

BUSINESS INTELLIGENCE WITH SPAGOBI

SPAGOBI COMPETENCY CENTER



Open Source Business Intelligence

data mining external processes charts ad hoc reporting reporting
free inquiry real time location intelligence collaboration olap
kpi office etl cockpits mobile master data management

SPAGOBI COMPETENCY CENTER, ENGINEERING

Business Intelligence with SpagoBI

Copyright © 2012 Engineering
All Rights Reserved
SpagoBI Competency Center
Corso Stati Uniti 23/c
35127 Padua, Italy
spagobi@eng.it
www.spagobi.org

Table of Contents

About the authors	VII
Lecture notes	VIII
Introduction.....	X
1. The SpagoBI Project.....	1
Open Source Business Intelligence.....	1
SpagoBI Suite.....	2
SpagoBI history	3
Open Source Strategy	7
Development Strategy.....	8
Ecosystem and Community.....	12
2. SpagoBI Architecture.....	14
SpagoBI Server	16
SpagoBI Meta	17
SpagoBI Studio.....	18
SpagoBI SDK	18
SpagoBI Applications	18
3. SpagoBI Meta	20
Goals and targets.....	21
Installation and configuration	22
The business model.....	32
Creation of the business model.....	34
Inquiring the business model	41

Deploy and use the business model	46
4. SpagoBI Studio	50
Goals and targets.....	51
Installation and configuration	54
Common tasks	61
Data set definition	61
Download and deploy.....	63
Developing documents.....	66
Report.....	68
Chart	92
Real time.....	122
Cockpit	130
Location Intelligence.....	137
5. SpagoBI Server	152
Goals and targets.....	154
Architecture.....	155
Delivery layer.....	156
Analytical layer	157
Data layer	159
Administrative tools and cross services.....	159
Multi-tenancy.....	161
Installation and configuration	162
The behavioral model	168
User and role configuration.....	169
Repository structure and rights	175
Analytical drivers	178
Publication environment.....	191
Analytical model.....	196
Main concepts	196
Document types.....	198
Register an analytical document.....	200
Correlation between analytical drivers	210

Controlled Visibility	212
Cross services.....	213
Document browser	213
Subscriptions	227
Massive export and scheduling	229
Administrative Tools.....	232
Scheduler.....	232
Import/export.....	246
Engine management.....	249
Map catalogue	255
Data sources	256
Data sets.....	260
6. Analytical engines	277
Report.....	278
SpagoBIBirtReportEngine.....	279
SpagoBIJasperReportEngine.....	280
SpagoBIAccessibleReportEngine	281
Multidimensional analysis	285
SpagoBIJPivotEngine	286
SpagoBIJPaloEngine.....	310
SpagoBIJPXMLAEngine	318
Chart	322
SpagoBIJFreeChartEngine.....	323
SpagoBIJSChartEngine	363
Analytical cockpit.....	399
SpagoBICompositeDocEngine.....	400
KPI.....	403
Build a KPI model.....	405
Calculate a KPI model.....	426
Create a KPI analytical document	427
View a KPI model.....	432
Data Mining	433

Basic concepts.....	434
SpagoBIWekaEngine	436
Advanced functionalities	448
Free Inquiry.....	455
SpagoBIQbeEngine	456
SpagoBISmartFilterEngine	480
Ad hoc reporting.....	491
SpagoBIWorksheetEngine.....	492
Location Intelligence.....	501
Basic concepts.....	502
SpagoBIGeoEngine	507
SpagoBIEORReportEngine.....	533
Real time	551
SpagoBIDashboardEngine	552
SpagoBIConsoleEngine.....	563
Mobile BI.....	620
Architecture	620
Installation	622
Navigator.....	623
SpagoBITableMobileEngine	625
SpagoBIChartMobileEngine	629
SpagoBICockpitMobileEngine	635
Office integration.....	639
SpagoBIOfficeEngine.....	639
Collaboration	641
SpagoBIDossierEngine	642
7. Operational engines	652
ETL.....	652
SpagoBITalendEngine.....	652
External processes	662
SpagoBIProcessEngine	662
Master data management.....	670

SpagoBIRegistryEngine.....	670
8. SpagoBI SDK	676
Goals and targets.....	676
Installation and configuration	677
SDK Web Services	682
Proxy.....	682
DocumentService	683
DataSetsSDKService.....	699
DataSourcesSDKService.....	704
EngineService.....	705
MapsSDKService.....	707
SDK Tag Library.....	709
SDK Javascript APIs.....	712
9. SpagoBI Applications	715
Goals and targets.....	715
Installation and configuration	717
Audit and monitoring	719
Audit documents	720
The last access point for the analysis is the document showing the list of LOVs. LOVs can be searched using a filter on the type of LOV, namely: query, fixed value, and script	725
Monitoring documents	727
Other SpagoBI Applications	727
Spago4Q.....	727
SpagoBI for Areas	728
SpagoBI for Elisa.....	729
SpagoBI for Adaxa Suite	730
Reference and resources.....	732
General information.....	732
Technical resources	733
Professional services.....	733
Figures and Tables	735

Index of figures	735
Index of tables	743

About the authors



Grazia Cazzin

Grazia, SpagoBI Project Leader and SpagoBI competency center director, is a technical manager at Engineering Group. With many years of experience in enterprise application development, data modeling, data warehousing, dimensional analysis and business intelligence, she has gained valuable expertise working in several market sectors (industry, finance, public administration), covering several thematic areas (ERP, MRP, MPS, Enterprise Portals, CRM, DWH and BI). She has served as Assistant Professor of Business Intelligence and DSS at the Faculty of Mathematical, Physical and Natural Sciences of the University of Turin (Italy).



SpagoBI Core Team

SpagoBI core team includes consultants, architects and developers that constantly work on SpagoBI code or projects, mixing technical competencies and business intelligence knowledge. Team members have significant experience in several market sectors, e.g., industry, finance, public administration, and healthcare. They contribute to SpagoBI growth by sharing their competencies and by proposing novel ideas and emerging topics in BI.

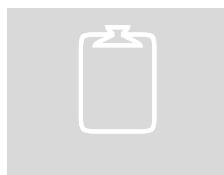
Lecture notes

Some graphical conventions have been adopted to allow readers to easily identify special contents such as notes, summaries, essential information and so on. All conventions are explained hereafter.



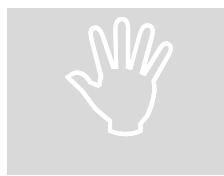
Notable content

This icon highlights relevant content.



Summary

This icon highlights short texts that sum up key messages.



Attention

This icon warns the reader about possible errors and problems using SpagoBI.



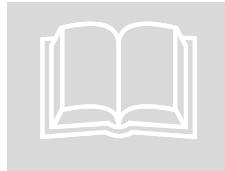
Advice

This icon identifies suggestions to better use SpagoBI.



Writing convention

This icon explains any stylistic choice.



Read more

This icon refers to additional documentation that may be useful to the reader. It usually points to external sources for non-native SpagoBI features.



Reference

This icon points to another section of the book dealing with a specific subject.



Work in progress

This icon highlights SpagoBI functionalities that are still under development or need improvement.

The following fonts have been conventionally adopted, to easily identify special words and expressions:

- **Menu**, **menu item** and **static label** directly refer to SpagoBI GUI
- **Input field** is a label referencing input fields in SpagoBI GUI
- *SpagoBI keyword* highlights SpagoBI terminology
- **Code example** is a piece of code showing configuration patterns or parts of a document template.

Introduction

Business Intelligence platforms enable users to build information support applications that help organizations in their decision-making processes for enterprise strategic and tactical management.

The BI market has recently witnessed two relevant trends. In the proprietary world, major IT vendors have been acquiring existing BI products, thus creating a small set of leading actors sharing the market. On the other hand, open source BI systems have emerged as a suitable solution for companies wishing to combine specific features from different tools, instead of adopting monolithic proprietary BI products.

Today SpagoBI is the only 100% open source, complete and flexible business intelligence suite. SpagoBI covers all areas of Business Intelligence with a wide set of analytical engines and cross functionalities. The suite has been developed to constantly meet real end-users' needs, carefully balancing open source flexibility and enterprise-grade software quality. Until now SpagoBI has been used in dozens of BI projects, making a constant effort to find the best solution for the end-user, saving time and economic resources.

After seven years spent managing BI projects with SpagoBI and developing the suite, time has come to share the acquired knowledge and expertise, letting more and more users get to learn SpagoBI and make the best use of its various features.

This book is a reference manual for those wishing to learn how to use SpagoBI. It targets different types of users, just as the different modules of SpagoBI do. All modules are described from the perspective of the main target user, showing examples and providing useful hints and references. Readers can choose whether to take a quick overview on the suite, or follow the detailed explanation of a specific tool.

Needless to say, a book cannot replace those design and analysis activities that are fundamental to the success of any BI project. Nevertheless, this manual represents a valid support for BI developers, integrators, domain experts and users interacting with SpagoBI in any phase of the project.

The book is organized as follows:

Chapter 1 provides an overview of SpagoBI project, its history and specific characteristics, its open source and development strategies, the ecosystem and SpagoBI community.

Chapter 2 briefly introduces SpagoBI suite architecture and its composing modules, which are described in the following chapters.

Chapter 3 presents SpagoBI Meta, the graphical environment to build and inquire business models.

Chapter 4 describes the environment for developing SpagoBI analytical documents: SpagoBI Studio. It provides installation and configuration instructions, as well as detailed explanations for each document designer.

Chapter 5 and 6 are focused on SpagoBI Server. The first presents its overall architecture, cross services and administrative tools, as well as the main conceptual models driving its logics. This includes the behavioural model, which rules visibility and permissions on documents; and the analytical model, a common framework to represent the BI project.

All analytical areas covered by SpagoBI are described in Chapter 6, which focuses on the main functionalities provided by each engine and offers a detailed step-by-step guide to build documents.

Chapter 7 is dedicated to users wishing to integrate SpagoBI with their own information system by relying on SpagoBI SDK.

Chapter 8 presents SpagoBI Applications through relevant examples.

Finally, useful references and resources are provided.

1. The SpagoBI Project

Open Source Business Intelligence

Business Intelligence platforms enable users to build information support applications that help organizations in their decision-making processes for enterprise strategic and tactical management. It is a vibrant and constantly growing sector in the Information Technology market, as confirmed every year by the main IT research and advisory companies. The strong competition that characterizes this market led to several acquisitions over time, until the outline of a few mega-vendors, which control almost all proprietary offering.

During the last years, the landscape of business intelligence platforms and products has undergone a radical process of transformation. Some historical suppliers merged creating a narrow circle of leading actors, while new models emerged aiming at reducing the total cost of acquisition and management as a whole: software as a service, cloud, mobile applications, appliances and analytics.

In this scenario open source plays a key role as it allows users and organisations to combine the best features of available tools, functionally enriching and adapting them to the typical architectural contexts of business intelligence enterprise solutions, which are strictly related to issues such as security, scalability and reliability.

In the past, good open source tools were provided that focused on specific Business Intelligence tasks, such as:

- JasperReport and BIRT as reporting tools
- JFreeChart as chart library
- Weka as data mining tool
- Jpivot/Mondrian as OLAP tool
- Kettle and Talend as ETL tools.

Each single tool was satisfactory in its domain, but its usage was limited to personal use. In fact all solutions were developer-oriented and they did not cover enterprise-level issues such as security, life cycle, scalability and delivery.

Starting from those tools, some Business Intelligence products appeared on the market that integrated them into an enterprise architecture. Each Business Intelligence product enhanced its approach and features over time, providing new and different BI capabilities released with different policies, according to the vendors' different business models.

Today, SpagoBI is the only entirely open source business intelligence suite adopting a peculiar approach that makes it unique in this domain.

SpagoBI Suite

SpagoBI is the only 100% open source, complete and flexible business intelligence suite.

SpagoBI is 100% open source forever, because it is released under an OSI¹ approved open source license, protecting the freedom of the code and allowing its commercial use, resulting in the development of various business models in its own eco-system. It adopts the “pure open source” business model as it is released as only one stable version. It is an alternative to the *dual-licensing* or *open core* model, which also includes a parallel proprietary software release as an enterprise or professional version. In order to guarantee the open source transparency, openness, quality, sustainability and availability over time,

¹ <http://www.opensource.org/>

SpagoBI software is download-able from the forge of OW2 Consortium², a no-profit and independent open source community.

SpagoBI is a complete suite because it covers the whole range of Business Intelligence needs, with innovative solutions, providing users with a wide range of analytical tools supporting developers, testers and administrators in their daily work. This maximizes the effectiveness of data analysis according to the principle: “the right tools for the right user”.

SpagoBI is a flexible suite, because it offers many engines for the same analytical area, allowing users to freely choose how to build their own solution. Based on open standards, avoiding a pre-defined software stack, SpagoBI provides the best solution for users’ needs, saving investments and providing quick results. The number of available engines and their characteristics have been growing over time according to SpagoBI project road-map. Moreover, the suite is suitable to various environments, as it can easily integrate with security systems, portals and legacy applications.

The suite is complemented and supported by a complete set of professional services, including training, consulting, development, support and maintenance, provided by SpagoBI Competency Center.

SpagoBI history

SpagoBI belongs to SpagoWorld³, the open source initiative by Engineering Group. The story began in 2001, with the need of the company to realize a Java Enterprise framework that could support project development for its customers. At the end of 2004, this framework was re-engineered and released as open source software on SourceForge - the Spago framework. SpagoBI

² <http://www.ow2.org>

³ <http://www.spagoworld.org>

project⁴ started in late 2004. In July 2005 SpagoBI was released for the first time on SourceForge, and then moved onto the forge of ObjectWeb Consortium. In 2006 SpagoWorld initiative was formally launched and the codebase of SpagoWorld projects was moved onto the forge of the new OW2 Consortium. In 2007 two more projects were released: Spagic, the open source universal middleware, and Spago4Q, which is a vertical application of SpagoBI focused on quality of products, processes and services. In 2010, when Engineering Group joined the Eclipse Foundation, the eclipse eBPM and eBAM projects, the latter lead by SpagoBI team, were launched.



FIGURE 1.1 – SpagoBI history

Nowadays SpagoBI Competency Center is a work unit of the Research and Innovation division of Engineering Group managing all the activities related with SpagoBI suite and open source business intelligence: development and management of SpagoBI and eBAM projects, supply of SpagoBI professional

⁴ The name SpagoBI refers to SpagoWorld initiative, marked by the root of the word "**Spago**". Spago (i.e., twine) is not an acronym, but an Italian name suitable for an integration approach (it ties, without compelling). It may also recall the widely known, fine Italian cuisine: well, if you like Spaghetti, Spago software may be an excellent choice for you!

services, management of GeoBI initiative, participation in BI-related OW2 Consortium initiatives, development of business intelligence projects and BI consultancies at Engineering Group.

When the first version of the Open Source Business Intelligence SpagoBI suite was released in 2005, within the framework of SpagoWorld initiative, the aim of the SpagoBI project was to build an integration platform which could offer a flexible and sustainable BI solution to organizations and companies, supporting the measurement and management of their critical assets and success factors.

At the beginning, the product was built by integrating the best available open source tools, making them usable at the enterprise level, by customizing and extending their basic functions on the basis of the information coming from the first business intelligence projects in which the product itself was used. This way, SpagoBI development team created new analytical engines and consequently started a virtuous circle, according to which BI project needs quickly turned into additional practical capabilities or features that contributed to the improvement of the product.

As a result, the product has become more and more powerful and usable. Specifically, innovative practical functionalities have been added at a first stage, which was substantially oriented towards the support of the basic business intelligence functions, such as reporting and multidimensional analysis. They included: representation of data analysis on geo referenced supports (Location Intelligence), data inquiry and collaborative functions, using annotations and cooperative workflow.

The first wave of projects realized by means of SpagoBI version 1.x was mainly motivated by users' wish to test this open source product with a very cautious but also innovative approach. Therefore, the first projects aimed at demonstrating that the product was able to cover BI users' real needs at a much more reasonable cost than proprietary solutions. As time goes by and with the increasing diffusion of the open source model and applications, users' and companies' approach to open source products has started to change. This applied to SpagoBI as well: beside testing strategies, the replacement of proprietary tools and the strategic adoption of open source platforms have become key decision drivers for SpagoBI adopters.

This was just the right context for a significant evolution of SpagoBI towards 2.x version, released at the end of January 2009. This made a considerable step ahead not only from the technological and architectural perspective, but also from the functional one.

From the architectural point of view, three important changes deeply affected the design of SpagoBI core:

- the components splitting towards a fully compliant Oriented Architecture: all features of the previous version were collected in SpagoBI Server and related to each other in a secure mode; in addition, external services were provided in the new SpagoBI SDK module;
- the integration of the CAS security module, which considerably improved access security in all suite services;
- the independence from external systems, such as portals and profiling systems, as well as CMS as the internal objects repository.

From a functional viewpoint, various BI capabilities have been added, such as the new GIS engine for the representation of the analysis on geographical maps; composed documents, allowing to build highly efficient and interactive dashboards; the KPI engine; new dashboards; subscription functions. Next to the Server and SDK modules, the SpagoBI Studio and Applications modules were also first introduced in the 2.x series, as the development environment and a set of pre-built analytical models, respectively.

It is important to notice that, once released, SpagoBI 2.0 version has been used in several business intelligence projects, directly managed by SpagoBI development team, ensuring solidity and response to the requirements of Engineering Group's major customers.

Finally, in mid-June 2011, version 3.0 of SpagoBI was released, marking an important turning point in the domain of enterprise-grade open source solutions. This version offers a module fully dedicated to metadata management (SpagoBI Meta), allowing users to define different information models within their own data domain and to develop new analytical documents through the complete data sources abstraction. It also includes a new ad-hoc reporting engine for the guided creation of customized reports and a new live-chart engine supporting the realization of advanced cockpits even on mobile

devices, such as iPad and Android tablets, as well as iPhone and Android smart phones. The 3.0 release also improved real-time data analysis, based on systems and events interacting with OLTP (On-line Transaction Processing) applications. The real time console was integrated in the framework of a new Eclipse project named extended Business Activity Monitoring (eBAM), where a Complex Event Processing (CEP) engine was also implemented.

With the 3.x series, SpagoBI has not only grown to become a complete BI suite, but it has also increased the range of solutions and services to support users in the adoption of the product (agile development methodology, on-demand, cloud and SaaS usage models), while enriching its offer of support and training services.

Today SpagoBI is an open source business intelligence suite providing features comparable with proprietary products, such as Agile BI, Mobile BI, Cloud BI, Real-time BI, and offering vendors and integrators a new window of opportunities in the Information Technology market.

Open Source Strategy

SpagoBI open source strategy is based on the following key principles:

- the development of enterprise-grade software supported by companies and communities (both industrial and open), allowing users – enterprises and organizations – to use mature, stable, scalable solutions and a wide range of support services;
- the full adoption of the open source model in order to foster knowledge and expertise exchange, which is a crucial element in a really free and open source environment;
- the awareness that open source communities have been evolving over time towards a wider *ecosystem* model, where companies of different size and business activity cooperate to improve open source products and bring them to a full enterprise quality level.

According to this strategy, SpagoBI belongs to a new generation of open source projects, characterized by:

- holistic approach and planned development process;
- suitability to complex technological domains aiming to support the achievement of business goals in various markets;
- mature process development;
- involvement of a wide community of users, companies and organizations, open source specialists, consultants and developers;
- guaranteed support services.

To this end, SpagoBI suite is released under an open source software license (MPLv2⁵) aiming at encouraging a wider adoption of the suite, while clearly stating three basic principles:

- it is both open source and free software (i.e., GNU (X)GPL compatible);
- it is a weak copyleft licence based on the protection of the free code of the suite;
- it is a commercial friendly license: the code of the suite can be “linked” to open source and proprietary third-party software⁶.

Development Strategy

SpagoBI competency center fosters the sustainable development of business intelligence projects and applications, in order to allow organizations and enterprises achieve their mission-critical goals, within a trustworthy environment, guaranteed by the software quality, the use of open standards and open source software, and innovation. The strategy adopted by SpagoBI competency center is based on sharing the developments with the community, consolidating the open source solutions at the enterprise level, strengthening

⁵ <http://www.mozilla.org/MPL/2.0/>

⁶ More information at: <http://www.spagoworld.org/xwiki/bin/view/SpagoBI/Licenses>

international partnerships and orienting the project road-map towards industrial initiatives.

In particular, SpagoBI vision is that user's needs are more valuable than the adopted product, in contrast with many solutions imposing products' constraints onto users' requirements. This way, a good solution can actually improve a business intelligence project, if it promotes:

- a project-oriented solution, which, freed from product fees, favors the project start-up, offering a new balance among the elements involved in the development phase - effort, cost, time and quality of results. This deeply differentiates SpagoBI from many other solutions, which are based on proprietary products and characterized by price discrepancies;
- a coherent design, thanks to the adoption of open standards, a modular architecture, the development of innovative components and the integration of the best open source solutions, supporting their integration and re-use into existing environments, thus increasing the value of the solutions that are already in use;
- an evolutionary or agile development process helping users achieve their goals since their first steps: starting small, thinking big, rapidly obtaining first results;
- a sustainable growth, which allows users to benefit of good features and customizable enterprise-level solutions (which are also high-quality, reliable and scalable), adequate support services at a reasonable cost (including installation/configuration, bug-fixing, maintenance, training, warranty), giving them the opportunity to become confident with open source solutions;
- good features that are actually customizable, adequate support services provided at a reasonable cost, giving users the opportunity to become confident with open source solutions;
- a shared business model among vendors and users in a win-win perspective: sharing business goals and priorities may be the best way to obtain the expected results.

According to this vision, SpagoBI has a distinctive and unique feature: its entirely open source approach, which gets over the cultural attitude of keeping

off new rules and ideas. The most adopted business models in the open source world, known as dual-licensing or open-core, follow the traditional paradigm of the proprietary market. This approach, which considers the product as the central element, relies on license sale as the main source of income and imposes software lock-in, tightly binding subscription services to the rights to use the software.

On the contrary, SpagoBI, thanks to its pure open source approach (i.e. a single stable version of the product, entirely distributed under an open source license, adding no “professional” or “enterprise” proprietary versions), imposes no software lock-in, separating any purchase of services from the right to use the software. Moreover, it adopts a project-centric model instead of a product-centric one. This has influenced the architectural choices, its business evolution, its business strategy and its own approach to the development of business intelligence projects.

The goal is not to make money by selling a product, but to provide new opportunities to create a business intelligence solution that meets end-users' real needs and fits their budget, involving them in the product development. The pure open source approach adopted by SpagoBI allows the suite to grow over time thanks to the experience and innovative ideas of the SpagoBI development team, which interprets users' requests and project needs.

This approach works because SpagoBI development team understands users' needs and transforms them into new requirements to be brought into the project roadmap. Along this process, new ideas are validated until they can be included into the open source solution, allowing to create new value for customers and, through the realized product, for all other users as well. This way, different business intelligence projects contribute in terms of requirements, code, test and funding. Moreover, this model fosters sharing of common developments and research tracks with various organizations and communities.

In the meantime, users asking for support are provided with a complete set of professional services including advice, maintenance, training, consultancy, software development, in-house and on-site project development.

Many SpagoBI innovative engines have been developed in the context of actual business intelligence projects (or software projects including analytical capabilities) required by SpagoBI competency center's customers. They represent the core inspiration source for SpagoBI development and growth. Those projects led, among others, to the development of the Geo Engines, the Composite Document, the KPI, the QbE and the ad-hoc reporting engines.

SpagoBI executive team constantly defines the SpagoBI project road-map as a balanced mix of the market analysis and innovative ideas, always involving customers and partners to create synergies between their needs and the project road-map. Thanks to this approach, customers get answers to their specific needs, and see their money invested in the development of their solution, not in software licenses. They obtain solutions that are technologically mainstream and innovative, being guaranteed about the continuous maintenance of the solution itself even in subsequent SpagoBI releases. At the same time, SpagoBI competency center - the software editor – manages to economically sustain the development of the solution and to constantly orient it to solving actual requirements, enforcing its effectiveness. In a few words, it is a true win-win approach.

This is a significant example of the importance of sharing development and creating synergies between multiple projects. Probably no single software project can support the development of a complete functionality on its own, but splitting investments into multiple projects makes the investments themselves more affordable and sustainable.

SpagoBI project is sustained by Engineering Group, an industrial ICT company. It is a different approach compared to many corporate-lead open source projects that are mainly funded by venture capitalist firms. This is part of the wide open source approach adopted by the company: the project itself must be sustainable, balancing industrial funding, revenues coming from the sale of professional services and contributions coming from the development of business intelligence projects, from the partners and the community. The wide adoption of SpagoBI suite in many enterprise-grade BI environments and its stable maturity demonstrate the effectiveness of this approach.

Ecosystem and Community

SpagoBI is an open source project which guarantees openness, not only thanks to the open availability of its results – mainly source code – thanks to its license policy, but also because its design, collaboration, assembling and adoption in the real world live through the community, which is the main driver of SpagoBI innovation and evolution.

SpagoBI community mainly consists of SpagoBI competency center's customers involved in BI projects, integrator partners and users. All of them create an eco-system where the achieved value is the result of complex interactions among the different participants sharing a common decision space. The final value is the mutual enriching relationship among the different agents composing this network, going far beyond the simple sum of single contributions. The larger the community, the greater the amount and quality of relationships connecting its members, the bigger the shared value.

This model, mainly focused on users' needs and real-world capabilities through the development of business intelligence projects, rather than on the product appeal and marketing, makes SpagoBI one of the best solutions and the most complete suite in the Open Source Business Intelligence domain, offering more flexibility than proprietary business intelligence solutions.

SpagoBI participates in OW2 Consortium, a global, independent not-for-profit open source community, open to companies, institutions and individuals, whose goal is the development of open source code infrastructure (middleware and generic applications). OW2 role is central in SpagoBI open source policy because it guarantees transparency, openness, quality, sustainability and availability of the open source code over time. SpagoBI's active commitment in OW2 Consortium is demonstrated by many activities, above all the following:

- leading – from the technological point of view – the OW2 BI Initiative, a joint effort of OW2 and no-OW2 members, set up to foster the growth of a business ecosystem in the Business Intelligence domain;
- participating in the OW2 Open Source Cloudware initiative, a joint effort of partners from Brazil, China, Europe and the U.S.A., all sharing the same goals in the open source cloud domain.

Moreover, SpagoBI competency center participates in the Eclipse Community with a leading role in the eclipse extended Business Activity Monitoring (eBAM) project, which adds real-time BI capabilities to SpagoBI suite.

SpagoBI project team closely works with a wide network of international partners, which includes systems integrators, technology partners and all those partners who want to contribute to the global success of the suite. All SpagoBI partners contribute to the growth of SpagoBI functionalities and are committed to bring the projects that integrate SpagoBI to rapid success.

Last but not least, SpagoBI ecosystem includes research laboratories and institutes, as well as several universities at the international level, thanks to the continuous support that SpagoBI team offers to researchers, professors and students, both in educational and research activities.

2. SpagoBI Architecture

SpagoBI overall architecture is composed of five modules:



SpagoBI Server, the core of the suite that offers all the core and analytical functionalities



SpagoBI Meta, the environment to define and manage meta-data and business models



SpagoBI Studio, the development environment



SpagoBI SDK, the integration layer that allows external tools and applications to interact with SpagoBI Server



SpagoBI Applications, collecting vertical models focused on a particular business domain.

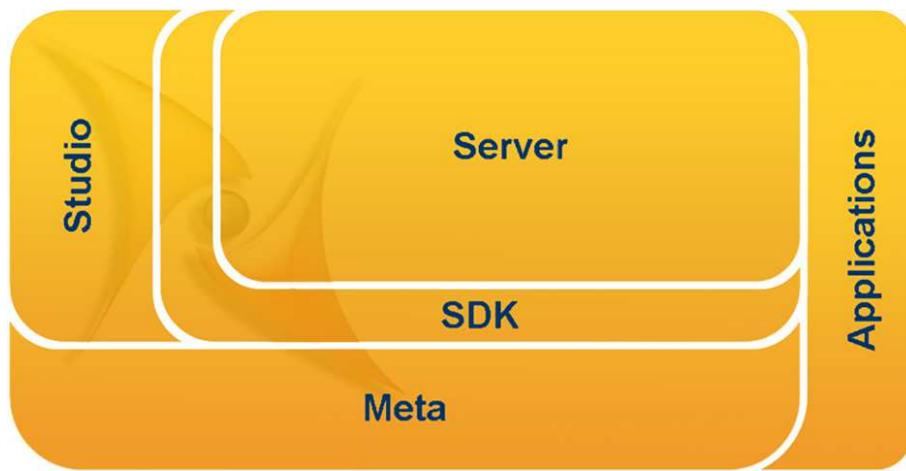


FIGURE 2.1 – SpagoBI suite main modules

The main module, which provides all the BI capabilities, is the SpagoBI Server. This is the only mandatory module, the one that offers a web-based end-user environment to run all types of analysis.

The Meta and Studio modules interact with the Server thanks to the SDK, mainly to authenticate the user, send or receive metamodels, analytical documents or data sets.

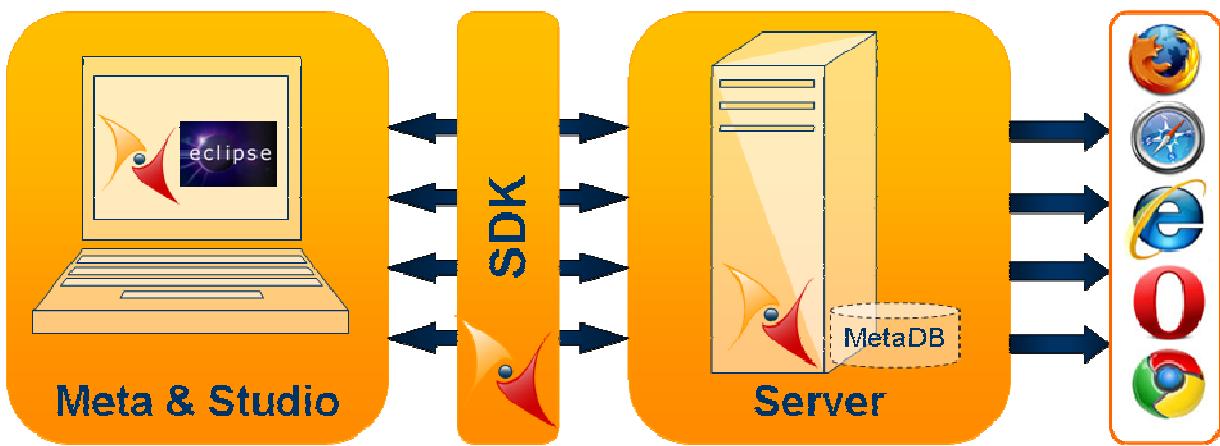


FIGURE 2.2 – Interaction between SpagoBI main modules

A typical BI project uses:

- A single SpagoBI Server : the main BI platform where all BI documents are stored and used by end users. The end-user GUI is entirely web based and can be accessed via a common web browser
- A few instances of SpagoBI Meta: the data manager uses SpagoBI Meta (client application) to define the business models over the enterprise data warehouse
- Several instances of SpagoBI Studio: developers use SpagoBI Studio (client application) to develop analytical documents using the business model, to get data and deploy documents into a remote SpagoBI Server.

The Meta and Studio modules are also available in a single package, which is particularly useful when the data developer is the data manager.

Each module has its own end-user target and covers specific tasks in a development process, as explained in the following paragraphs.

SpagoBI Server

SpagoBI Server is the environment for:

- End-users: they can use their analytical documents by means of this unified access point via a web browser
- System administrators: they can manage the server through a web interface.

For end-users, this module offers all typical BI functionalities :

- Reporting
- Multidimensional analysis (OLAP)
- Charts
- Interactive cockpits
- KPIs
- Data Mining
- Free Inquiry and driven data-selection

- Ad-hoc reporting
- Location Intelligence
- RT dashboards and consoles
- Mobile
- Office automation
- Collaborative tools.

Moreover, some operational BI features are available:

- ETL
- External processes
- Master data management.

For the administrator, SpagoBI Server allows to manage all relevant tasks such as the lifecycle of analytical documents, internal repository, users and roles, security, and presentation environment.

From a technical point of view, SpagoBI Server is a web application deployed into a J2EE application server such as Tomcat, JBoss, WebSphere and WebLogic. It can run on any operating system that supports JVM 1.6 and it works with a private repository hosted on a common RDBMS (MySQL, PostgreSQL, Oracle, Ingres, HSQL, etc.). The application can be accessed through almost all major web browsers (IE, Firefox, Opera, Safari) and it can work as a simple web application or be included in a standard portal server (Liferay, eXo, WebSphere, etc.)⁷.

SpagoBI Meta

SpagoBI Meta is the environment for data owners who define the semantic layer on which the BI documents will be based.

⁷ The complete list of certified environments is available at:
<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/CertifiedEnvironments>

It is a graphical environment to design and inquire business models (semantic layer) and then release them into one or more SpagoBI Servers.

From a technical point of view, SpagoBI Meta is an eclipse plug-in, so it works as a full-client application related to one or more SpagoBI Servers via web services.

SpagoBI Studio

SpagoBI Studio is the environment for BI developers who create analytical documents such as charts, reports and dashboards.

It is a graphical environment to design, test and deploy analytical documents into one or more SpagoBI Servers, using the business model or even directly inquire data sources.

From a technical point of view, SpagoBI Studio is an eclipse plug-in, so it works as a full-client application related to one or more SpagoBI Servers via web services.

SpagoBI Meta and Studio can even be plugged into the same eclipse local installation.

SpagoBI SDK

SpagoBI SDK is the standard development kit for java developers who need to integrate SpagoBI with their applications.

It is a collection of web services, tag libraries or APIs that allow to relate an external application to SpagoBI Server, allowing to use some SpagoBI documents and functionalities in this environment.

SpagoBI Applications

SpagoBI Applications are ready to use as soon as the administrator installs them.

Some pre-built analytical models can be imported by the administrator into SpagoBI Server and be immediately released for the end-user.

An application normally includes:

- Data model (DWH)
- Data or ETL to load data from custom sources
- Analytical documents (reports, charts, cockpits, etc.)

Usually, they don't include code.

3. SpagoBI Meta

In general terms, *metadata* means ‘data about data’ and in the BI domain it usually refers to data used to describe and enrich the original ones in terms of:

- meaning and structure of corporate data
- where they come from
- how they are organized
- how they are valued
- how they are accessed
- how they are used
- how they can be interpreted
- what they are
- what they mean
- what they are for
- etc.

BI literature usually distinguishes between technical and business metadata, according to their nature, use and preferred users.

Technical metadata are commonly used by analysts and developers to build and manage the DWH, and to ensure system security. Therefore, they mainly contain descriptions of data sources (physical patterns), transformations performed on data, constraints and information about data usage.

Technical metadata are used to abstract from data sources, define a business model independently from data storage, generate automatic documentation, build data lineage and manage changes over data structures.

Business metadata are designed to support end users in accessing and using/interpreting business data in the DWH. They typically include many types of re-classification, categorization, description, semantic information, high-level calculation rules.

Business metadata are used to enrich basic data with additional information, categorize documents and provide additional contents to make their interpretation easier.

SpagoBI Meta is the module of SpagoBI suite specifically focused on technical metadata management, whereas business metadata are managed by SpagoBI Server.

Goals and targets

SpagoBI Meta is the module dedicated to technical metadata management and inquiry. It offers a tool supporting the database reverse engineering process, where users can define the semantic layer to be used during the development of analytical documents, so as to define data selection criteria even without any particular expertise on the data domain and query languages.

The target users of the meta module are:

- data owners and data domain experts, who define one or more models that represent informative islands from a business point of view
- administrators who manage or build shared semantic layers under the analytical documents
- developers who inquire semantic layers to build their analytical documents.

In small projects, a single or a few developers are usually in charge of managing all activities. So they can build and inquire semantic layers, playing *de facto* all roles.

Installation and configuration

In order to use the SpagoBI Meta module, the first step is to check that your environment satisfies the following prerequisites:

- it has a JDK 1.6.x already installed
- it has the JAVA_HOME variable already set
- it has a certified operative system (Windows, Linux, Unix are generally accepted)⁸.

If so, the second step is to download the suitable package from OW2 forge at:

- <http://forge.ow2.org/projects/spagobi> to look at SpagoBI latest releases
- http://forge.ow2.org/project/showfiles.php?group_id=204 to choose the latest release among all SpagoBI releases.

Now choose the right package according to the specific release and distribution matching your environment.



SpagoBI Meta version

Using the same version of both SpagoBI Server and SpagoBI Meta is always recommended, even if not mandatory. This is to be sure that both modules are using the same SDK release.

You can choose between:

- SpagoBI Meta, if you are the data owner or the data domain expert and you work on metadata models only
-

⁸ The list of certified environments mainly depends on the environments supported by Eclipse. Refer to the SpagoBI Meta release notes to know which Eclipse version is included in each released package.

- SpagoBI Studio (which includes the Meta functionalities, too) if you are a developer, the administrator or a one-man-project and you also work on document developments.

SpagoBIPivotEngine-bin-2.0.0_01272009.zip	20,208.3	Any	.zip
SpagoBIQbeEngine-bin-2.0.0_012/2009.zip	24,318.4	Any	.zip
SpagoBITalandEngine-bin-2.0.0_01272009.zip	4,090.5	Any	.zip
SpagoBIWekaEngine-bin-2.0.0_01272009.zip	10,624.4	Any	.zip
SpagoBI 2.0 - Script db		2009-01-27	
mysql-dbscript-2.0.0_01272009.zip	9.7	Any	.zip
SpagoBI 2.0 - Source		2009-01-27	
SpagoBI-src-2.0_01272009.zip	242,408.1	Any	Source .zip
c. SpagoBI Meta 			
SpagoBI 3.3		2011-12-22	
SpagoBIMeta_3.3_linux_20111220.zip	367,352.9	Any	.zip
SpagoBIMeta_3.3_linux64_20111220.zip	367,342.9	Any	.zip
SpagoBIMeta_3.3_Win_20111222.zip	370,759.2	Any	.zip
SpagoBI 3.2 RC		2011-11-03	
SpagoBIMeta_3.2-RC_11042011_Win.zip	321,758.0	Any	.zip
SpagoBI 3.1		2011-07-22	
SpagoBIMeta_3.1_Linux_20110721.zip	315,178.6	Any	.zip
SpagoBIMeta_3.1_Linux64_20110824.zip	315,379.5	Any	.zip
SpagoBIMeta_3.1_Win_20110721.zip	317,571.5	Any	.zip
SpagoBI 3.0		2011-06-16	
SpagoBIMeta_3.0_linux_20110620.zip	311,627.4	Any	.zip
SpagoBIMeta_3.0_win_20110516.zip	323,209.8	Any	.zip
d. SpagoBI Studio 			
SpagoBI 3.3		2011-12-22	
SpagoBISTudio_3.3_Linux_20111222.zip	408,092.8	Any	.zip
SpagoBISTudio_3.3_Linux64_20111222.zip	408,073.2	Any	.zip
SpagoBISTudio_3.3_wini_20120120.zip	411,728.7	Any	.zip
SpagoBI 3.2 RC		2011-11-03	
SpagoBISTudio_3.2-RC_11042011_Win.zip	361,448.1	Any	.zip
SpagoBI 3.1		2011-07-22	
SpagoBISTudio_3.1_Linux_20110721.zip	354,305.7	Any	.zip
SpagoBISTudio_3.1_Linux64_20110824.zip	354,531.3	Any	.zip
SpagoBISTudio_3.1_Win_20110721.zip	355,500.7	Any	.zip

FIGURE 3.1 – Download SpagoBI Meta package from OW2 forge



Eclipse

Both SpagoBI Meta and Studio have been developed as Eclipse plug-ins. Therefore, any limit on supported environments and some technical features and configuration settings derive from Eclipse development model. Please refer to the foundation website at <http://www.eclipse.org/>.

At this point, you just have to unzip the downloaded package under a folder, having a SpagoBIMeta_<version>_<distribution>_<date> subfolder. Here you can find and run a SpagoBIMeta.exe or SpagoBI.sh script to start and select the preferred workspace.

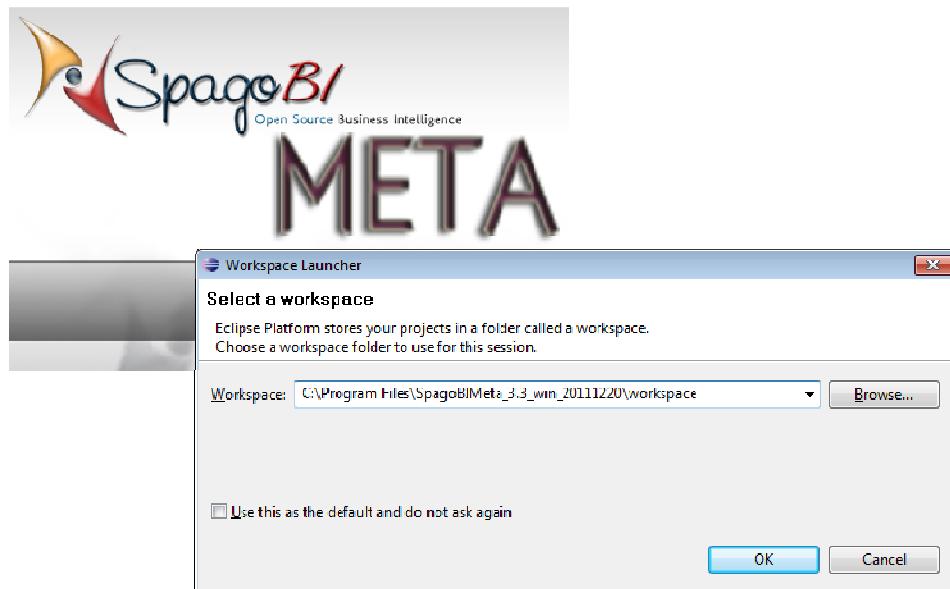


FIGURE 3.2 – Run SpagoBI Meta by selecting the workspace

The selected workspace works as a local repository for all the developed objects (metamodels and data sets). However documents are neither visible nor reusable by other users until they are published on SpagoBI Server.

To get documents published on the server, configure SpagoBI Meta and connect it to SpagoBI Server, which will be the deployment environment to provide end-users with all certified objects. The main steps to configure SpagoBI Meta are:

- select the SpagoBI perspective
- create a SpagoBI project
- set SpagoBI Server connections
- set data sources.

At first execution of SpagoBI Meta, a welcome page appears with references to technical documentation (the wiki⁹, SpagoBI free community tool) and the preferred working perspective.

⁹ <http://wiki.spagobi.org/xwiki/bin/view/Main/>

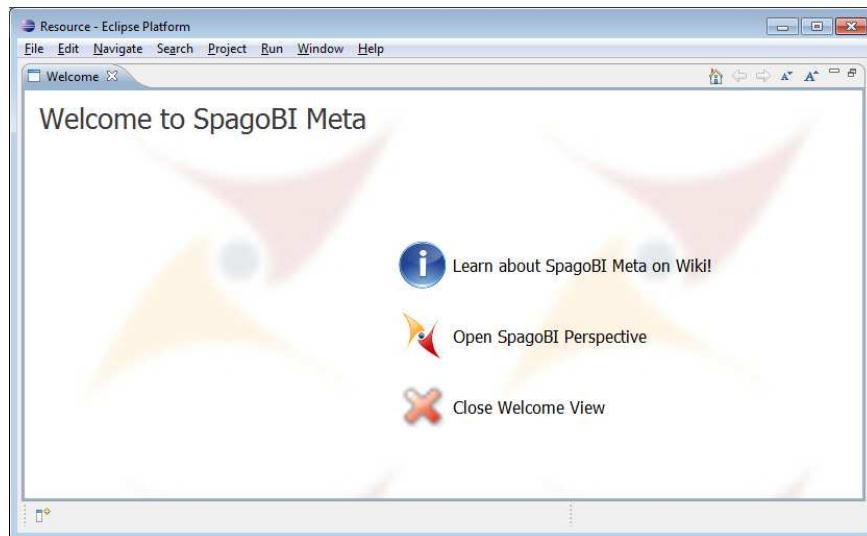


FIGURE 3.3 – SpagoBI Meta welcome page

The first step is to create a SpagoBI project by selecting the SpagoBI icon from the toolbar located at the top of the page.

