to the first document execution, and then click on another product to see its sales details.

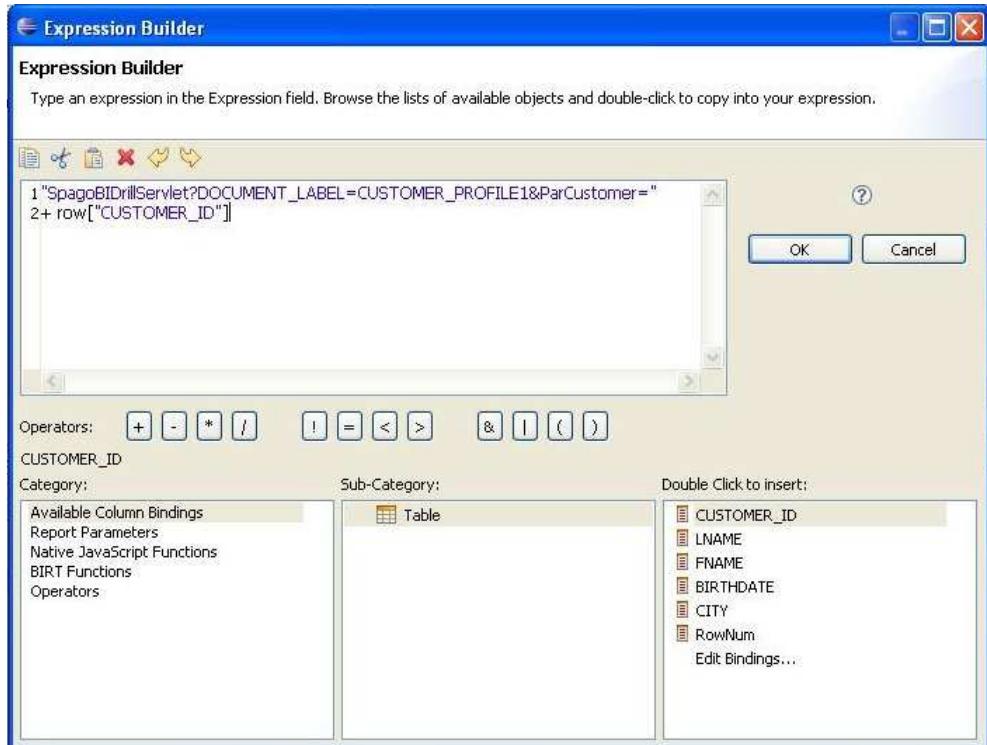
### 7.11.2. SPAGOBI BIRT REPORT ENGINE

Open the report with Birt 2.2.0, suppose you have a text detail field, click on it, you will see its properties. Click on "Hyperlink" section, then on "Open hyperlink options":



**Figure 123 - Hyperlink configuration on Birt report designer**

Select "URI" as hyperlink type, target "Self" and build the location formula as follows:


**Figure 124 - Hyperlink expression**

You obtain an hyperlink as below:

ID	CITY	SURNAME	NAME	BIRTHDATE
3	Issaquah	Derry	Jeanne	21/giu/10
5	Novato	Gutierrez	Maya	10/mag/51
6	Lynnwood	Damstra	Robert	08/ott/42
10	Lake Oswego	Stanz	Darren	26/ago/49
11	La Mesa	Murraim	Jonathan	20/giu/67
12	Chula Vista	Creek	Jewel	18/ott/71
14	Lincoln Acres	Rutledge	Bryan	09/lug/12
17	Arcadia	Marshall	Brenda	20/mar/28
18	Altadena	Wolter	Daniel	21/set/14
19	Oakland	Collins	Dianne	20/lug/53
20	Spring Valley	Baker	Beverly	16/apr/74
21	Renton	Castillo	Pedro	04/nov/18
22	Bellingham	Borges	Laurie	07/ott/47
23	La Jolla	Wyro	Shauna	24/ott/16

**Figure 125 - Birt report hyperlink for cross navigation**

### 7.11.3. SPAGOBIQBEENGINE

It is possible to build hyperlink using calculated fields (see "Query By Example" chapter). If you are familiar with groovy scripts and Qbe formulas definition, you will easily understand how the formula

```
<FORMULA id="cField1"
  onEntity="it.foodmart.Product"
  mode="auto"
  mappings="productId->input1,productName->input2"
  resultingFieldName="Link Detail"
  script="productlnk.groovy"
  inExport="true" />
```

and the groovy script

```
return "<a href=../SpagoBIDrillServlet?DOCUMENT_LABEL=PROD_DETAIL&ParMonth=5&ParProduct=${input1}> Detail ${input2} </a>"
```

create the link that you can see below:



Link Detail	Product Identifier	Product Name	Brand Name
Detail Washington Berry Juice	1	Washington Berry Juice	Washington
Detail Washington Mango Drink	2	Washington Mango Drink	Washington
Detail Washington Cream Soda	4	Washington Cream Soda	Washington
Detail Jeffers Oatmeal	12	Jeffers Oatmeal	Jeffers
Detail Blue Label Canned Beets	16	Blue Label Canned Beets	Blue Label
Detail Blue Label Chicken Soup	19	Blue Label Chicken Soup	Blue Label
Detail Blue Label Canned Tuna in Water	30	Blue Label Canned Tuna in Water	Blue Label
Detail Blue Label Large Canned Shrimp	32	Blue Label Large Canned Shrimp	Blue Label
Detail Blue Label Fancy Canned Anchovies	33	Blue Label Fancy Canned Anchovies	Blue Label
Detail Blue Label Fancy Canned Clams	34	Blue Label Fancy Canned Clams	Blue Label
Detail Blue Label Fancy Canned Oysters	35	Blue Label Fancy Canned Oysters	Blue Label
Detail Blue Label Fancy Canned Sardines	36	Blue Label Fancy Canned Sardines	Blue Label
Detail Green Ribbon Canned Mixed Fruit	38	Green Ribbon Canned Mixed Fruit	Green Ribbon
Detail King Rosy Sunglasses	40	King Rosy Sunglasses	King
Detail Queen Eyeglass Screwdriver	41	Queen Eyeglass Screwdriver	Queen
Detail Queen City Map	42	Queen City Map	Queen
Detail Club Low Fat Cottage Cheese	43	Club Low Fat Cottage Cheese	Club
Detail Club Low Fat Sour Cream	44	Club Low Fat Sour Cream	Club
Detail Club Jack Cheese	46	Club Jack Cheese	Club
Detail Club 2% Milk	56	Club 2% Milk	Club

Copyright © 2000-2006 eXo Platform SARL

http://localhost:8080/SpagoBIQbeEngine/SpagoBIDrillServlet?DOCUMENT\_LABEL=PROD\_DETAIL&ParMonth=5&ParProduct=30

**Figure 126 - Link field on query result for cross navigation**

### 7.11.4. SPAGOBIGEOENGINE

Let's take a look on the following template example:

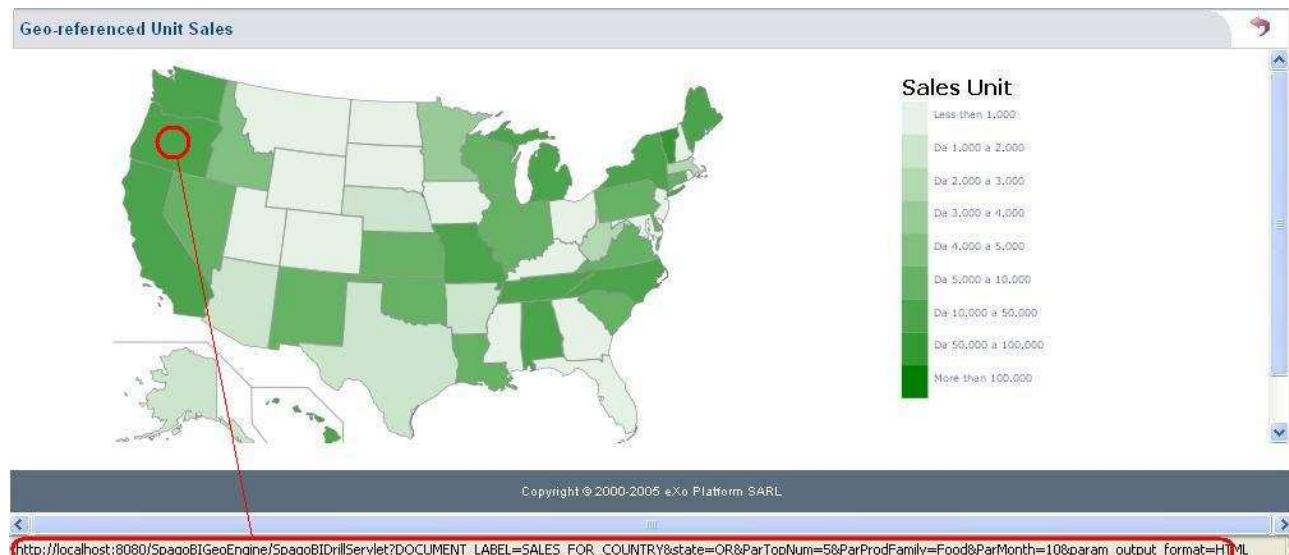
```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<MAP name="usa" >
    <DATAMART_PROVIDER connection_name="spagobigeo"
        query="select id,sum(SALES_UNIT) as sales, PROD_FAMILY from STATE_UNIT
            where PROD_FAMILY LIKE '%${ProdFamily}%' group by id, PROD_FAMILY"
        column_id="id"
        column_value="sales">
        <DRILL document="SALES_FOR_COUNTRY">
            <PARAM type="relative" name="state" value="id" />
            <PARAM type="absolute" name="ParTopNum" value="5" />
            <PARAM type="relative" name="ParProdFamily" value="PROD_FAMILY" />
            <PARAM type="absolute" name="ParMonth" value="10" />
            <PARAM type="absolute" name="param_output_format" value="HTML" />
        </DRILL>
    </DATAMART_PROVIDER>
    <CONFIGURATION>
        [...]
    </CONFIGURATION>
</MAP>

```

Note the "DRILL" tag: the "document" attribute is the label of the target document. A "PARAM" tag define a link parameter: it can be absolute or relative. Absolute parameters have a single value that is specified by the "value" attribute; relative parameters value is the value of the resultset column specified by the "value" attribute.



**Figure 127 – Hyperlink in svg map for cross navigation**

### 7.11.5. SPAGOBIJPIVOTENGINE

You can produce a link in the Scriptable drill-through table: configure the schema extension file (in this case SpagoBIJPivotEngine\WEB-INF\classes\FoodMart.ext.xml) as follows:

```

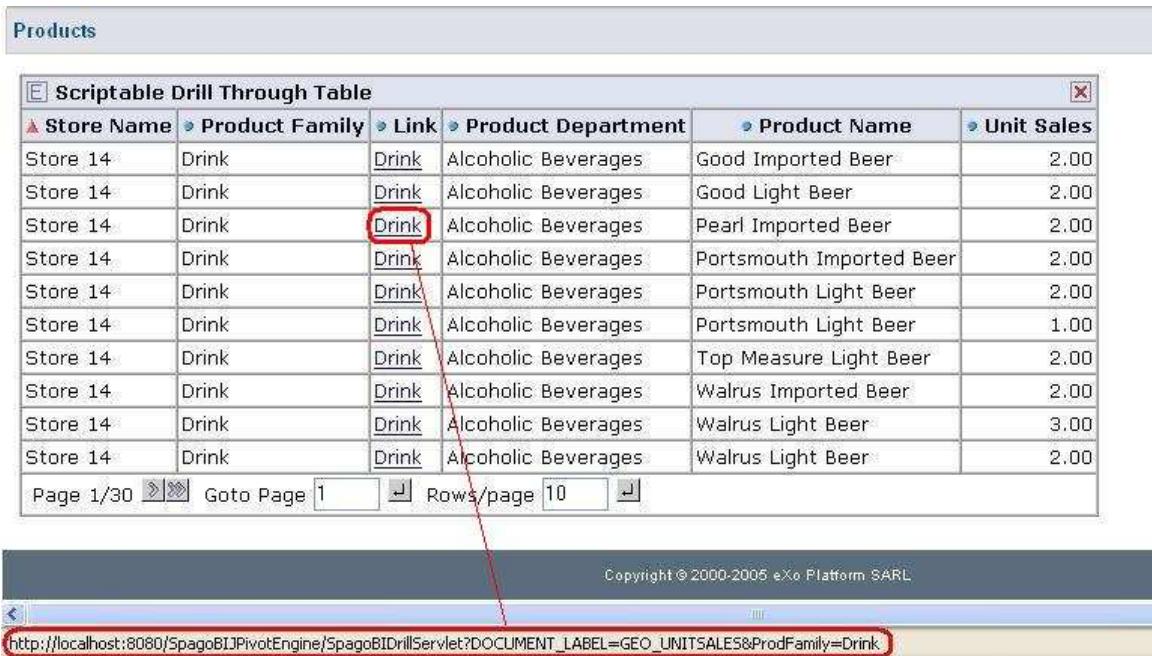
<?xml version="1.0"?>
<extension title="Scriptable Drill Through Table" maxResults="300"
    scriptRootUrl="/groovy-scripts">
    <script title="Link" position="3" file="link.groovy"/>
</extension>