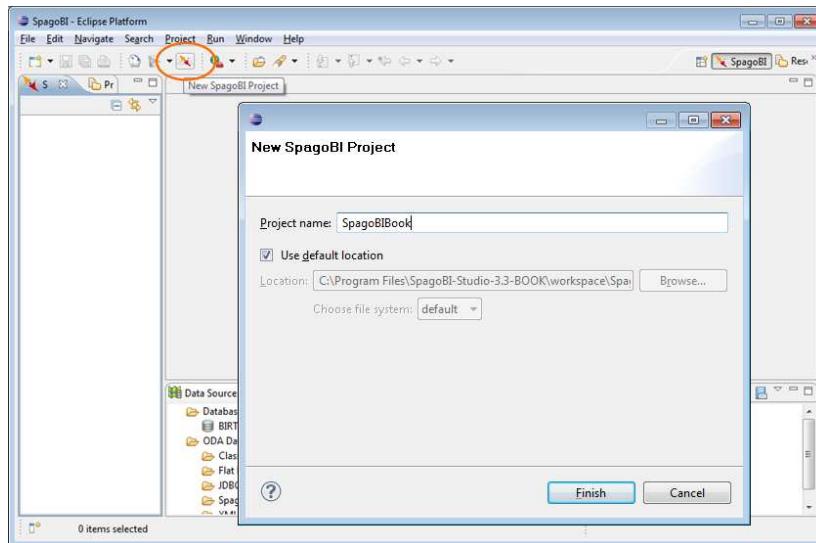


FIGURE 3.4 – Create a SpagoBI project

Once the project has been created, a standard folder tree appears.

The predefined folders are:

- **Business Models:** to host all metadata models created by the user

- **Business Queries:** to host all queries defined by the user over a metadata model
- **Resources:** to define all technical resources to be used in the project (i.e., the reference to SpagoBI Server)
- **Private folders:** to host personal or project documentation that may be useful for the user during his work. This folder can be freely managed by the user.

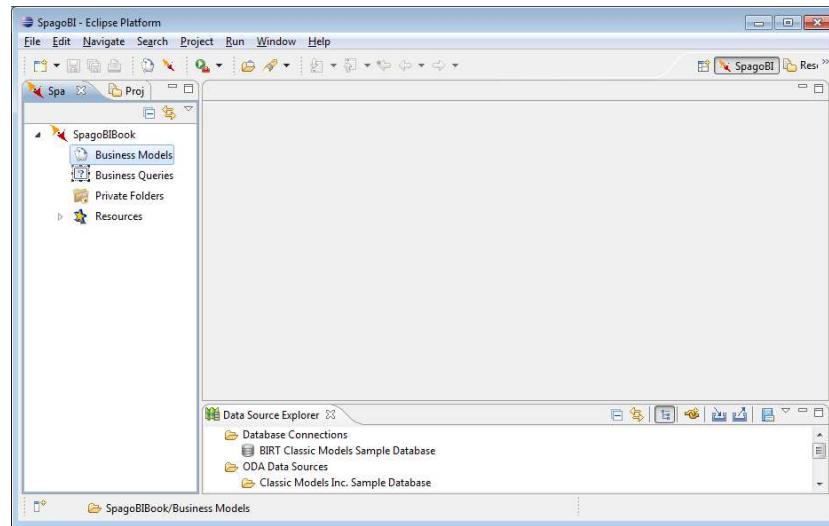


FIGURE 3.5 – SpagoBI Meta folders tree

Now you can configure references to one or more SpagoBI Servers. In other words, each user can work for many projects that can be:

- hosted on different servers
- hosted on the same server with different user accounts.

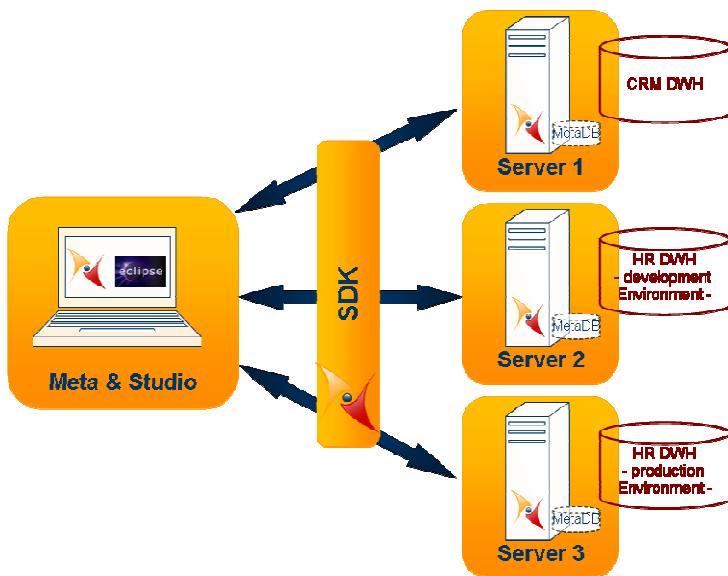


FIGURE 3.6 – SpagoBI Meta towards SpagoBI Servers

To define a server connection, click on the **New Server** item of the **Server** contextual menu. You will be asked for:

- **Server name:** a logical name to identify the server. The name is used on the local workspace only and has no relation with the physical one
- **Url:** the http URL where the server is hosted and reachable.
- **User:** the user who authenticates on SpagoBI Server, setting his access rights in terms of what kind of operations he can do (upload and download a model or a data set) and what parts of the Server repository he can access.
- **Password:** user's password
- **Active:** a flag that indicates the active server. It is particularly useful when the user is working with multiple servers. The active server indicates that every upload and download operation refers to this SpagoBI Server instance.

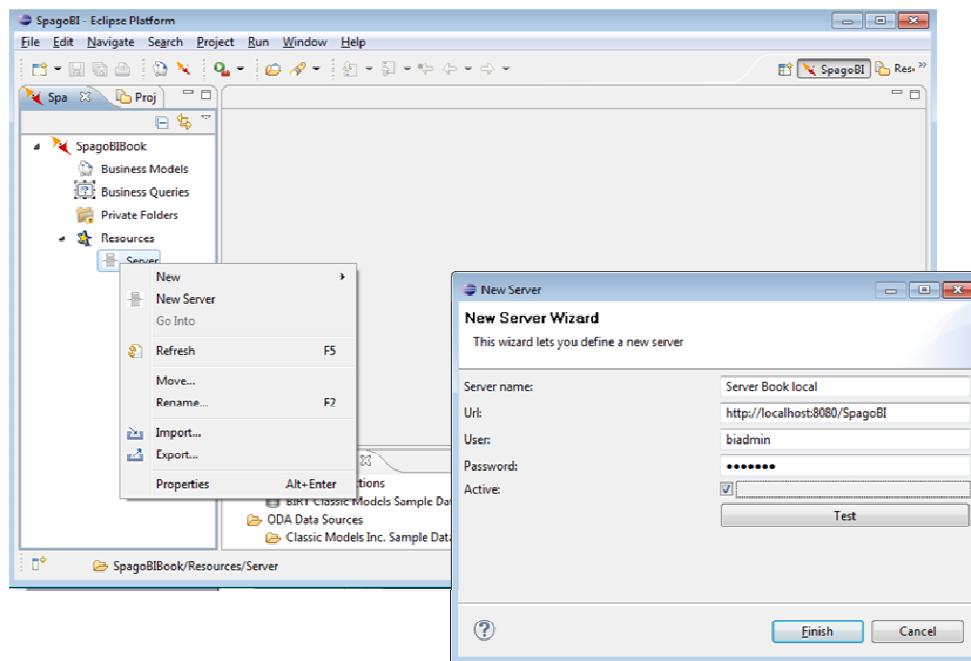
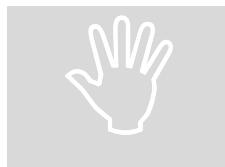


FIGURE 3.7 – Connection to a SpagoBI Server from SpagoBI Meta



Connection to SpagoBI Server

If something in your network configuration has been changed from your first run of SpagoBI Meta, the connection test of SpagoBI Meta to the Server could fail. Most often this problem is due to the proxy settings in your Eclipse environment. If this is not the case, try to run SpagoBI Meta from the command line with the `-clean` option to reset working settings. (SpagoBI.exe `-clean`)

The last step consists in setting data sources. As for servers, it is possible to define one or more data source connections to work with multiple metadata models that refer to different data sources.

At present, SpagoBI manages metamodels based on RDBMS only. Therefore, database connections are the only supported data sources.



DWH

From a technical point of view, nothing prevents you from using an operational database (OLTP) as data source. However this is not recommended. A DWH is preferred to guarantee performances, consistency and stability.

The user can refer to a local database or a remote one, depending on the general project resources. Obviously, if the user refers to a local database, this must be a copy (in terms of schema, not necessarily data) of the one referred by SpagoBI Server on which the metamodel will be deployed.

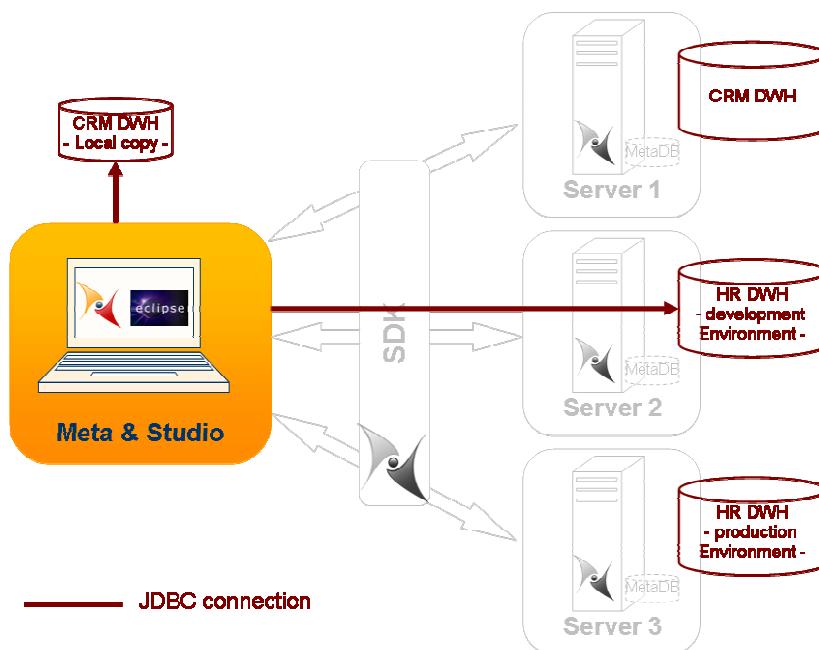


FIGURE 3.8 – Data sources on SpagoBI Meta

The procedure to define a data base connection follows the Eclipse standard procedure:

- Select the **New** item on the contextual menu of the **Data Base Connection** in the **Data Source Explorer** panel
- Create a connection profile choosing the RDBMS type and giving it a name

- Choose the right JDBC driver or add a new one
- Set connection parameters
- Test and save the new connection profile.

In the following, we show step by step how to create a data base connection.

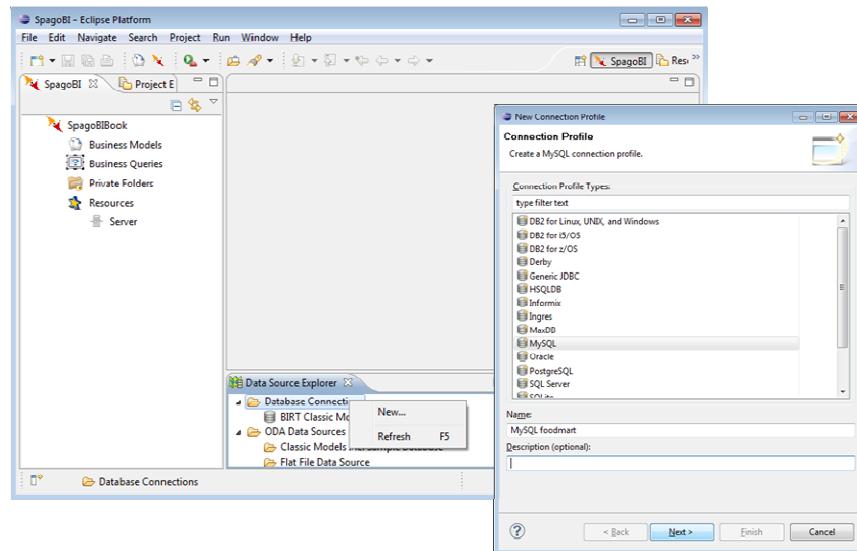


FIGURE 3.9 – Step 1 and 2 : Create a new connection profile selecting the right RDBMS type

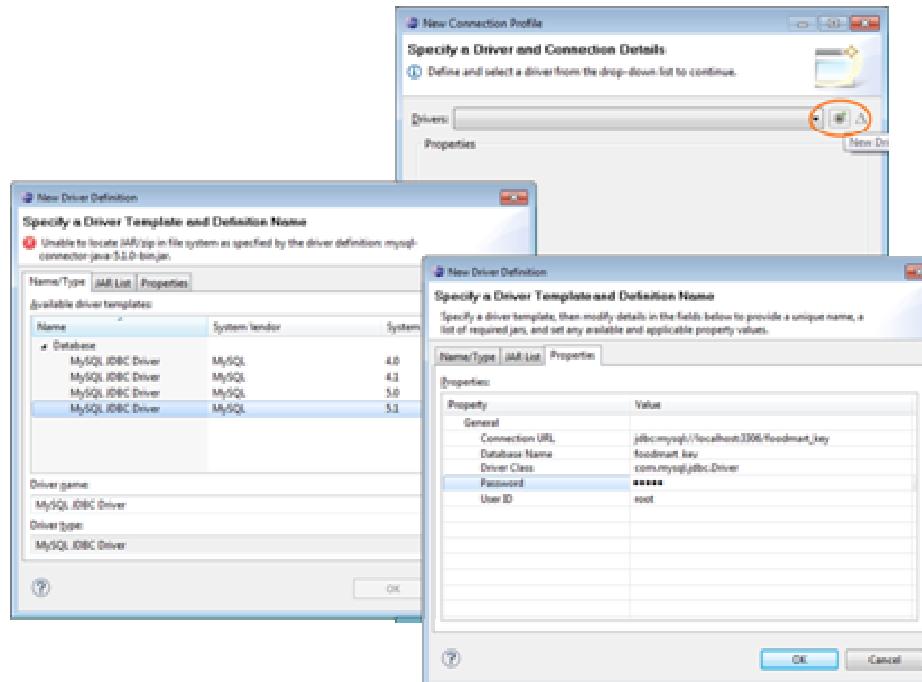


FIGURE 3.10 – Step 3 and 4 : Choose or add the right JDBC driver and set its properties

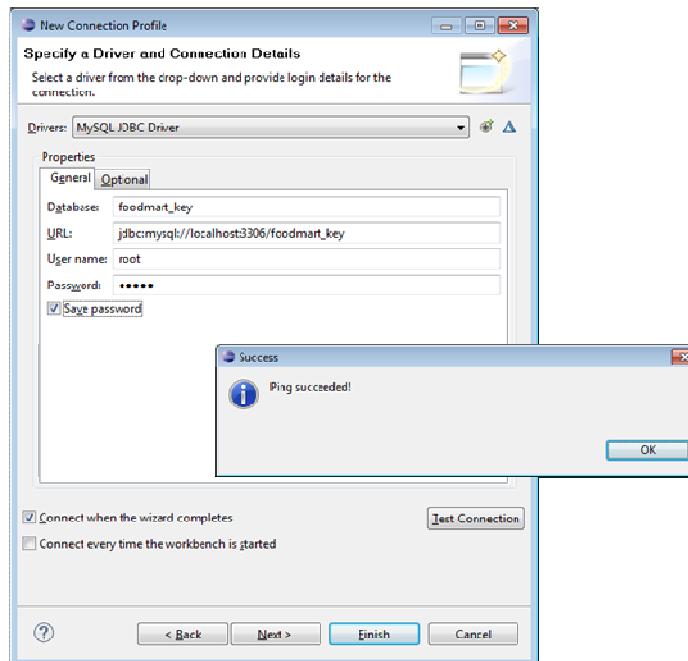


FIGURE 3.11 – Step 5 : Test and save the new connection profile

At this point, SpagoBI Meta is ready to work!

The business model

The *Business Model* (hereafter BM) is the semantic layer built over the data warehouse to redraw the data domain in terms of business elements. This allows to model and access data from the viewpoint of the end-user instead of the DBA.

The definition of a BM does not create new data structures, but produces a mapping between the business view and the technical one. Basically the mapping takes place at two different levels of abstraction, which correspond to the so called *physical model* and the business model itself.

The physical model is generated by choosing some (or all) tables from the data warehouse. At this level we have a representation of the data warehouse structure. The physical model defines the columns belonging to each table, the data type and constraints present within a column, as well as the (primary and foreign) keys and their relationships, if any. In short, the physical model shows data as they are organized in the data warehouse.

The next level of abstraction is the business model. The BM consists of business classes (notice that we do not refer to tables anymore). Classes may have attributes, as well as relationships with other classes. Business classes represent concepts or perspectives on data that do not necessarily correspond to existing structures in the data warehouse. The business model is indeed built by mapping its classes to the physical model.



The business model abstraction

Decoupling business data modeling from database design provides business users with a customized, transparent view on the data of interest, without affecting the physical model or the structure of the underlying database.

Now let us take a closer look at the steps involved in the creation of a BM. To create a new BM, select **New Model** in the contextual menu on the Business Models tree item.

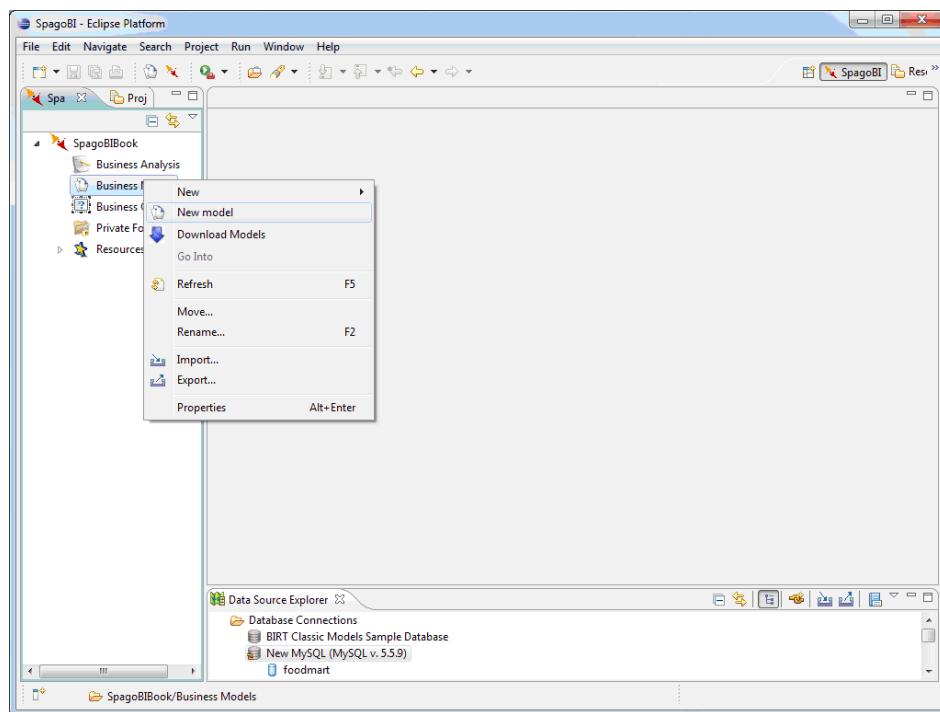


FIGURE 3.12 – Create a new business model

After choosing the type of the desired connection, you will be shown the initial data warehouse schema: this allows the creation of the physical model. At the beginning the left panel shows all tables in the data warehouse, while the right panel is empty. This means that no tables have been imported in the physical model. Tables can be imported by clicking on the arrows icon.

The most common usage is to select all tables by clicking on the double arrow icon . Alternatively, single classes can be selected by clicking on the single arrow icon . Whenever only a subset of tables is imported to the physical model, it is important to check that no pending references to foreign keys are left. Classes can be removed from the physical model by clicking on the back arrow icons.

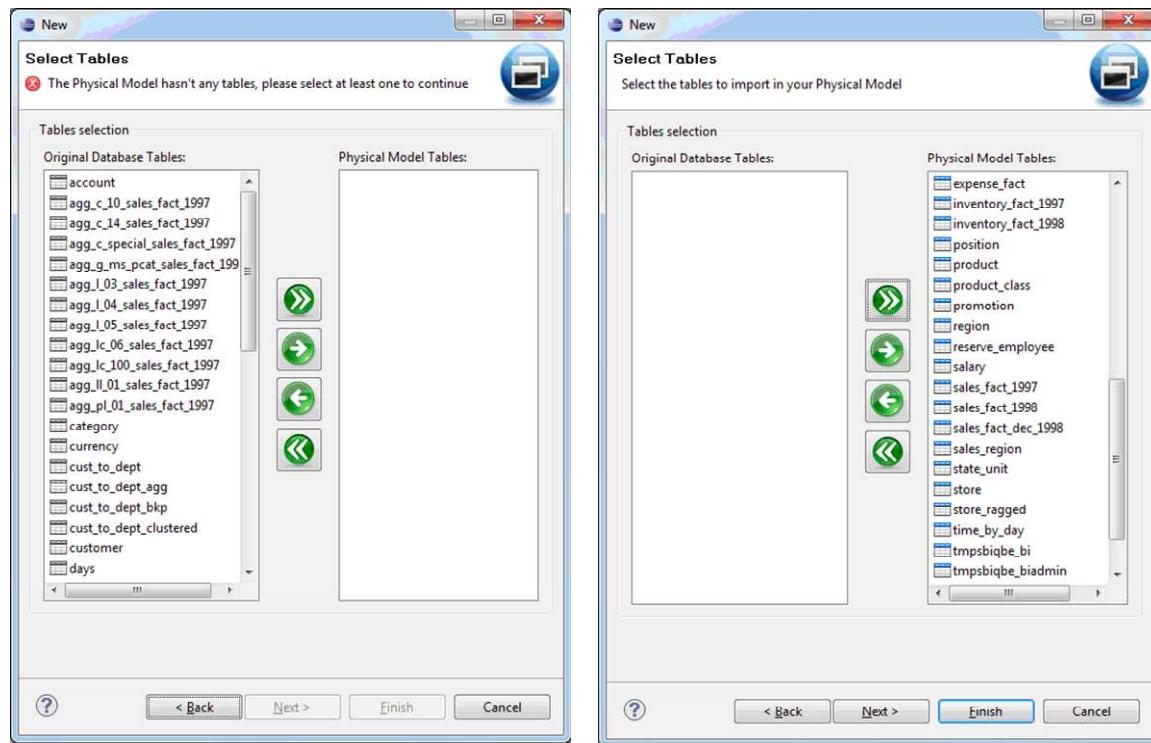


FIGURE 3.13 – Create a physical model importing all tables from the database

Creation of the business model

Once the physical model has been generated, we are ready to create the BM. There are two different ways to create and populate a BM. The first option is to initially select (a subset of) tables from the physical model, then possibly rename them and their attributes, and add relationships between classes.

To do so, select Next in the physical model creation panel. In the following window the left panel shows the tables in the data warehouse, the right panel shows the business classes in the BM. Tables can be mapped onto business classes using the arrows. According to this simple mapping approach, each class basically corresponds to one table in the physical model, although renaming and modification of attributes and relationships is always possible.

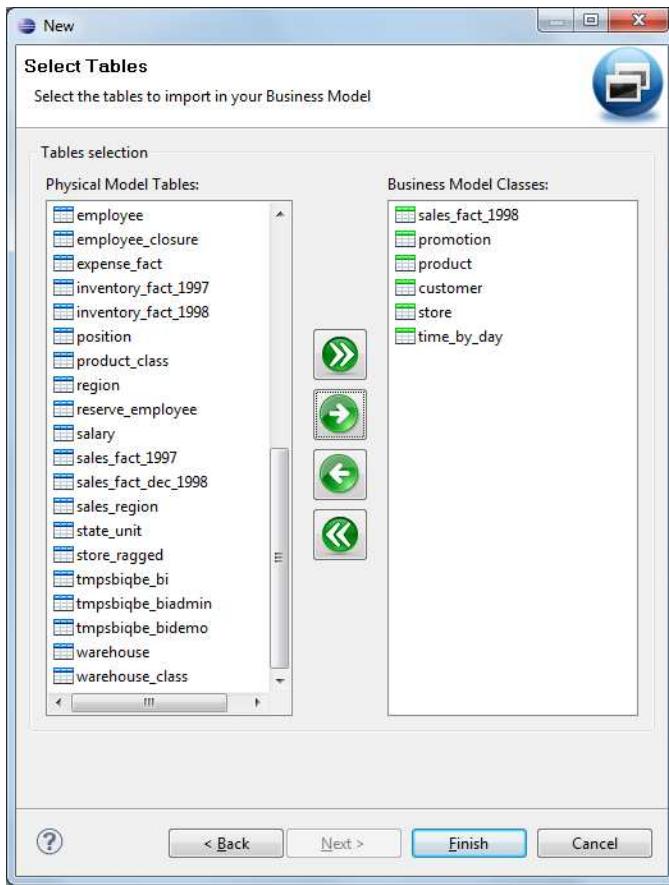


FIGURE 3.14 – Create a business model with an initial selection of tables from the physical model

Another option is to create an empty BM and populate it by creating one business class at a time. To do so, select **Finish** in the physical model creation panel and an empty BM will be created. Classes can then be added to the BM from the main window. Notice that classes can always be added and/or modified, regardless of how the BM was formerly created.

To add a new business class to the BM, you can either directly drag and drop tables from the physical model (on the right panel of the main window) or use the contextual menu by right clicking on the BM. Drag and drop allows to import the selected tables as business classes.

In the contextual menu select either **Add business class** or **Add empty business class**. In the first case you will be prompted with a mapping window to select a table from the physical model and then the columns to be imported as

attributes of the class. In the second case, after choosing a meaningful name for the class, you can start editing the new class. By right clicking on **Edit attributes**, you will be asked to select a table and (some of) its columns.

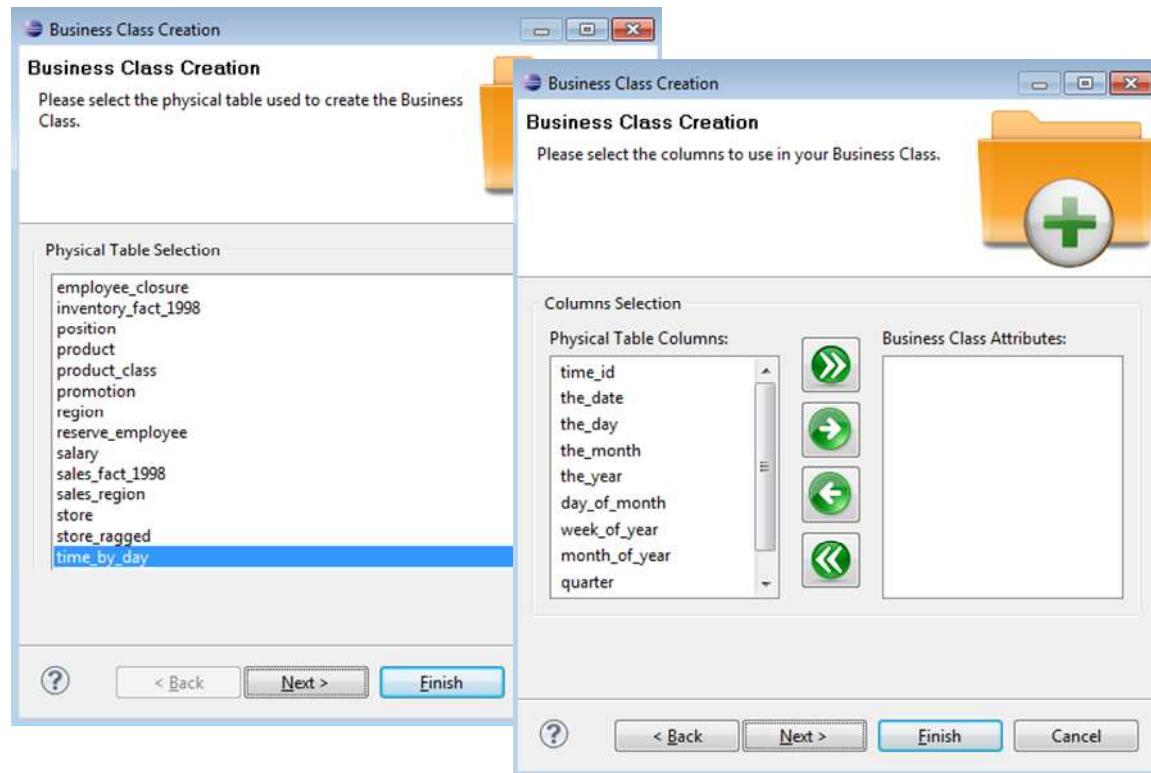


FIGURE3.15 – Add a business class to the business model

A more powerful option is to drag attributes from different tables of the physical model and drop them directly into the business class: this allows the user to create custom classes that make use of data in the physical model, without being bound to a specific table. This way each user can build the business model starting from his needs and representation of the business domain, not from the structure of the data warehouse. Clearly this is a delicate process and should be carried out carefully, in order to obtain a business model that is both meaningful and correct.

Make sure that each class has a primary key. By default primary keys will be imported from the physical model, but in case you make changes or you create a class from scratch, remember to set the attribute(s) that constitute the primary

key. You can recognize primary keys because they are denoted with the symbol



Before starting to inquire the model and deploying it, you can further manipulate the BM. Clicking on a class or attribute's name, you can rename them. The rename operation does not affect the mapping with the corresponding table, but it provides a more readable or expressive name than the one used in the data warehouse.

Using the contextual menu, you can also modify class attributes, new relationships, as well as edit class identifiers. Class properties can also be modified using the **Properties** menu (select **Show Properties View** in the contextual menu). Here you can define whether a selected table is a dimension or a fact within the model, and if a column is defined as an attribute or a measure (and which aggregate function shall be used in queries).

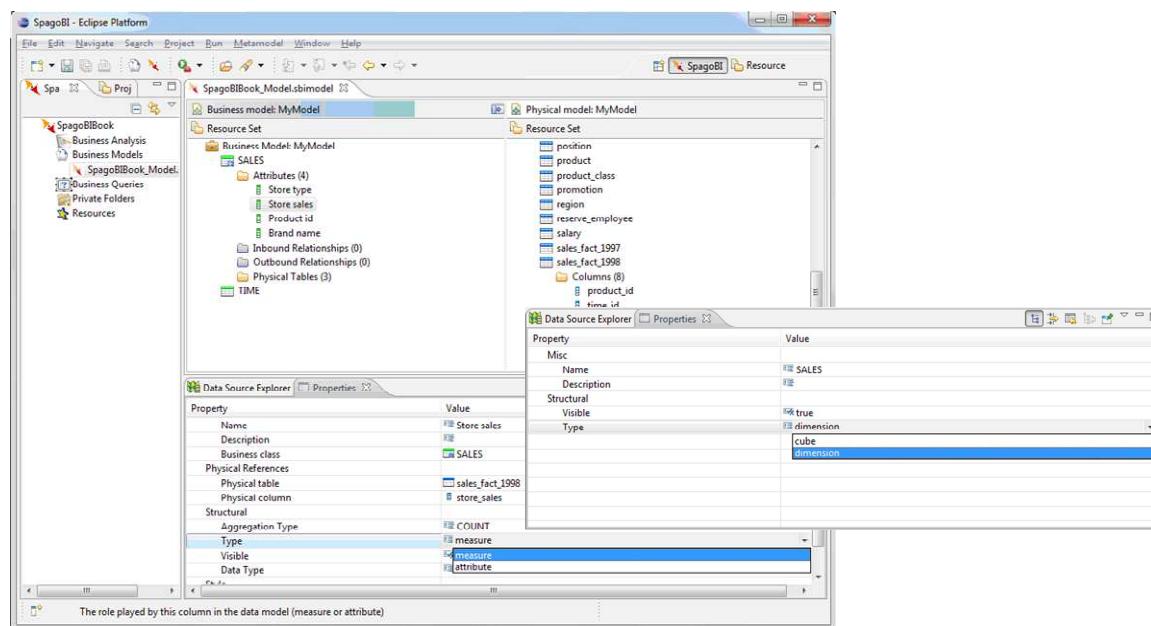


FIGURE 3.16 – Edit attribute and class properties

Finally, in the contextual menu it is possible to create an attribute as a calculated field by selecting the corresponding menu item. Adding calculated fields allows the creation of class attributes (or measures) that were not originally defined in the physical model. This enhances the expressivity of the

business model and its transparency, compared to the underlying data warehouse.

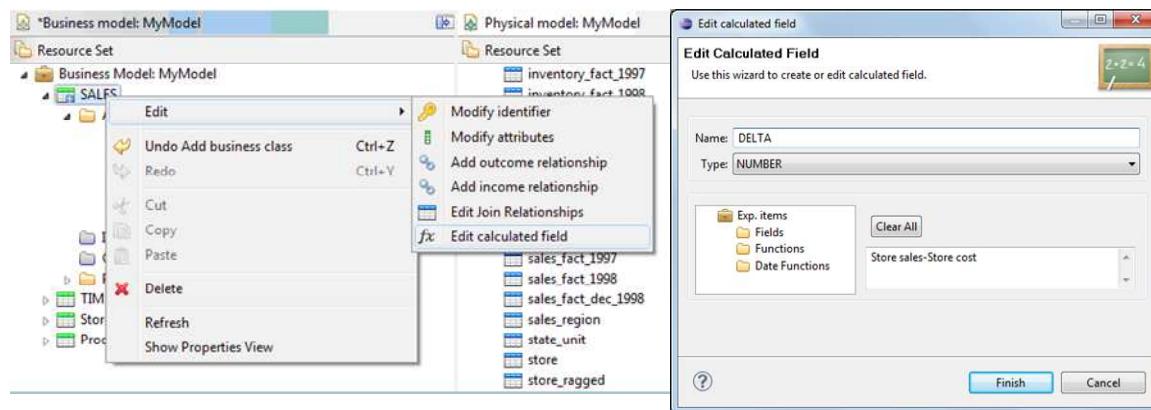


FIGURE 3.17 – Create a calculated field in the business model

Defining relations between classes

An important aspect of defining a BM concerns the definition of relations. Relations basically set connections between different business classes, or some of their attributes. These connections represent join criteria between data that are automatically applied whenever a query is executed on the BM, without need to specify them each time.

The simplest type of relation between classes corresponds to the definition of a foreign key between the corresponding tables in the data warehouse. Foreign keys defined at the DWH level are automatically imported into the BM. In the main window, these relations are identified with the prefix **FK_**, as shown in FIGURE 3.18.

In addition to foreign keys, further relations can be defined at the BM level. These relations do not affect the physical model but they are stored in the BM only. Whenever a query is executed on the BM, they will be taken into account as additional join constraints between the corresponding tables and columns.

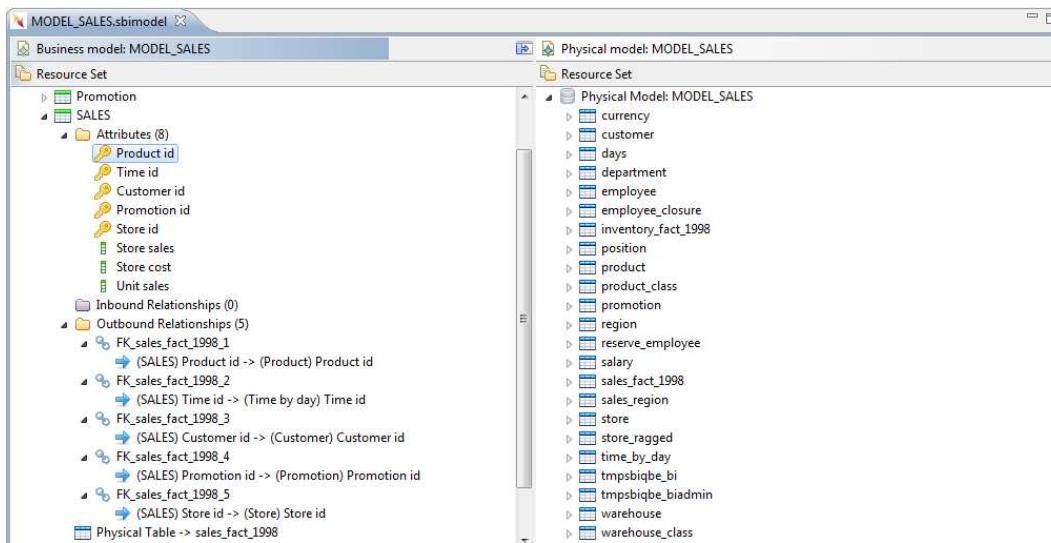


FIGURE 3.18 - Foreign keys in SpagoBI meta model



Relations in the Business Model

Relations defined at the BM level are mapped onto join conditions every time the model is queried. Be careful defining relations, to ensure that query results are always consistent.

To add a relation, right click on the BM instance: in the Edit menu, you can either choose **Add inbound relationship** or **Add outbound relationship**. The difference between these two types resides in the direction of the relation with respect to the destination table. The concept of directed relation is similar to that of a foreign key in a relational database as it represents a referential constraint within the BM. Roughly speaking, if a relation is outbound for class A and inbound for class B, B acts as the referenced class, while A acts as the referencing class. If the physical model is modelled as DWH, classes representing measures will generally have outbound relations that are inbound for the corresponding dimensions.

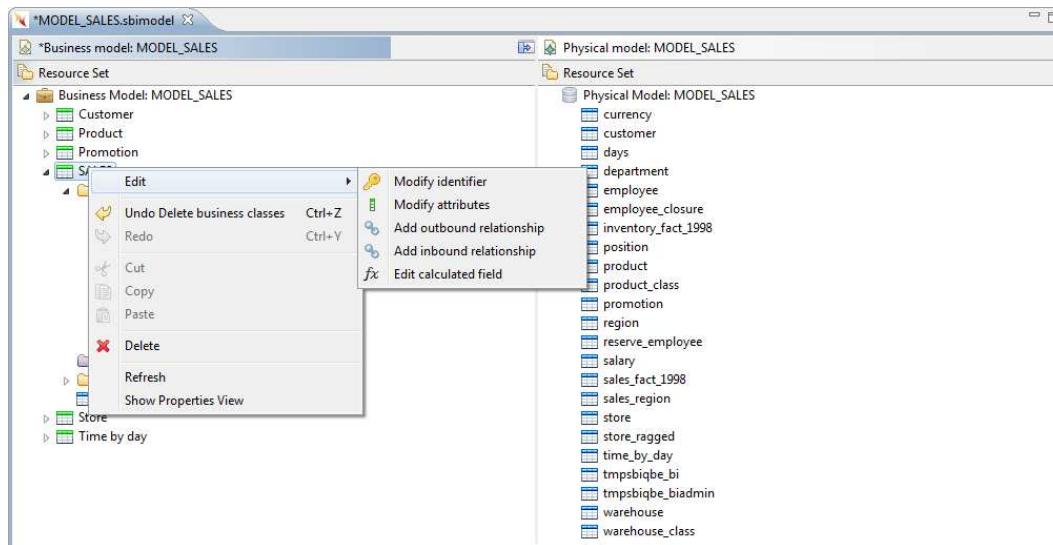


FIGURE 3.19 – Add an outbound relation to a business class

Relations defined in the BM are identified with the BR_ prefix, as shown in FIGURE 3.20.

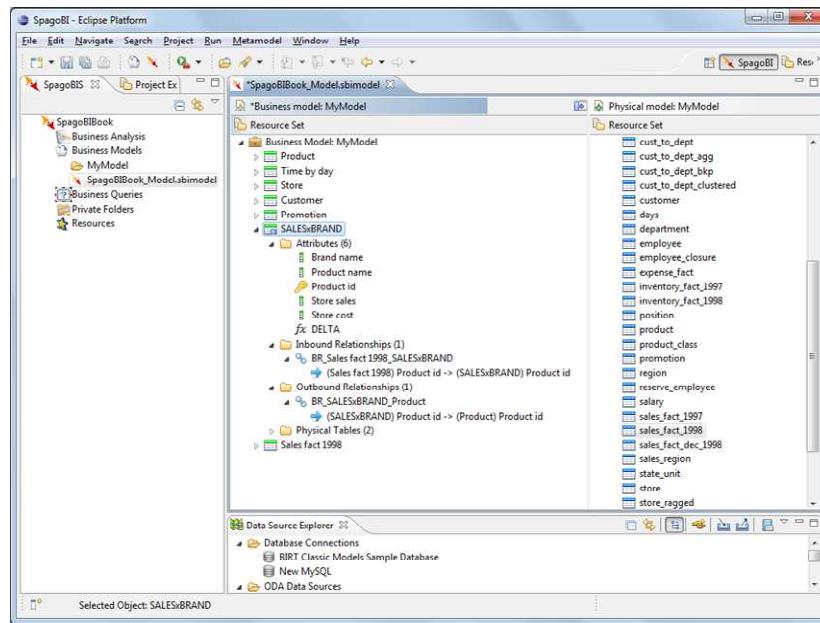


FIGURE 3.20 – Business classes with BM relations

Inquiring the business model

So far we have seen how to create and modify a BM. In this section we will show how to query it.

Instead of writing and executing SQL queries, the business user can create queries at the BM level via an advanced graphical editor. To open the editor, right click on the BM and select **Create > Query**.

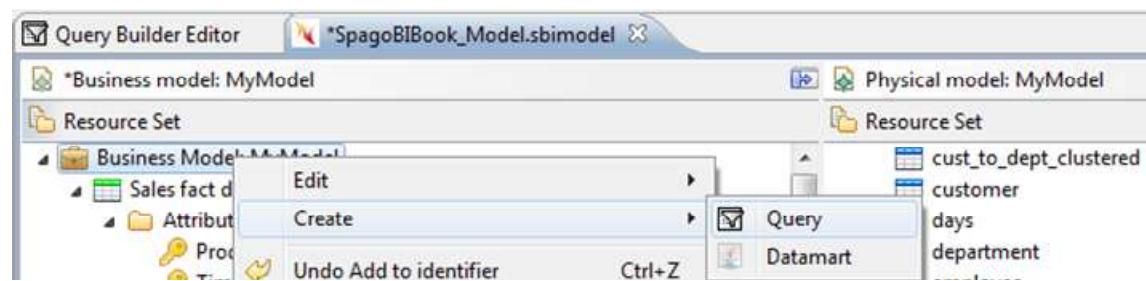


FIGURE 3.21 – Create a query on a business model

A graphical editor will open to start building queries (see FIGURE 3.21). On the left side the BM is shown as a tree; on the right the query editor allows the definition of queries. Let us first have a look at the BM tree, which is the result of the modelling process described above.