```

and edit file SpagoBIJPivotEngine\WEB-INF\classes\groovy-scripts\link.groovy as:

```
output = [URL:"SpagoBIDrillServlet?DOCUMENT_LABEL=GEO_UNITSALES&ProdFamily=" +
input["Product Family"], Value:input["Product Family"]]
```

you obtain:



Scriptable Drill Through Table					
Store Name	Product Family	Link	Product Department	Product Name	Unit Sales
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Good Imported Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Good Light Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Pearl Imported Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Portsmouth Imported Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Portsmouth Light Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Portsmouth Light Beer	1.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Top Measure Light Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Walrus Imported Beer	2.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Walrus Light Beer	3.00
Store 14	Drink	<a href="#">Drink</a>	Alcoholic Beverages	Walrus Light Beer	2.00

Page 1/30 Goto Page 1 Rows/page 10

Copyright © 2000-2005 eXo Platform SARL

http://localhost:8080/SpagoBIJPivotEngine/SpagoBIDrillServlet?DOCUMENT\_LABEL=GEO\_UNITSALES&ProdFamily=Drink

**Figure 128 – Hyperlink in Scriptable drill through table for cross navigation**

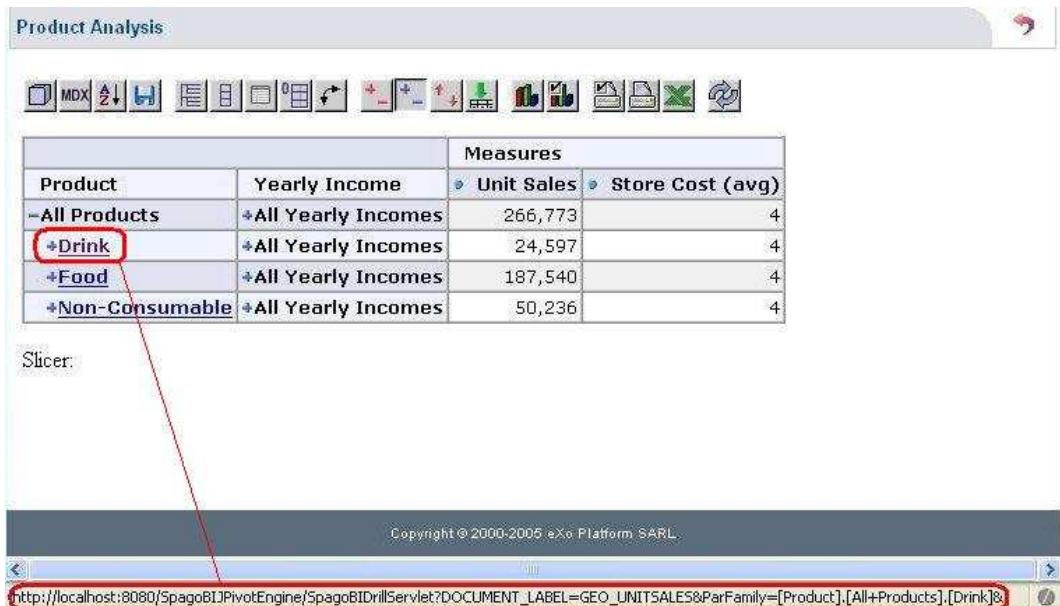
[Note: in SpagoBI Demo unfortunately drill through table does not work, since there is a problem with HSQLDB]

You can also produce a link in the main JPivot table: configure a "clickable" tag in the document template as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<olap>
    <cube reference='/WEB-INF/queries/FoodMart.xml' />
    <MDXquery>
        select {[Measures].[Unit Sales], [Measures].[Store Cost (avg)]} ON COLUMNS, Hierarchize(Union({[Product].[All Products]}, {[Product].[All Products].Children})) ON ROWS
    from [Products]
    <clickable urlPattern="/SpagoBIDrillServlet"
        uniqueName="[Product].[Product Family]">
        <clickParameter name="DOCUMENT_LABEL" value="GEO_UNITSALES" />
        <clickParameter name="ParFamily" value="{0}" />
        <clickParameter name="param_output_format" value="svg" />
    </clickable>
    </MDXquery>
    [...]
</olap>
```

Members of level specified by "uniqueName" attribute will have a link with the parameters specified; {0} is the name of the current member.

Please pay attention to the fact that the member name is complete, for example "[Product].[All Products].[Food]" and not simply "Food".



The screenshot shows the SpagoBI Product Analysis interface. At the top, there is a toolbar with various icons. Below the toolbar is a JPivot table titled "Product Analysis". The table has two columns: "Product" and "Measures". The "Measures" column contains "Yearly Income", "Unit Sales", and "Store Cost (avg)". The "Product" column lists categories: "-All Products", "+Drink", "+Food", and "+Non-Consumable". Each category has a corresponding row with a link labeled "+All Yearly Incomes". The "Unit Sales" column shows values: 266,773, 24,597, 187,540, and 50,236 respectively. The "Store Cost (avg)" column shows values: 4, 4, 4, and 4. Below the table, there is a section labeled "Slicer:" followed by a dropdown menu. At the bottom of the interface, there is a footer with copyright information and a URL: "Copyright © 2000-2005 eXo Platform SARL" and "http://localhost:8080/SpagoBI/PivotEngine/SpagoBIDrillServlet?DOCUMENT\_LABEL=GEO\_UNITSALES&ParFamily=[Product].[All+Products].[Drink]&". A red box highlights the URL in the browser's address bar.

**Figure 129 – Hyperlink in JPivot table for cross navigation**

## 8. In more depth

### 8.1. 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.

---

### 8.2. 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.

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.

### **8.3. USER ROLES**

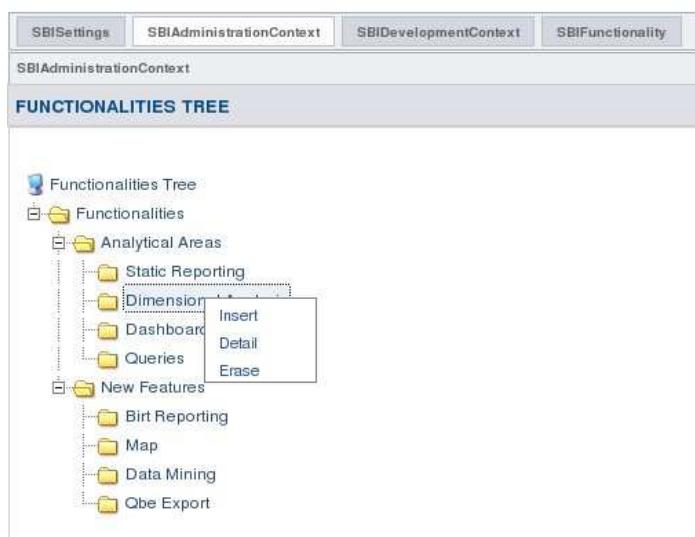
Every user is characterized by one or more functional roles.

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.

### **8.4. 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 130 - 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.

## 8.5. USER DEFINITION AND ROLES MANAGEMENT

---

Work in progress.



## 8.6. PORTAL DEFINITION

---

Work in progress.



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

### 8.7.1. 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 it's necessary to define a new **datamart** object.

\*\*\*\*\*

In order to recall the datamart object from SpagoBI, you must define a DATAMART type document (using the SpagoBIQbeEngine as external engine) where the template is a simple xml file like the following:

```
<QBE>
  <DATASOURCE name="hsq1Foodmart"
    dialect="org.hibernate.dialect.HSQLDialect" />
```

```
<DATAMART      name="foodmart" />
</QBE>
```

You have to specify the datasource logical name (as defined inside exo-home/webapps/SpagoBIQbeEngine/WEB-INF/conf/data-access.xml), the hibernate dialect to be used and the name of the datamart to be executed.

This kind of document does no require any parameters.



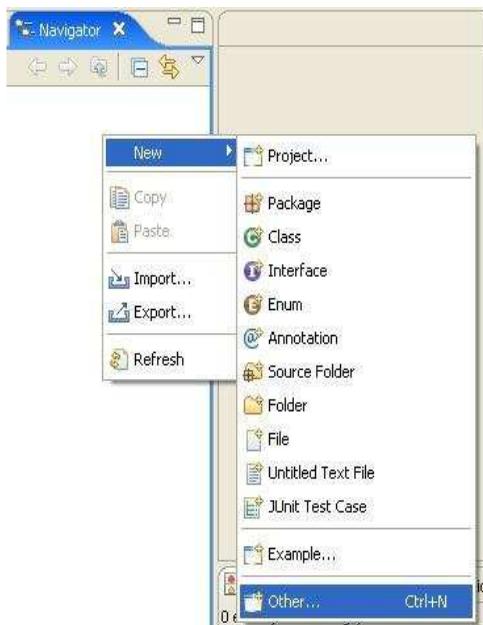
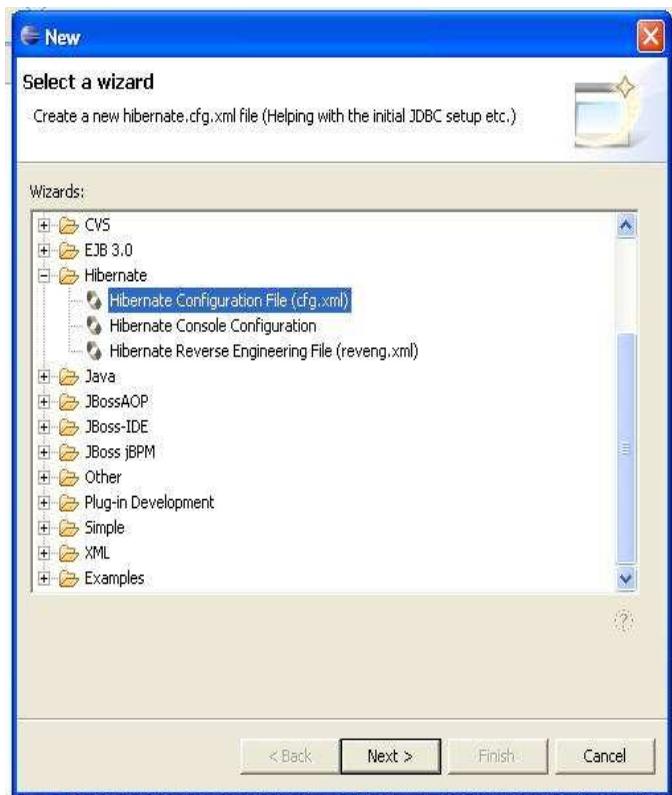
In SpagoBI older versions than 1.9.2 the Qbe was an internal engine and the document configuration was quite different.

### 8.7.2. HOW TO CREATE DATABASE MODEL JAR FILE

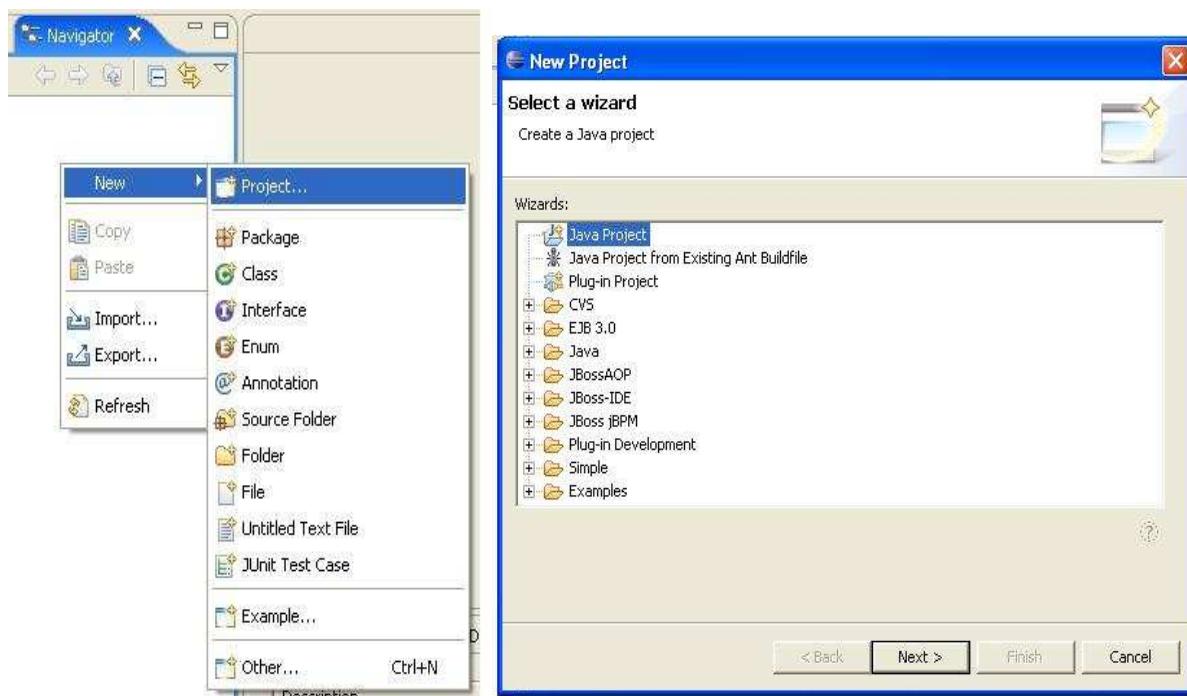
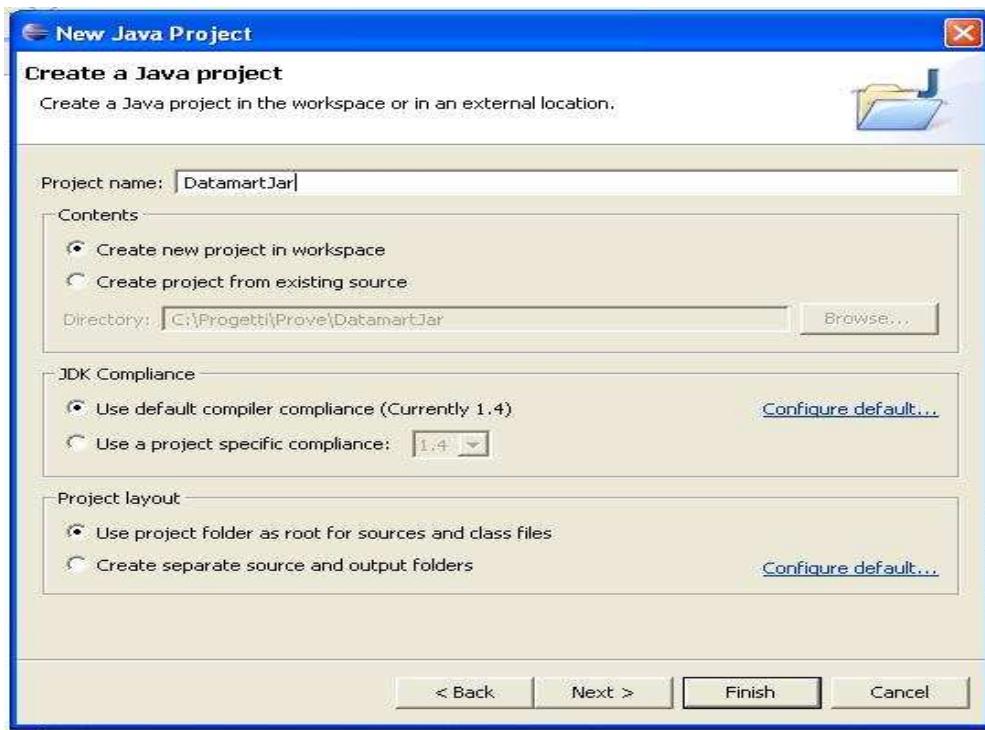
Since a database model jar is composed by a set of hibernate mapping files and relative java classes it's possible to create it manually, only using java and xml editors. However, this operation can be too long and error prone, so, it's better to use some automatic tools like 'HibernateTools-3.1'.