Classes can be either defined as cubes (that is, facts in a DWH model) or dimensions. Each class has a sub-tree, where its attributes, measures and calculated fields are listed. The sub-tree also shows classes sharing a relation with the root class: in particular, the referencing class (having the relation as outbound) will list the referenced class (having the relation as inbound) in its tree. Note that any class defined in the BM will also be listed by itself, regardless of any defined relation. In the following we will see how this affects query execution.

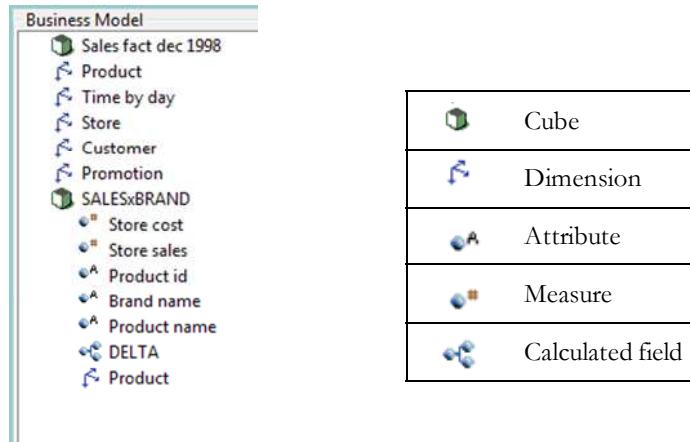


FIGURE 3.22 – Tree representation of the business model, with symbols

The query editor consists of three main sections: one for selection and two for constraints (WHERE and HAVING). Intuitively, a query can be specified by adding fields to the respective sections.

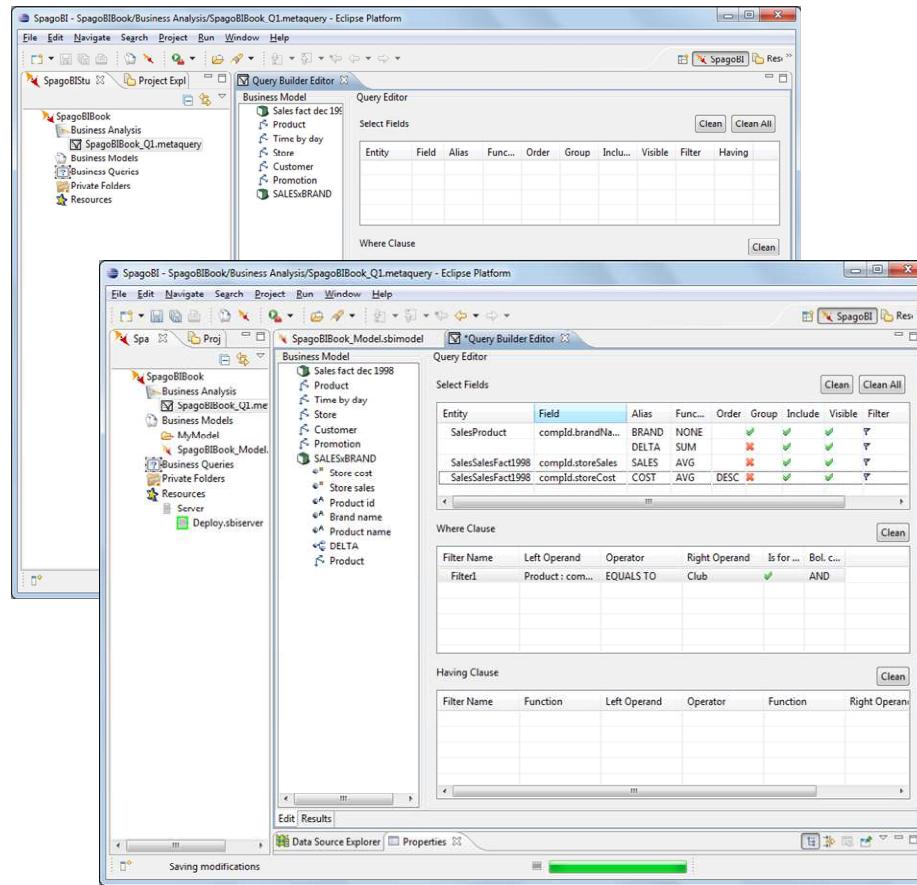


FIGURE 3.23 – SpagoBI Meta graphical query editor

Fields are populated via drag & drop or double click on the left side tree, which shows the structure of the BM. Each field can be renamed by giving a value to the alias column. You can define aggregation functions by clicking on the field of the corresponding column. Similarly, grouping and ordering can be set by ticking the corresponding column.

Clicking on the filter/having symbol, a new row will be created in the Where /Having section, respectively. These rows are pre-initialized with the selected field. Alternatively, a row can be created via drag & drop from the left tree.

Filters are composed of a left operand, a right operand and an operator. Operators include common mathematical/logical operators, as well as value assignment. Both operands can be processed using the **Function** column, for

example by aggregating them. Filters can also be combined in AND or OR condition.



Query on multiple business classes

If a query involves more than one business class, it is important to correctly insert them into the query editor. To set a join relation between two classes, the referenced class must be selected from within the tree of the referencing class.

All classes referenced by another class via a relation are listed as sub-entities in the tree representing the business class (on the left panel). If the user drags and drops a class from within another class (that is, in its sub-tree), SpagoBI Meta will automatically resolve and apply the join condition defined in the relation.

If otherwise two classes are dragged and dropped on the editor area, without being related one with the other, the query will result in the Cartesian product of the two. This has a negative impact on the performance of the query, particularly if the user meant to set a join relation between some attributes of those classes. In other words, to set a join relation between two business classes in a query, the user must always select the referenced class from within the tree of the referencing class.

The results of the query execution are visible in the **Results** tab, at the bottom of the page. By clicking on the **Edit** tab the user can return to the query editor and modify the query.

Once the query is done, remember to save it by clicking on the Save icon in the menu bar. Now you can deploy it on SpagoBI Server.

To deploy the query, right click on the query item in the left panel. In the deploy panel, assign a label, a name and a description to all datasets in SpagoBI. Then select the correct data source among those available on the server. If you have defined a local database in the Meta environment, be careful to select the data source with the same schema on the Server.

When a query is deployed on the server, it will be saved as a dataset in the server and it will be accessible under the **Resources > Data set** menu.



Datasets

For a detailed explanation of dataset creation, storage and access, please refer to section Data sets, in chapter 5 – SpagoBI Server.

A dataset built from a query on a business model is always deployed as a QbE dataset. This means that it can be opened and edited on SpagoBI Server using the QbE Engine.



QbE

See section Free Inquiry, in Chapter 6 – Analytical Engines, to learn how to execute and modify the dataset you just created with SpagoBI Meta, as well as how to build ad-hoc reports on top of it.

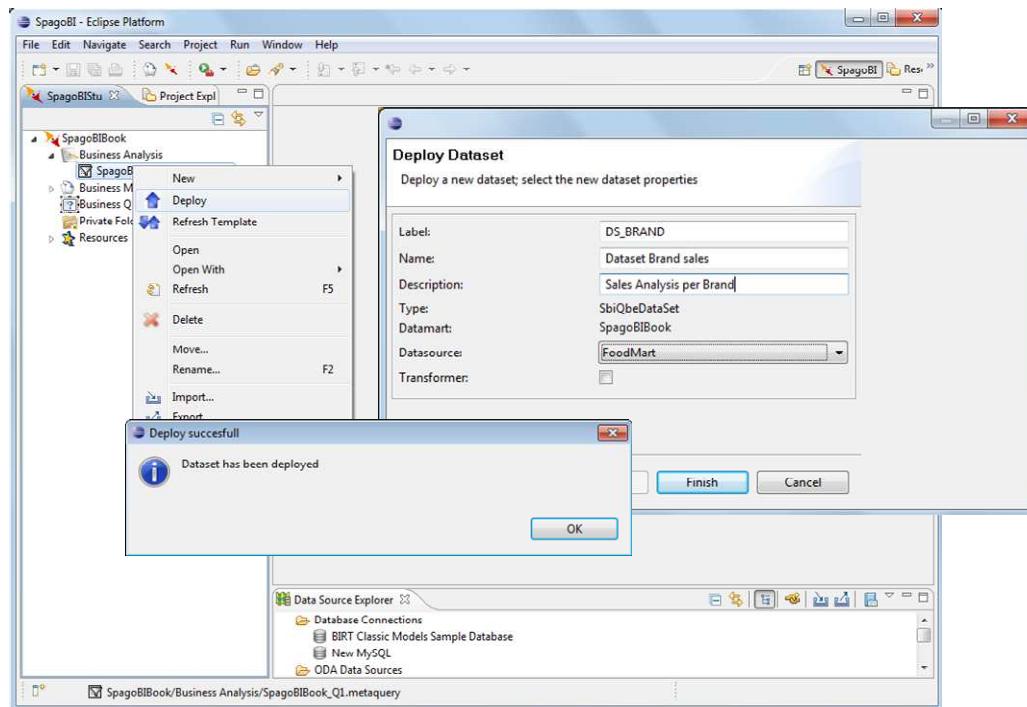


FIGURE 3.24 – Deployment of a BM query into SpagoBI Server

Deploy and use the business model

Besides the query, the BM itself can be deployed on SpagoBI Server as well. This will allow users to view the model, build high level queries on it using the QbE Engine (QbE datasets) and finally develop documents using the defined datasets.

The BM can be deployed in a similar way to the query. Just right click on the model item and choose **Upload Datamart and Model on Server**. This operation will trigger the creation of a new datamart in the server catalog. For each business model deployed on server, a new package will be stored at `/resources/qbe/datamarts/<your_datamart_name>` under the main installation folder. The name of the subfolder containing the package is the label of the metamodel.

Furthermore, a datamart QbE document will be created within the personal folder of the user deploying the BM. This way the model can be edited and queried on the server using the QbE Engine.

By default the BM will be uploaded into the user's personal folder (for example, administrator). However, this may not be appropriate when the BM needs to be accessible to other users, for instance BI developers who build reports based on the metamodel but are not allowed to modify the metamodel itself. In this case, the document should be made accessible to other users once deployed on the Server.

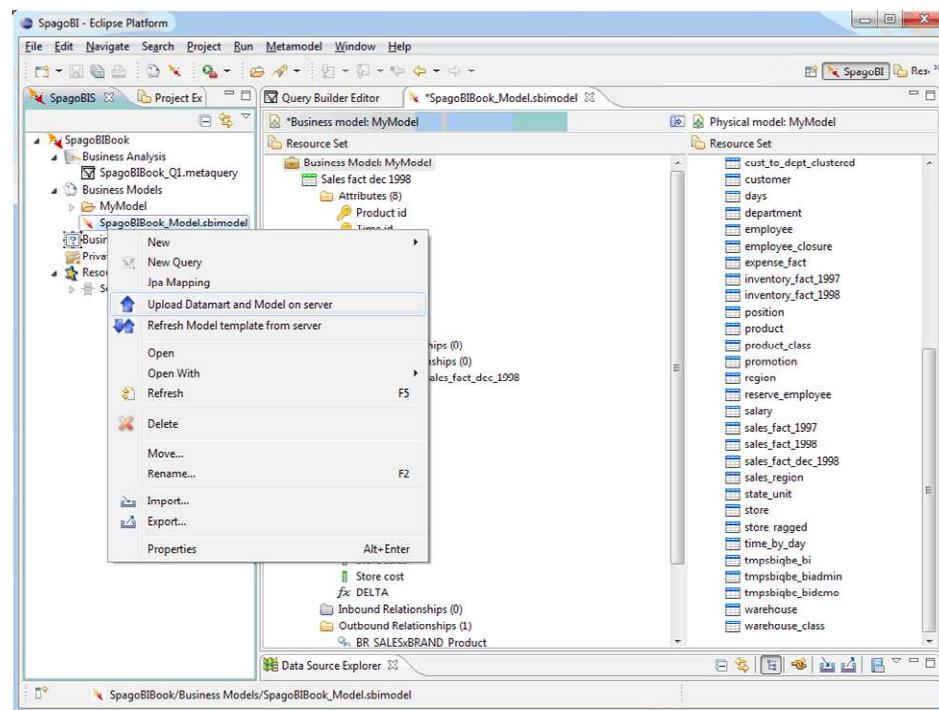


FIGURE 3.25 – Upload the business model onto SpagoBI Server

SpagoBI allows to manage the visibility on documents, through the visibility options on containing folders. To this end, the QbE document of the deployed BM should be copied in a folder that is accessible to the selected users. To do so, open the detail view of the document and check the folder where you wish to copy the document.

Other users can download the Model on their local Meta environment by simply selecting **Download Models** in the contextual menu (see FIGURE 3.12).

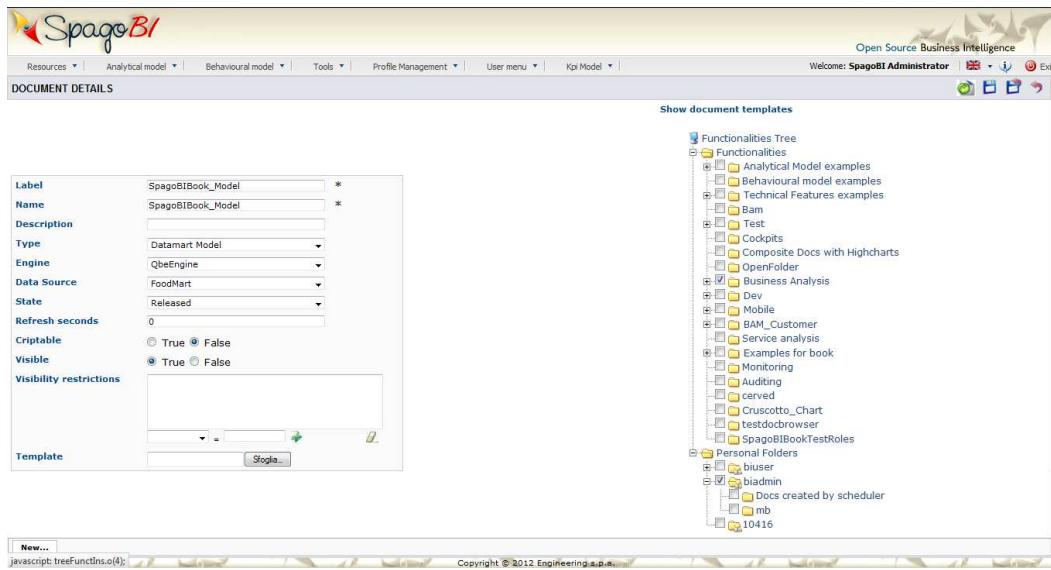


FIGURE 3.26 - Save the BM at different locations in the functionality tree



Setting visibility on documents

To fully understand how SpagoBI manages visibility and rights on documents, please refer to section Behavioral Model, chapter 5 – SpagoBI Server.

Once deployed, the BM can be queried both using the web interface (Server) or the design environment (Meta). Nevertheless, it can be edited from the Meta environment *only*. Any change to the model shall be managed by editing the metamodel locally and re-deploying it on the server. Changes are locally stored and then uploaded into the corresponding document on the Server.

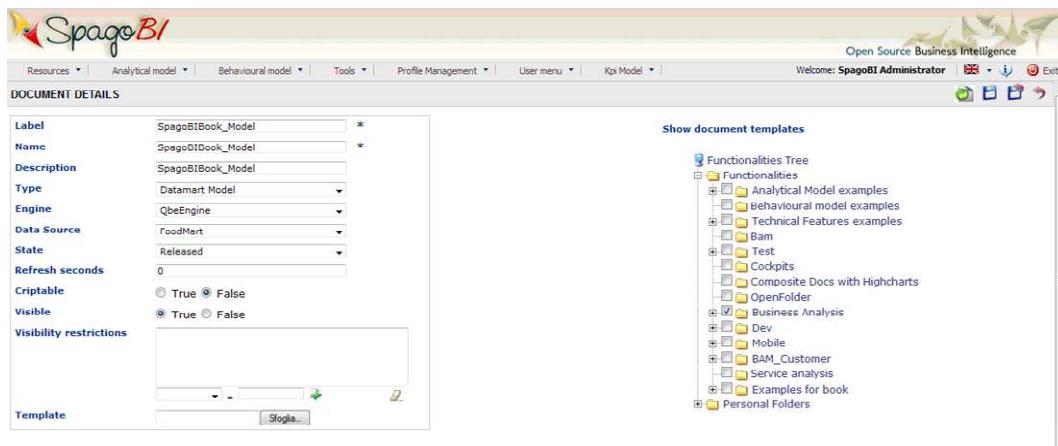


FIGURE 3.27 – Extend visibility on documents describing the BM to other users

This requires special attention when the BM has been deployed as a QbE document and copied in multiple folders (to enhance its visibility). The re-deployment of a model will only affect the document that was originally bound to the local model, either via **Upload Model** or **Download Model**. Other instances of the same BM (QbE documents on the server) will not be affected by these changes.

To manually edit and re-deploy a business model that cannot be reached using the Meta deployment mechanism (for example because it is in a different folder), you can directly upload the **.Shimodel* file stored in the local environment.



Business Model Version Control

At present, SpagoBI does not support any automated mechanisms to alert users about the possible impact of a BM modification on existing datasets and/or documents. Be careful when editing and re-deploying a BM.

4. SpagoBI Studio

Each SpagoBI document (e.g., report, olap, chart, cockpit, etc.), has its own technical metadata stored in SpagoBI internal repository. The most relevant technical metadata describing document structure, content and behaviour are:

- Template, which defines the document layout
- Data set, which defines how data of each document should be read
- Analytical drivers, which allow to define the drivers used by the document at runtime and to produce the right form for parameters.



SpagoBI Server

Look at chapter 5 – SpagoBI Server to learn more on the concepts of template, analytical document and driver.

SpagoBI Studio supports BI developers drawing the template for each analytical document by a graphical interface and simple wizards. Each document type has its own designer and manages the relation with data sources and data sets.



Designers

Graphical designers are currently available for some SpagoBI documents only. Other ones will follow over time (i.e., real time console, mobile and GIS). All designers share common tasks and a general approach.

Goals and targets

SpagoBI Studio is an eclipse-based integrated development environment (IDE) providing BI developers with all the needed functionalities to design, develop, test, deploy and maintain SpagoBI analytical documents. As said above, each document is mainly associated to a template describing its layout and a data set defining how data will fill it.

SpagoBI Studio assists the developer in writing these templates and/or data sets by means of a graphical user interface and of easy-to-use wizards.

These data sets are based on specific business models created through SpagoBI Meta, possibly integrated within SpagoBI Studio.

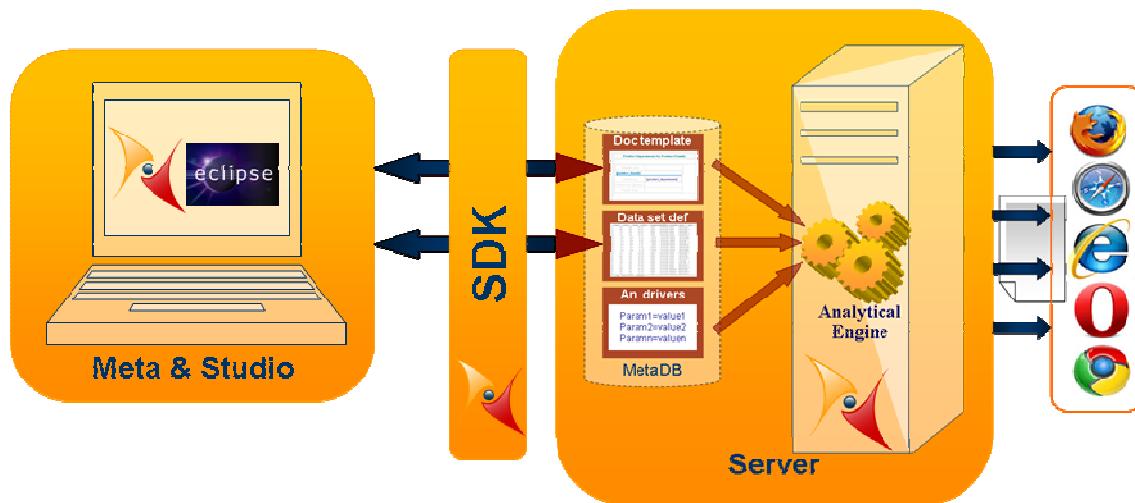


FIGURE 4.1 – Relation between SpagoBI Studio and SpagoBI Server

This module is not mandatory because an expert developer can even work directly on the server, managing documents and data sets by hand, thanks to the web interface for administrators and developers. Usually, this way is faster when only small changes are required on already released documents, whereas the Studio is particularly useful when a developer works on new documents.

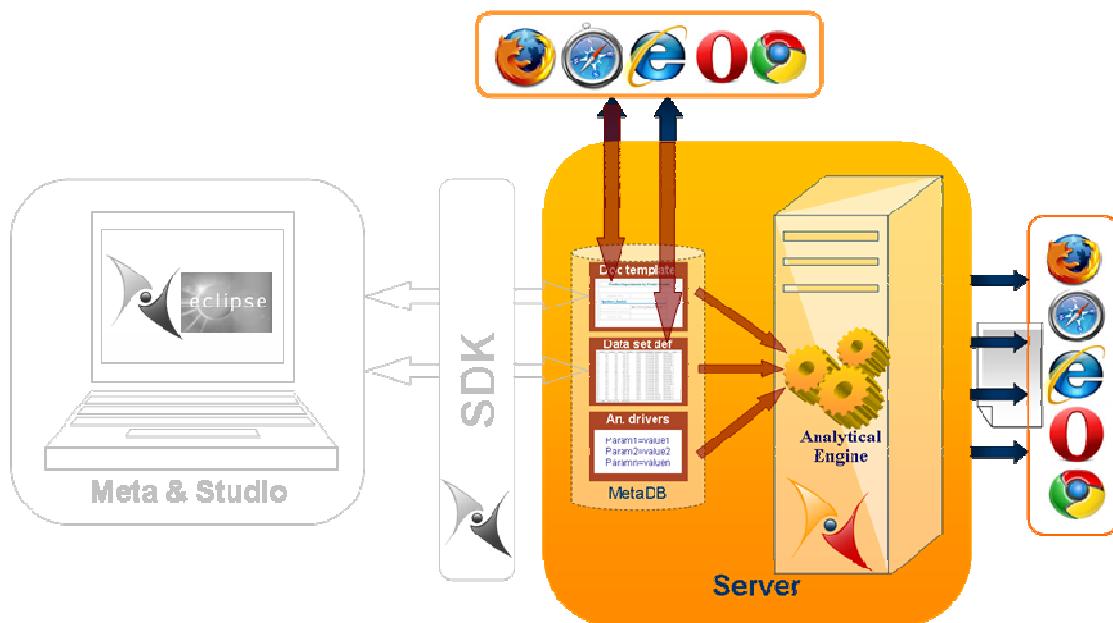


FIGURE 4.2 – Manual management of document templates and data sets

The target users of the Studio module are:

- BI developers, who define analytical documents and data sets to be released onto a remote SpagoBI Server
- administrators who define or update analytical documents and data sets.

In other words, SpagoBI Studio covers the development processes of more technical documents. On the other hand, high-level documents are created directly through SpagoBI Server, where a power user can access graphical designers without need to use the Studio, which requires more technical skills to manage the installation and configuration process.

Specifically, the documents which are not covered by SpagoBI Studio because oriented to power users are:

- Worksheet for ad-hoc reporting
- KPI
- Smart Filter, for driven data selection
- Analytical Dossier, for collaboration.

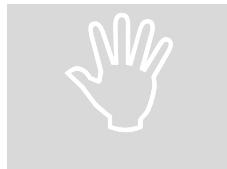
The following table summarizes where each document type can be built. It also shows the already available designers and the expected ones.

Document type	Engine	Designers	State
Report	SpagoBIJasperReportEngine SpagoBIBirtReportEngine SpagoBIAccessibleReportEngine	SpagoBI Studio SpagoBI Studio SpagoBI Studio	Available Available To be done
Chart	SpagoBIJFreeChartEngine HighChart (SpagoBIJFreeChartEngine) SpagoBIJSChartsEngine	SpagoBI Studio SpagoBI Studio SpagoBI Studio	Available Available To be done
OLAP	SpagoBIJPivotEngine /Mondrian SpagoBIJPaloEngine /Mondrian SpagoBIJPXMLAEngine	SpagoBI Meta/Server SpagoBI Meta/Server SpagoBI Meta/Server	Partially available To be done To be done
Interactive cockpit	SpagoBICompositeDocEngine	SpagoBI Studio	Available
KPI	SpagoBIKpiEngine	SpagoBI Server	Available To be done
Data Mining	SpagoBIWekaEngine	SpagoBI Studio	To be done
Free Inquiry - Smart Filter	SpagoBISmartFilterEngine	SpagoBI Server	Available
Free inquiry	SpagoBIQbeEngine	SpagoBI Meta/Server	Available
Ad-hoc reporting	SpagoBIWorksheetEngine	SpagoBI Server	Available
Location Intelligence	SpagoBIGeoEngine SpagoBIGisEngine	SpagoBI Studio SpagoBI Studio	Available To be done
Real-time Console	SpagoBIConsoleEngine	SpagoBI Studio	To be done
DashBoard	SpagoBIDashboardEngine	SpagoBI Studio	Available

Mobile	SpagoBITableMobileEngine SpagoBIChartMobileEngine SpagoBICockpitMobileEngine	SpagoBI Studio SpagoBI Studio SpagoBI Studio	To be done To be done To be done

FIGURE 4.3 – SpagoBI features, engines and designers

Installation and configuration



Studio and Meta

The installation procedure of SpagoBI Studio is similar to the one used for the Meta module. Don't skip this paragraph even if you've read the one concerning the installation and configuration of SpagoBI Meta, because some little but relevant differences occur.

To use SpagoBI Studio, first check that your environment satisfies the following prerequisites:

- it has a JDK 1.6.x already installed
- it has the JAVA_HOME variable already set
- it has a certified operative system (Windows, Linux, Unix are generally accepted)¹⁰
- it has the iReport plugin already installed, if you wish to develop JasperReport documents¹¹.

If so, the second step is to download the suitable package from OW2 forge at:

¹⁰ The list of certified environments mainly depends on the ones supported by Eclipse. Please, refer to the SpagoBI Studio release notes to find out the Eclipse version included in each released package.

¹¹ Refer to SpagoBI Studio release notes to find out the required iReport version for each released package.

- <http://forge.ow2.org/projects/spagobi> to look at SpagoBI latest releases
- http://forge.ow2.org/project/showfiles.php?group_id=204 to choose the latest release among all SpagoBI releases.

Now choose the right package according to the specific release and distribution matching your environment.

SpagoBIJPivotEngine-bin-2.0.0_01272009.zip	20,208.3	Any	.zip
SpagoBIObeEngine-bin-2.0.0_01272009.zip	24,318.4	Any	.zip
SpagoBITalendEngine-bin-2.0.0_01272009.zip	4,090.5	Any	.zip
SpagoDIWekaEngine-bin-2.0.0_01272009.zip	10,624.4	Any	.zip
SpagoBI 2.0 - Script db		2009-01-27	
mysql-dbscript-2.0.0_01272009.zip	9.7	Any	.zip
SpagoBI 2.0 - Source		2009-01-27	
SpagoBI-src-2.0_01272009.zip	242,408.1	Any	Source .zip
c. SpagoBI Meta			
SpagoBI 3.3		2011-12-22	
SpagoBIMeta_3.3_linux_20111220.zip	367,362.9	Any	.zip
SpagoBIMeta_3.3_linux64_20111220.zip	367,342.9	Any	.zip
SpagoBIMeta_3.3_Win_20111222.zip	370,769.2	Any	.zip
SpagoBI 3.2 RC		2011-11-03	
SpagoBIMeta_3.2-RC_11042011_Win.zip	321,768.0	Any	.zip
SpagoBI 3.1		2011-07-22	
SpagoBIMeta_3.1_Linux_20110721.zip	315,178.6	Any	.zip
SpagoBIMeta_3.1_Linux64_20110024.zip	315,279.5	Any	.zip
SpagoBIMeta_3.1_Win_20110721.zip	317,571.5	Any	.zip
SpagoBI 3.0		2011-06-16	
SpagoBIMeta_3.0_linux_20110620.zip	311,627.4	Any	.zip
SpagoBIMeta_3.0_win_20110616.zip	323,209.8	Any	.zip
d. SpagoBI Studio			
SpagoBI 3.3		2011-12-22	
SpagoBISTudio_3.3_Linux_20111222.zip	400,092.0	Any	.zip
SpagoBISTudio_3.3_Linux64_20111222.zip	408,073.2	Any	.zip
SpagoBISTudio_3.3_Win_20120120.zip	411,728.7	Any	.zip
SpagoBI 3.2 RC		2011-11-03	
SpagoBISTudio_3.2-RC_11042011_Win.zip	361,448.1	Any	.zip
SpagoBI 3.1		2011-07-22	
SpagoBISTudio_3.1_Linux_20110721.zip	354,305.7	Any	.zip
SpagoBISTudio_3.1_Linux64_20110824.zip	354,531.3	Any	.zip
SpagoBISTudio_3.1_Win_20110721.zip	355,600.7	Any	.zip

FIGURE 4.4 – Download SpagoBI Studio package from OW2 forge



SpagoBI Studio version

Using the same version of both SpagoBI Server and SpagoBI Studio is always recommended, even if not mandatory. This is to be sure that both modules are using the same SDK release and the same library version for the development of analytical documents.



Eclipse

Both SpagoBI Meta and Studio have been developed as Eclipse plug-ins. Therefore, any limit on supported environments and some technical features and configuration settings derive from Eclipse development model. Please refer to the foundation website at <http://www.eclipse.org/>.

At this point, you just have to unzip the downloaded package under a folder, having a SpagoBISTudio_<version>_<distribution>_<date> subfolder. Here you can find and run a SpagoBISTudio.exe or SpagoBI.sh script to start and select the preferred workspace.



FIGURE 4.5 – Run SpagoBI Studio and select the workspace

The selected workspace works as a local repository for all the developed objects (reports, chart, cockpits, etc.). However documents are neither visible nor reusable by other users until they are published on SpagoBI Server.

To this end, configure SpagoBI Studio and connect it to SpagoBI Server, which will be the deployment environment to provide end-users with all certified objects. The main steps to configure SpagoBI Studio are:

- select the SpagoBI perspective
- create a SpagoBI project

- set SpagoBI Server connections.

At first execution of SpagoBI Studio, a welcome page appears with references to technical documentation (the wiki¹², SpagoBI free community tool) and the preferred working perspective.

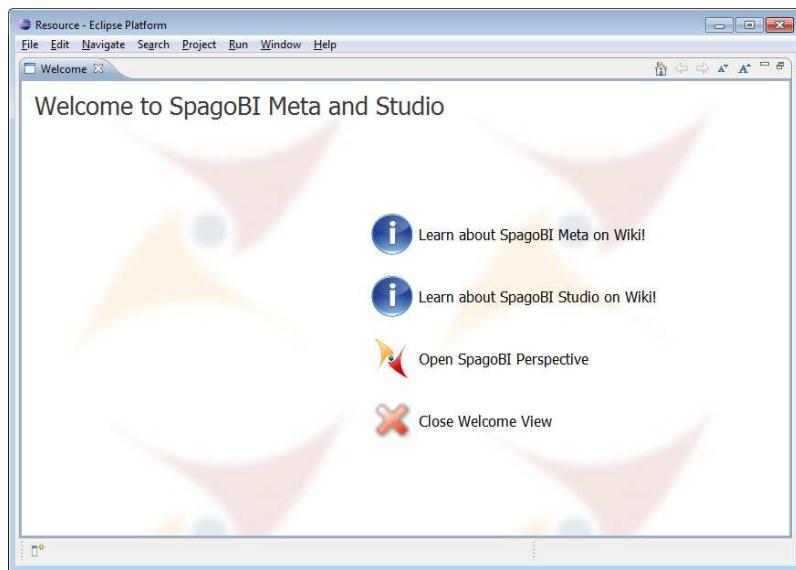


FIGURE 4.6 – SpagoBI Studio welcome page

The first step is to create a SpagoBI project by selecting the SpagoBI icon from the toolbar located at the top of the page.

¹² <http://wiki.spagobi.org/xwiki/bin/view/Main/>

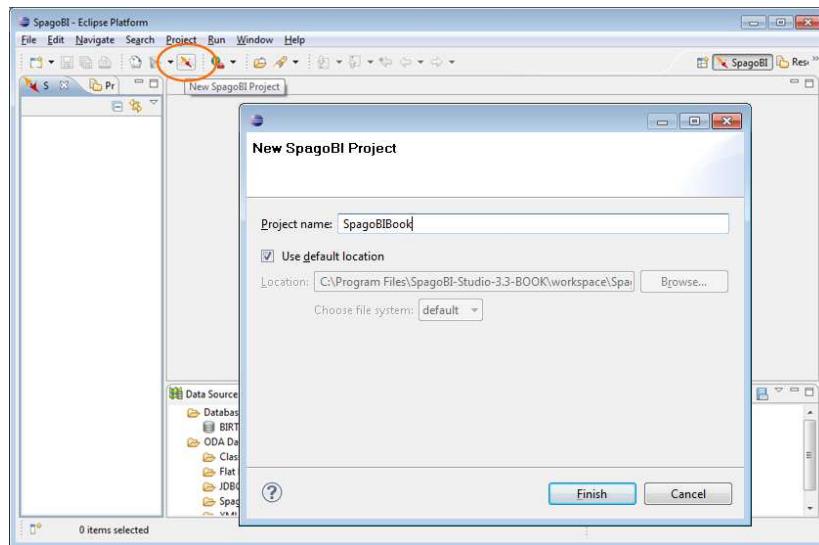


FIGURE 4.7 – Create a SpagoBI project

Once the project has been created, a standard folder tree appears.

The predefined folders are:

- **Business Analysis:** it contains all developed documents (reports, charts, cockpits, etc.) that can be uploaded into or downloaded from SpagoBI Server. This folder can be structured with subfolders, to freely organize one's own documents.
- **Business Models:** they contain all metadata models created by the user (if he has a data owner role too) or downloaded from SpagoBI Server
- **Business Queries:** they contain all queries defined by the user over a created or downloaded metadata model
- **Private folders:** they contain the user's personal or project documentation. This folder can be freely managed by the user.
- **Resources:** they contain all technical resources used in the project (i.e., the reference to a SpagoBI Server).

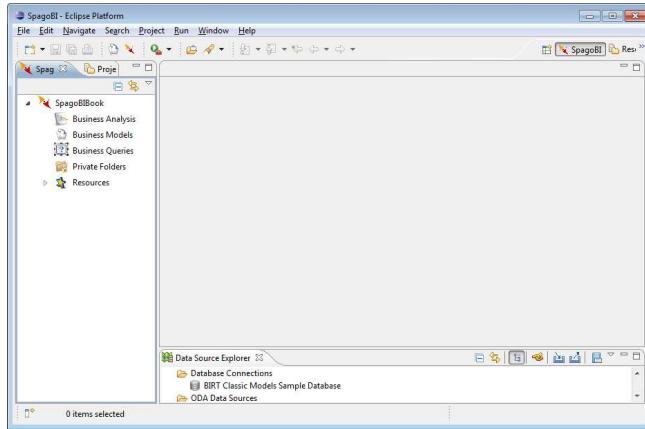


FIGURE 4.8 – SpagoBI Studio folder tree

Now you can configure references to one or more SpagoBI Servers. In other words, each user can work for many projects that can be:

- hosted on different servers
- hosted on the same server with different user accounts.

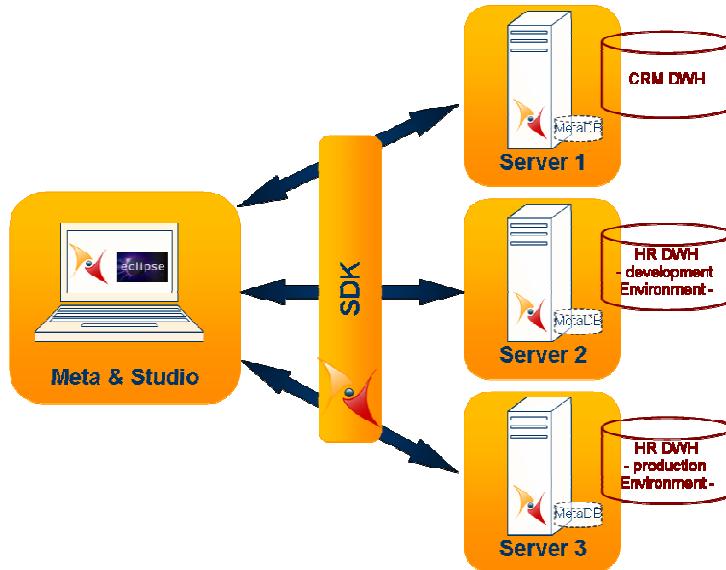


FIGURE 4.9 – SpagoBI Studio towards SpagoBI Servers

To define a server connection, click on the **New Server** item of the **Server** contextual menu. A form will ask you for:

- **Server name:** a logical name to identify the server. The name is used on the local workspace only and has no relation with the physical one
- **Url:** the http url where the server is hosted and reachable.
- **User:** the user who authenticates on SpagoBI Server, setting his access rights in terms of what kind of operations he can do (upload and download a model or a data set) and what parts of the Server repository he can access.
- **Password:** the user's password
- **Active:** a flag that indicates the active server. It is particularly useful when the user is working with multiple servers. The active server indicates that every upload and download operation refers to this SpagoBI Server instance.

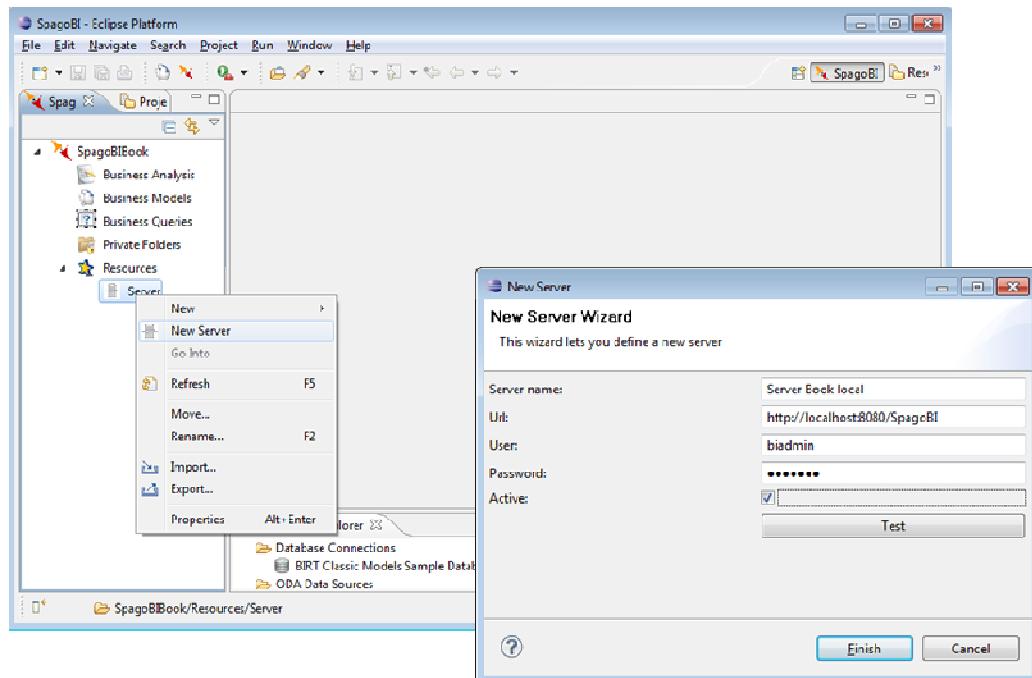
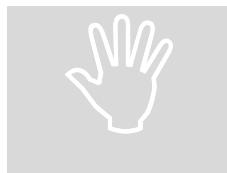


FIGURE 4.10 – Connection to a SpagoBI Server from SpagoBI Studio



Connection to SpagoBI Server

If something in your network configuration has been changed from your first run of SpagoBI Studio, the connection test of SpagoBI Studio to the Server could fail. Most often this problem is due to the proxy settings in your Eclipse environment. If this is not the case, try to run SpagoBI Studio from the command line with the –clean option (SpagoBI.exe –clean) to reset working settings.

At this point, SpagoBI Studio is ready to work!

Common tasks

The interaction with SpagoBI Server takes place in the same way for all types of analytical documents. The tasks to be performed follow:

- Define your dataset(s)
- Download a template from the Server
- Deploy a template or a new document to the Server.

Data set definition

Each document type has its own way to define how to get data from an internal data source, according to a data set definition. This allows the document to directly access the RDBMS, through the SQL loading script, which can be encoded within the template or externally (i.e., stored as SpagoBI Server resource), but without any abstraction from data sources.

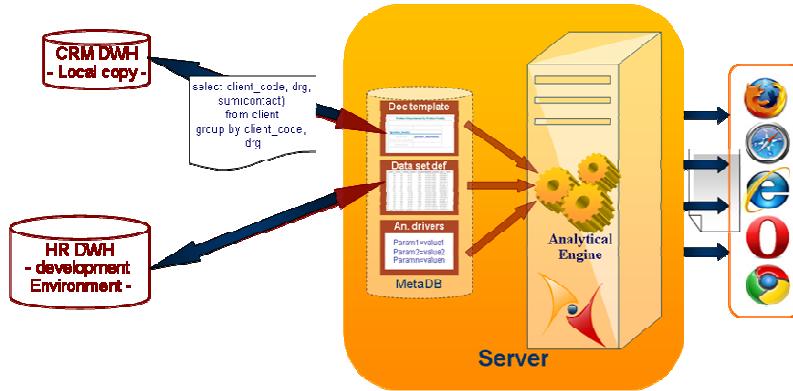


FIGURE 4.11 – Direct access to RDBMS from SpagoBI Studio

Alternatively, a document can use an external data set, produced as an inquiry over a business model and deployed on the active SpagoBI Server. This way, the developer does not need to write any SQL statement but he can produce new data sets using the graphical wizard as follow:

- create or download an already deployed Business Model from a SpagoBI Server
- create a new business query
- deploy the new business query on SpagoBI Server.

Obviously, if the developer wants to re-use an already released business query, the last two steps are unnecessary.

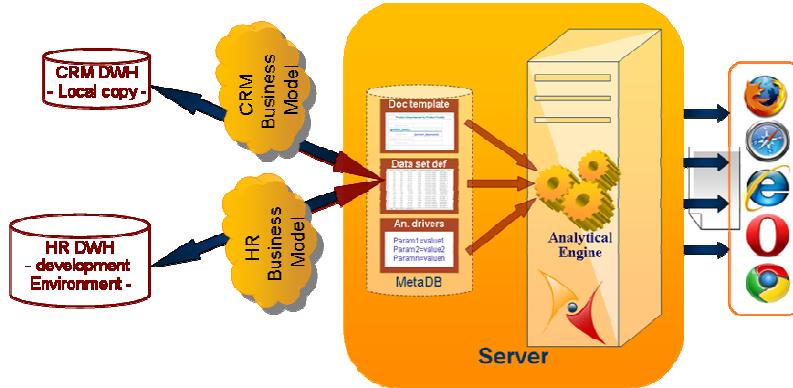


FIGURE 4.12 – Access the RDBMS from SpagoBI Studio by means of a metamodel



Business Model and Business Inquiries

To learn more about how to build a business model and inquire it, please refer to chapter 3 - SpagoBI Meta.