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

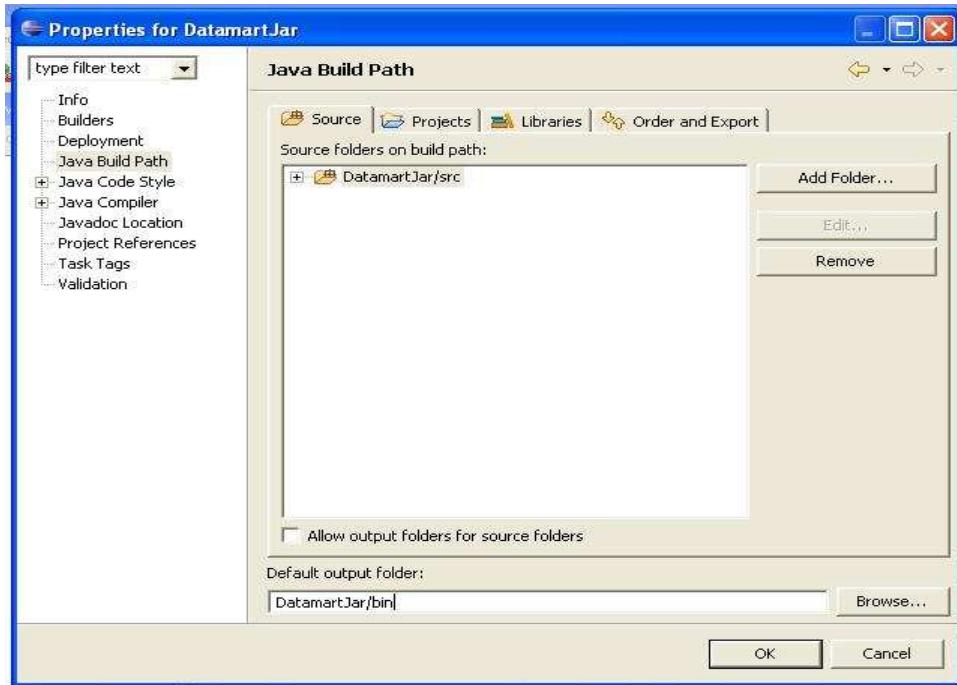
- download eclipse 3.1 sdk. It's strongly recommended to install the tools over a new and clean installation of eclipse 3.1. It's possible to download the 3.1 version from the url <http://download.eclipse.org/eclipse/downloads/index.php>
- install the eclipse platform: simply unzip the zipped downloaded 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/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 131 - Eclipse Navigator and Figure 132 - Hibernate options).


**Figure 131 - Eclipse Navigator**

**Figure 132 - 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 133 - New Java Project and Figure 134 - Default properties of a Java project.

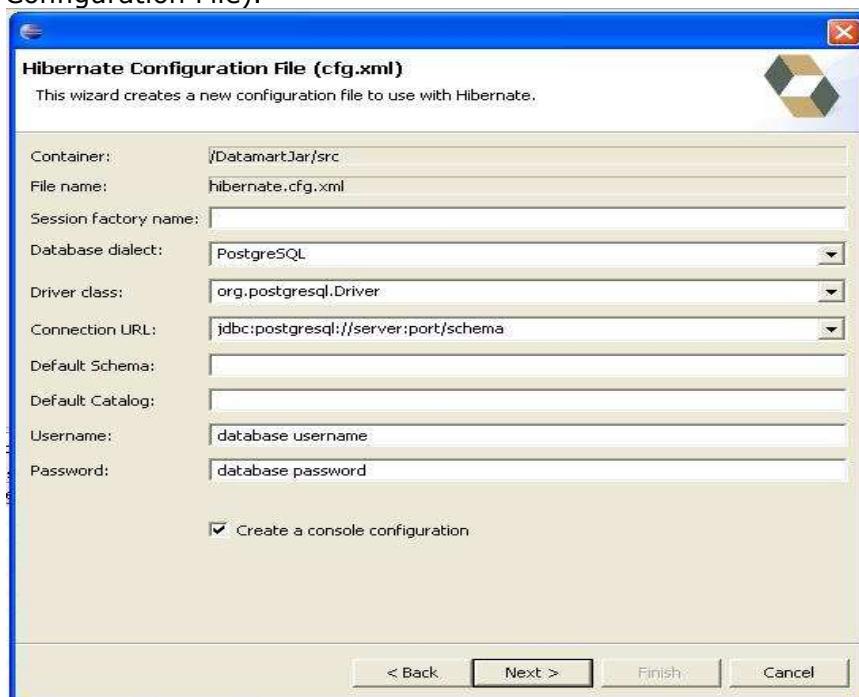

**Figure 133 - New Java Project**

**Figure 134 - 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 135 - Java project build path options).

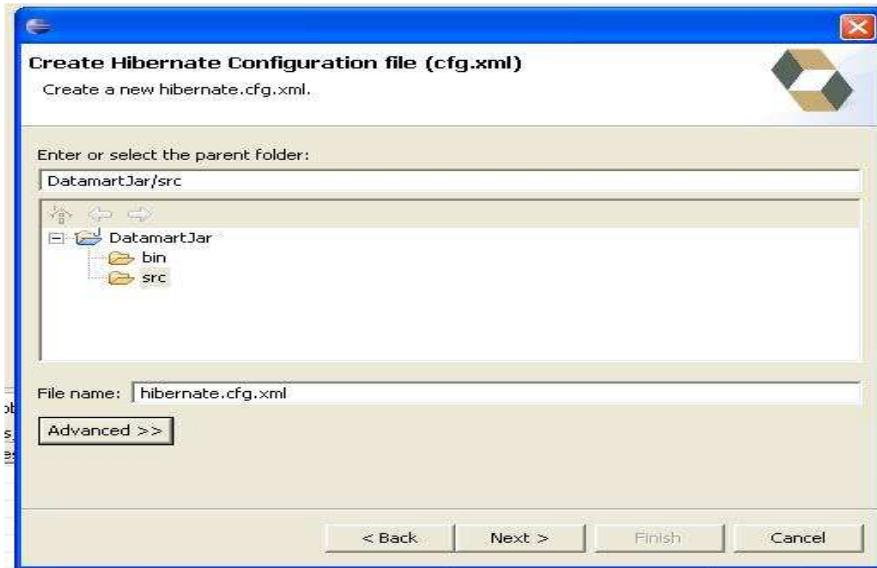


**Figure 135 - Java project build path options**

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 136 - New Hibernate Configuration File and Figure 137 - New Hibernate Configuration File).

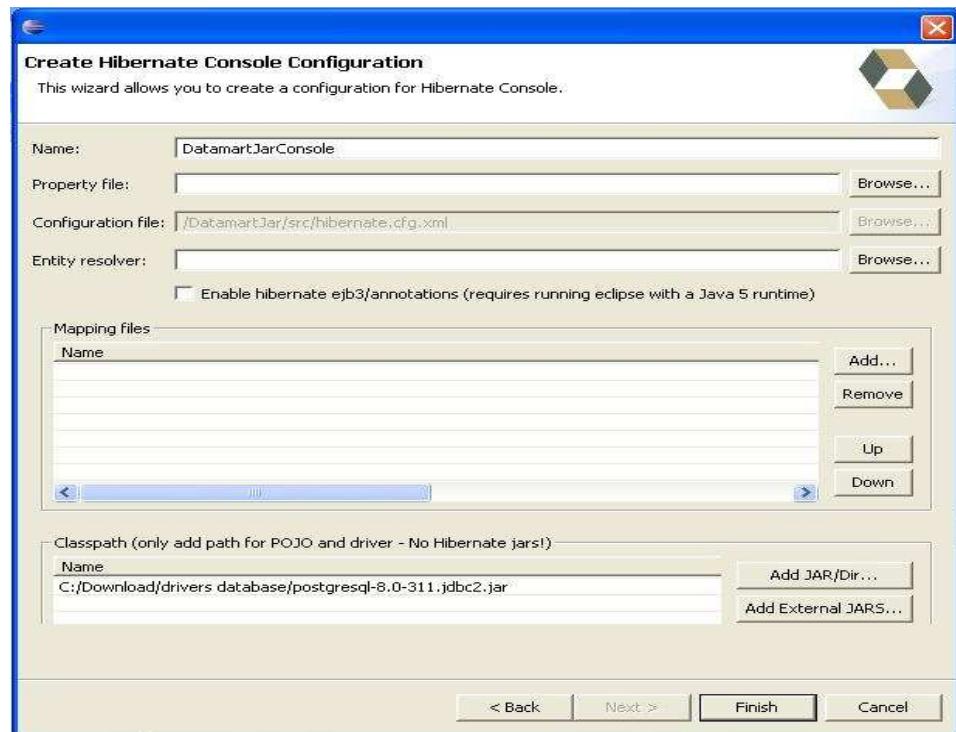


**Figure 136 - New Hibernate Configuration File**



**Figure 137 – 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 138 - New Hibernate Console Configuration)



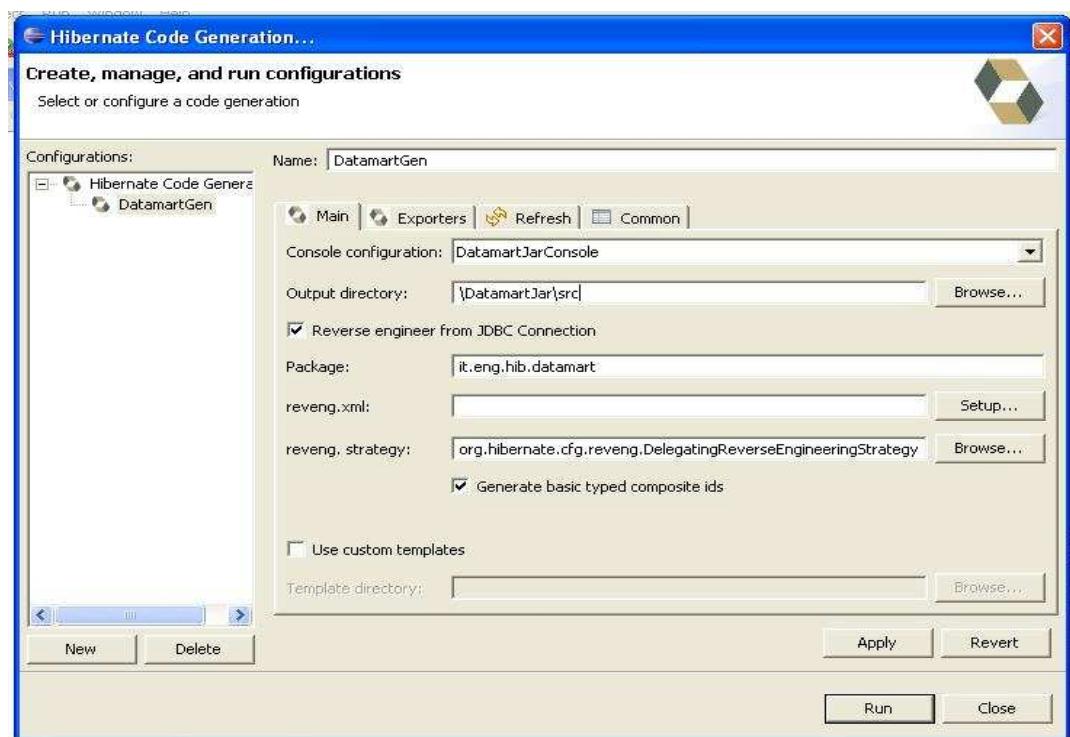
**Figure 138 - New Hibernate Console Configuration**

Exec the hibernate code generation tool (Figure 139 - Hibernate code generation)



**Figure 139 - Hibernate code generation**

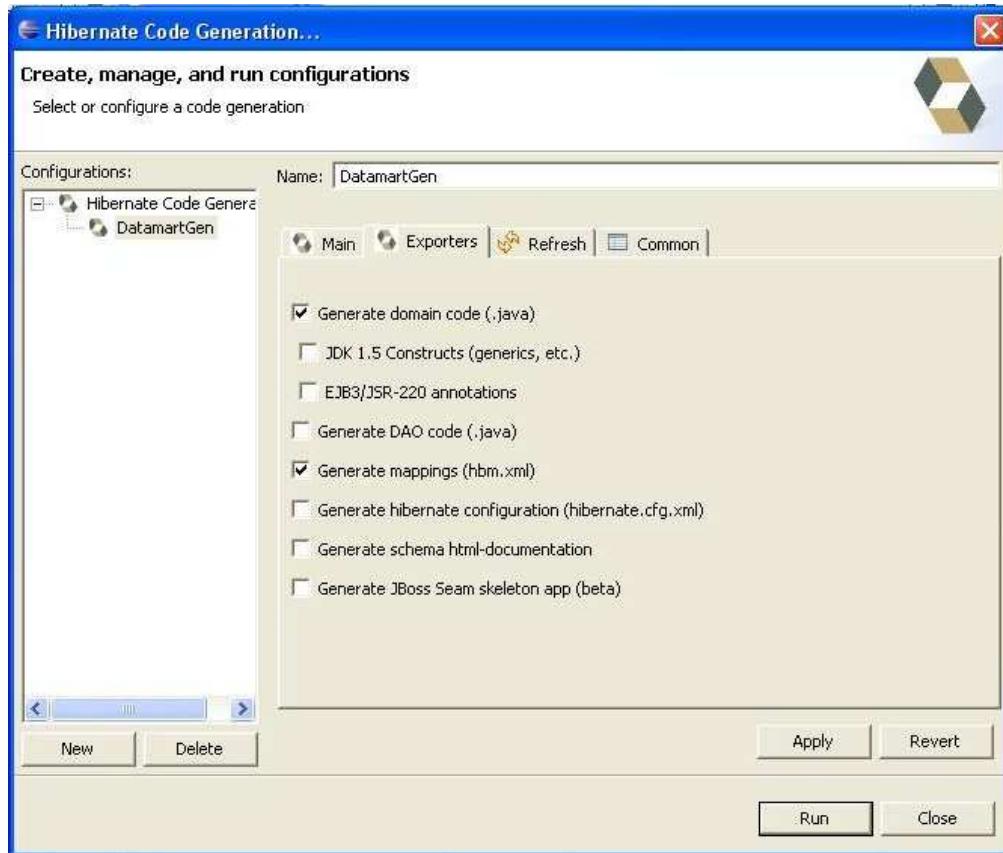
In the main tab give a name to the generation, select as output folder the src folder of the project, check the 'reverse engineer from jdbc connection' option, give a package