SQL code vs Business Inquiry

Even if building a data set using the metamodel is easier than writing a SQL script, this method will never be obsolete. In fact, accessing a RDBMS through advanced SQL code can guarantee performances and loading logic that are not always possible with metadata abstraction.

Download and deploy

To modify an already deployed document, first download the related template from the SpagoBI Server repository.

Right-click on the **Business Analysis** folder or on one of its subfolders. In the contextual menu, select the **Download** option. At this point, the functionality tree appears, allowing you to choose the documents to be downloaded.

These documents will be available in the local folder that you have previously selected. Document details (i.e., label, description, state, engine and parameters) are stored as metadata in the local repository. Metadata can be refreshed from the Server by clicking on the **Refresh** button in the **SpagoBI > Document Metadata** tab of the **Properties** section. To open Properties, right-click on the document item and select **Properties**.

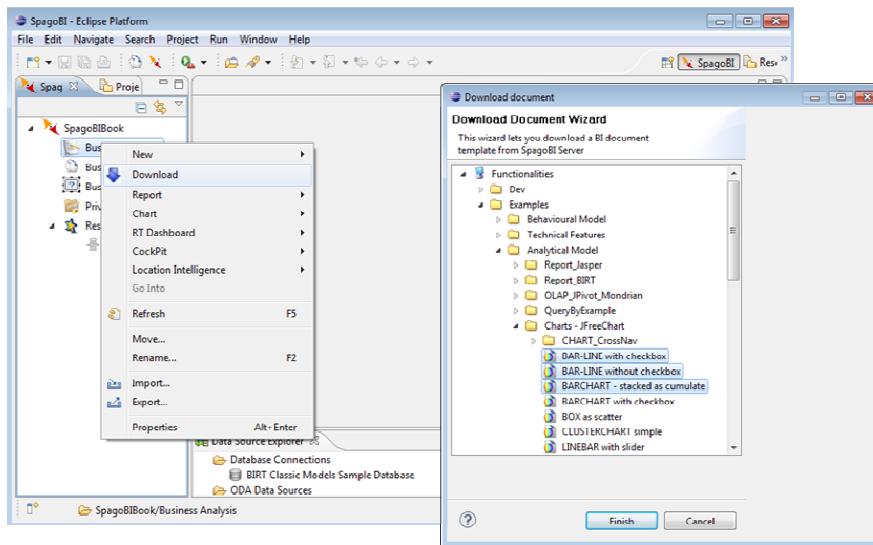


FIGURE 4.13 – Download the document template from a Server through SpagoBI Studio

In a similar way, after a document update, the **Deploy** option of the same menu sends the new template to the Server, ready for use.

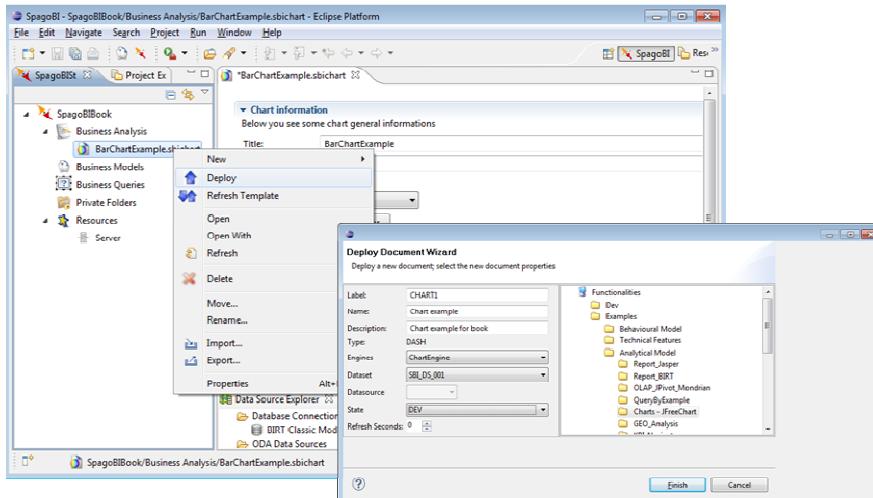


FIGURE 4.14 – Deploy the document template from SpagoBI Studio to a Server

Another possible situation is when the designer creates a new template from scratch and deploys it on the Server. At first deploy, a link between the template and a document on the Server is created. It will last until the document on the Server is deleted or its label is modified. In those cases, you will need to re-deploy the template from the Studio.

To deploy a template, right-click and select **Deploy**. You will be prompted a form for basic metadata on the new document. Required and/or pre-filled input data may change according to the document type. However, they usually include:

- **Label**: free label as short code
- **Name**: name of the document
- **Description**: long description
- **Type**: document type (report, chart, cockpit, etc.)
- **Engine**: it depends on the document type and contains the engines available for the selected document type (i.e. Jasper and BIRT for reporting, JFreeChart and HighChart for charts, etc.)
- **Data Set**: the already deployed data set for documents that use external ones
- **Data source**: the reference to the data source that will be used on SpagoBI Server for documents that have an internal data set, in order to work with official source instead of local or working RDBMS
- **State**: the initial state of the document (development, test, released, suspended) according to their life cycle management policy
- **Refresh seconds**: the automatic refresh time
- **Position**: the folder in the remote SpagoBI Server repository where documents are deployed, indirectly setting who can use it and its first authorization level.



Analytical documents

The described form sets basic metadata, generally managed as technical metadata on SpagoBI Server. To learn more about the management of analytical documents and their technical metadata, please refer to chapter 5 – SpagoBI Server.

These document details are stored as metadata in the local repository and used to register it in the central repository of the Server as well. To look at their local

values, select the **Properties** item from the document contextual menu and choose SpagoBI.

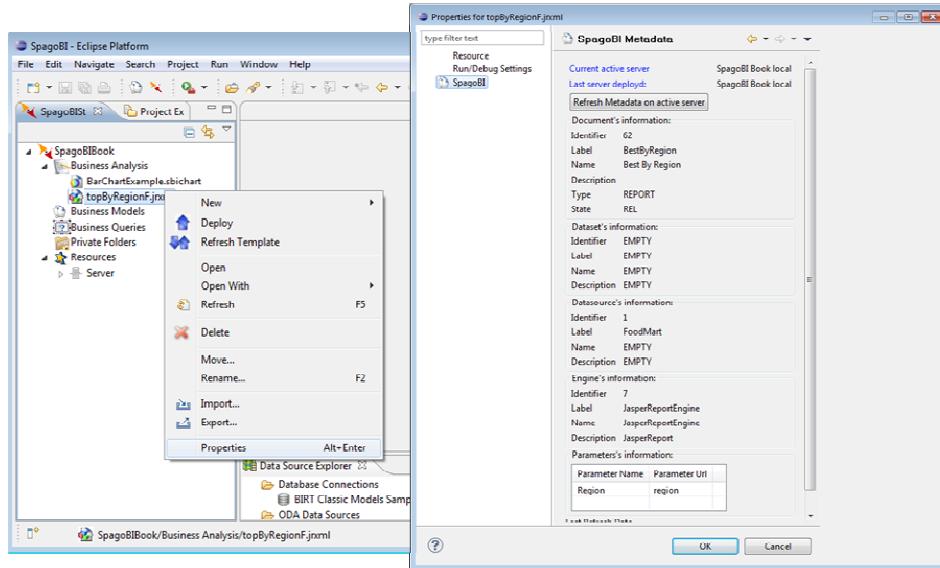


FIGURE 4.15 – Document metadata

Directly from there, local metadata can be refreshed anytime on the active server, by simply pressing the Refresh Metadata on active server button.

Developing documents

To create a new document, first choose the desired type and engine from the contextual menu. This appears in the **Business Analysis** folder of the local repository, together with the corresponding graphical editor.

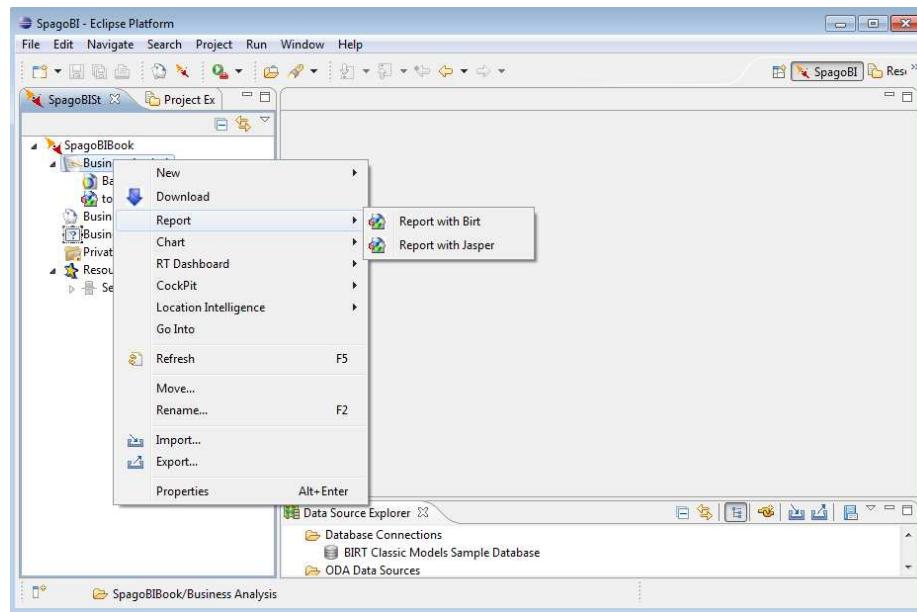


FIGURE 4.16 – Create a new document choosing type and engine

Once the document is designed, it is stored as a local file, marked out with an icon and a specific file extension:

- **.sbichart**: document template for charts that use the JFreeChart engine
- **.sbihchart**: document template for charts that use the HighChart engine
- **.sbidash**: real-time document templates that use the dashboard engine
- **.sbidoccomp**: document templates for cockpits that use the ComposedDocument engine
- **.sbigeo**: document template for maps supporting Location Intelligence that use the GEO engine
- **.jrxml**: document template for reports that use the Jasper engine
- **.rptdesign**: document template for reports that use the BIRT engine.

A double click on one of these files allows to open the document template, with its related graphical editor.

Report

Designer for SpagoBIBirtReportEngine

The design and deployment of a BIRT report includes the following steps:

- create the empty document
- switch to the report designer perspective
- create the data source
- create the dataset
- design the report via the graphical interface
- deploy the report on the server.

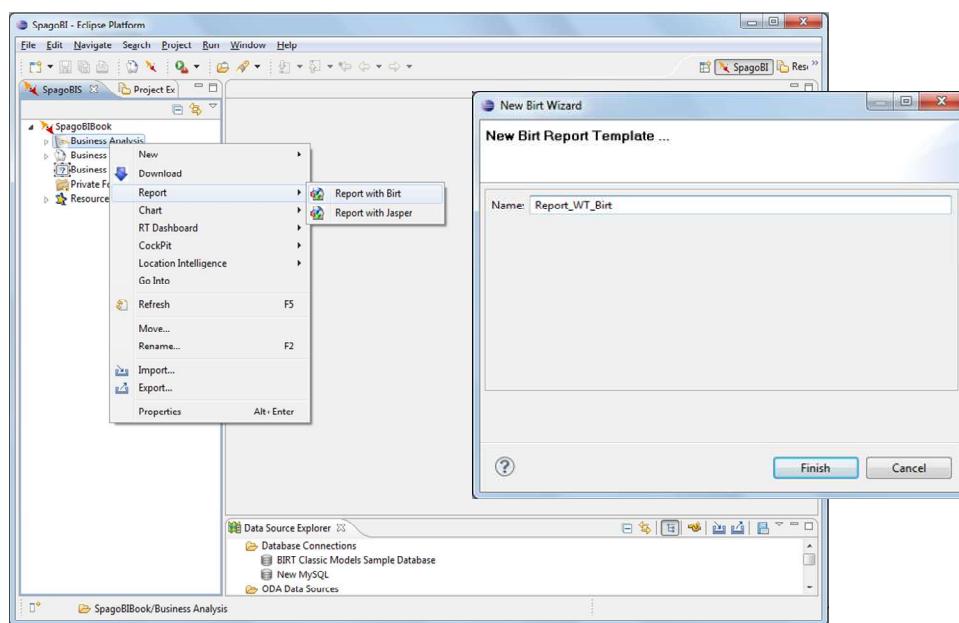


FIGURE 4.17 – Step 1: Create a BIRT report

To create a new BIRT report, right click on the **Business Analysis** folder and select **Report > Report with BIRT**. This will open an editor where you can choose the name of your document. The new document will be created under the **Business Analysis** folder.

Double click on it to open the editor. At this point, you are still working in the SpagoBI perspective. To design the report, switch to the actual BIRT designer perspective. Click on the perspective icon of the Eclipse editor and select the Report Designer among the available perspectives.

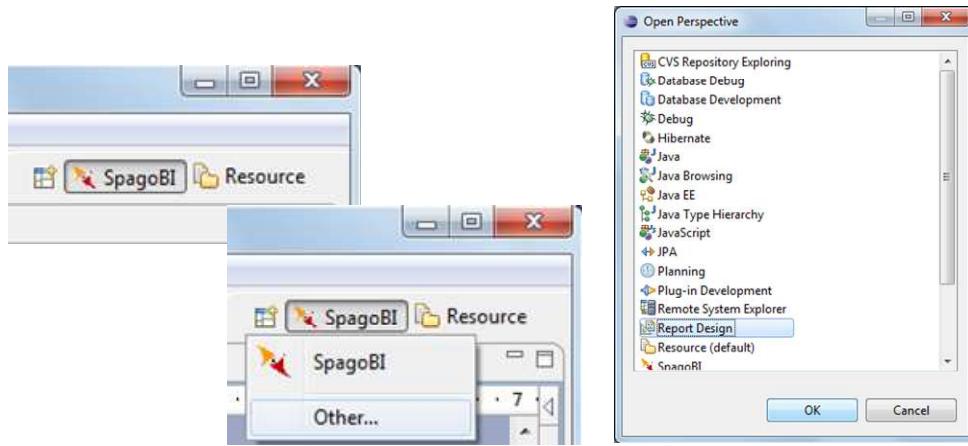


FIGURE 4.18 – Switch to the report designer perspective

The next steps are the creation of a datasource and of a dataset. As previously described in the section Dataset Definition, SpagoBI Studio allows the development of analytical documents using either internal or external datasets. In this specific example, we will show how to create a report with an internal dataset.

First of all, in case of an internal dataset, define a **JDBC Data Source**. Right click on the **Data Source** item and select the corresponding data source. A pop up editor will open, prompting you for the connection settings:

- **Driver class**
- **Database URL**
- **Username** and **password**



BIRT Data Source

For detailed information about configuration of data sources in BIRT, please refer to BIRT documentation and manuals.
www.eclipse.org/birt/

Note that these configuration parameters will be used by the report to connect to the database, if it is executed locally (i.e., within the Studio). This may be the same database referenced in SpagoBI Server or a local database (provided that they share the same schema).

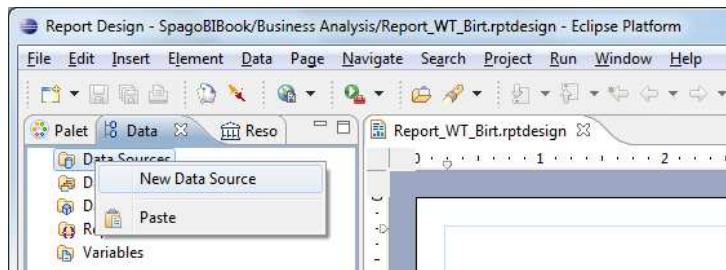


FIGURE 4.19 – Step 2: Creation of a data source

Since you are setting a local reference to a database inside the report, remember to set an additional information: this will enable SpagoBI Server to correctly execute the report, by connecting to the data source referenced within the server and not inside the report. Basically you need to tell the server to override the data source configuration.

Therefore, add a parameter to the report, called `connectionName`. Then select **Property Binding** in the Data Source editor and set the property `JNDI URL` to the value of the `connectionName` parameter.



JNDI URL

Do not forget to define the `connectionName` parameter in your BIRT report and set the `JNDI URL` accordingly. Without these settings your BIRT report may be unable to access data once it is deployed on the server.

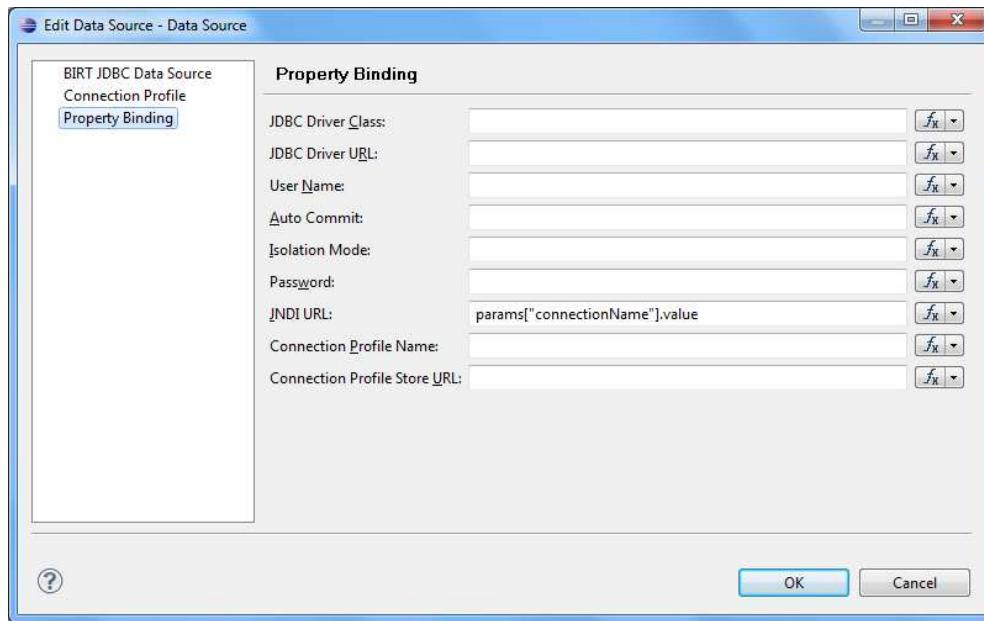


FIGURE 4.20 – Setting the connectionName parameter in the Data Source editor

Once the data source is defined, you can create your dataset. Right-click on the **Data Set** item and select **New Data Set**. In the next window, select the data source, the type of query and give a name to the dataset. The scope of this name is limited to your report, because we are defining an internal dataset.

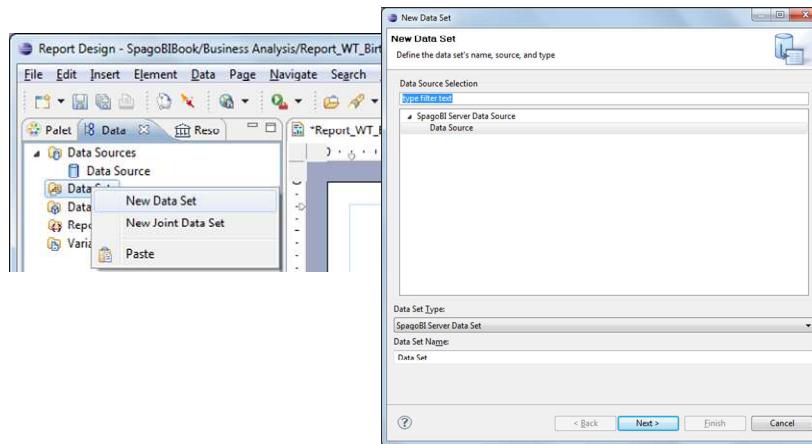


FIGURE 4.21 – Step 3: Data Set definition

Now you can define your dataset by writing the SQL query in the editor and testing the results. At any time, you can modify the dataset by clicking on it, which will re-open the query editor.

Let us design a very simple report, which contains a table showing the data from the defined dataset. The easiest way to create a table from a dataset is to drag & drop the dataset from the tree menu into the editor area.

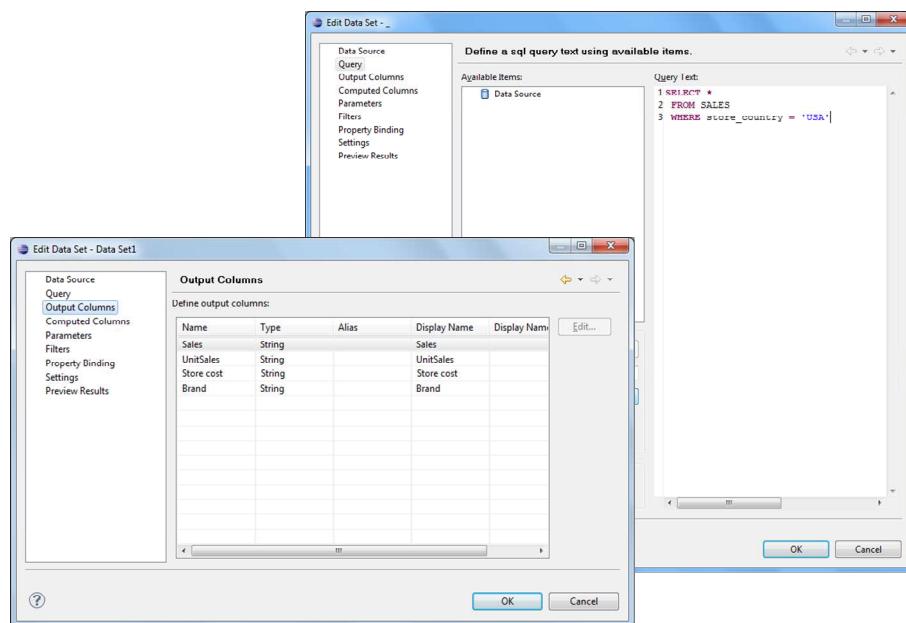


FIGURE 4.22 – Dataset editor, with preview

The most generic way, which applies to all graphical elements, consists in switching to the **Palette** menu on the left panel, keeping the designer in the central panel. Drag and drop the table into the editor area. Consider that this can be done with all other elements listed in the Palette. At this point, you can edit the table (as well as any other graphical element on the report) using the **Property Editor** tab below the editor area.

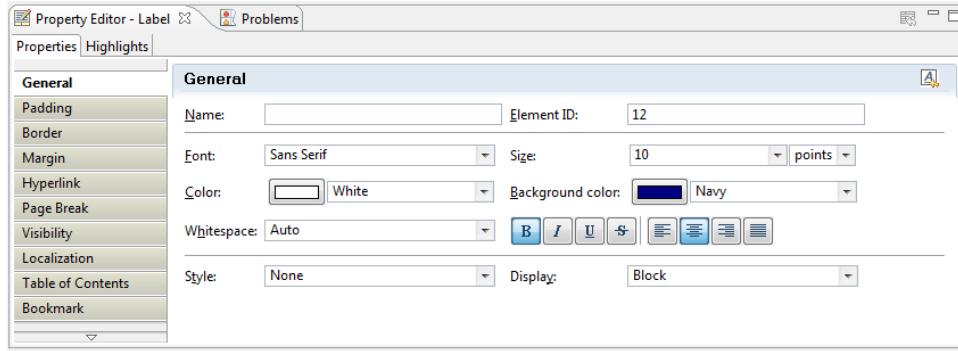


FIGURE 4.23 – BIRT Property Editor

While developing a report, it is particularly useful to test it regularly. To this end, click on the **Preview** tab below the editor area. To revert back to the editor, just click on the **Layout** tab. In the **Master Page** tab, you can set the dimensions and layout of the report; the **Script** tab supports advanced scripting functionalities; finally, the **XML Source** tab shows the editable source code of your report.

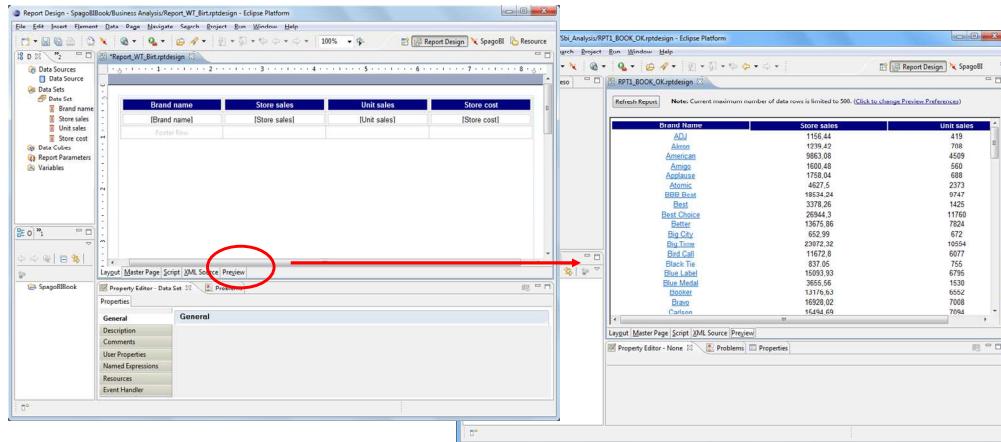
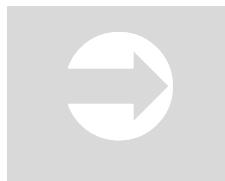


FIGURE 4.24 – BIRT Designer preview

Once your report is done, you can deploy it on SpagoBI Server.



Deploy on SpagoBI Server

Please refer to the section Download and Deploy in this chapter to find out more on report deployment.

The BIRT report designer allows the creation of complex reports, with different graphical elements such as cross tabs, charts, images and different text areas. In this section we do not provide any details on graphical development but we focus on specific aspects of SpagoBI BIRT Report Engine.



BIRT Designer

For a detailed explanation of report design, please refer to BIRT documentation at www.eclipse.org/birt/

External Data Set

In the afore-described example, we built a report using an internal dataset, i.e., a dataset defined within the report. This has two main implications. First, the dataset is not visible outside the report execution: for example, it cannot be directly reused by other reports. Second, an internal dataset is always defined as a SQL query and it cannot take advantage of SpagoBI business model abstraction.

For these reasons, SpagoBI allows the definition of *external datasets* in reports. An external dataset is defined in SpagoBI Server and, as a consequence, it is visible to all documents on the server (i.e., it can be used by any of them, if properly linked to the document). External datasets can either be SQL datasets or QbE datasets, that is, datasets defined by queries over a business model.



Business Model Query

Please refer to chapter 3 – SpagoBI Meta to see how to define and query a business model.

An external dataset can be included into any BIRT report by downloading it from a SpagoBI Server. Specifically:

- define a SpagoBI Server datasource
- download a dataset from the SpagoBI Server datasource.

We always start by right-clicking on the **Data Source** item. Select **SpagoBI Server Data Source** and set the appropriate input configuration:

- **Server URL**
- **Username** and **password** used to log into the Server (e.g., biadmin).

After filling in the configuration fields, test the connection and save it. The new data source will appear in the left tree menu. Instead of connecting to a database via a JDBC driver, connect to the server as the source of data. Obviously, the actual data source and dataset must have previously been defined on the Server.



Datasource and dataset in SpagoBI Server

Please refer to chapter 5 – SpagoBI Server for a general overview on dataset definition and management.

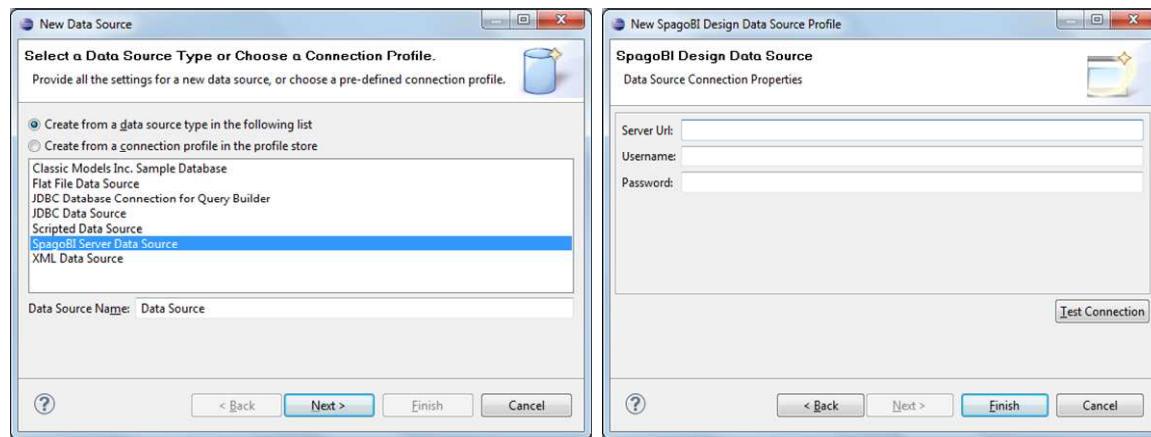


FIGURE 4.25 – Definition of a SpagoBI Server Data Source

To select the dataset, click on **New Data Set** as above, but this time select the **SpagoBI Data Source** that you have just defined. Now, instead of choosing a new name for the dataset, insert the correct label of the dataset that you want to import from the Server. If the label is correct, the dataset will be imported in the report by clicking on **Finish**.

Notice that the imported dataset may be a SQL or a QbE one. Since both types of datasets are stored in the same repository by SpagoBI Server, we are enabled to use any BM query in the development of a report.

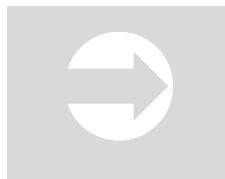


Use of BM queries in report development

The ideal use of a business model is to define queries over the BM via SpagoBI Meta, deploy them on SpagoBI Server and reuse them on SpagoBI Studio as external datasets.

Adding parameters to reports

Most times reports show data analysis that depend on variable parameters, such as time. SpagoBI BIRT Report Engine allows the designer to add parameters to a report and link them to analytical drivers defined in SpagoBI Server.



Analytical Drivers

For a complete understanding of analytical drivers, please refer to section The Behavioral Model in chapter 5 – SpagoBI Server.

To use these parameters, you first need to add them to your report. Right-click on **Report Parameters** in the tree panel and select **New Parameter**. Here you can set the data type and choose a name for your parameter.

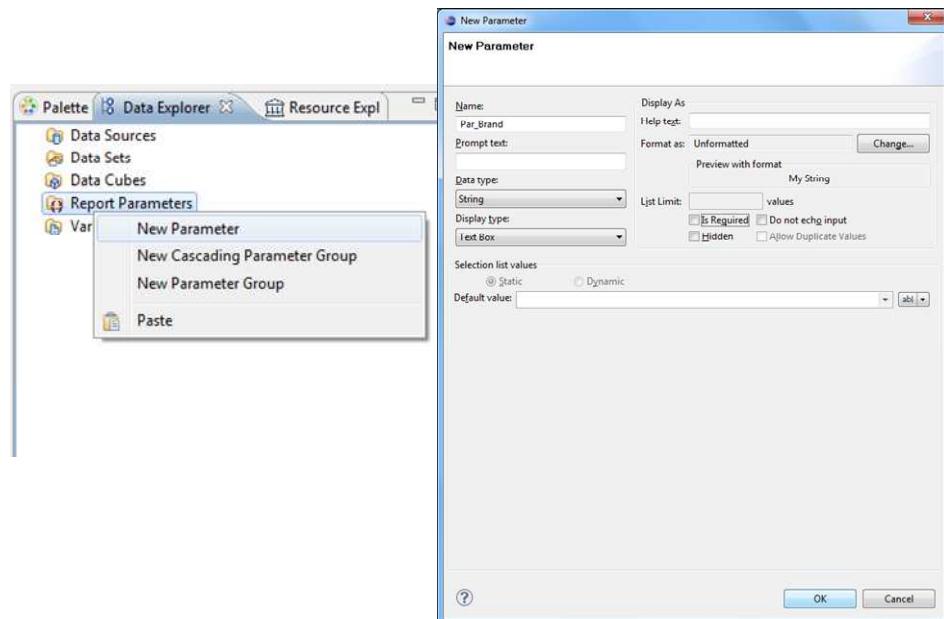
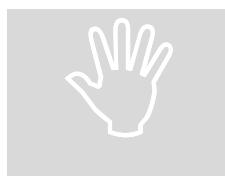


FIGURE 4.26 – Creation of a new parameter in a BIRT report



Parameters URI

Be careful when assigning a name to a parameter inside a report. This name must correspond to the parameter's URI when you deploy the document on SpagoBI Server.

Once you have defined all parameters, open the (or create a new) dataset. Parameters are identified by a question mark ?. For each ? that you insert in

your query, you must set the corresponding link in the **Parameters** tab: this will allow parameters substitution at report execution time.

Note that you must set a link for each question mark, even if the same parameter occurs multiple times in the same query.

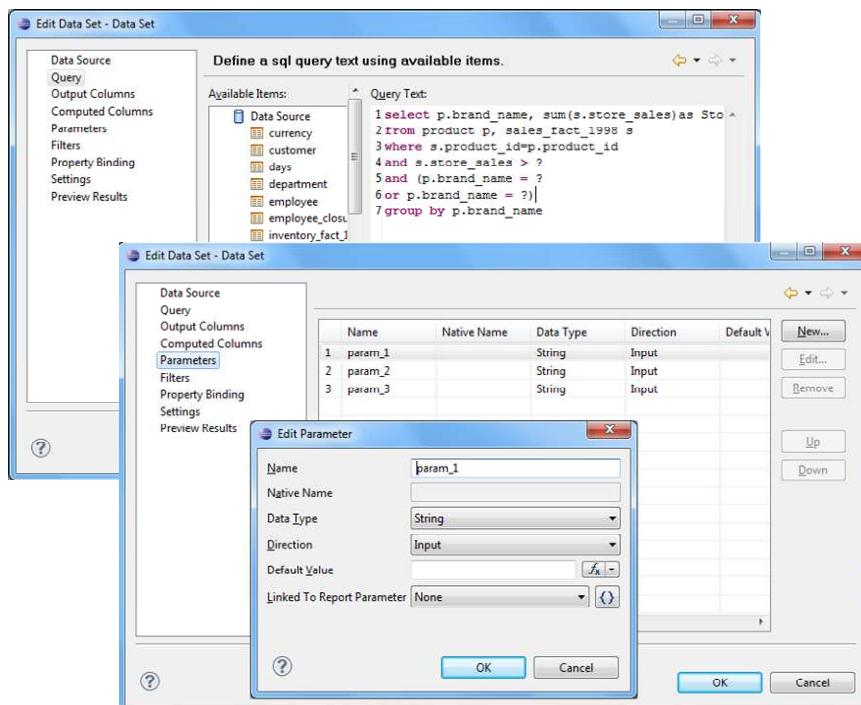


FIGURE 4.27 – Insert parameters into the dataset definition

Parameters can also be used within some graphical elements, such as dynamic text, with the following syntax:

```
params[ "name_of_parameter" ].value
```

Cross Navigation

A powerful feature of SpagoBI analytical documents is cross-navigation, i.e., the ability to navigate documents in a browser-like fashion following logical data flows. Although cross-navigation is uniformly provided on all documents

executed in SpagoBI Server, each type of document has its own modality to set the link pointing to another document.

Notice that the pointer can reference any SpagoBI document, regardless of the source document. For example, a BIRT report can point to a chart, a console, a geo or any other analytical document.

To allow cross-navigation in a BIRT report, you need to add a hyperlink to the element you want to be clickable. Most report elements can host an hyperlink. For example, let us add an hyperlink to a cell in the table of our previous example.



Cross Navigation

For further information on cross navigation and how it works, please refer to the corresponding section in chapter 5 – SpagoBI Server.

Click on the table cell and select the **Hyperlink** item in the **Properties** tab. By clicking on Edit, the hyperlink editor will open and show three input fields:

- **Location:** write here the URI
- **Target:** select **Self**
- **Tooltip:** write the text you wish to appear on the link.

To edit the Location, click on the right drop down button and select the JavaScript syntax. This will open BIRT JavaScript editor. Here you must set the hyperlink for cross-navigation as a string, according to the following syntax:

```
"javascript:parent.execCrossNavigation(this.name,
'CrossNav_Destination_Doc', 'Param_1=' + params["Param_1"].value
+ "&Param_2=" + row["my_table_column"] + "&Param_3=Fixed_val"
+',','');
self');");
// CrossNav_Destination_Doc is the label of the destination
document
// Param_1 is assigned the value of the parameter Param_1
// Param_2 is assigned the current value of a table cell (column)
// Param_3 is assigned a fixed value
```

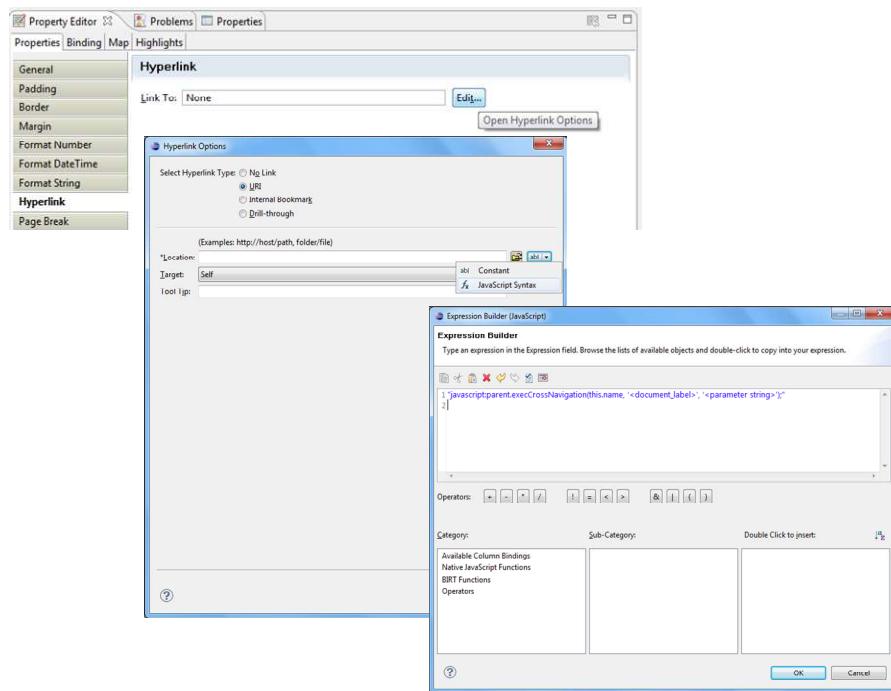


FIGURE 4.28 — Add a hyperlink to a BIRT report for cross-navigation

The syntax of the string is fixed, while you need to set the appropriate label of the destination document, as well as assign values to the parameters that will be passed to the destination document. The JavaScript editor may help you to insert dataset column bindings and report parameters automatically.

The case of *multi-value parameters* requires special attention since SpagoBI represents the values of a multi-value parameter as a string. To avoid errors in the Javascript syntax, multi-value parameters must be separately managed.



Cross navigation with multi-value parameters

On SpagoBI wiki you can find useful information about cross navigation, and particularly on how to manage links with multi-value parameters.

<https://wiki.spagobi.org/xwiki/bin/view/Main/>

Designer for SpagoBI JasperReportEngine

Unlike the BIRT designer, iReport is not integrated in Eclipse. Therefore, before starting developing your report, download the Jasper report designer: iReport.



Download iReport

Download the iReport designer at
<http://jasperforge.org/projects/ireport>.

Although the recommended version is 4.0.2, SpagoBI Studio can work with more recent versions as well.

Then you will have to set the path to the iReport designer in SpagoBI Studio. Open **Window > Preferences > SpagoBI Editor Configuration** and set the correct path to the ireport.exe file.

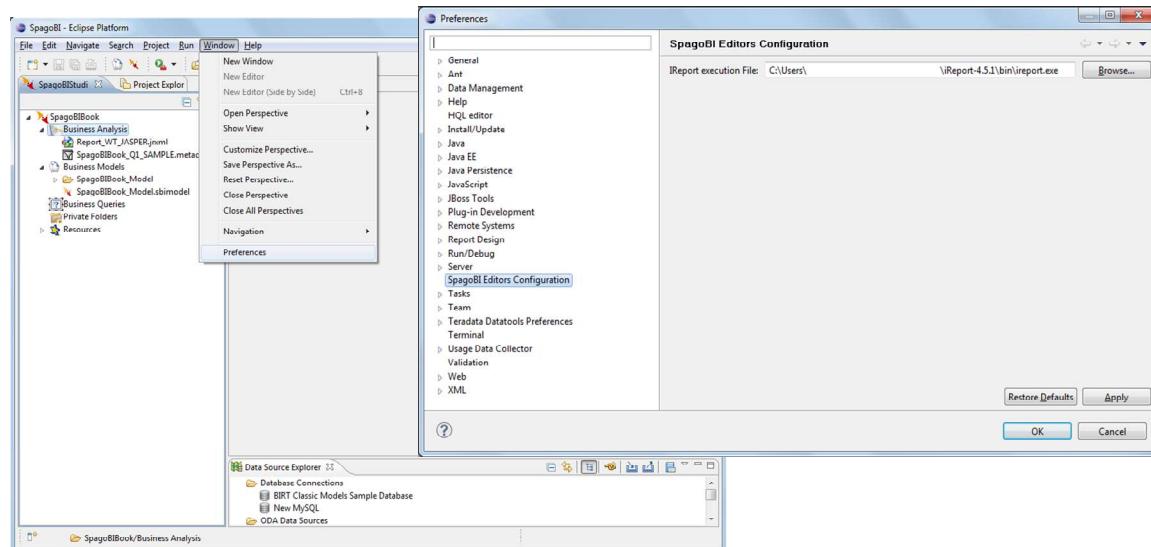


FIGURE 4.29 – Configuration of the iReport editor within SpagoBI Studio

Similarly to the case of a BIRT report, the design and deployment of a Jasper report with SpagoBI Studio consists of the following steps:

- create the empty document

- switch to the report designer perspective
- create the data source
- create the dataset
- design the report via the graphical interface
- deploy the report on the server.

To create a new Jasper report, right-click on the **Business Analysis** folder and select **Report > Report with Jasper**. This will open an editor where you can choose the name for your document. The new document will be created under the **Business Analysis** folder.

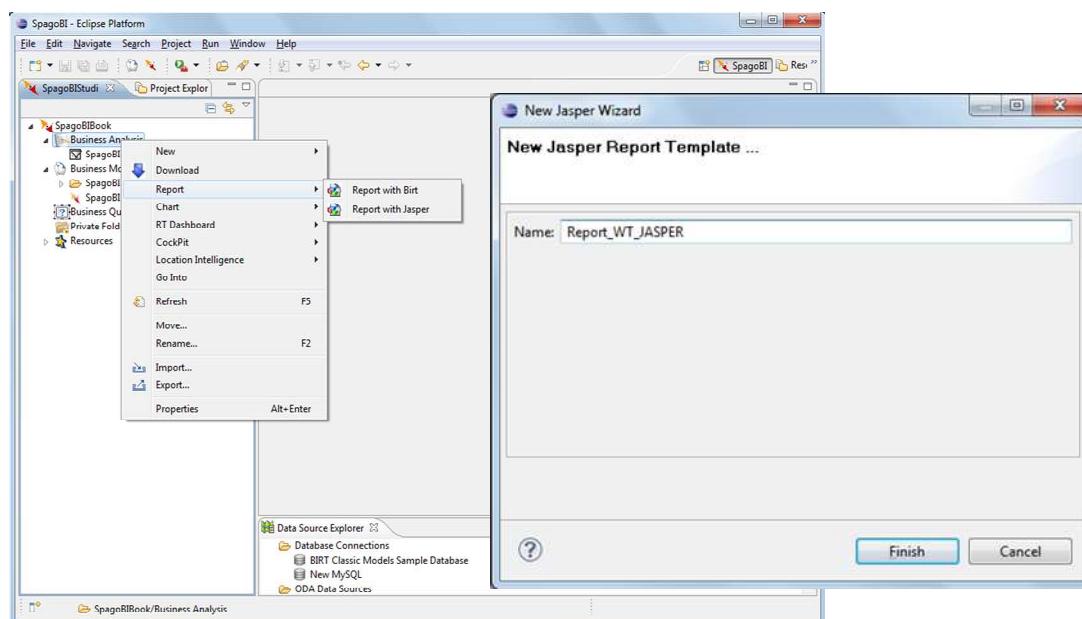


FIGURE 4.30 – Step 1: Create a Jasper report

Double click on the report to open the editor: the iReport editor will be launched inside SpagoBI Studio. Now you are ready to start developing your report.



iReport Designer

Before trying to edit your Jasper report, make sure that the link to the iReport editor is correctly set in **Preferences**.