name for the generated java classes and fill the 'reveng strategy' field with a '**org.hibernate.cfg.reveng.DelegatingReverseEngineeringStrategy**' value (Figure 140 - Hibernate code generation main options).



**Figure 140 - Hibernate code generation main options**

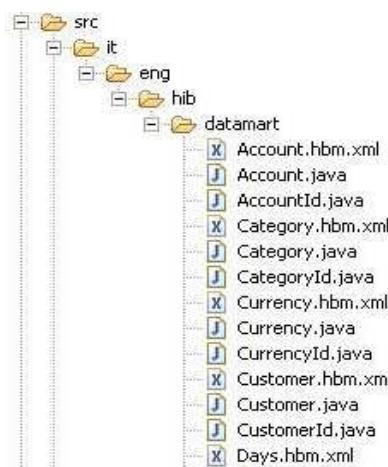
If you want to filter the tables to process for the generation click on the 'setup' button and create a new reverse engineering file. Into the new window select the 'refresh' button and wait until the schema of the database is showed into the left box. From the database schema select the table you want 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 two options as in the Figure 141 - Hibernate code generation exporters options. The "Generate hibernate configuration (hibernate.cfg.xml)" must be un-checked.



**Figure 141 - 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 142 - Java classes and Hibernate mapping files.



**Figure 142 - 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

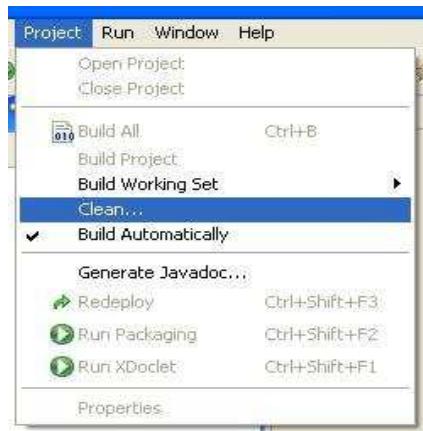
So the files for english / italian labels will be named:

label\_en.properties

label\_it.properties

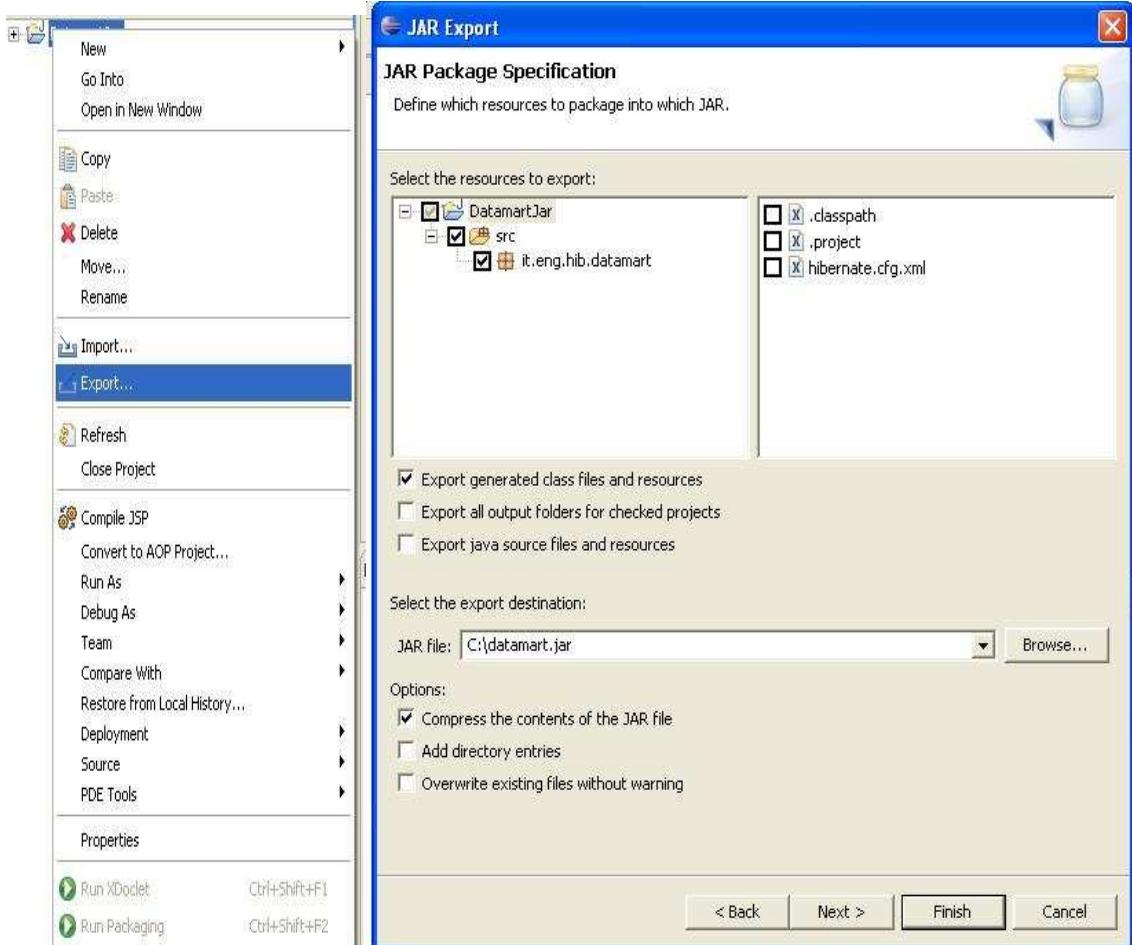
Qbe will automatically choose the correct file based on the portal or browser language. (in case there isn't a suitable file it will use the default one).

To create the jar file first compile the project selecting the Projects/clean menu (Figure 143 - Compile project)



**Figure 143 - 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 144 - Export jar file).



**Figure 144 - Export jar file**

The datamart jar model can be taken from the destination directory and loaded into a SpagoBIQbeEngine as a new datamart object.

## 9. Glossary

---

Work in progress.



---

Analytical document

Portal

Analytical portal

Parameter

LOV (list of values)

User role

Report

OLAP

Data Mining

Dashboard

Scorecard