The next steps consist in the creation of a datasource and of a dataset. As described in section Dataset Definition, SpagoBI Studio allows the development of analytical documents using either internal or external datasets. In this example we will show how to create a report with an internal dataset.

Click on the small icon of the menu bar. The data source creation editor will open. Select a JDBC data source, set the appropriate parameters and give the data source a name.

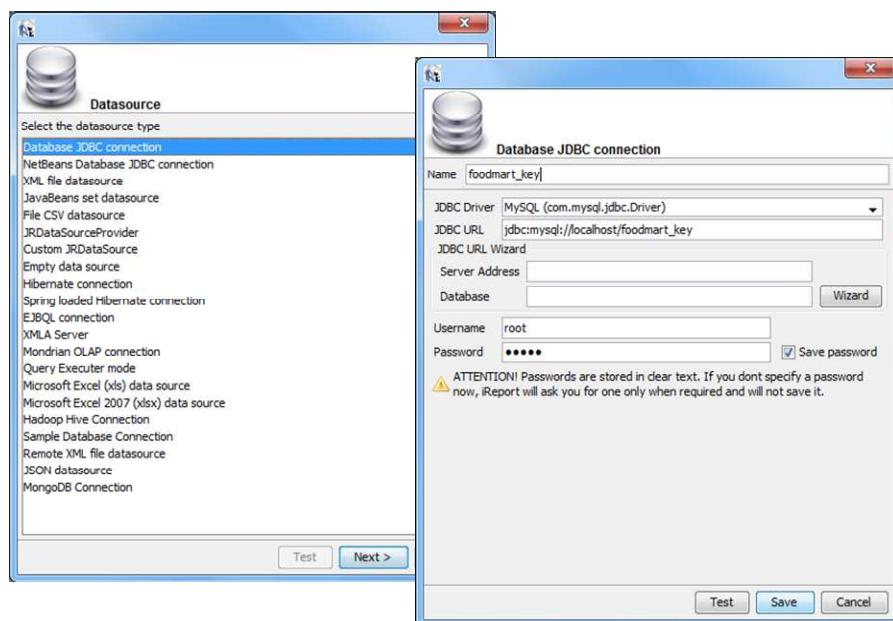


FIGURE 4.31 – Creation of a JDBC data source in a Jasper report

Once you have defined the data source, create your dataset. Note that Jasper only allows one main dataset. Secondary datasets can be added if needed (the procedure is explained in the particular case of external datasets, later in this section).

Right-click on the report item and select **Add dataset**. The dataset editor will guide you through the dataset definition. You can either manually edit the SQL query or use the design query tool provided by iReport.

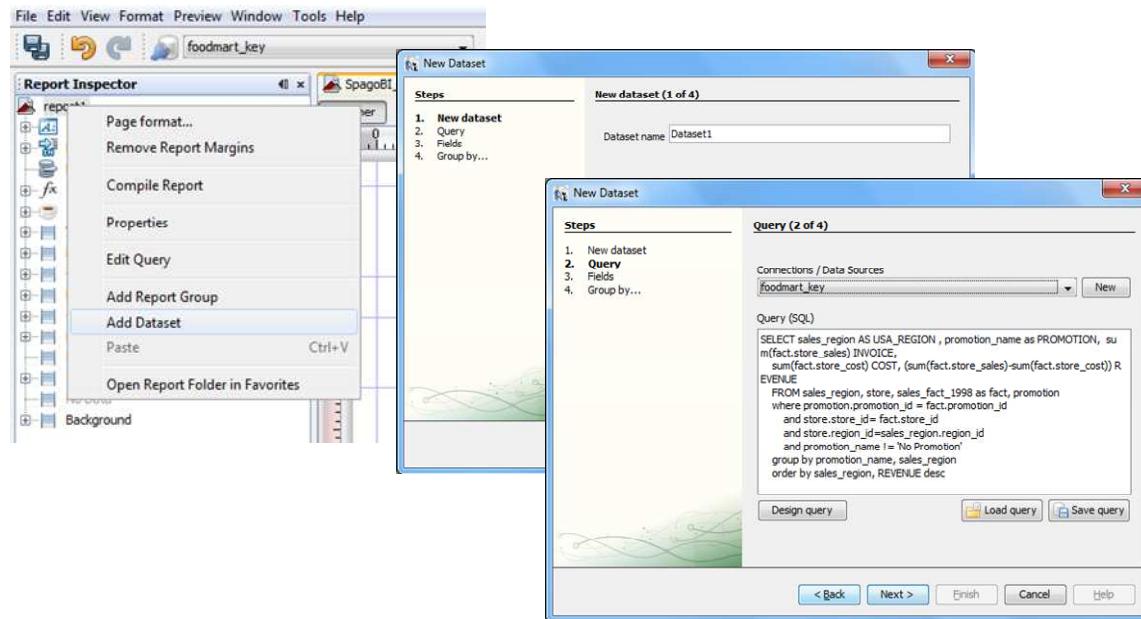


FIGURE 4.32 – Step 3: Data Set definition

The tree located in the left part of the window shows the elements of the report. On the right, the **Palette** shows all graphical elements you can add. You can choose to see your report within the **Designer**, to inspect the **XML** code or to look at the **Preview** of your report. If you click on the icon you can edit and preview your query.

As you can see, the iReport designer allows the creation of complex reports, with different graphical elements such as cross tabs, charts, images and different text areas.

This section does not provide any detail about graphical development but it focuses on specific aspects of SpagoBI Jasper Report Engine. It allows you to design a very simple report, which contains a table showing data from the defined dataset.



Jasper report configuration and editing

For more details about report configuration and graphical editing, please refer to the project webpage:

<http://jasperforge.org/>

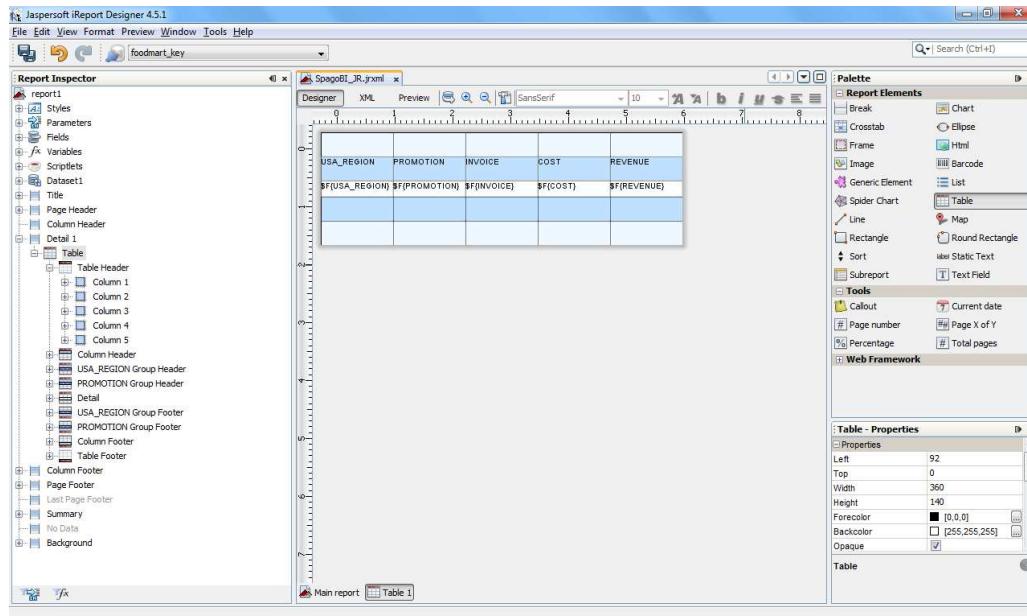


FIGURE 4.33 – iReport graphical editor

To create a table, click on the element in the **Palette** and use the wizard. To insert dataset columns into the report, use the following syntax:

```
$F{name_of_dataset_column}
```

Once your report is ready, deploy it on SpagoBI Server.



Deploy on SpagoBI Server

Please refer to section Download and Deploy in this chapter for more details on report deployment.

External Data Set

In the above-described example, we built a report using an internal dataset, i.e., a dataset defined within the report. This has two main implications. First, the dataset is not visible outside the report execution: for example, it cannot be directly reused by other reports. Second, an internal dataset is always defined as a SQL query and it cannot take advantage of SpagoBI business model abstraction.

For these reasons, SpagoBI allows the definition of *external datasets* in reports. An external dataset is defined in SpagoBI Server and is therefore visible to all documents on the server (i.e., it can be used by any of them, if properly linked to the document). External datasets can either be SQL datasets or QbE datasets, that is, datasets defined by queries over a business model.



Business Model Query

Please refer to chapter 3 - SpagoBI Meta to see how to define and query a business model.

An external dataset can be included into any Jasper report. Both the main dataset and secondary datasets can be SpagoBI datasets; however, the steps are slightly different.

To link a SpagoBI (external) dataset to the main dataset of a Jasper report, you should the following steps:

- define the dataset within the report. This will be used when the report executes locally (i.e., in the iReport designer environment);
- define the dataset on SpagoBI with the exact same label as the local one. These two steps can be swapped, just remember to use the same label;
- upload the report on SpagoBI server and set the dataset name in the upload interface.

When the document is executed, SpagoBI will check if the document is associated to a dataset and, if this is the case, it will retrieve data from the dataset defined on the server.



Datasource and dataset in SpagoBI Server

Please refer to chapter 5 – SpagoBI Server for a general overview on dataset definition and management.

As said above, the main dataset is just one for a Jasper report. If you need to add multiple external datasets, you can exploit secondary datasets. The steps are the following:

- Create a parameter of type *net.sf.jasperreports.engine.JRDataSource* with the label of the external dataset. You can select the type in the dataset properties.
- Right-click on the report name in the **Report Inspector** panel (left) and select **Add dataset..**. Give the dataset the same label as above.
- Create a graphical element in the report and right-click on its name in the **Report Inspector** panel (left). Select **Edit datasource** to open the panel **Dataset Run**. Here you can associate the element to a secondary dataset by opening the Expression Editor (see FIGURE 4.36).

The same procedure can be repeated for as many secondary datasets as you wish.

When the document is uploaded and executed on Server, SpagoBI will first look for the main dataset (as described in the previous section). Then, it will check if any of the report element is associated to a parameter of type *JRDataSource*. If this is the case, it will retrieve values from the SpagoBI dataset with the parameter's label. Note that the dataset should also be defined locally if you want to execute the report within iReport designer.

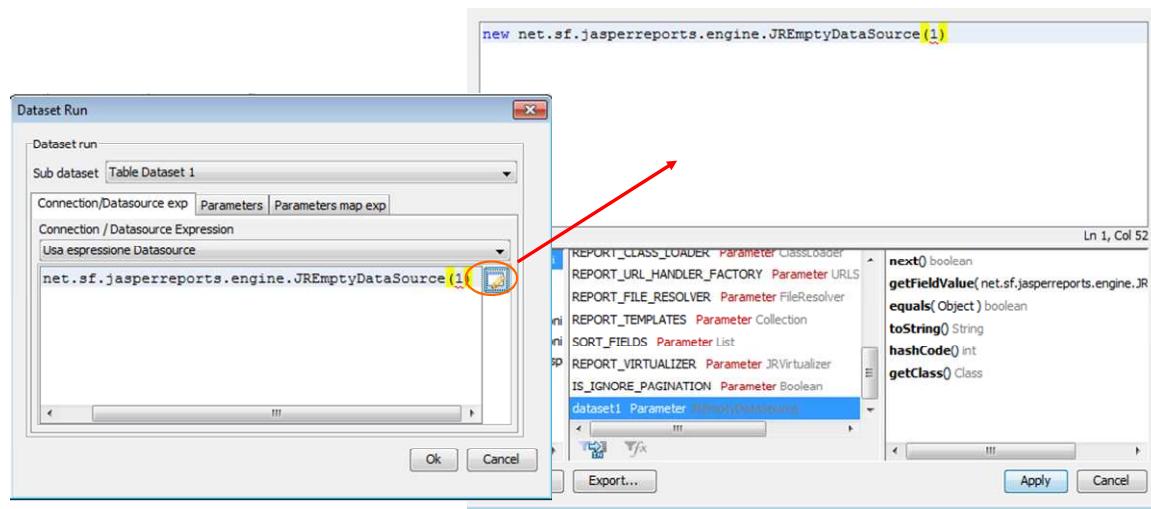


FIGURE 4.36 - Linking external dataset to Jasper report

Notice that the imported dataset may be a SQL or a QbE one. Since both types of datasets are stored in the same repository by SpagoBI Server, we are enabled to use any BM query in the development of a report.



Use of BM queries in report development

The ideal use of a business model is to define queries over the BM via SpagoBI Meta, deploy them on SpagoBI Server and reuse them on SpagoBI Studio as external datasets.

When you deploy a report with an external dataset, remember to set the dataset name in the deploy window. This allows SpagoBI to link the dataset defined on the Server to your report.

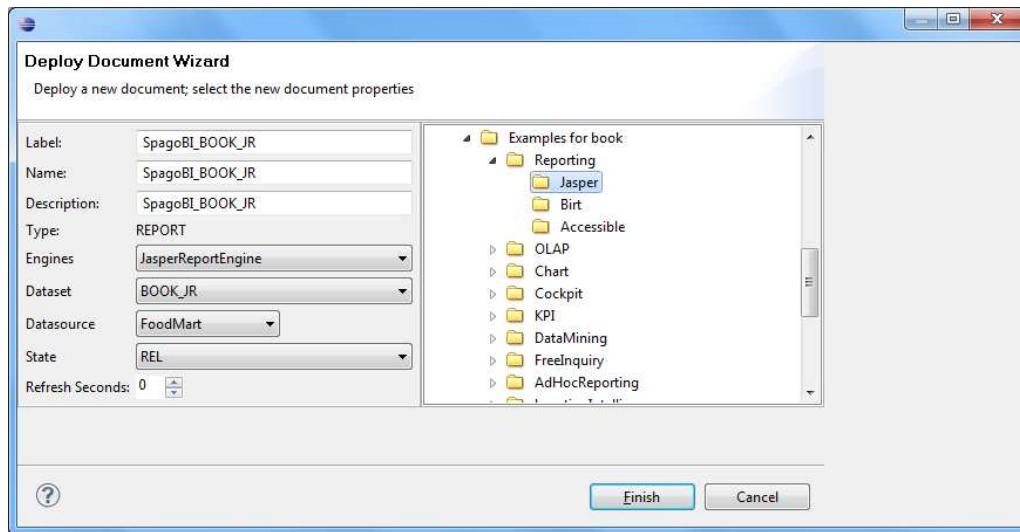
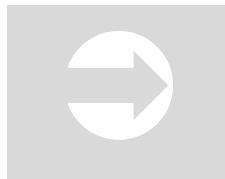


FIGURE 4.37 – Deploy a Jasper report with an external dataset

Adding parameters to reports

Most times reports show data analysis that depend on variable parameters, such as time. SpagoBI Jasper Report Engine allows the designer to add parameters to a report and link them to analytical drivers defined in SpagoBI Server.



Analytical Drivers

For a complete understanding of analytical drivers, please refer to section The Behavioral Model in chapter 5 – SpagoBI Server.

To use parameters, first add them to your report. Right-click on **Parameters** in the tree panel and select **Add Parameter**. Set the data type and choose a name for your parameter.

Parameters can be used within a dataset or referenced in the report, using the following syntax:

`$P{name_of_parameter}`



Parameters URI

Be careful when assigning a name to a parameter inside a report. This name must correspond to the parameter's URI when you deploy the document on SpagoBI Server.

Cross Navigation

Although cross-navigation is uniformly provided on all documents executed in SpagoBI Server, each type of document has its own modality to set the link pointing to another document.

Notice that the pointer can reference any SpagoBI document, regardless of the source document. For example, a Jasper report can point to a chart, a console, a geo or any other analytical document.

To allow cross-navigation in a Jasper report, you need to add a hyperlink to the element you want to be clickable. Most report elements can host an hyperlink. For example, let us add an hyperlink to a cell in the table of our previous example.



Cross Navigation

For further information on cross navigation and how it works, please refer to the corresponding section in chapter 5 – SpagoBI Server.

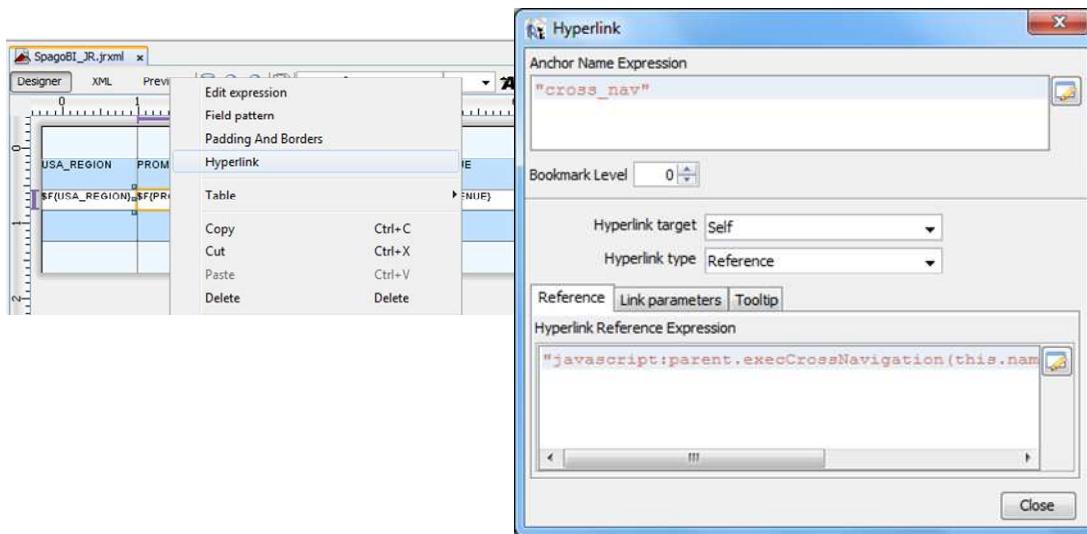


FIGURE 4.38 – Add a hyperlink for cross-navigation to a Jasper report

Right-click on the table cell and select **Hyperlink** in the contextual menu. The hyperlink editor will open and show three input fields:

- **Anchor name:** the name of the hyperlink
- **Hyperlink target:** select **Self**
- **Hyperlink type:** select **Reference**
- **Reference:** the actual link.

Set the hyperlink for cross-navigation as a string, according to the following syntax:

```
"javascript:parent.execCrossNavigation(this.name,
'CrossNav_Destination_Doc','&Param_1='"+$F{Param_1}+"&Param_2+
"&Param_3=Fixed_val" + "','','','self');"

// CrossNav_Destination_Doc is the label of the destination
document
// Param_1 is assigned the value of the parameter Param_1
// Param_2 is assigned the current value of a table cell (column)
// Param_3 is assigned a fixed value
```

The syntax of the string is fixed, while you need to set the appropriate label of the destination document, as well as assign values to the parameters that will be

passed to the destination document. The JavaScript editor may help you to insert dataset column bindings and report parameters automatically.

Chart

SpagoBI Studio provides a graphical editor for creating and configuring a wide range of charts.

Unlike reports, which may have internal or external datasets, charts are always linked to external datasets, i.e., defined on SpagoBI Server. Datasets can either be queries or business inquiries, which are queries over a metamodel (in other words, a business model built using SpagoBI Meta).



Business Model Query

Please refer to chapter 3 – SpagoBI Meta for more details on how to define and query a business model.



Dataset Definition

To learn how to build a dataset of type “query”, please refer to section Data Sets in chapter 5 – SpagoBI Server.

In both cases the dataset must be created on the Server before starting to draw the chart. The dataset will be actually linked to the chart template only at deploy time.

Nevertheless, it is important to have a clear understanding of the dataset and its columns before starting to develop the chart. Data must be returned in a coherent format according to the specific type of chart. Moreover, column names can be used to customize the template, to show categories and series, as explained below.

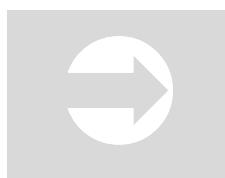


Deploy on SpagoBI Server

Please refer to section Download and Deploy in this chapter for more details on chart deployment.

Datasets can also include parameters, like any other SpagoBI document. Since datasets are not managed directly by the dashboard editor but on SpagoBI Server only, parameters are linked to the corresponding analytical drivers when the document is created on the Server.

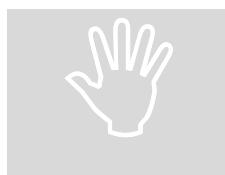
Once parameters have been linked to the document via the dataset, you can use them in the chart: for example, you can reference them in the title or pass their value via cross navigation to another document.



Analytical Drivers

To learn how analytical drivers are linked to document parameters, please refer to section Analytical Model in chapter 5 – SpagoBI Server.

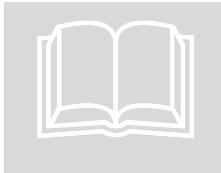
The chart editor allows the configuration of most graphical options for charts. It may still happen that more advanced configuration features are needed. In this case, you can open the chart template with the XML editor and manually edit it. Right-click on the chart item and select **Open with > Chart Editor**. In the XML editor you can make modifications to the template and save it.



Interleaving manual and automatic editing

If you open a template that you have previously edited by hand, manual edits may not be preserved after saving it in editor modality.

The following sections offer an overview of most features provided by the graphical editor.



Highcharts

Since the Highcharts library will be discontinued starting 4.0 release, we omit the documentation, which will be out of date soon.

Designer for SpagoBIJFreeChartEngine

To create a new chart document, right-click on the **Business Analysis** folder and select **Chart**. Then select **Chart with JFreechart**. The chart wizard allows you to select the type of desidered chart among the available ones:

- Dial chart
- Bar chart
- Pie chart
- Cluster chart
- Box chart
- XY chart
- Scatter chart.

Select a type of chart in the drop down menu inside the wizard and you will see its preview. Once you have chosen a name and a type, click on **Finish**. A new .sbichart file is created under the **Business Analysis** folder and the editor panel opens.

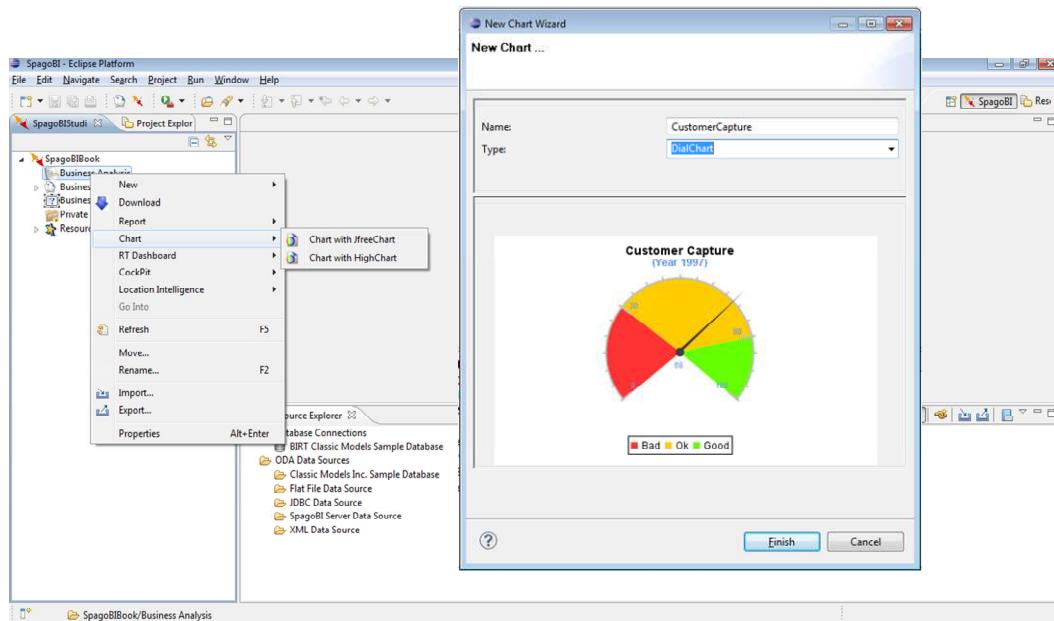


FIGURE 4.39 – Chart creation and chart wizard

The editor includes different sections. Some sections are common to all charts, while other ones are used by some of them only.

Common Sections

The following sections are common to all charts.

▪ Chart Information

It allows you to choose title and subtitle, chart sub-type (based on the type of chart) and background color.

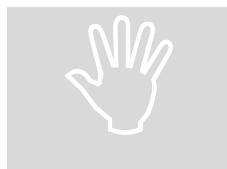


Chart sub-type

By changing the chart sub-type, all other editor sections are updated. Remember to save your input data before changing the sub-type!

In both title and subtitle you can insert parameters as well as profile attributes, according to the following syntax:

```
$P{parameter_name}  
 ${profile_attribute_name}
```

When the document is executed, the placeholder (as defined above) will be replaced with the actual value of the parameter/profile attribute. For example, if your chart shows the historical analysis concerning a specific year (given as parameter), you can show the year in the title.



Profile attributes

To learn more on profile attributes, please refer to section The Behavioral Model in chapter 5 – SpagoBI Server.

▪ **Chart Dimension**

It allows you to set the pixel dimension of your chart.

▪ **Chart Style**

It allows you to set the style of some elements contained in your chart. The set of elements may vary according to the type of chart and it includes: title, subtitle, x and y axes, value labels.

▪ **Chart Configuration**

It allows you to set configuration parameters. Given that parameters are highly specific to chart types, details are provided for each type of chart in the remainder of the section.

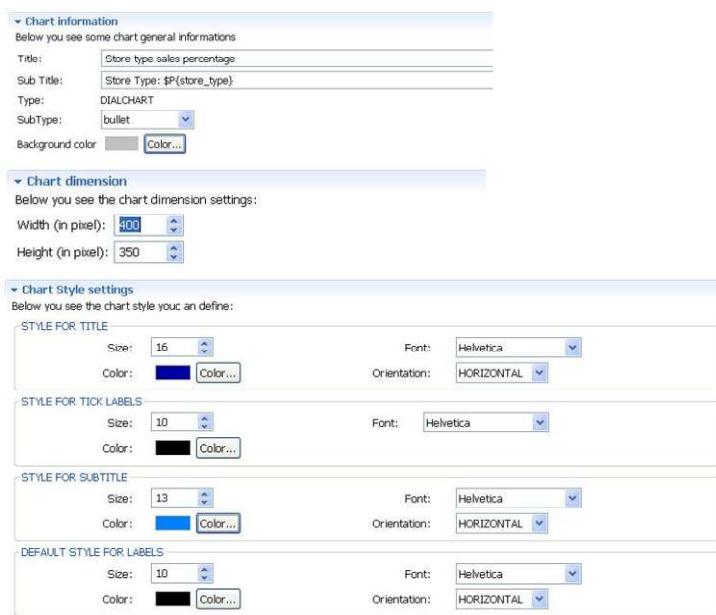


FIGURE 4.40 – Chart information, dimension and style settings

▪ Drill (cross navigation)

Setting drill values allows cross navigation. This option is available for some types of charts: linkable bar, stacked bar and linkable pie, with some differences. Drill parameters are summarized below:

Drill parameters	
Document label	The label of the target document: this is the unique document label in SpagoBI Server
Category URL name	Name (URL) of the parameter corresponding to the category linked in cross navigation
Series URL name	Name (URL) of the parameter corresponding to the series linked in cross navigation (bar chart only)
Add parameters (to be propagated)	
Name	Name (URL) of the parameter to be propagated in cross navigation

Type	Absolute	Does not depend on the specific document execution. Absolute parameters are passed to the target document with the value you manually assign to it in the editor area.
	Relative	Does not have a pre-defined value. It will be assigned the value of the analytical driver whose name is written in the Value cell.
Value	Absolute parameter	Actual value that will be passed to the target document
	Relative parameter	Name of the analytical driver providing the dynamic value for the parameter
Special Parameters		
Title (optional)	Title to be given to the destination document. By default it is the name of the document	
Target (optional)	Self	The target document is opened within the same window and breadcrumbs are updated
	Tab	If executing in a browser, the target document is opened in a new tab. Otherwise, same as Self.
	Update	Useful when source and target documents are the same, to update the document with new values

TABLE 4.1 - Drill parameters

■ Series Intervals

This section is only available for charts, since it follows the concept of series or interval (e.g., bar charts). It allows you to configure series options like color, series title, axis and other drawing details.

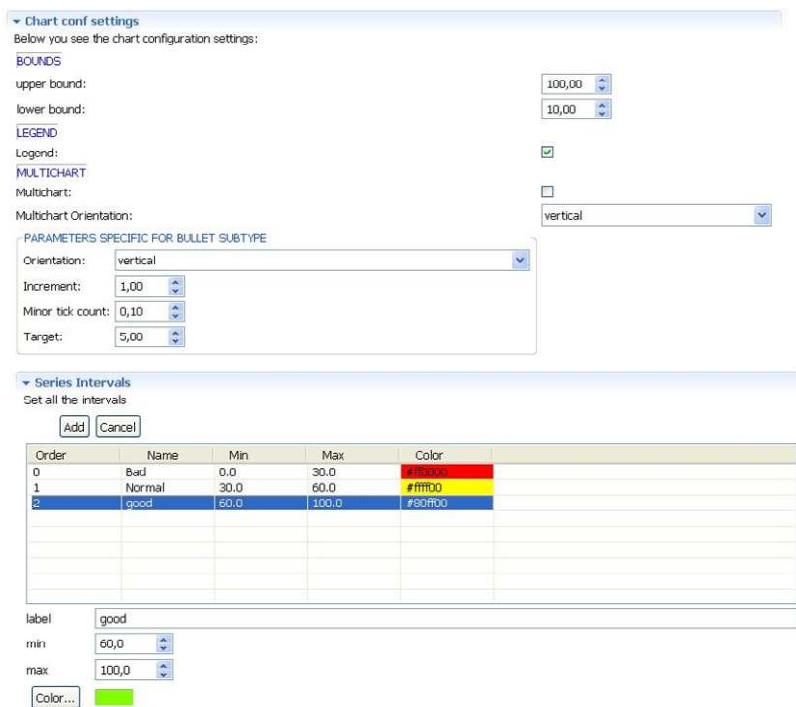
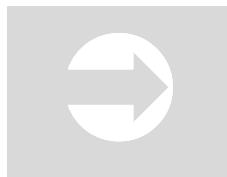


FIGURE 4.41 – Configuration Settings and Series Intervals

▪ Dataset Metadata (Read Only)

This table includes a summary of the datasets associated to the document, including name and type of each dataset column. It may be useful, for example, when you need to customize specific series.

Note that this section is read only. Therefore, it shows information on the dataset only if this information has already been generated on the Server. This requires that: a dataset has been associated to the chart document on SpagoBI Server, and the dataset itself has been tested at least once on the Server.



Deploy on SpagoBI Server

Please refer to section Download and Deploy in this chapter for more details on chart deployment.



Associate Datasets to Documents

Please refer to chapter 5 – SpagoBI Server for Dataset creation and testing, and association to documents (behavioral model).

▪ **Dataset structure**

In addition to these configuration sections defined in the graphical editor, each type of chart requires the dataset to be specified according to some criteria. This allows the engine to correctly interpret the dataset. In the remainder of the section, the dataset structure is described for each type of chart.



Dataset definition for charts

The dataset linked to a chart must be properly defined. Please refer to section SpagoBIJFreechartEngine at Chapter 6 – Analytical engines, for a more detailed explanation of dataset structures.

Now let us see all chart types in detail and how to set specific options for each of them.

Dial Chart

Dial charts show a value on a given scale, within an interval defined by a maximum and a minimum value and possibly sub-intervals. They can be useful, for example, to show performance indicators or any other single measure having reference values.

Some setting options are common to all dial charts in SpagoBI Studio, namely:

- **Lower** – the minimum value of the scale

- **Upper** – the maximum value of the scale
- **Increment** – the interval between two ticks
- **Minortickcount** – the number of sub-ticks between two ticks
- **Legend** – if true, the legend will be shown.

Style configuration for charts include: title and subtitle (see Style Configuration above), as well as tick and value labels.

For all dial charts a graphical designer is available to set the intervals within the absolute minimum and maximum value (see Series Intervals above). As shown in FIGURE 4.41, each interval is characterized by:

- a label
- a minimum and a maximum value (for the considered interval)
- a color.

Clicking on the **Add** button, you can create a new interval, which you can later modify clicking on the row. The only exception is the *Thermometer* chart, where intervals (if any) must be 3 and have pre-defined labels (see below).

- **Dataset structure**

The dataset has the following structure:

```
SELECT 30 as value from dual
```

Where the column returning the value to be drawn in the dial chart must be just one and must be called **value**.

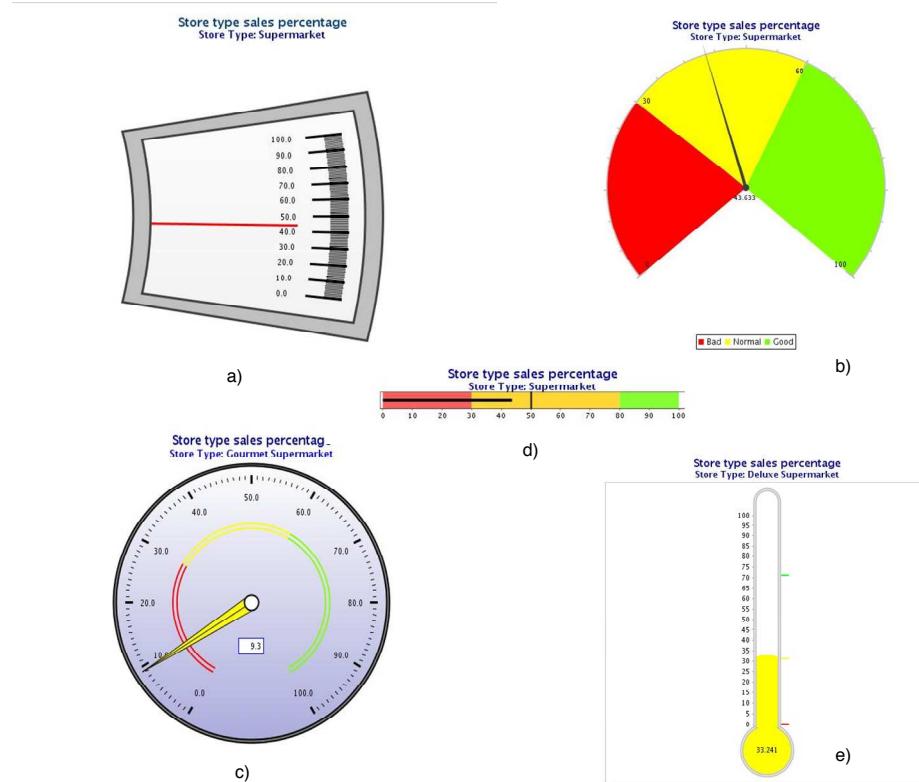


FIGURE 4.42 – Dial charts: a) Simple Dial b) Meter Dial c) Speedometer d) Bullet Dial and e) Thermometer

SpagoBI Studio JFreechart Designer offers different types of charts, some of which with additional configuration parameters:

- **Simple Dial.** Here you can additionally define the chart orientation: horizontal or vertical.
- **Meter Dial.**
- **Speedometer.**
- **Bullet Dial.** You can additionally set a target value.
- **Thermometer.** As said above, in this chart you cannot freely define intervals: there are exactly three intervals, whose labels must have the following values: **normal**, **warning** and **critical**. The editor automatically sets the number and labels of the intervals, if the user adds a new one.

Bar Chart

Bar charts are designed on two orthogonal axes. The x axis shows the *categories* of a chart, i.e., points on the scale values have been calculated on. For example, if a chart shows monthly values, then categories will be months.

Values distributed over categories belong to one or more *series*. For example, a chart showing daily store sales and costs has two different series. The y axis serves as a reference scale for actual values in a series.

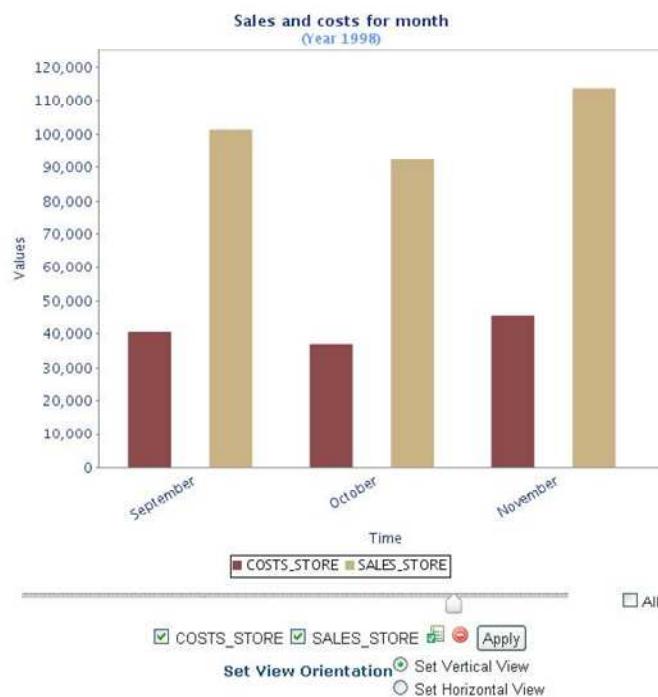


FIGURE 4.43 – Simple bar chart with months as categories and two value series

The JFreechart Designer offers several different bar charts, with specific features and options. A number of settings is common to all bar charts, as follows:

- **Chart Configuration**

This section includes several options, which are grouped by the editor into sub-sections for improved readability.

Filters	
Filter categories	If checked, it enables to filter categories using a slider
Filter categories group	If checked, it enables to filter category groups using a slider
Filter series	Allows the filtering of single series using a checkbox
Show filter series button	If true, the options “All” and “None” for filtering series will be shown
Number of visible categories	If greater than the total number of categories, the slider is not shown
Dynamically change the number of visible categories	The number of visible categories can be modified on the chart by the user
Number of visible series	Number of series that will be shown
Legend	
Draw legend	If true, the legend will be shown
Legend position	Possible options are: left, up, bottom, right
Slider	
View slider	If true, the slider will be shown (provided that the number of visible categories is less than the total)
Slider position	Possible options: top, bottom
Slider starts from end	If true, at first execution the slider is set to the end
Others	
Name of category label	Label for the category axis
Name of value label	Label for the value axis
Maximum bar width	Sets a maximum width for bars (bar width is dynamically calculated)
Only integer values on value axis	If true, only integer values will be shown
Position of value axis	By default on the left (or at the bottom for a horizontal layout). It can be forced to right (or top)
Show value labels	If true, values will be shown with the bars
Enable tooltips	If true, tooltips are enabled. Tooltips must be returned from the dataset.

Orientation	Possible options: vertical, horizontal
-------------	--

TABLE 4.2 - Chart configuration options

■ Chart Style

Recall from paragraph Common Sections that the editor allows the configuration of style settings. General style options include: title and subtitle.

In the specific case of bar charts, style configuration options also include labels for: default label style, X and Y axes style, and value labels style.

■ Series Labels Parameters

Here you can customize the visualization of series in your chart. For each series you want to visualize, click on **Add Series** and insert the corresponding name, as returned by the dataset. This will allow the linkage of the chart to the dataset.

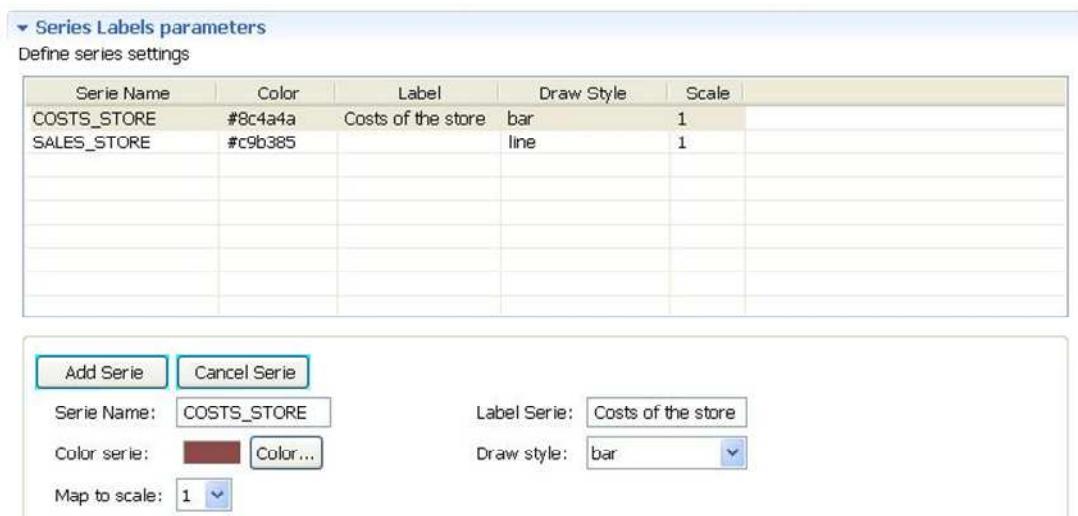
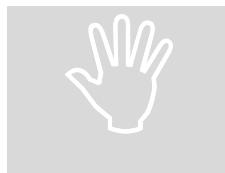


FIGURE 4.44 – Series labels configuration

Depending on the specific bar chart sub-type, you can set additional options. For example the color of the series, the label that will be shown in the chart and the type of drawing (available for bar-line charts only).



Adding series to a chart

When you add a series to a chart, you link the chart to the dataset associated to the chart. Remember that the series must be returned by that dataset and that their names must correspond.

▪ Dataset structure

The dataset must have the following structure:

```
SELECT 'January' as x, 2000 as SALES_STORE, 1500 AS COSTS_STORE,
500 as REVENUE FROM dual
UNION
SELECT 'February' as x, 2000 as SALES_STORE, 1700 AS
COSTS_STORE, 300 as REVENUE FROM dual
UNION
SELECT 'February' as x, 2500 as SALES_STORE, 1700 AS
COSTS_STORE, 800 as REVENUE FROM dual
```

Where:

- Column **x** represents categories (on the X axis)
- Other columns become series
- The optional column **cat_group** allows the grouping of categories. If filters on group categories are active, setting this column enables group-based filtering.

So far we have seen common options for all types of bar charts. In the remainder of this section we will provide an overview of the available bar charts.

Simple Bar

This is the basic bar chart. In addition to all configuration and style options described above, you can add text to series bars using the **Add label**.

Linkable Bar

This is the linkable version of the simple bar, i.e., enabled for cross navigation. Clicking on a bar will redirect you to another SpagoBI document, passing the information related to the selected category and/or series.



Cross Navigation

Please refer to section Cross Navigation in chapter 5 – SpagoBI Server for a full explanation of this mechanism.

If you select the Linkable Bar type, you will be prompted with the Drill parameters section. The graphical editor collects relevant information to build the link for cross navigation.

Drill parameters

Set all the drill parameters

Document Label:	Detail_report_for_store
Category Url Name:	month
Serie Url Name:	store

ADD PARAMETER

Name	Value	Type
year	year	RELATIVE
state	ITA	ABSOLUTE
title	Report detail from cross.	ABSOLUTE
target	self	ABSOLUTE

Parameter Name: title
 Parameter Value: Report detail from cross
 Parameter Type: ABSOLUTE

FIGURE 4.45 – Set cross navigation from a bar chart document

Stacked Bar (with Cumulate option)

This type of chart shows bars as stacked, instead of side by side. This chart type is linkable, so the editor will show the Drill parameters section.

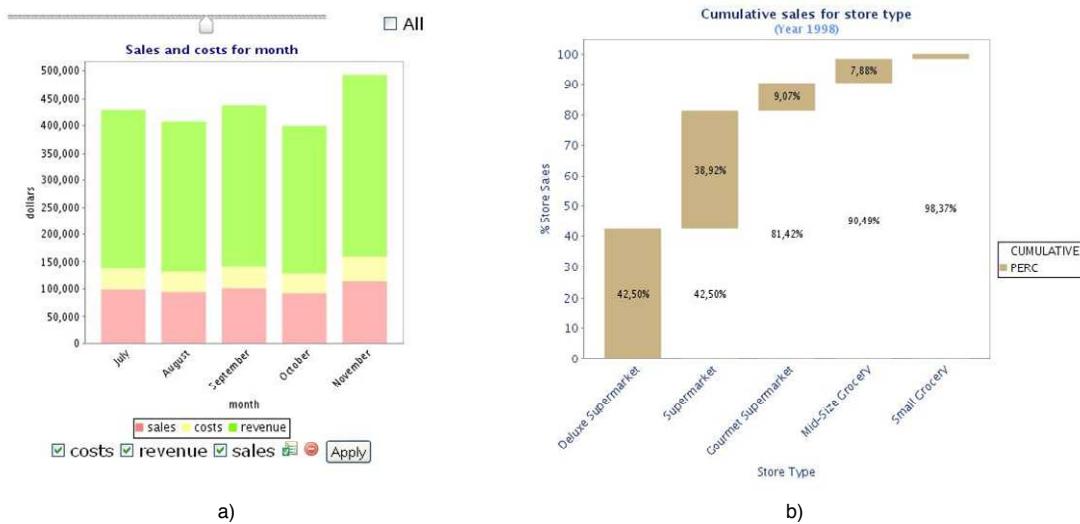


FIGURE 4.46 – Stacked bar charts: a) basic and b) with cumulate and percentage options

In addition to all configuration parameters for bar charts, this sub-type allows the user to set further options and create more advanced charts.

Stacked bar configuration options	
Cumulative	Add a cumulative series
Make percentage	If selected, values are calculated in percentage
Show percentage values	Show values in percentage
Value label position	Set value labels inside or outside bars
Add label	Add text to labels. Text must be returned by the dataset

TABLE 4.3 - Stacked bar configuration options

The Series configuration section also includes an additional table, where you can specify colors for series by clicking on the **Add colour** item. Note that this setting overrides any color setting defined in the common section.

Stacked Bar Group

This is similar to the stacked bar but it also allows to define sub-categories. Additional configuration options include:

Stacked bar group configuration options	
Subcategory label	The label for sub-categories
Show percentage values	Show values in percentage
Number of visible groups	Number of groups that will be shown
Number of series per group	Number of series for each group

TABLE 4.4 - Stacked bar group configuration options

Similarly to the stacked bar, the Series configuration section includes an additional table, where you can specify colors for series by clicking on the **Add colour** item. Note that this setting overrides any color setting defined in the common section.

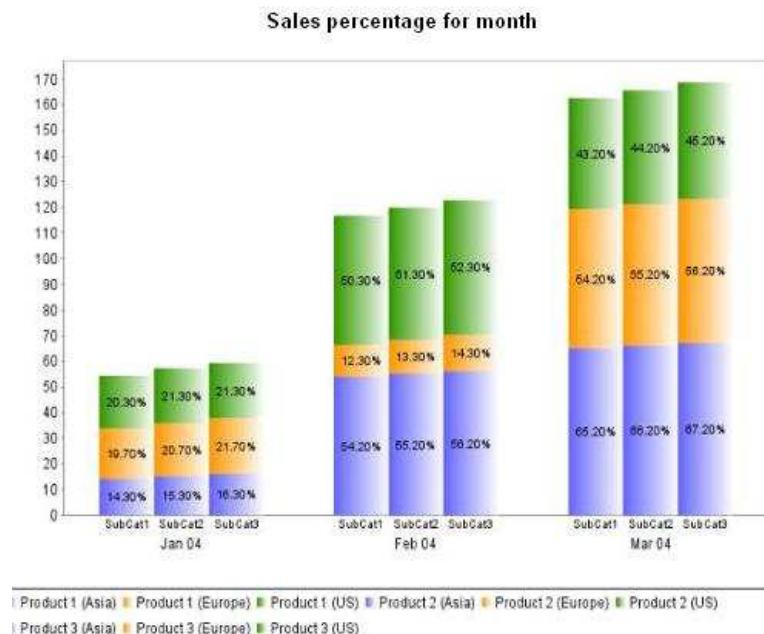


FIGURE 4.47 – Stacked bar group with sub-categories

Overlaid Bar Line

The distinctive feature of this bar chart type is that it can represent series as either bars or lines. To provide appropriate reference axis for both types of value representation, you can add a secondary Y axis and choose to which one each value series will refer. As shown in FIGURE 4.44, for each series you shall specify the type of representation (bar o line) and the reference axis.

Additional configuration parameters for this type of bar chart are:

Overlaid bar line configuration options	
Second axis label	The label for the secondary Y axis. In case of no value, the axis will not be drawn.
Add label	Add text to labels. Text must be returned by the dataset
Stacked bar renderer	If true, bar series referring to the first axis will be shown as stacked
Stacked bar renderer 2	If true, bar series referring to the second axis will be shown as stacked
Value label position	Set value labels inside or outside bars

TABLE 4.5 - Overlaid bar line configuration options

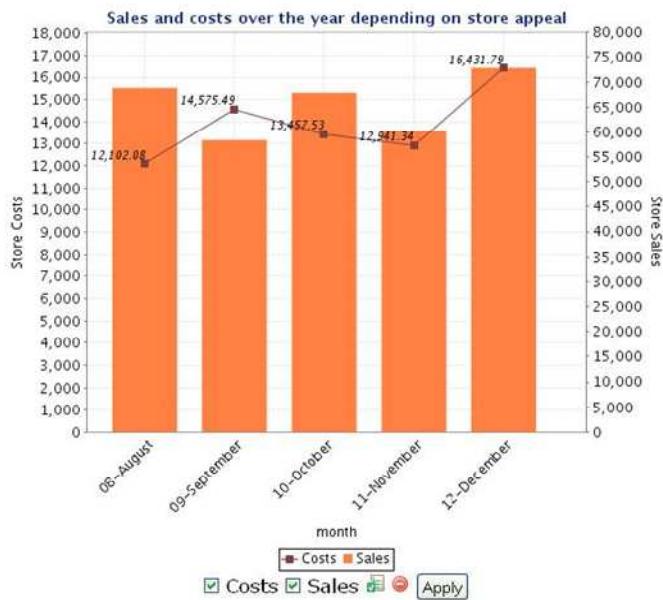


FIGURE 4.48 – Overlaid bar line chart with double Y axis

Pie Chart

Pie charts show values associated to a set of categories, with respect to the total value, represented by the whole pie.

▪ Chart Configuration

Using the Chart information panel (described in the Common Sections paragraph) you can set the chart title, subtitle and category labels. Additional configuration parameters include:

Pie chart configuration options	
How to view values	Value or percentage
How many dimensions	Possible options: 2D or 3D

TABLE 4.6 - Pie chart configuration options

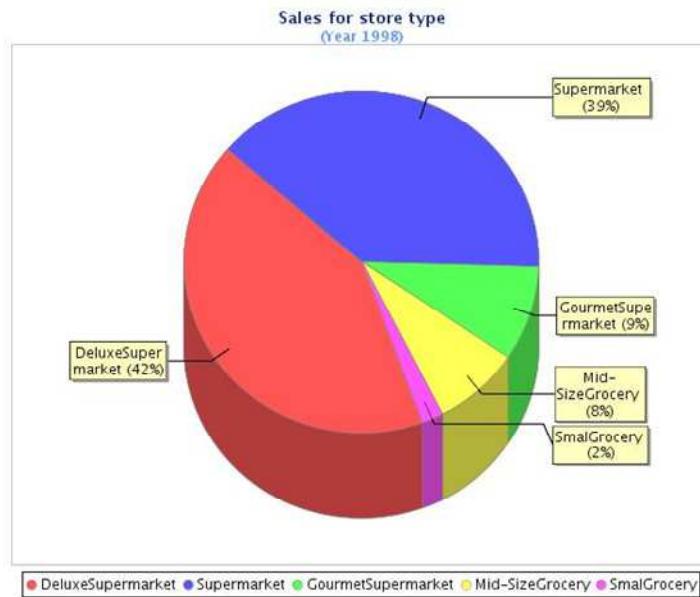


FIGURE 4.49 – Simple pie chart in 3D

▪ Dataset structure

The dataset must have the following structure:

```
select 30 as X, 20 as Y, 50 as Z from dual
```

Where each column represents a category (i.e., a slice of pie).

There are two possible types of chart.

Simple pie chart

This is the basic pie and can be customized with the above-mentioned options.

Linkable Pie

This is the linkable version of the simple pie, i.e., enabled for cross navigation. If you select this sub-type, you will be prompted with the Drill parameters section. Drill parameters are described in the Common Sections paragraph.

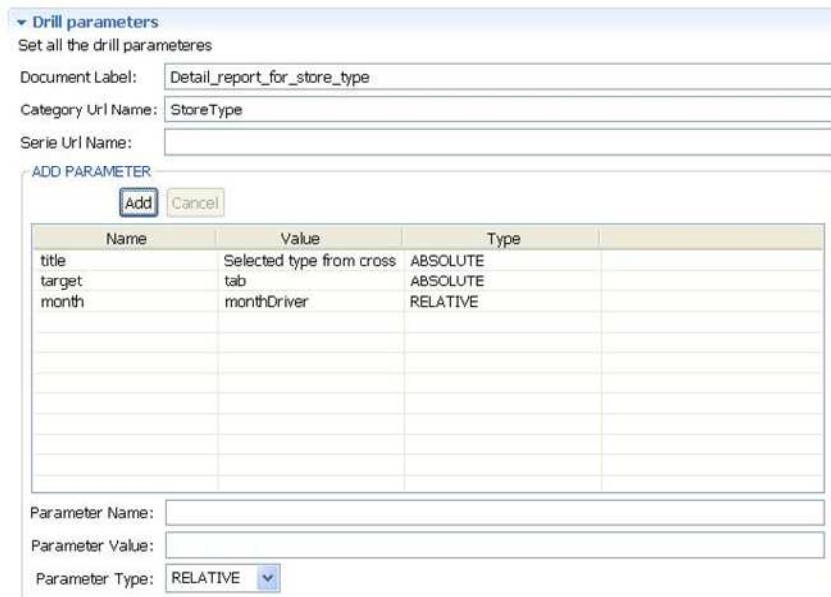
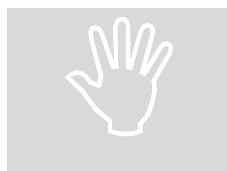


FIGURE 4.50 – Drill parameters section for a linkable pie chart



Cross Navigation in Linkable Pies

Note that pie charts only have one series. Therefore, the only parameters passed to the target document is the category.

Cluster Chart

Cluster charts represent clusters, i.e., sets of discrete values that can be grouped together according to some criteria. Cluster charts have two axes: one for categories (X) and the other one for values (Y). Values can be divided into series.

For example, you can define clusters of customers based on the number of unit sales over time, for a specific type of good.

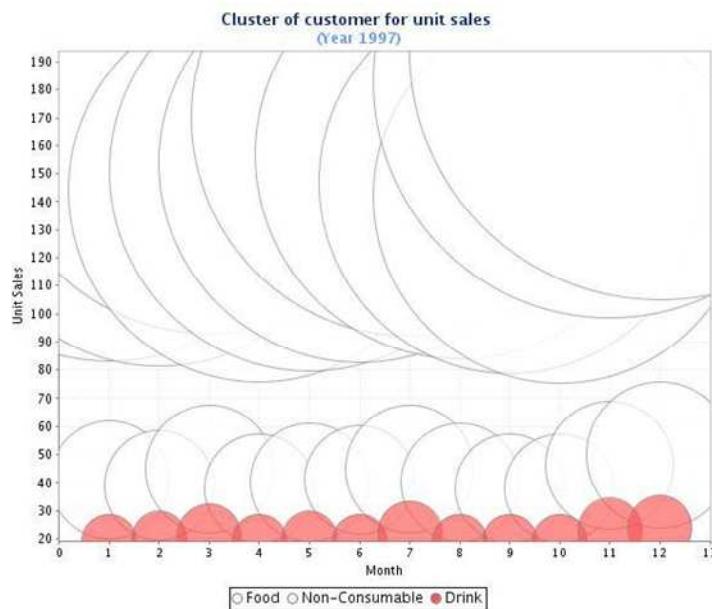


FIGURE 4.51 – Simple cluster chart with one series selected (red)

▪ Dataset Structure

The dataset should have the following structure:

```
select 'TRUE' is_sel, 'Drink' series_name, 1 x, 15000 y, 0.8 z
from dual
union
select 'TRUE' is_sel,'Drink' series_name, 2 x, 12000 y, 0.7 z
from dual
union
select 'TRUE' is_sel,'Drink' series_name, 3 x, 14500 y, 0.6 z
from dual
union
select 'FALSE' is_sel, 'Food' series_name, 1 x, 15300 y, 0.1 z
from dual
union
select 'FALSE' is_sel, 'Food' series_name, 4 x, 11000 y, 0.3 z
from dual
union
select 'FALSE' is_sel, 'Food' series_name, 2 x, 12100 y, 0.5 z
from dual
```

Where:

- Column X defines a point on the X axis

- Column Y defines a point on the Y axis
- Column Z defines the size of clusters
- Column `series_name` defines the series (where the name can be freely chosen)
- There is also an optional boolean column, whose name is specified in the template (in our example it is called `is_sel`). This column states whether the series should be selected: if true, the series is selected, if false it is not. If not specified, series will be assigned default colors.

- **Chart Style**

Using the Chart Style panel (see Common Sections) you can set the title, subtitle, X and Y axes labels and a default label.

- **Chart Configuration**

There is only one type of cluster chart, the Simple Cluster Chart. Its specific configuration parameters are:

Simple cluster chart configuration options	
X axis label	The label for the category axis
Y axis label	The label for the value axis
Choose a column	The dataset column used to define whether the series will be visible
Default color	Color for all non selected series

TABLE 4.7 - Simple cluster chart configuration options

Box Chart

A box chart depicts groups of numerical data through their five-number summaries: the sample minimum, lower quartile, median, upper quartile, and the sample maximum.

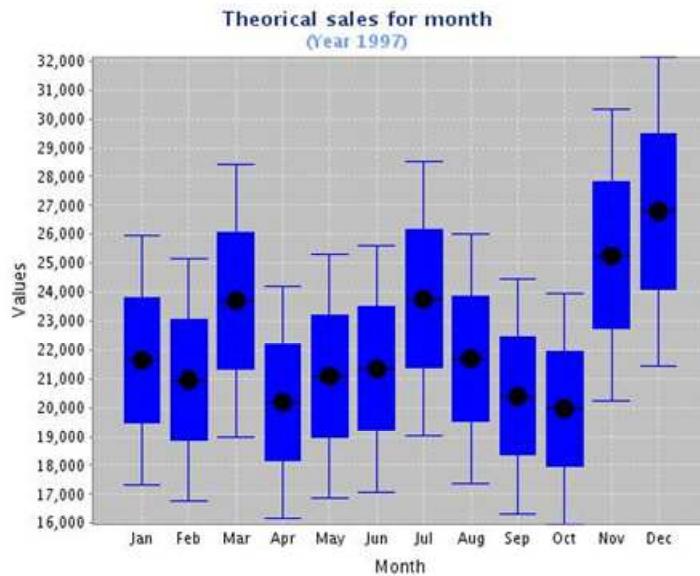


FIGURE 4.52 – Simple box chart

There is only one type of box chart, called Simple Box Chart, with the following configuration options.

▪ Chart Style

Using the Chart Style panel (see Common Sections), you can set the title, subtitle, X and Y axes labels and a default style for the axes' labels.

▪ Chart Configuration

Simple box chart configuration options	
X axis label	The label for the category axis
Y axis label	The label for the value axis
Number of visualization	Number of series to be visualized

TABLE 4.8 - Simple box chart configuration options

▪ Dataset structure

The dataset must have the following structure:

```
select '0-10' as x, 2 as ser1, 3 as ser2, 4 as ser3, 2 as ser4, 5
as ser5 from dual
```

```

UNION
select '11-20' as x, 3 as ser1, 1 as ser2, 2 as ser3, 4 as ser4,
7 as ser5 from dual

```

Where:

- X is the category (mandatory)
- Other columns represent series and their names can be set freely.

Scatter Chart

Scatter charts allows the visualization of values depending on two variables. Values are represented as points on a Cartesian space, where the position of a point on the x axis is given by the value of the first variable, and the position on the y axis is given by the second variable.

For example, in FIGURE 4.53 each point is a store: on the x axis we show the repartition of sales (in percentage) with respect to the average sales value (on the y axis).

- **Chart Style**

Using the Chart Style panel (see Common Sections) you can set the title, subtitle, X and Y axes labels and a default label style for X and Y.

- **Chart Configuration**

Configuration parameters for the scatter chart are:

Scatter chart configuration options	
X axis label	The label for the category axis
Y axis label	The label for the value axis
Number of visualization	Number of series to be visualized
Default color	Color for all non selected series
View annotation	Add text
Draw legend	If true, the legend will be shown

Legend position	Position of the legend
-----------------	------------------------

TABLE 4.9 - Scatter chart configuration options

■ Dataset structure

The dataset must have the following structure:

```
select 'Deluxe Supermarket' as x, 3 as x0, 4 as y0, 10 as x1, 5
as y1 from dual
union
select 'Gourmet Supermarket' as x, 3 as x0, 4 as y0, 4 as x1, 2
as y1 from dual
union
select 'Supermarket' as x, 3 as x0, 4 as y0, 9 as x1, 7 as y1
from dual
```

Where:

- Column x defines the series
- The list of columns called X_i , with $i = 0, 1, 2, \dots$ and corresponding Y_i define positions on the chart.

There are two sub-types of scatter chart.

Simple Scatter Chart

This is the basic version and can be configured using the parameters described above.

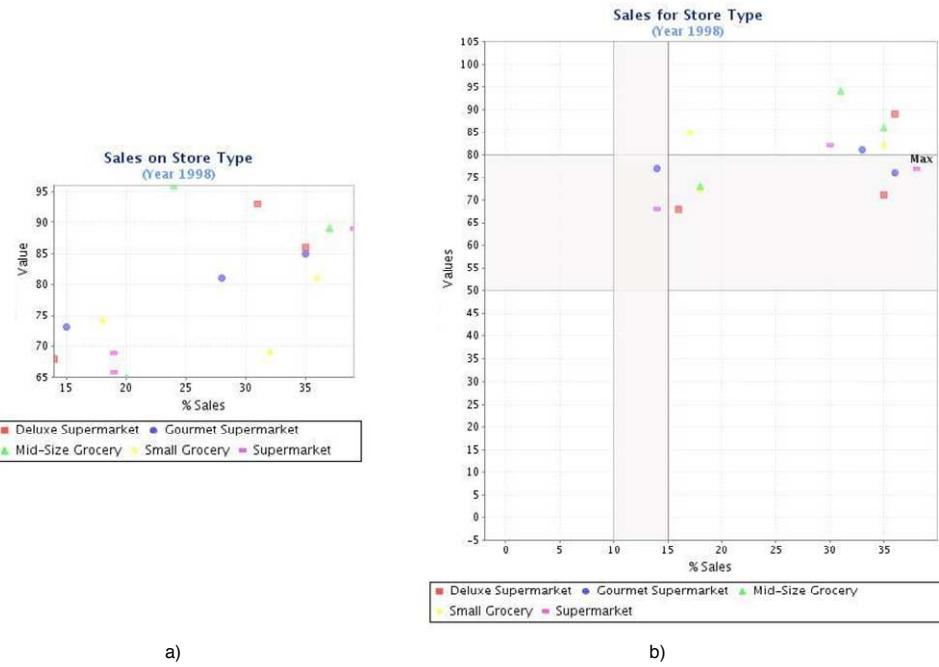


FIGURE 4.53 – Scatter charts: a) simple and b) with markers

Marker Scatter Chart

In addition to the simple scatter chart, it allows you to define range and markers, to inscribe values within pre-defined regions.

Additional configuration options for the definition of markers and range are:

Markers scatter chart configuration options	
X axis range	Minimum and maximum for the x axis
Y axis range	Minimum and maximum for the y axis
X axis marker	Includes: min, max, value, label and color
Y axis marker	Includes: min, max, value, label and color

TABLE 4.10 - Markers scatter chart configuration options



FIGURE 4.54 – Configuration of markers and range for a scatter chart

XY Block Chart

The Block Chart represents blocks whose color depends on a third dimension (Z). It is also called XY because it is based on a Cartesian representation.

The XY Block chart has the following configuration options.

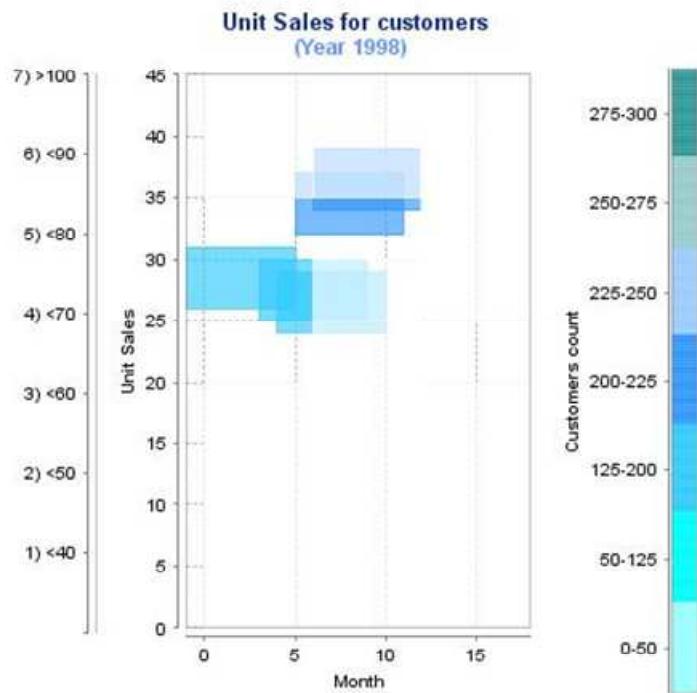


FIGURE 4.55 – Block chart

▪ Chart Style

Using the Chart Style panel (see Common Sections), you can set: title, subtitle, X and Y axes labels, and range labels.

▪ Chart Configuration

Configuration parameters for the scatter chart are:

Block chart configuration options	
X axis label	The label for the X axis
Y axis label	The label for the Y axis
Z axis label	The label for the Z axis (represented with colors)
X range	
Low value for x range	Minimum for X axis range
High value for x range	Maximum for X axis range
Y range	

Low value for y range	Minimum for Y axis range
High value for y range	Maximum for Y axis range
Z range	
Low value for z range	Minimum for Z axis range
High value for z range	Maximum for Z axis range
Label	Label for the range
Color	Color for the range
Grid width	Width of blocks
Grid height	Height of blocks

TABLE 4.11 - Block chart configuration options

▪ Dataset structure

The dataset must return one column for each dimension (x, y, z) as follows:

```
select 5 as x, 20000 as y, 125 as z from dual
union
select 10 as x, 20000 as y, 249 as z from dual
```

Real time

Designer for SpagoBIDashboardEngine

SpagoBI Studio allows you to create real time dashboards using the graphical editor. This avoids the burden of manually defining the template, although you can always create real time documents by hand and deploy them on SpagoBI Server.

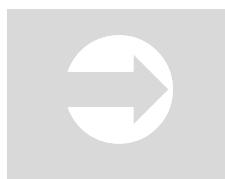


Designer for real time documents

SpagoBI Studio currently supports the design of real time dashboards. A wizard for the real time console will be available in the near future.

Similarly to charts, real time dashboards are linked to external datasets only, i.e., defined on SpagoBI Server. Datasets can either be queries or business inquiries,

which are queries over a metamodel (in other words, a business model built with SpagoBI Meta).



Business Model Query

Please refer to chapter 3 – SpagoBI Meta to see how to define and query a business model.



Dataset Creation

To learn how to build a dataset of type “query”, please refer to section Data Sets in chapter 5 – SpagoBI Server.

In both cases the dataset must be created on the Server before starting to develop the dashboard. The dataset will be actually linked to the dashboard template only at deploy time.



Deploy on SpagoBI Server

Please refer to the Download and Deploy section in this chapter for chart deployment.

It is important to have a clear understanding of the dataset and its columns before starting to develop the dashboard. Data must be returned in a coherent format with the specific type of dashboard. In addition, dataset columns can be used to customize the style and configuration of the dashboard.

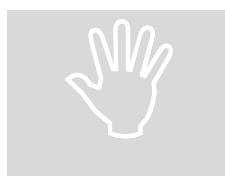
Datasets can also include parameters, like any other SpagoBI document. Since datasets are not managed directly by the dashboard editor but on SpagoBI Server only, parameters are linked to the corresponding analytical drivers when the document is created on the Server.



Analytical Drivers

To learn how analytical drivers are linked to document parameters, please refer to section Analytical Model in chapter 5 – SpagoBI Server.

The dashboard editor allows the configuration of most graphical options for charts. It may still happen that more advanced configuration features are needed. If this is the case, you can open the chart template with the XML editor and manually edit it. Right click on the chart item and select **Open with > Dashboard Editor**. In the XML editor you can make changes to the template and save it.



Interleaving manual and automatic editing

If you open a template that you have previously edited by hand, manual edits may not be preserved after saving it in editor modality.

To create a new chart document, right-click on the **Business Analysis** folder and select **RT Dashboard > Dashboard basic component**. The dashboard wizard allows you to select the type of dashboard among those available:

- Rotation
- Live lines
- Live Table.

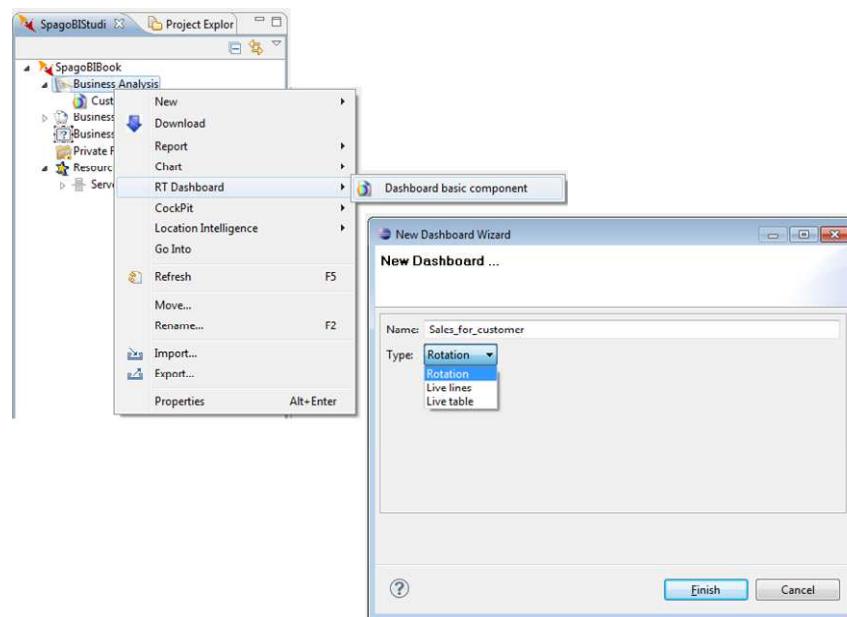


FIGURE 4.56 – Create a new real time dashboard

Common Sections

The following sections are common to all dashboard components.

▪ Dashboard Information

This section shows a read-only summary of dashboard name and type.

▪ Dashboard Internal Settings

This section offers different configuration options based on the component type you have selected. The common options to all three subtypes are:

Dashboard configuration options

Internal area width	Width of the internal area
Internal area height	Height of the internal area
Refresh rate	Refresh time (in ms) of the value

TABLE 4.12 - Dashboard configuration options

■ Dashboard Dimensions

Here you can set the dimensions of the dashboard in pixels.

■ Dataset structure

In addition to these configuration sections defined in the graphical editor, each type of chart requires the dataset to be specified according to some criteria. This allows the engine to correctly interpret the dataset. In the remainder of the section, describe the dataset structure for each type of chart.



Dataset definition for dashboards

The dataset linked to a chart must be properly defined. Please refer to section SpagoBIDashboardInternalEngine of chapter 6 – Analytical Engines for a more detailed explanation of dataset structures.

▼ Dashboard information
 Below you see the Dashboard type and associated movie:
 Type: Rotation
 Movie: /dashboards/rot.lzx.swf

▼ Dashboard dimension
 Below you see the Dashboard dimension settings:
 Width (in pixel):
 Height (in pixel):

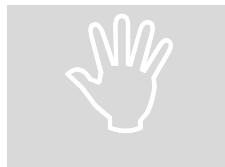
▼ Dashboard internal settings
 Below you see the Dashboard internal settings:
 Name of the value column:
 Dashboard internal area width:
 Dashboard internal area height:
 Refresh rate (in ms):
 Minimum value:
 Low circular sector upper edge value:
 High circular sector lower edge value:
 Maximum value:
 Low circular sector color: █
 Medium circular sector color: █
 High circular sector color: █
 Number of main divisions:

FIGURE 4.57 – Dashboard editor: fixed sections (on the left) and specific settings (on the right)

In the following we provide details about each dashboard sub-type and its characterizing options.

Rotation

The rotation dashboard, commonly defined as speedometer, shows a value within a pre-defined range, possibly divided into sub-intervals. The value is dynamically refreshed to reflect real time variations.



Speedometers

The speedometer is also available as a dial chart. However, its value is set at first execution, while in the real time speedometer it is dynamically updated.

The specific configuration parameters for the rotation dashboard component are summarized below. They are also shown in FIGURE 4.57.

Rotation dashboard configuration options	
Minimum value	Minimum value of the interval scale
Maximum value	Maximum value of the interval scale
Low circular sector upper edge value	Threshold between the lower section and the middle section
High circular sector upper edge value	Threshold between the middle section and the upper section
Low circular sector color	Color of the lower section
Medium circular sector color	Color of the middle section
High circular sector color	Color of the upper section
Number of main divisions	Number of sections in the dashboard

TABLE 4.13 - Rotation dashboard configuration options

▪ Dataset structure

The dataset must return one column whose name is **value**, as follows:

```
SELECT 2.3 AS value FROM dual
```

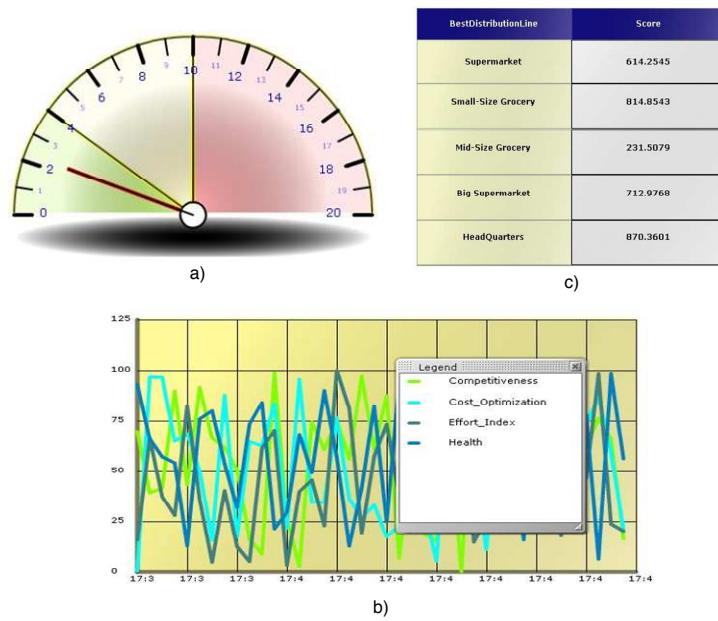


FIGURE 4.58 – Dashboard components: a) Rotation b) Live line and c) Live table

Live Line

The specific configuration parameters for the live line are explained below:

Live line configuration options	
Start time	Start time for the x axis (hh:mm) Default is the current time
Stop time	Stop time for the x axis (hh:mm) Default is the current time
X axis grid step	Distance between ticks on the X axis
Y axis grid step	Distance between ticks on the Y axis
X axis minimum value	Minimum value on the X axis
X axis maximum value	Maximum value on the X axis

TABLE 4.14 - Live line configuration options

Dashboard internal settings	
Below you see the Dashboard internal settings:	
Dashboard internal area width:	500
Dashboard internal area height:	300
Refresh rate (in ms):	15000
Start time (hh:mm) (current time if not specified):	
Stop time (hh:mm) (calculated if not specified):	
X axis grid step (hh:mm):	
Number of X axis grid steps:	10
Y axis minimum value:	0
Y axis maximum value:	200
Y axis grid step:	25
Color of line 1:	■ Color...
Color of line 2:	■ Color...
Color of line 3:	■ Color...
Color of line 4:	■ Color...

Dashboard internal settings	
Below you see the Dashboard internal settings:	
Dashboard internal area width:	300
Dashboard internal area height:	180
Left column name:	xLabel
Refresh rate (in ms):	15000
Number of table rows:	5
Rows spacing:	0
Header height (in % respect to table):	20
Left column cells background color:	■ Color...
Other columns cells background color:	■ Color...
Background opacity gradient:	0.5
Cells opacity gradient:	0.3
Header font size:	12
Cells font size:	10

a)

b)

FIGURE 4.59 – Dashboard settings for: a) live line and b) live table

▪ Dataset structure

The dataset must return the value that each line has at a given moment, such as:

```
Select 90 as Effort_Index, 75 as Competitiveness, 55 as
Cost_Optimization, 70 as Health from dual
```

Live Table

The specific configuration parameters for the live line are shown in FIGURE 4.59 and explained below:

Live table configuration options	
Left column name	Name for the column on the left
Number of rows	Number of rows in the table
Rows spacing	Spacing between rows
Header height (%)	Height of the table header in percentage wrt. table
Left column cell background color	Background color for the left column

Other cells background color	Background color for other columns
Background opacity gradient	Opacity gradient of the background
Cells opacity gradient	Opacity gradient of cells
Header font size	Font size for the header
Cells font size	Font size for cells

TABLE 4.15 - Live table configuration options

The dataset must return the name and the value for each row in the table:

```
select 'Supermarket' as BestDistributionLine, 614.245 as Score
from customer where customer_id = 1
union
select 'Small-Size Grocery' as BestDistributionLine, 814.853 as
Score from customer where customer_id = 1
```

Note that column names must match the Left column name defined in the configuration options above.

Cockpit

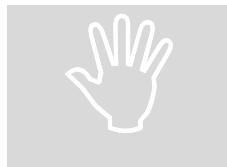
Designer for SpagoBICompositeDocEngine

With SpagoBI Studio you can design composed documents (also known as cockpits) via a graphical interface, without having to deal with the XML template.

In order for SpagoBI to create a composed document, all composing documents must be deployed both locally and on the Server so that documents' metadata are visible. To check that metadata are defined, right click on the single document, select **Properties**, then **SpagoBI** in the window that will open.

In case you updated the document on the server, e.g., adding a parameter, refresh metadata locally by clicking **Refresh** in the above-mentioned **SpagoBI Metadata** tab.

Composing documents may be already on the Server or they can be created on the Studio from scratch. In the first case, you need to download them from the Server. In the second case, you need to deploy the document on the Server.



Deploy composing documents

Before starting to build your cockpit, make sure that all composing documents are deployed locally and on the server, and that documents' metadata are visible. If a document has no metadata associated, it cannot be added to the cockpit.

To create a new cockpit, right-click on the **Business Analysis** folder and select **Cockpit > Composed document**. This will open an editor where you can choose the name for your document. The new document will be created under the **Business Analysis** folder. Double click on the document to open the editor.

Building the document requires the following steps:

- Define the layout and size of the cockpit
- Insert the composing documents in the layout.

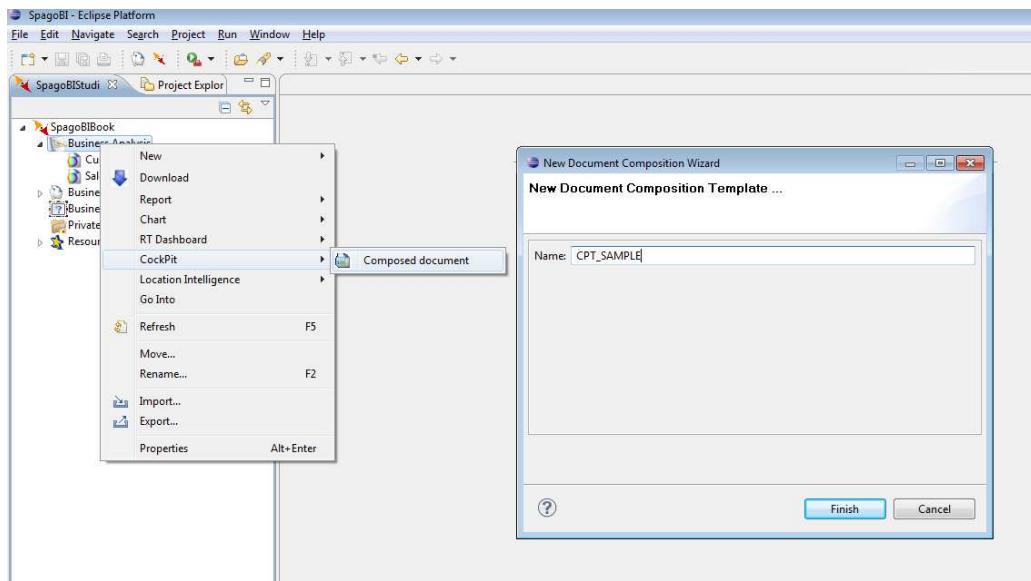


FIGURE 4.60 – Create a composed document in SpagoBI Studio

To define the layout of a composed document, you basically need to add empty spaces (*Containers*), define their size and position, and fill these spaces with composing documents.

Right-click on the document area and select **Add Doc**. A new container will appear in the editor area with a number. You can drag the container to place it in the right place and you can create new containers. If you want to resize a container, right-click on it and select **Resize**. You can also delete the container by right-clicking and selecting **Delete Container**.

The next step is to fill each container with an actual document. Pick a document under the **Business Analysis** folder and drop it in the chosen container. Remember that one container can host one document only: if you want to add a document you must first add a new container. Once documents are added, save and deploy the cockpit.

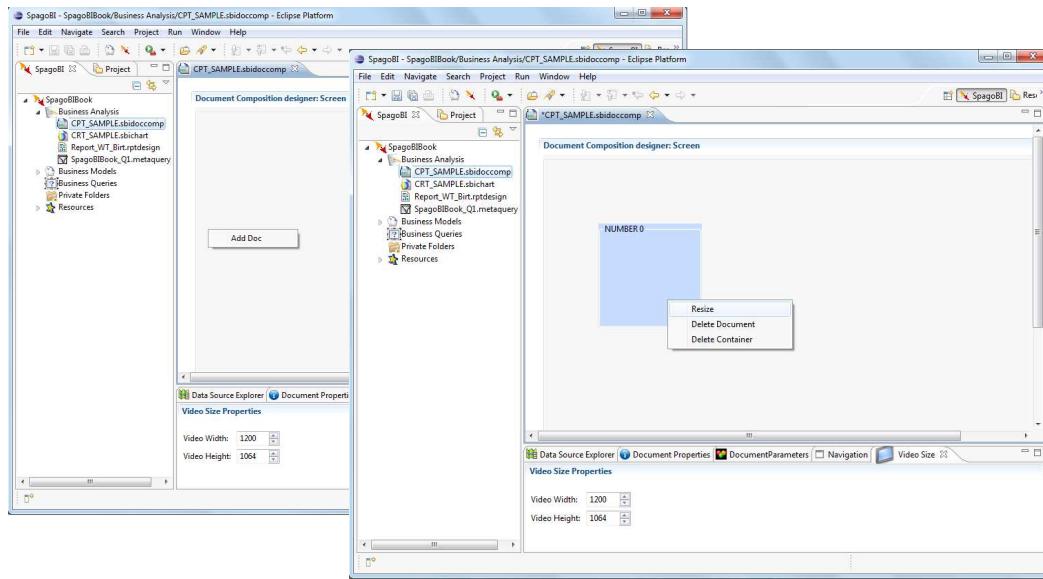


FIGURE 4.61 – Step 1: Create a container in the editor area



Deploy on SpagoBI Server

Please refer to section Download and Deploy in this chapter for more details on cockpit deployment.

So far we have created a cockpit, which is a static document: data are shown separately in each composing document and there is no way to navigate the document. It would be more interesting to explore data via logical links from one document to another.

For example, supposing we wish to select a brand from the table on the left and see data about sales and costs refreshed for that specific brand (see FIGURE 4.62).

Now we need to add one or more *internal navigation* paths. First of all, we should carefully think about the path we wish to create among data. In other words, which data should flow from one document to the other(s) to contextualize the analysis.

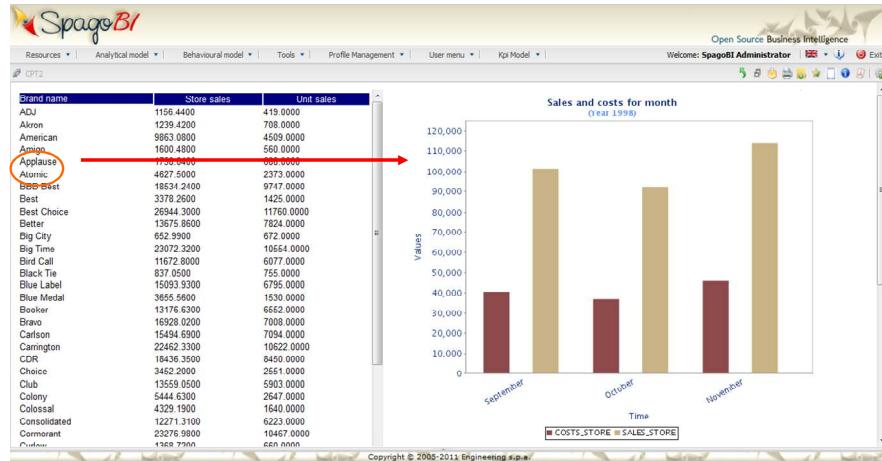


FIGURE 4.62 - Internal navigation path in a composed document



Designing cockpit internal navigation

Before starting to implement internal navigation, take some time to design it logically. This is important as it determines the visual impact and analysis effectiveness of the cockpit.

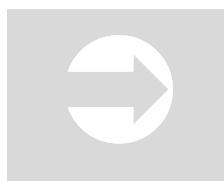
Once this path is logically defined, look for the actual parameters and analytical drivers allowing you to implement this data flow. This process is similar to adding parameters to a single document, but it is more complex because input parameters for one document are provided by another (or multiple) documents in a same cockpit.

The definition of parameters and analytical drivers requires the following steps:

- Insert parameters in each composing document
- Define analytical drivers for each parameter on the Server
- Modify each composing document on the Server to have the proper analytical drivers associated
- Update metadata in the Studio editor to be sure that analytical drivers have been correctly associated to each composing document
- Define output parameters in each composing document

- Set navigation paths among documents.

Let us go back to the cockpit example and add a navigation path among the composing documents. We will show a very simple navigation path with one parameter only. However, you can add as many parameters as you wish, with different navigation directions.



Manual configuration of internal navigation

Cockpit internal navigation can be manually set by editing the cockpit template. See the section Cockpit in chapter 5 – SpagoBI Server.

First of all, we add a parameter called **Brand** to the second report (the bar chart on the right). We modify the dataset accordingly, to return sales and costs for a given brand. In this case, we will use the BIRT designer since we are dealing with a BIRT report. Depending on the type of your report, you shall modify it using the appropriate editor. Finally we deploy the modified document on the Server.



Parameters and analytical drivers

Please refer to section Analytical model, in chapter 5 – SpagoBI Server, for a detailed explanation of how to create analytical drivers and associate them to documents.

Afterwards, access the Server and create the new parameter. Associate it to the document and link it to the appropriate analytical driver on the Server. If the driver is not available on the Server, create a new one from scratch. Test the report as a stand-alone document and if everything is fine, switch back to the Studio.

After updating the document on the Studio, refresh the document metadata in the **Properties > SpagoBI > Document Metadata** tab. Here a parameter should be visible in the **Document Parameters** tab, as shown in FIGURE 4.63.

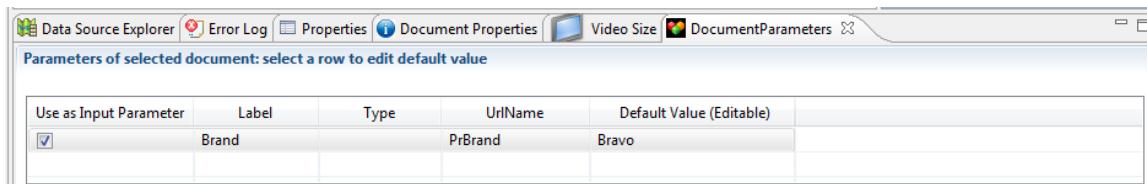
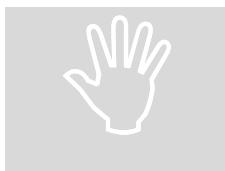


FIGURE 4.63 - Document Parameters tab showing parameters associated to a document



Adding parameters to navigate a cockpit

You can add a parameter to a stand-alone document using SpagoBI Studio. However, if you want to use it to internally navigate a cockpit, you must first deploy the document and associate the parameter to it on the Server.

Generally speaking, when we create a composed document we may reuse components that already exist as single documents. As a consequence, you shall manage the parameters that are already defined in these documents, some of which may be useful to navigate the cockpit, while some others may not (even though they are still needed to execute the single document correctly).

If a document composing a cockpit has one or more parameters, there are two possible choices:

- Include the parameter in the navigation path of the composed document
- Leave the parameter unused in the navigation path.

Both choices are implemented via the **Document Parameters** tab. To include a parameter you need to check it in the tab. Leave it unchecked if you don't want it to be propagated along the composed document.

If you include the parameter in the navigation, you can:

- Set a navigation path from/to this parameter or
- Assign a default value to the parameter.

In the latter case, write the value in the Document Parameters tab. In the first case, add a navigation in the **Navigation** tab (as described below). Note that the choice depends on the specific navigation you are designing for your cockpit.

A navigation path needs both a source and a destination: the former is the *output parameter*, the latter is the *input parameter*. In our example we open the **Document Parameters** tab to define: the output parameter **Brand** on the table (left) and the input parameter **Brand** on the bar chart (right).

The final step is to define actual navigation paths between documents, using the **Navigation** tab, as shown in FIGURE 4.64.

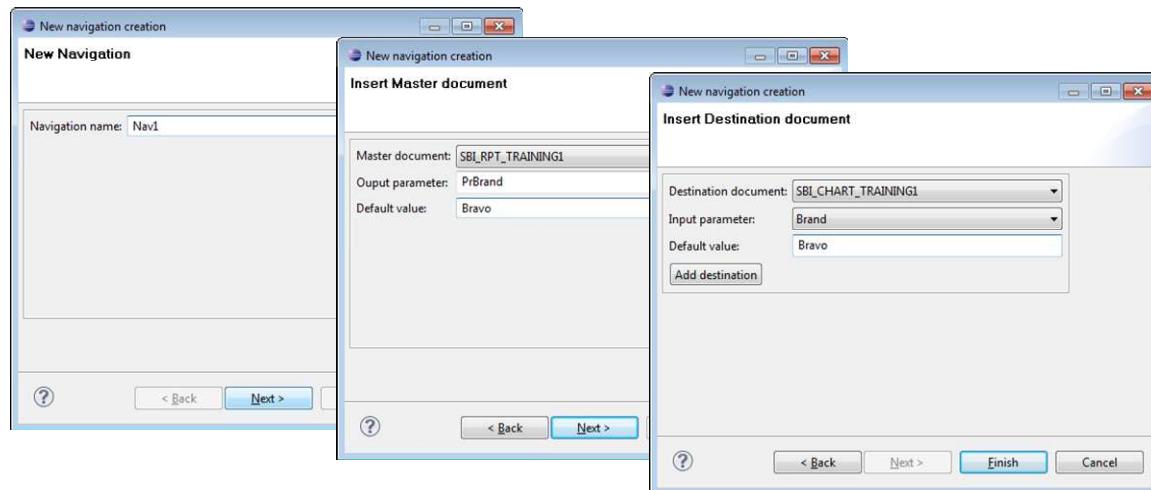


FIGURE 4.64 - Setting a navigation path between a source and a destination document of the cockpit

Finally, deploy the document on the Server and test it.

Location Intelligence

SpagoBI Studio provides a graphical editor supporting the development of analytical documents for location intelligence.

Geo-referential documents are always linked to external datasets, i.e., defined on SpagoBI Server. Datasets can either be queries or business inquiries, which are queries over a metamodel (in other words, a business model built with SpagoBI Meta).



Business Model Query

Please refer to chapter 3 – SpagoBI Meta to see how to define and query a business model.



Dataset Definition

To learn how to build a dataset of type query, please refer to section Data Sets in chapter 5 – SpagoBI Server.

In both cases the dataset must be created on the Server before starting to draw the document. The dataset will be actually linked to the template only at deploy time.

Nevertheless, it is important to have a clear understanding of the dataset and its columns before starting to develop the document, because column names are used to customize the template, and to show categories and series.

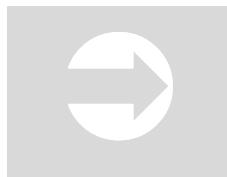


Deploy on SpagoBI Server

Please refer to section Download and Deploy in this chapter for GEO document deployment.

Datasets can also include parameters, like any other SpagoBI document. Since datasets are not managed directly by the editor but on SpagoBI Server only, parameters are linked to the corresponding analytical drivers when the document is created on the Server.

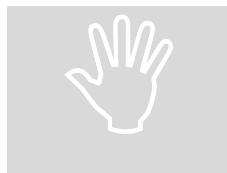
Once parameters have been linked to the document via the dataset, you can use them in the document: for example, you can reference them in the title or pass their value to another document via cross navigation.



Analytical Drivers

To learn how analytical drivers are linked to document parameters, please refer to section Analytical Model, in chapter 5 – SpagoBI Server.

The editor allows the configuration of most graphical options for georeferential documents. It may still happen that more advanced configuration features are needed. In this case, open the template with the XML editor and manually edit it. Right click on the chart item and select **Open with > Text Editor**. In the XML editor, make your changes to the template and save it.



Interleaving manual and automatic editing

If you open a template that you have previously edited by hand, manual edits may not be preserved after saving it in editor modality.

The remainder of this chapter describes how to develop documents using the editor for SpagoBI Geo Engine.



Editor for SpagoBI GIS Engine

A dedicated editor for GIS documents is not available yet, but it will be in the future. For the time being, you can develop GIS documents using SpagoBI Server.

Designer for SpagoBISGeoEngine

To create a new GEO document, right-click on the **Business Analysis** folder and select **Location Intelligence > Geo**. The document wizard allows you to select the name of your document. Once you click on **Finish**, a new **.sbigeo**

file is created under the **Business Analysis** folder. Double click on it and the editor panel opens.

The editor consists of three different sections.

- **Geo Designer**

It allows you to choose the dataset and its columns, define measures and choose the map with associated features.

- **Hierarchies**

It allows you to define hierarchies and levels.

- **Cross Navigation**

It allows you to set cross navigation parameters.

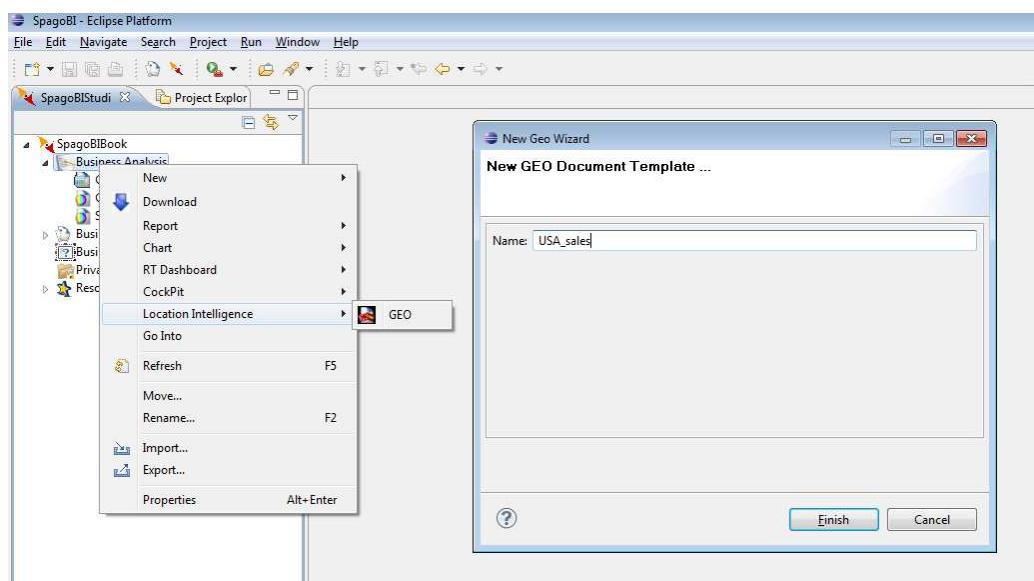


FIGURE 4.65 – Create a GEO document using SpagoBI Studio

The last section concerns the rendering of the map and includes three subsections:

- **GUI Settings – Windows**

To configure legend windows

- **GUI Settings – Param**

To set additional parameters on the map

- **GUI Settings – Labels**

To add labels, if needed.



Location Intelligence & GEO Analytical Documents

For a comprehensive discussion about location intelligence and the SpagoBI Geo Engine, please refer to the corresponding section in chapter 5 – SpagoBI Server.

The next paragraphs show how to build a GEO document using SpagoBI Studio designer, by discussing each section in detail.



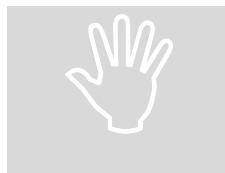
Geo Editor Sections

These sections are based on a specific logical order, i.e., which section you need to configure first when creating a document. For this reason, we may sometimes jump from one section to another.

GEO Designer

The designer is divided into two parts, which correspond to the main elements of a GEO document: the dataset and the map. On the left, you can select and configure columns from the dataset, while on the right you can select the map and its features.

Both the list of available datasets and the list of maps are downloaded from the Server. Therefore, datasets and maps must have been previously created/uploaded on SpagoBI Server in order to use them in the Studio.



Active connection to SpagoBI Server

Unlike all other editors, the GEO designer requires a working connection to the Server. This allows to retrieve datasets metadata and maps.

In the right panel, you can select a map from the map catalogue on the Server. Besides, the designer provides:

- An editable description of the map
- A check box to set this map as the default one
- A check box to choose the base color of the map.

Let us move onto the left panel. When you select a dataset, you must choose which column will be the *geo-identifier* (geo id) for a given feature in the map. The geo id represents the column in the dataset matching the selected feature in the map. This allows the plotting of dataset measures on the map. For example, if we choose **region_id** as the geo id and we choose the feature **States** in the map, we are plotting sales values on states (see FIGURE 4.66).

The screenshot shows the GEO designer interface with two main panels: 'Data Set' and 'Map'.

Data Set Panel:

Column name	Type	Select	Aggregation mode
region_id	java.lang.Long	geoid	sum
store_country	java.lang.String		
sales_state	java.lang.String		
sales_region	java.lang.String		
store_city	java.lang.String		
store_sales	java.math.BigDecimal	measure	sum
unit_sales	java.math.BigDecimal	measure	sum
store_cost	java.math.BigDecimal		

Once hierarchies are set:
right click on 'measure' typed column to add KPI settings
right click on 'geoid' typed column to add granularity level.

Map Panel:

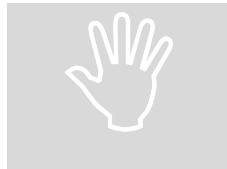
Feature name	Description	Select Default	Default Color
State_Colors		<input type="radio"/>	<input checked="" type="checkbox"/> Color
States		<input checked="" type="radio"/>	<input type="checkbox"/> Color
State_borders		<input type="radio"/>	<input type="checkbox"/> Color
Country_borders		<input type="radio"/>	<input type="checkbox"/> Color
Labels_State_Names		<input type="radio"/>	<input type="checkbox"/> Color
centroids_states		<input type="radio"/>	<input type="checkbox"/> Color

FIGURE 4.66 – Geo Designer

To help the BI developer in this task, columns are automatically filled according to the selected dataset. For each column the following data are shown:

- Name

- Type
- A combo box to define the column as the geo-identifier or a measure
- A combo box to select the aggregation function for measures.



Dataset metadata

The designer needs to access the dataset metadata stored on the Server. To avoid errors, it is advisable to execute a preview of the dataset on the Server before using it in the GEO designer.

If you right click on a measure you will be prompted with a window, where you can set a number of configuration options for that measure:

Measure Configuration Parameters		
isDefault		Set the measure as default
Description		Textual description
Aggregation function		Aggregation function on the map
Threshold type		Choose the type of threshold and its value (range or groups_number, depending on the type)
	Static	The range parameter contains intervals separated by commas, e.g., “0,256,512”
	Percentual	The range parameter contains the percentage of intervals wrt. Total, e.g., “30,20,30”
	Uniform	The GROUPS_NUMBER parameter specifies how many intervals with uniform distribution to define
	Quantile	The GROUPS_NUMBER parameter specifies how many intervals with quantile distribution to define
Threshold param value		Value of the threshold parameter. Name changes based on threshold type.

Threshold lb value	Minimum value for the threshold. May be “none”. Lower values are considered “outbound”
Threshold ub value	Maximum value for the threshold. May be “none”. Upper values are considered “outbound”
Colors type	Options: static and gradient
Colors param value	Color, either static or base for the gradient
Colors outbound	Color for out-of-range values
Colors null value	Color for null values

TABLE 4.16 - Measure Configuration Parameters

Right clicking on the dataset column that has been set as geo id, another pop up window will open. Here you can choose to which hierarchy and level you want to associate that column. Since we have not defined hierarchies yet, we need to move to the next section of the editor.

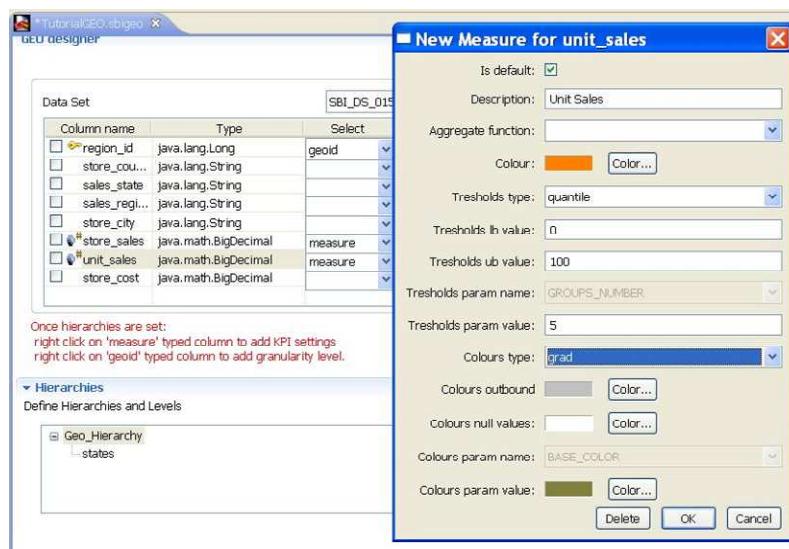


FIGURE 4.67 – Measure configuration in the GEO deisgner

Hierarchies

A GEO document is based on the definition of one or more *hierarchies*. Hierarchies are composed of different *levels*, which allow to view the map (and data plotted on it) from a coarse to a fine level of granularity. For example, we can define a hierarchy with three levels: the first is the state, the second is the region and the third is the province.

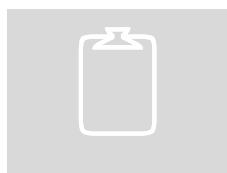
You can create a new hierarchy using the dedicated panel. Right click and select **New Hierarchy**. Then you can edit the hierarchy by selecting **New Level**. For each level you have to specify:

- A name
- The dataset column to which the level should be bound
- A description
- The map feature to which the level should be associated.

Note that you need to first specify both the dataset columns and the map features before defining a level.

Now we have to link the hierarchy level to the dataset column in the **Geo Designer** section as well. In particular, the column referenced as geo id must be associated to a hierarchy and a level within this hierarchy.

To do so, go back to the **Geo Designer** and right-click on the column geo id. In the pop up window, link the column to the desired hierarchy level.



How to proceed

First of all, choose the dataset and the map. Select the geo id column and the map features. Then, create a hierarchy and associate it to the geo id and the feature. Finally, go back to the geo id column, right click and link it to the hierarchy.

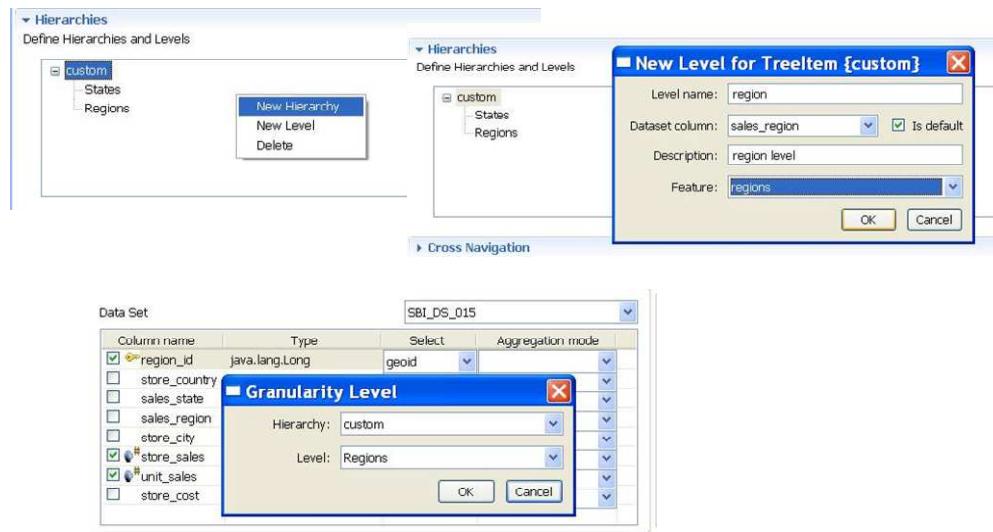


FIGURE 4.68 – Creation of a hierarchy for a GEO document

Cross Navigation

Once defined hierarchies and levels, you can decide to add a cross navigation link to a specific level. This means that the measure value corresponding to the map feature at that level will be passed to the target document.



Cross Navigation

For further information on cross navigation and how it works, please refer to the corresponding section in chapter 5 – SpagoBI Server.

Click on the icon to add a new link, select a hierarchy and a level from the drop down menus. Click on the icon to add a new parameter, on the icon to view defined ones and on the icon to delete one parameter.

For each parameter you should specify:

- Name
- Type: absolute or relative

- Value
- Scope: **dataset** if the value should be read from the dataset, **environment** if the value is passed to the Geo engine.

Some parameters have a special meaning and are not mandatory:

Special parameters for cross navigation		
Document label	The label of the target document: this is the unique document label in SpagoBI Server	
Title (optional)	Title to be given to the destination document. By default it is the name of the document	
Target (optional)	Self	The target document is opened within the same window and breadcrumbs are updated
	Tab	If executing in a browser, the target document is opened in a new tab. Otherwise, same as Self.
	Update	Useful when source and target documents are the same, to update the document with new values

TABLE 4.17 - Special parameters for cross navigation

GUI Settings

This part of the editor allows you to customize the graphical layout of your GEO document. Graphical settings are optional. If not set, the document will be displayed with default settings.

The Windows editor is used to set visibility and configuration options for windows that can be displayed with the map. A **Defaults** box allows the definition of default settings for all windows. Each box contains configuration options for a specific window, namely:

- **Measures**
- **Legend**
- **Navigation**

- Layers
- Color-picker.

Within each box, a set of pre-defined configuration parameters can be selected from a drop down menu and assigned a value.

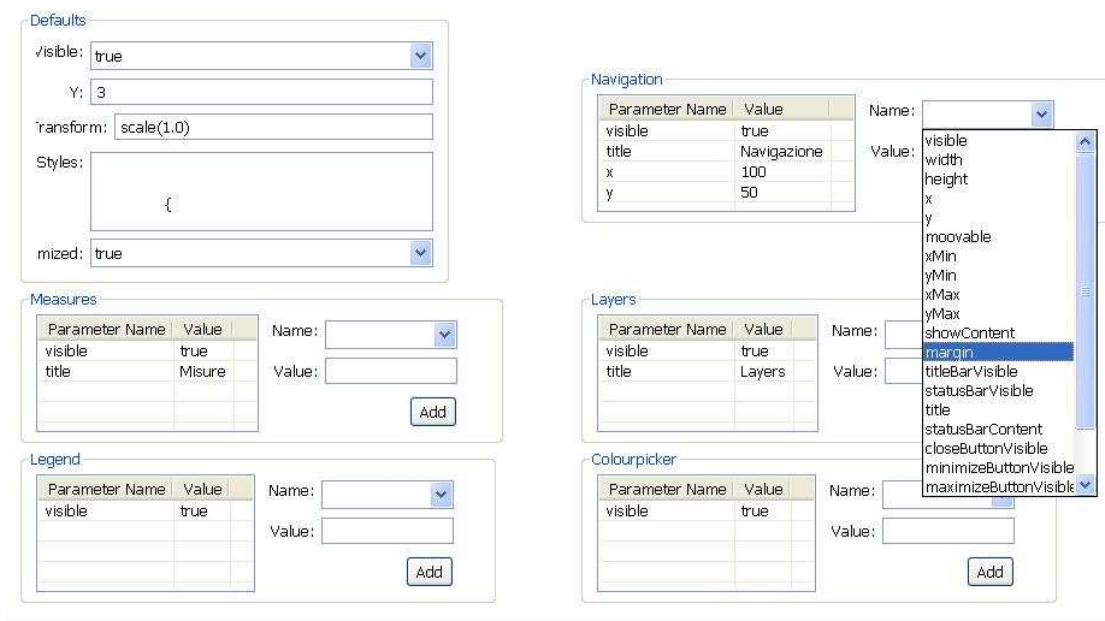


FIGURE 4.69 – GUI Settings for Windows

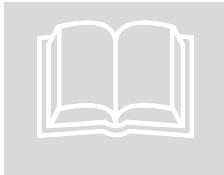
GUI Windows configuration parameters	
Visible	Set visibility for the specific window
Width	Width of the window
Height	Height of the window
X	Position of the left side in ViewBox coordinates
Y	Position of the left side in ViewBox coordinates
Movable	If true, the window can be moved
xMin	Set the left bound for the movable area
yMin	Set the upper bound for the movable area
xMax	Set the right bound for the movable area
yMax	Set the lower bound for the movable area
showContent	If true, content inside the window will be visible when the window is moved

Margin	Margin in ViewBox coordinates. Use this to configure text and buttons in title and status bar.
TitleBar Visible	If true, title bar will be visible
StatusBar Visible	If true, status bar will be visible
closeButton Visible	If true, window has a close button
minimizeButton Visible	If true, window has a minimize button
Minimized	If true, window will be minimized by default
Transform	List of transformations to be applied to the window. See SVG specification.
Styles	JSON object describing styles for window components (see below)

TABLE 4.18 - GUI Windows configuration parameters

Here is an example of JSON style object:

```
![CDATA[
{
  winPlaceholderStyles: {"fill": "none", "stroke": "dimgray", "stroke-width": 1.5}
, windowStyles: {"fill": "#fffce6", "stroke": "dimgray", "stroke-width": 1}
, titlebarStyles: {"fill": "steelblue", "stroke": "dimgray", "stroke-width": 1}
, titlebarHeight: 17
, statusbarStyles:
{"fill": "aliceblue", "stroke": "dimgray", "stroke-width": 1}
, statusbarHeight: 13
, titletextStyles: {"font-family": "Arial, Helvetica", "font-size": 14, "fill": "white"}
, statustextStyles: {"font-family": "Arial, Helvetica", "font-size": 10, "fill": "dimgray"}
, buttonStyles: {"fill": "steelblue", "stroke": "white", "stroke-width": 2}
}
]]
```



SVG Specification

Read more about SVG specification at
<http://www.w3.org/TR/SVG/coords.html>

GUI Settings Params

Here you can configure the general properties of the map. The list of available parameters is summarized below.

GUI Params configuration parameters	
defaultDrillNav	It indicates whether the initial navigation modality shall be cross-type or drill-type
highlightOnMouseOver	It indicates whether the areas of the map shall color when the mouse passes over them
normalizeChartValues	It indicated whether the values shall be normalized on a 0-100 scale. This is useful if values have different unit measures or high variance.
chartScale	Scale factor applied to the map. Default is 1.0.
chartWidth	Width of the map image
chartHeight	Height of the map image
valueFont	Font of values on the map
valueScale	Scale factor applied to values on the map. Default is 1.0.

TABLE 4.19 - GUI Params configuration parameters

a)

GUI Settings - Params
Insert parameters for GUI Settings Params

Parameter Name	Value
chartScale	1.0
chartWidth	80
chartHeight	160
valueFont	16px

Name: valueScale
Value: 1.0

b)

GUI Settings - Labels
Insert parameters for GUI Settings Labels

Parameter Name	Value	Name	Value
font-size	12		

Position: header-left header-center header-right
 footer-left footer-center footer-right
Text: USA \$(day)
Format: 01/03/2012 17:50

Parameter Name	Value	Name	Value

Position: header-left header-center header-right
 footer-left footer-center footer-right
Text:
Format: dd/MM/yyyy HH:mm

FIGURE 4.70 – GUI Settings section for: a) Parameters and b) Labels

GUI Settings Label

In this section you can add labels to a GEO document. To add a new label, click on **Add** and a new configuration block will appear.

Each configuration block allows you to configure the label. First, set the position of labels, which can be placed in the header or in the footer: on the left, on the right or at the center of the header/footer.

Then, write text according to a given syntax and decide how to format it. For example, you can write the following text to visualize day and current time:

```
'Last update ${day} at ${hour}  
// day and hour are parameters' names
```

Finally you can set a number of configuration numbers. This set of parameters depend on the properties of the text field for the current label.



SVG Specification

The complete list of these parameters is available at
<http://www.w3.org/TR/SVG/coords.html>

5. SpagoBI Server

SpagoBI Server is the main module of the suite, the one providing the whole analytical power of the product and all administrative functionalities.

That is the main reference for all potential users and usages; it drives the development trend in terms of features, services and delivery models.

It represents an enterprise level solution for BI, supporting the whole project life-cycle, managing security and guaranteeing scalability, clustering and high availability architectures.

SpagoBI Server is the only mandatory module, the complete Business Intelligence product covering the whole range of BI capabilities, and providing flexible and innovative solutions for advanced BI domains. In particular, it includes:

- **Reporting.** Development of structured reports, exportable in several formats
- **Multidimensional analysis (OLAP).** Navigation of data along hierarchies and dimensions
- **Charts.** Single ready-to-use graphical and interactive widgets
- **Interactive cockpits.** Aggregation of several documents into a single view: interactive and intuitive usage
- **KPIs.** Creation, management, view and browsing of KPI hierarchical models

- **Data Mining.** Models and algorithms to find out hidden information patterns from great volumes of data.
- **Free Inquiry.** Graphical query building and creation of customized, data-driven selection forms.
- **Ad-hoc reporting.** End-user self-made analysis.
- **Location Intelligence.** Correlation between geographic and business data.
- **Real-time.** Dashboards and consoles to monitor events in real time and act on data.
- **Mobile.** Based on the common touch-screen interaction paradigm, the combination of BI with mobility features
- **Office automation.** Publication of personal documents in BI environments.
- **Collaborative tools.** Organized dossiers, with comments and notes
- **ETL.** Traditional process of extraction, transformation and loading, including internal migration of data to/from operational systems
- **External processes.** Bidirectional interaction with operational and/or external systems.
- **Master data.** Manual management of data.

Moreover, SpagoBI Server works even at the operative level, with:

- **ETL**, not only for the continuous loading of source data into the DWH, but even for the internal movement of data, high-level consolidations or returning of the produced information to the operational systems
- **External processes**, for a bidirectional interaction with operational systems and external ones
- **Master data**, to manually manage domain data.

SpagoBI Server provides one or more solutions for each domain, all related to the behavioural model that regulates visibility over documents and data.

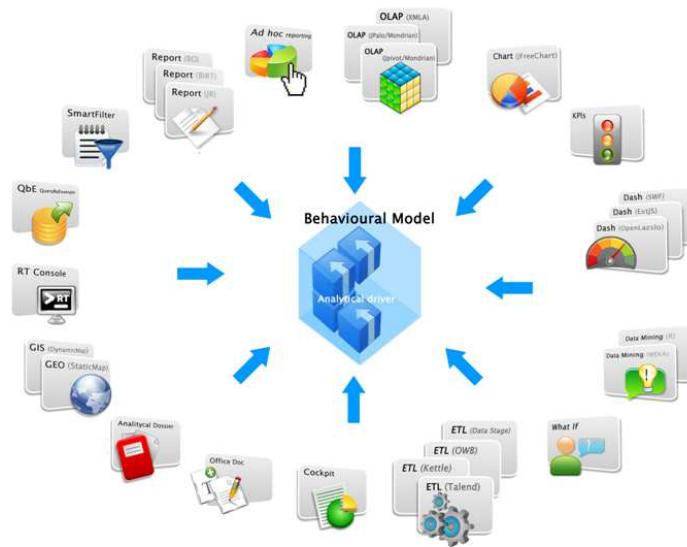


FIGURE 5.1 – Conceptual model of SpagoBI Server

This way, the BI suite adopts an evolutionary approach, starting from a few areas and growing over time. The server instance reflects this strategy, guaranteeing security and consistency, thanks to the independence of the behavioural model.

Moreover, SpagoBI has a distributed logic and handles more instances of a same engine. This allows to distribute the workload on several servers and to guarantee the linear system scalability.

Goals and targets

SpagoBI Server is a web application deployed into a standard J2EE application server (Tomcat, JBoss, WebSphere, etc.) or into a standard portal server (Liferay, eXo, etc.). Therefore, it does not require any particular environment

and the user can directly access and use his BI solution based on SpagoBI, simply using a common browser (IE, Firefox, Opera, Safari)¹³.

Being the Business Intelligence product, SpagoBI Server is the module used by many types of users:

- end users, who can access pre-built BI environments, use official documents and analysis, or freely build their own ones
- administrator, who manages the BI Server instance, sets authorization policies, and handles the document life cycle
- behavioural model owner, which manages the visibility on data and users' roles
- developers, who directly release or manage analytical documents.

Note that, especially in small-scale projects, different roles may be assigned to the same person(s).

Architecture

SpagoBI Server architecture is functionally layered on three main levels:

- **Delivery layer**, which manages all possible usages of the Server by end-users or from external applications
- **Analytical layer**, providing all analytical functionalities of the product
- **Data layer**, which regulates data loading through many access strategies.

Every layer of the functional architecture is composed of different application modules, as shown in the following figure.

¹³ The list of certified environments is just as an indication. Please, refer to <http://www.spagoworld.org/xwiki/bin/view/SpagoBI/CertifiedEnvironments> for the official list of certified environments.

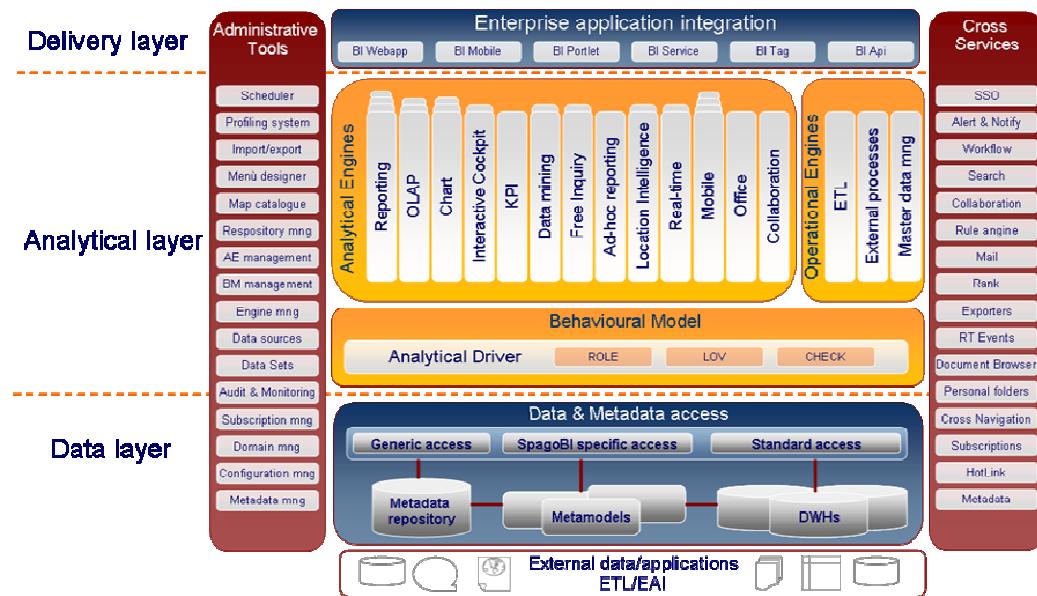


FIGURE 5.2 – SpagoBI Server architecture

Delivery layer

The Delivery Layer covers all publication requirements that allow business intelligence services to be used by end-users and to be accessible from third-party applications.

It can be accessed through different modalities:

- **BI Webapp**: it is the default SpagoBI use mode. A customizable web application is provided, working on a standard application server (i.e Tomcat, JBoss, WAS)¹⁴. The administrator can define the layout and specific views for each end-user type.
- **BI Mobile**: some types of analysis are also accessible through a mobile device, thanks to the interaction between SpagoBI Server and the remote client interface, in order to display one's reports, charts or cockpits on one's own tablet or smartphone

¹⁴ The complete list of certified environments is available at:
<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/CertifiedEnvironments>

- **BI Portlet:** users can access and use all functionalities through a portlet on a standard portal server (i.e. eXo, Liferay, WebSphere Portal)¹⁵, especially when this is preferable to a standard web application (BI Webapp).

SpagoBI Server can be integrated with external applications equipped with their own publication environments, thanks to the functionalities provided by SpagoBI SDK:

- **BI Service:** web services allowing SpagoBI components to interact with external applications or to collect the results of static documents (report, image of a chart, etc.)
- **BI Tag:** tag libraries to encapsulate a dynamic document (OLAP, GEO, etc.) into a different context
- **BI API:** even for the integration of enterprise applications behind or without the end-user GUI.

Analytical layer

The Analytical Layer is the core of the Server, providing all the analytical and operational capabilities in a secure and profiled mode. Its main components are:

- **Analytical Engines**, covering all the BI analytical requirements in terms of analysis (Report, OLAP, Charts, Cockpits, KPIs, etc.) and providing many engines for each type, so as to guarantee the highest flexibility and customers' satisfaction
- **Operational Engines**, to interact with OLTP systems by means of returning ETL or processes, and manage basic BI registries such as master data or lookup domains
- **Behavioral Model**, which regulates the visibility over documents and data, according to end-users' roles. It allows to reduce the number of analytical documents to be developed and managed by the user,

¹⁵ The complete list of certified environments is available at:
<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/CertifiedEnvironments>

guaranteeing the uniform and consistent growth of the solution over time.

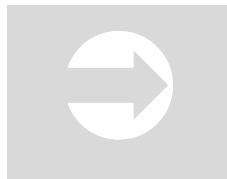
Offering multiple engines for the same analytical/operational area and/or multiple instances for a same engine, SpagoBI logic and architecture provides various benefits, such as:

- less workload on the engine instances, to guarantee high performances
- openness to improve or enrich the suite thanks to the integration of new engines and functionalities, minimizing the impact on the exiting environments
- high flexibility and modularity
- a high level of scalability over time, with minimum economic, infrastructural and application-level impact
- meeting requests coming from a wide range of users, coming from various sectors, through a complete range of engines
- scalability of users and servers with no additional costs.



Analytical and Operational engines

Even if SpagoBI covers many analytical and operational domains, this does not mean that each SpagoBI installation must include all its engines. Thanks to the modular architecture, you can start with a few domains and engines, and add others only when required, avoiding useless workload. This way, BI projects can easily follow evolutive approaches and methodologies.



Analytical layer

A single and detailed paragraph is dedicated to each component of the analytical layer. Please refer to the related section of this book for further information.

Data layer

The Data Layer allows data and metadata storage and usage. BI data is usually located in a data warehouse, whose design is out of the BI product scope and strictly related to the specific customer's world. SpagoBI offers a specific ETL tool allowing to load data at this level, covering the whole BI stack.



DWH

From a technical point of view, there is no limit to the use of an operational database (OLTP) as data source. However, its use is not recommended. A DWH is preferred to guarantee performances, consistency and stability.

SpagoBI can directly access the DWH through a standard JDBC/JNDI resource, or it can use a specific access strategy based on metamodels, built through SpagoBI Meta.

All SpagoBI metadata are stored in a private repository hosted on a generic RDBMS and accessed by means of a generic description based on hibernate technology. SpagoBI metadata contains technical information, business metadata and metamodels registry.

Administrative tools and cross services

Besides its analytical, delivery and data access capabilities, SpagoBI Server provides all the administrative tools to handle the whole instance and many cross-product services to make its functionalities powerful. Both of them cross all architecture layers because they have an end-user graphical interface, an application logic related to the analytical engines and documents and they all manage data and metadata.

The **administrative tools** support developers, testers and administrators in their daily work, providing various functionalities, such as:

- scheduler
- profiling system

- import/export capabilities
- menu designer
- map catalogue
- management of repository, analytical model, behavioural model and engines
- configuration of data sources and data sets
- audit & monitoring analysis
- subscriptions
- management of value domains, configuration settings and metadata.

The **cross services** include the common features of the product, shared by all analytical engines and documents:

- single sign on
- alert and notification
- workflow
- search engine
- collaborative tools
- rules engine
- sending e-mails
- ranking
- multiformat exporter
- RT events
- document browser
- personal folders
- cross navigation
- subscription service
- hot link

- metadata view.



Administrative tools and cross services

A single and detailed paragraph is dedicated to each relevant administrative tool and cross service. Please, refer to the related section of this book for any further information.

Multi-tenancy

Starting version 3.5, SpagoBI supports multi-tenancy, i.e., the use of the same SpagoBI instance by different organizations, called *tenants*. In a multi-tenancy architecture, each tenant owns and manages his users, documents, analytical drivers and so on, which are completely independent from those owned by other tenants.

In the current version, multi-tenancy is implemented according to certain criteria. As far as the behavioral model is concerned, the following rules apply:

- Each user belongs to one tenant only
- Each user is defined unambiguously over the whole set of users, belonging to all tenants. This means that if a user has been defined for a given tenant, no user with the same name can be defined for another tenant.
- Roles and profile attributes can be defined across tenants. This means that users belonging to different tenants may assume the same role or have the same attribute profiles.

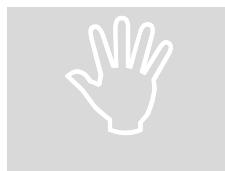


The behavioral model

The behavioral model rules visibility and rights on all analytical documents. For a full understanding of its meaning and functionalities, please refer to the next section in this chapter.

The following elements are shared across tenants:

- General configuration and domain management
- Layout of the web interface is also shared across tenants
- Repository of static resources (e.g., OLAP schemas, QbE datamarts, HTML static pages)
- Data source configuration in the application server.



Data sources for multi-tenancy

Although the data source configuration is shared across tenants in the application server, each tenant will have to define his own data sources at application level.

At the end of the next section, we will explain how to create and manage tenants in SpagoBI.

Installation and configuration

The first step to use SpagoBI Server is to check that your environment satisfies the following prerequisites:

- It has a JDK 1.6.x already installed
- It has the JAVA_HOME variable already set
- It has a certified operative system (Windows, Linux, Unix are generally accepted)¹⁶
- It has a supported browser (IE, Firefox, Chrome)¹⁷.

If so, the second step is to download the suitable package from OW2 forge at:

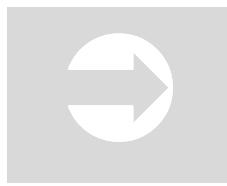
¹⁶ The complete list of certified environments is available at:
<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/CertifiedEnvironments>.

¹⁷ The complete list of certified environments is available at:
<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/CertifiedEnvironments>

- <http://forge.ow2.org/projects/spagobi> to look at SpagoBI latest releases
- http://forge.ow2.org/project/showfiles.php?group_id=204 to choose the latest release among all SpagoBI releases.

Now choose the right package according to the preferred installation procedure:

- **Basic installation**, starting with a ready-to-run, complete SpagoBI Server instance and demo, including all analytical engines, with a predefined environment and setting.
- **Custom installation**, starting with a free installation, choosing the target environment and SpagoBI Server composition.



SpagoBI Server installation and configuration

The complete installation process, both basic and custom, is detailed on the projects' wiki, section How to use SpagoBI > Server documentation. See

http://wiki.spagobi.org/xwiki/bin/view/spagobi_server/

For a basic installation, the reference package is the All-In-One-SpagoBI-<version>-<date>.zip.

		2011-11-30
SpagoBI 3.x		
SpagoBI-3.x-BOOK-index.pdf	55.2	Any .pdf
SpagoBI-3.x-quick-start.pdf	533.1	Any .pdf
SpagoBI 2.4		2009-12-18
SpagoBIMeta-Alpha_12182009.zip	3,590.8	Any .zip
SpagoBI-Studio-Chart-2.4.0_12182009.zip	5,636.2	Any .zip
SpagoBI-Studio-Core-2.4.0_12182009.zip	7,006.4	Any .zip
SpagoBI-Studio-DecComp-2.4.0_12182009.zip	6,367.0	Any .zip
SpagoBI-Studio-Gec-2.4.0_12172009.zip	0,501.0	Any .zip
b. SpagoBI Server		
		2011-12-22
SpagoBI 3.3 - Binary		
All-In-One-SpagoBI-3.3-01242012.zip	500,357.5	Any .zip
ClassServer-3.3.0_12222011.zip	13,085.5	Any .zip
SpagoBI-AccessibilityEngine-bin-3.3.0_12222011.zip	5,965.6	Any .zip
SpagoBI-bin-3.3.0_12222011.zip	64,261.9	Any .zip
SpagoBI-EarReportEngine-bin-3.3.0_01242012.zip	93,729.2	Any .zip
SpagoBI-CommonEngine-bin-3.3.0_12222011.zip	5,922.0	Any .zip
SpagoBI-ConsoleEngine-bin-3.3.0_12222011.zip	34,118.3	Any .zip
SpagoBI-GeoEngine-bin-3.3.0_12222011.zip	20,974.1	Any .zip
SpagoBI-GeoReportEngine-bin_3.3.0_12222011.zip	35,290.7	Any .zip
SpagoBI-JasperReportEngine-bin-3.3.0_12222011.zip	48,310.8	Any .zip
SpagoBI-PaloEngine-bin-3.3.0_12222011.zip	47,141.6	Any .zip
SpagoBI-PrintEngine-bin-3.3.0_12222011.zip	24,180.9	Any .zip
SpagoBI-XMIEngine-bin-1.3.0_12222011.zip	25,226.8	Any .zip
SpagoBI-ObeEngine-bin-3.3.0_12222011.zip	58,159.1	Any .zip
SpagoBI-TalendEngine-bin-3.3.0_12222011.zip	5,431.7	Any .zip
SpagoBI-WekaEngine-bin-3.3.0_12222011.zip	14,137.7	Any .zip
SpagoBI 3.3 - Script db		2011-12-22
ingres-dbscript-3.3.0_12222011.zip	35.8	Any .zip
mysql-dbscript-3.3.0_12222011.zip	31.6	Any .zip
oracle-dbscript-3.3.0_12222011.zip	32.9	Any .zip
postgres-dbscript-3.3.0_12222011.zip	31.9	Any .zip
sqlserver-dbscript-3.3.0_12222011.zip	15.7	Any .zip

FIGURE 5.3 – Download SpagoBI Server ready-to-run package from OW2 forge

At this point you just have to unzip the downloaded package under the desired folder, having a SpagoBIServer-<version> subfolder. SpagoBI Server can be started using the following scripts:

- ./bin/SpagoBIStartup.bat for windows environments
- ./bin/SpagoBIStartup.sh for unix/linux environments.

Opening the browser at <http://localhost:8080/SpagoBI>, the login page appears.

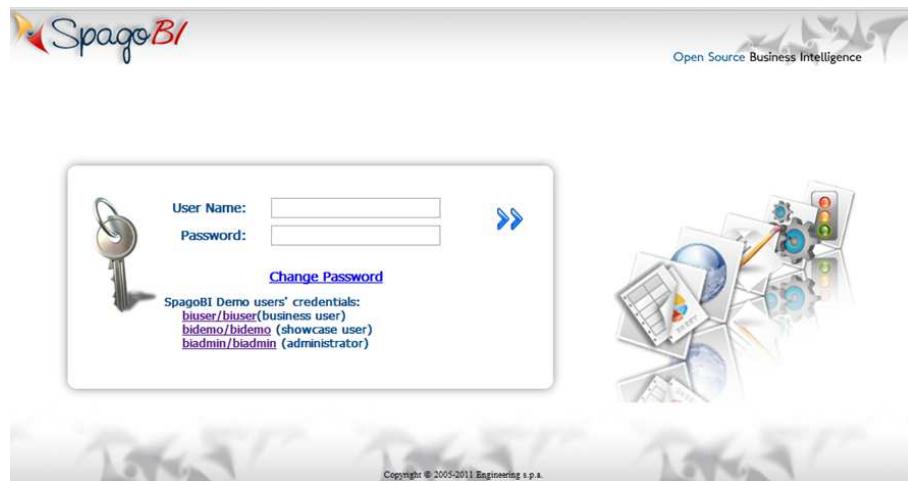


FIGURE 5.4 – SpagoBI Server login page

A standard application is ready to run on the predefined environment, based on Tomcat and HSQL database for the SpagoBI repository and the demo DWH.

To add a connection to a real DWH and begin with one's own business intelligence project, the steps are:

- add the relative JDBC driver to the folder SpagoBIServer-<version>/lib and restart SpagoBI Server
- login as administrator (biadmin/biamdin in the standard all-in-one distribution)
- select the **Data source** item from the **Resources** menu
 - click on the lens icon  on the top-right corner of the page
 - register the new data source, by setting the right string connection, user, password and JDBC driver, as shown in the following figure.

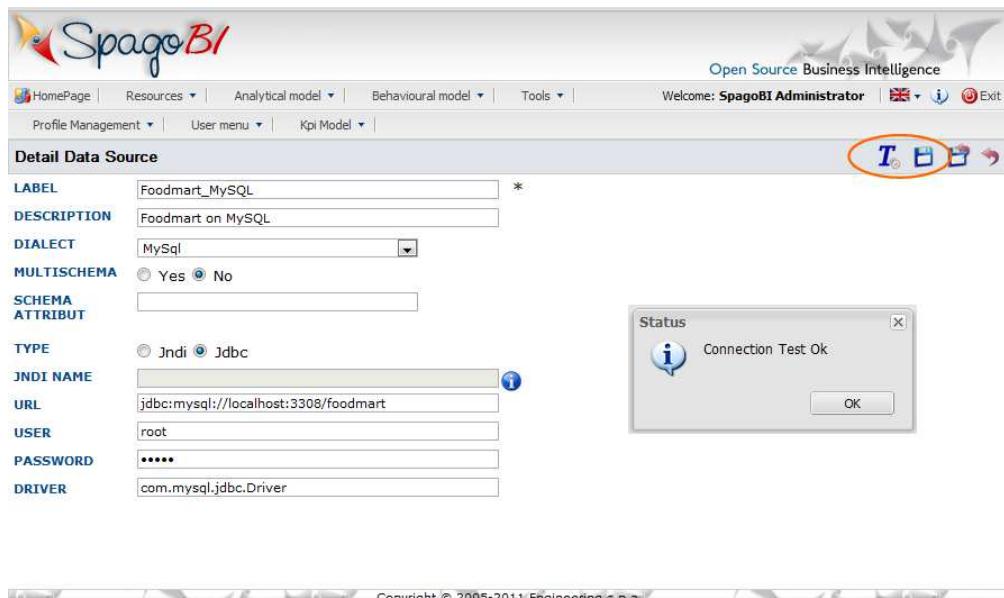
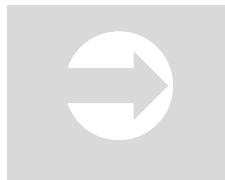


FIGURE 5.5 – Data source definition

- test the new data source using the related icon located at the top-right corner of the page.



Data source configuration

Data sources may also be defined as JNDI resources through a slightly more complex procedure, depending on the used application server. How to configure a data source as a JNDI connection for Tomcat is explained on a dedicated paragraph at the project's wiki page, http://wiki.spagobi.org/xwiki/bin/view/spagobi_server/

Now the new DWH is connected. You can start a new Business Intelligence project with SpagoBI!

Multi-tenancy management

At first execution, SpagoBI will create the default tenant, called “SPAGOBI”. If you do not need multiple tenants, you can simply ignore this feature as SpagoBI will normally work with the default tenant.

In case you wish to change the name of the default tenant, you must do this before first server execution. In particular, you should perform the following operations:

- Add the following code to file
SpagoBI/WEB-INF/conf/config/tenants.xml

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<TENANTS>
    <TENANT name="<nome del tenant>" />
</TENANTS>
```

- In file SpagoBI/WEB-INF/conf/config/internal_profiling.xml
Change the name of the default tenant for all pre-defined users.

If you don't need additional tenants, you are done.

In case you need to add multiple tenants, here is the procedure you should follow:

- Stop the application server
- Edit file SpagoBI/WEB-INF/conf/config/tenants.xml to add more tenants. An example with two tenants is shown below:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<TENANTS>
    <TENANT name="TENANT A" />
    <TENANT name="TENANT B" />
</TENANTS>
```

- Edit file SpagoBI/WEB-INF/conf/config/internal_profiling.xml to add at least one administrator. This will allow the administrator to further define roles, users, documents and so on. Below an example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<INTERNAL_PROFILING_INITIALIZER>
    <DEFAULT_USERS>
        ....
        <default users for TENANT A>
        ....
```

```

        <USER userId="<user id administrator>"  

password="<password administrator>" fullName="full name  

administrator" organization="TENANT B">  

        <ATTRIBUTE name="attribute tenant B" value="TENANT B"  

/>  

        <ROLE name="/spagobi/admin"/>  

    </USER>  

</DEFAULT_USERS>  

<DEFAULT_ATTRIBUTES>  

    ....  

    < default attributes for TENANT A>  

    ....  

    <ATTRIBUTE name="attribute tenant B"  

description="attribute tenant B" organization="TENANT B"/>  

</DEFAULT_ATTRIBUTES>  

<DEFAULT_ROLES>  

    ....  

    <default roles for TENANT A>  

    ....  

    <ROLE roleName="/spagobi/admin"  

description="/spagobi/admin" roleTypeCD="ADMIN"  

organization="TENANT B"/>  

</DEFAULT_ROLES>  

</INTERNAL_PROFILING_INITIALIZER>

```

- Restart the application server.

You are done! At server startup the new tenant will be created with its default users, which will then create the analytical and behavioral model for that tenant.

The behavioral model

The Behavioral Model regulates the visibility on documents and data according to the roles and profiles of the end users. The behavioral model offers many advantages in a BI project, including:

- reducing the required number of analytical documents to be developed and maintained
- coding visibility rules once only and apply them to several documents, each one with its own analytical logics
- ensuring a uniform growth of the project over time

- guaranteeing the respect of the visibility rules over time, with no limitation on the number of engines and analytical documents that can be added over time.

The behavioral model is based on four main concepts:

- **user profile**, defining the user's roles and attributes
- **repository rights**, which defines the users' rights in terms of document accessibility
- **analytical drivers**, defining which data can be shown to the user within a single document
- **presentation environment settings**, defining how the user can reach and run his own documents.

In other words, the behavioral model mainly answers the following questions:

- **WHO** uses the business intelligence solution (user profile)
- **WHAT** is visible to users, in terms of documents and data (repository rights and analytical drivers)
- **HOW** users work with their documents (analytical drivers and presentation environment settings).

User and role configuration

SpagoBI users are defined by:

- identities
- role
- profiles.

The *identity* of a user consists of all data used to identify that user, i.e., a username and a password, as well as a human readable full name.

The *profile* of a user consists of a set of properties called *attributes*, describing general information about the user, e.g., age and gender, but also domain-specific properties, such as the organizational unit to which he belongs. Some attributes, such as name and email, are defined by default in SpagoBI. Other can be added by the model administrator.

The *role* of a user represents a categorization of a group of users. These roles may correspond to specific positions in the company, e.g., “general manager” or a “sales director”, or to a position with respect to the BI project, e.g., “data administrator” and “BI developer”. Different users may have the same role, as well as the same user may have multiple roles.

It is possible to create several roles in SpagoBI, based on the project needs. However, all roles must belong to a pre-defined SpagoBI *role type*. A role type is a higher-level categorization provided by SpagoBI, aimed at mapping roles to the different functionalities of the suite. Technically speaking, a role type is a role for SpagoBI as it can be directly associated to a user.

Pre-defined roles are summarized in the following table. The first four roles are technical roles, while the last one – the user - is the actual end user. Each role type has a default user associated to it. Other users can be created and associated to a role type.

Role type	Description	Standard user
ADMIN	General administrator. Manages all SpagoBI functionalities	biadmin
MODEL_ADMIN	Model administrator. Manages the Behavioural Model and its associated functionalities	bimodel
DEV_ROLE	Developer. Creates and modifies BI documents	bidev
TEST_ROLE	Test user. In charge of testing documents.	bitest
USER	Final user. Executes documents visible to him.	biuser

TABLE 5.1 - SpagoBI role types

When a user logs into SpagoBI with his username and password, his profile is loaded right after authentication. The full name is shown at the right top corner of the page, as shown in FIGURE 5.6. Authentication can be handled internally by SpagoBI or delegated to an external Single Sign-On (SSO) system.



Authentication Management

The choice of handling authentication internally or delegating it to an external SSO system typically depends on the presence of an authentication system already in place. In this case, SpagoBI can seamlessly integrate with the existing authentication infrastructure.

Once the user has logged in, his role is loaded. Roles are managed internally. In case the user is associated with multiple roles, he will be asked to choose which role shall be played when executing a document. Alternatively, by clicking on the icon on the right top (see FIGURE 5.6), he can select a default role that will be kept valid until the user's logout.

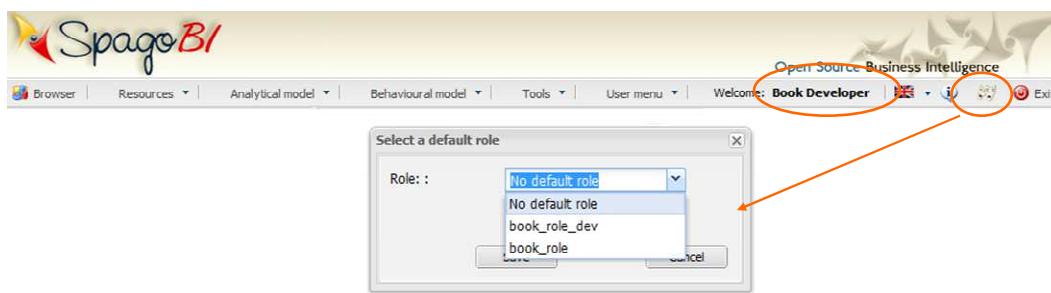


FIGURE 5.6 – User roles in SpagoBI

Profile Management Tools

The steps to create a BM follow:

- Create profile attributes
- Create roles
- Create users and associate attribute values and roles to them.

SpagoBI supports the management of user profiles and roles through the **Profile Management** menu on the main toolbar. This menu is only visible to SpagoBI administrator and to the model administrator, since users and roles management is a critical operation that requires an appropriate level of responsibility.

The **Profile Management** menu contains three sub-menu items:

- **Profile Attribute Management.** To define new profile attributes and manage the existing ones.
- **Role Management.** To create new roles and manage permissions for each role.
- **User Management.** To create users, manage their identities, assign values to their profile attributes and associate them with roles.

In the following, we show how the model administrator can define user profiles and roles using these functionalities. Note that SpagoBI profile management can also be integrated with external profiling systems.

Clicking on **Profile Attribute Management**, the list of currently defined attributes is shown. To add a new attribute, click on the **Add** button: a new row is added to the list, where you can insert the name and description of the new attribute. To delete an attribute, select the corresponding row and click on **Delete**.

Attributes defined in this section will be available to all user profiles. It is not mandatory to assign a value to each attribute for each user. However, all the attributes that have a specific value create the user profile.

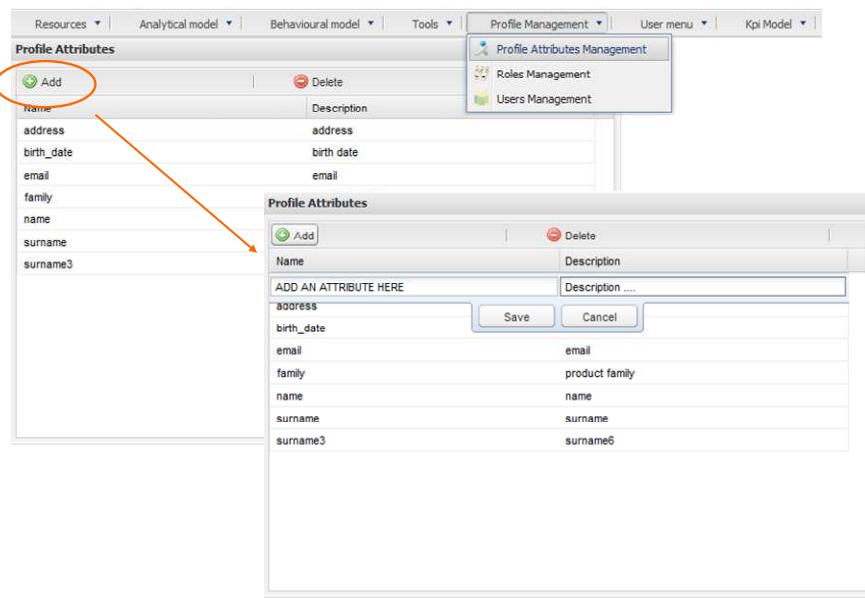


FIGURE 5.7 – Profile attribute management

Once the attribute are defined, the model administrator can define roles, using the **Role Management** functionality. The Role Management tool is two-sided: on the left you can see the list of already defined roles. At the beginning of a project, only default roles are visible. To add a new role, click on the **Add** button and move to the right panel. To delete a role, simply select the row and click on **Delete**.



Role Management

The behavioural model should be built taking into account the specificity of each organization and the needs of the BI project. Therefore, it is a good practice to define specific roles for the BI project and avoid using SpagoBI technical roles only.

In the right panel there are two tabs. The **Detail** tab allows the administrator to define role name and role type (mandatory). The role type constrains the visibility of that role based on the categorization in TABLE 5.1. A code and a description can be added, too, as shown in FIGURE 5.8. Remember to save the role before switching to the other tab.

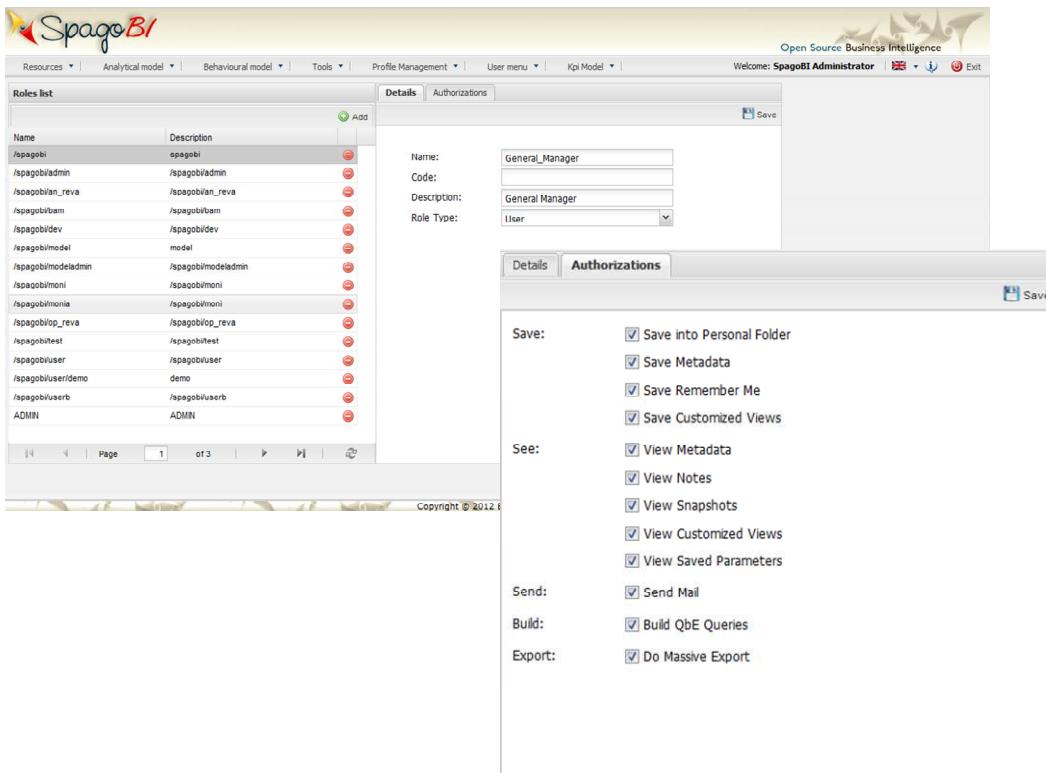


FIGURE 5.8 – Role management

The **Authorizations** tab sets rights for the considered role. Rights are pre-defined and grouped into categories, as shown in FIGURE 5.9.



Assigning permissions to roles

In a BI project scenario, the number of authorizations granted to a role tends to follow the expertise of the role itself: the more permissions a role has, more technical the role is. This is to avoid improper usage of the suite functionalities by non expert users.

The last step is the creation of actual users. This operation may be repeated over time as new users are granted access to the BI project. The **User Management** functionality has two sides. In the left panel, the administrator can see all existing users, create a new one clicking on **Add** and removing a user clicking on **Delete**.

The right panel includes three tabs, corresponding to the three elements characterizing users (as explained at the beginning of this section):

- **Detail.** In this tab the administrator sets identification data, including username and password.
- **Roles.** Using this tab the administrator assigns one or more roles to the user.
- **Attributes.** In this tab the administrator assigns values to one or more attributes of the user profile.

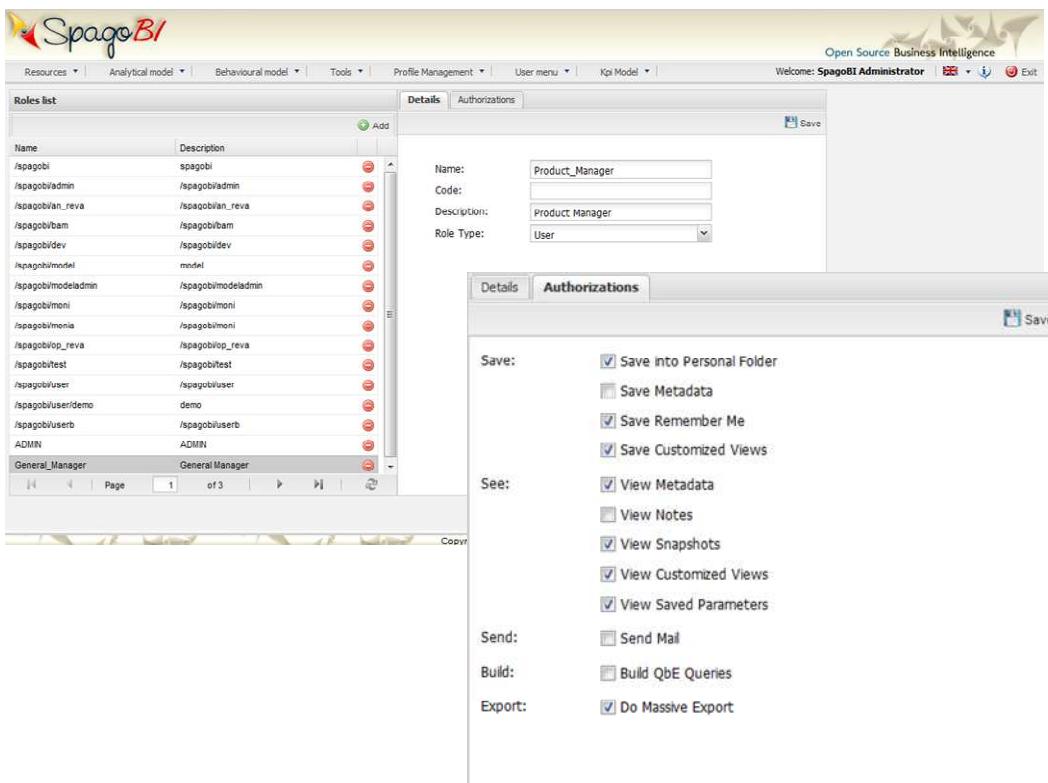


FIGURE 5.9 – User management

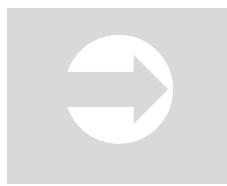
Repository structure and rights

SpagoBI adopts a file system to organize analytical documents in hierarchies and folders. This file system is called the Functionalities tree and is accessible via **Analytical Model > Functionalities Tree**.

There are two main reasons to organize documents into folders and hierarchies. First, a file system makes it easier for users and developers to navigate the BI project. For example, it is possible to create a folder that contains all analytical documents of a specific sub-domain or analysis type.

More importantly, the Functionality tree allows administrators to rule access to single documents via authorizations on containing folders. In other words, the administrator can create a folder, put a set of documents into it and grant access to that folder to some users (or roles). This way, authorized users will (only) see those documents, while the documents situated outside the folder will not be visible.

By default permissions are set at folder level. This guarantees that a user can not see anything outside that folder (unless he has permissions on other folders as well). It is also possible to further restrict the visibility scope of a user by associating rights to specific values of the profile attributes.



Restricting document visibility

Besides visibility limitations inherited by the containing folders, single documents can be subject to further restrictions. Learn more at section Analytical Model of this chapter.

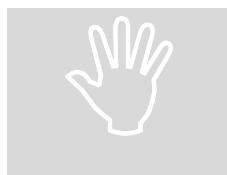
To create a new folder, select **Analytical Model > Functionalities Management**. The functionality tree is shown. Clicking on an item you can select one of the following options:

- Insert. To add a new child to the tree. Select this to create a new folder and go to the next step.
- Detail. To see the details of an item.
- Erase. To delete an item. This option is available only if the folder does not have any children nodes in the tree.
- Move down. To move the item down into the hierarchy.

Once you select Insert, the Functionality Details opens (see FIGURE 5.10). Enter a label and name for the new folder (functionality). In the table, assign permissions to roles. There are four types of permission:

- Development
- Test
- Execution
- Creation (for metamodels and QbE).

To assign permissions to roles, check the related boxes. Each user with that role will have permissions on that folder, except in case of specific restrictions on the single document.



Permission inheritance

A subfolder inherits the permissions assigned to the parent folder. While it is possible to further restrict inherited permissions, the opposite is not allowed: rights cannot be extended going down the hierarchy.

Role	Development	Test	Execution	Creation
/spagobi (spagobi)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/admin (/spagobi/admin)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/spagobi/an_reva (/spagobi/an_reva)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
/spagobi/bam (/spagobi/bam)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/spagobi/dev (/spagobi/dev)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/model (model)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/modeladmin (/spagobi/modeladmin)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/moni (/spagobi/moni)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/moni (/spagobi/moni)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/op_reva (/spagobi/op_reva)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
/spagobi/test (/spagobi/test)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/user (/spagobi/user)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/user/demo (demo)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
/spagobi/userb (/spagobi/userb)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

FIGURE 5.10 – Create a new folder and assign permissions

Analytical drivers

An *analytical driver* (hereafter simply *driver*) models a concept or a piece of data frequently used as a distinguishing criterion on the global data context. A driver highlights the concepts guiding the analysis, providing a unique representation of them and describing how they are shown and checked according to the end-users' roles.

When connected to analytical documents, a driver produces an explicit or implicit parameter used to filter data.

Each driver provides many *use modes*, defining:

- Who is involved in a specific use mode, in terms of list of end-user roles, considering that a role can be associated to a single use mode only.
- What data he can access and how they are presented to the end-user for his potential selection. This information is provided by the so called *List of Value* (LOV).
- How to verify the validity of the chosen values. This information is provided by the so called *Check*.

So, each use mode refers to an initial visualization method and content (LOV), to one or more validation rules (Check) and to one or more end-user roles (roles).

The logic of a driver is represented by the following picture:

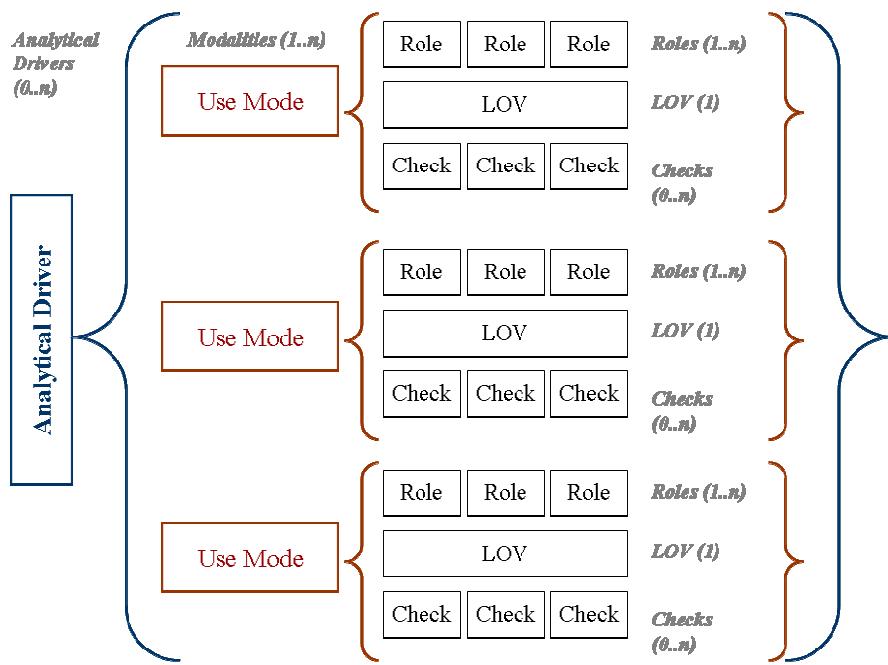


FIGURE 5.11 – Analytical driver schema

For example, if the concept of 'Product Family' is a common driver and discriminator for the enterprise analysis, an analytical driver will be coded, with all its behavioural rules, such as:

- if the user is a call center operator or a user that provides internal support, he can manually write the product family he wants to select. This value will be formally verified (it must be a text) and checked on the product family registry.
- if the user is a product brand director or an operative secretary, he can choose the value from a preloaded list of all the product families belonging to his brand. The selected value will be formally verified (it must have the right prefix) and checked on the product family registry
- if the user is a product family director or an executive secretary or a person of his staff, the brand automatically comes from his profile. For this reason, the value does not need any check.

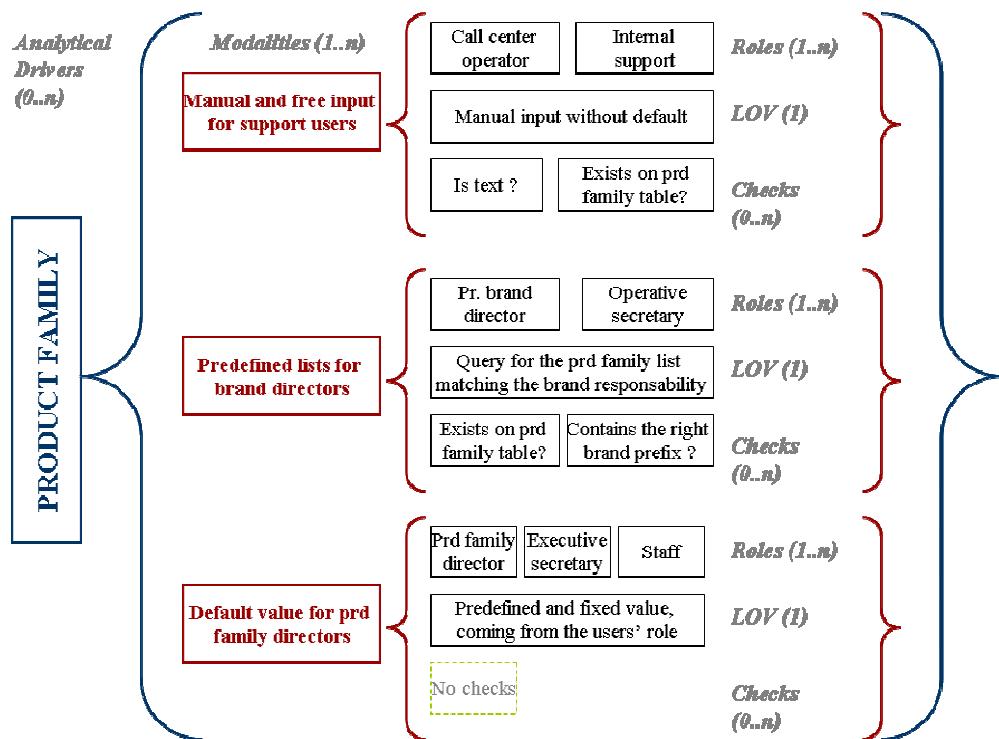


FIGURE 5.12 – Analytical driver example

Once defined, a driver can be related to many documents, driving their behavior and filters in a common way. This way, a user who runs different documents that use the same drivers, always receives the same parameter form, applying the same filters over shown data.

In fact, when an authenticated user (with its roles and profile) runs an analytical document, its technical metadata are read, mainly in terms of document template and related drivers. Based on them, a customized page for the parameters input is produced, according to the driver logic for the end-user role. The selected values are then validated and the final output comes to the user.

The following figure shows this process:

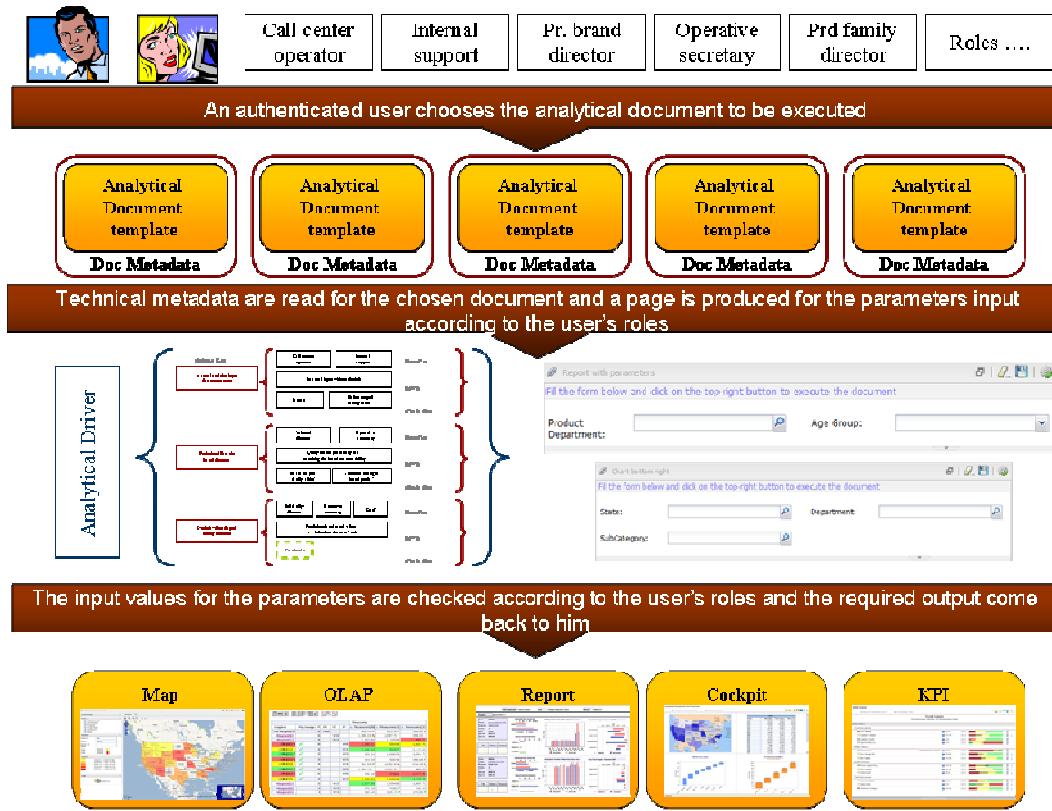


FIGURE 5.13 – Analytical drivers at work

Thanks to analytical drivers, a single document is able to cover the analytical demands of various categories of users, with noticeable advantages in terms of:

- reduction of the number of documents to be developed and maintained
- consistency in the request for parameters
- complexity reduction in the development of documents, thanks to the separation between security matters and massive development
- simple maintenance of the security (visibility over data) over time, despite the increase of developed documents or added engines.

The following paragraphs explain how to create an analytical driver and its basic components.

Creating a list of value

A LOV is a collection of data organized in attribute-value fashion. For example, the following LOV retrieves id, name and food family for a product:

```
{195, High Top Almonds, Food};  
{522, Tell Tale Walnuts, Food};  
{844, Very Good Soda, Drink};
```

There may be multiple attributes in a LOV, but only one of them is the core value that is actually used in the analytical driver. Other values have a descriptive function: they can be used to provide a human readable description of the LOV, as well as to store information used, for example, to correlate analytical drivers. In our example, the core value is the customer id, while the others are additional data describing the customer.

SpagoBI allows to create different types of LOV:

- **Query.** SQL query to retrieve values from the database
- **Script.** Groovy or JavaScript to dynamically return values
- **List of fixed values.** Values are defined statically at LOV creation time
- **Java objects.** External object invoked by name that returns the list of values.
- **Dataset** (starting from version 3.5). Dataset already defined in SpagoBI Server that is used to retrieve values. Note that the dataset must not contain parameters, while profile attributes are allowed.

Query and script LOVs can be parameterized using profile attributes. This means that at LOV execution time, the value of the attribute in the user's profile is assigned to the placeholder in the LOV query/script. The syntax to add profile attributes is:

```
 ${attribute_name}
```

For example, a SQL query that retrieves all brand names grouped by product family, based on the value of the `Pr_Family` attribute, is the following:

```
SELECT DISTINCT C.PRODUCT_FAMILY, P.BRAND_NAME
```

```
FROM PRODUCT P, PRODUCT_CLASS C
WHERE C.PRODUCT_CLASS_ID=P.PRODUCT_CLASS_ID
AND C.PRODUCT_FAMILY = ${PR_FAMILY}
```

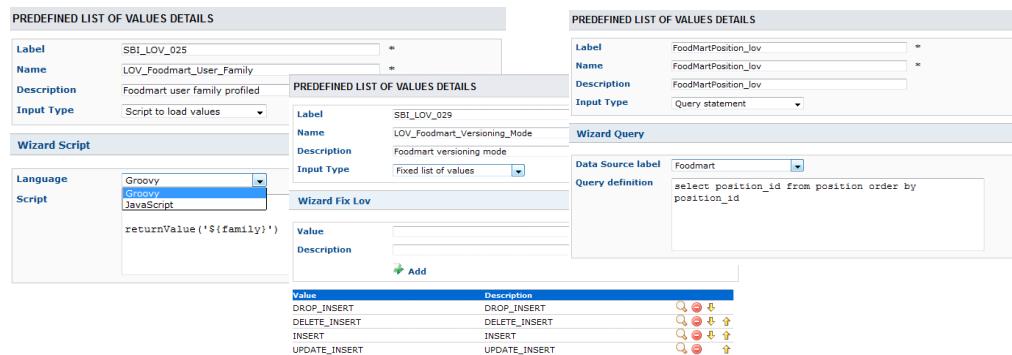
To create and manage list of values (LOVs), select **Behavioral Model > Lovs Management** from the top toolbar. The entire list of available LOVs appears.

For each LOV, the list shows the label, name, description, the type of LOV and the number of analytical drivers currently using that LOV. At the end of the row, click on the icon to see the details of the selected LOV and on the icon to delete it. Notice that you cannot delete a LOV if a driver is currently using it.

LABEL	NAME	DESCRIPTION	INPUT TYPE	USED BY N. ANALYTICAL DRIVERS
SBI_LOV_025	LOV_Foodmart_User_Family	Foodmart user family profiled	SCRIPT	1
SBI_LOV_030	LOV_Foodmart_Versioning_Col	Foodmart Version column	SCRIPT	2
SBI_LOV_028	LOV_Foodmart_Versioning_Flag_Y	Foodmart Versioning flag Y	SCRIPT	1
SBI_LOV_029	LOV_Foodmart_Versioning_Mode	Foodmart versioning mode	FIX_LOV	1
SBI_LOV_006	LOV_Foodmart_Warehouse_Profitability	Foodmart Warehouse Profitability	QUERY	1
SBI_LOV_004	LOV_Foodmart_Warehouses	Foodmart Warehouses	QUERY	1
SBI_LOV_013	LOV_Foodmart_Year	Foodmart Year	FIX_LOV	1
FoodMartPosition_lov	FoodMartPosition_lov	FoodMartPosition_lov	QUERY	2
SBI_LOV_044	LOV_KPI_Behaviour	KPI behaviour	FIX_LOV	1
LOV_FIX_DATE	LOV_FIX_DATE	LOV_FIX_DATE	FIX_LOV	1
LOV_KPI_HIERARCHIES	LOV_KPI_HIERARCHIES	LOV_KPI_HIERARCHIES	QUERY	1
LOV_KPI_OU	LOV_KPI_OU	LOV_KPI_OU	QUERY	1
SBI_LOV_AM_001	LOV_ORDERING_TYPE	LOV_ORDERING_TYPE	FIX_LOV	1
SBI_LOV_AM_002	LOV_ORDER_BY	LOV_ORDER_BY	FIX_LOV	1
SBI_LOV_AM_003	LOV_SBI_DOC_LABEL	LOV_SBI_DOC_LABEL	QUERY	1
SBI_LOV_AM_004	LOV_SBI_DOC_TYPE	LOV_SBI_DOC_TYPE	QUERY	1
SBI_LOV_AM_005	LOV_SBI_ENGINE	LOV_SBI_ENGINE	QUERY	1
SBI_LOV_AM_006	LOV_TYPE	LOV_TYPE	QUERY	1
MB_product	MB_product	MB_product	QUERY	1
SBI_LOV_MONTH	LOV_Month	Months in fact table	QUERY	1
SBI_LOV_027	LOV_Foodmart_Num_Claster	Number of clusters	FIX_LOV	1
SBI_LOV_000	LOV_Output_Type	Output Type	FIX_LOV	1
SBI_LOV_002	LOV_Foodmart_Products_PM	Products for Product Manager	QUERY	1
SBI_LOV_003	LOV_Foodmart_Products_FD	Products for Family Director	QUERY	1
REPORTED VALUES	REPORTED VALUES	REGISTER VALUES	FIX_LOV	1

FIGURE 5.14 – LOV Management

To create a new LOV, click on the  icon at the top right corner. The LOV creation interface will open, where you can set label, name and description, choose the LOV type and define its values accordingly.

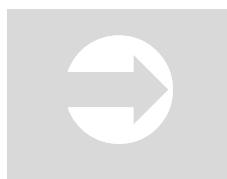


Value	Description	Value	Description
DROP_INSERT	DROP_INSERT	DROP_INSERT	DROP_INSERT
DELETE_INSERT	DELETE_INSERT	DELETE_INSERT	DELETE_INSERT
INSERT	INSERT	INSERT	INSERT
UPDATE_INSERT	UPDATE_INSERT	UPDATE_INSERT	UPDATE_INSERT

FIGURE 5.15 – LOV creation, with different possible types

Once completed the form, click on  to test the LOV. It is particularly important to test the LOV before saving it, since it allows to detect errors before the LOV is actually used in a driver and associated to a document.

After testing, you will be able to define which column is the actual value of the LOV, i.e., which value will be passed to the analytical driver using this LOV. Only one column can be the value attribute, while the others are descriptive. Descriptive columns can also be used for correlating drivers.



Correlating analytical drivers

Drivers can be correlated so that the value of the first driver is used as a parameter to select values in the second. Read more at section Analytical Model, in this chapter.

Predefined List of values details - Test results			
Column	Value column	Description column	Visible columns
product_id	<input checked="" type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
product_name	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>
product_family	<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>
<input type="button" value="Filter All"/> <input type="button" value="Print"/> <input type="button" value="Close"/>			
product_id	product_name	product_family	
195	High Top Almonds	Food	
205	High Top Mixed Nuts	Food	
206	High Top Canned Peanuts	Food	
211	High Top Walnuts	Food	
214	High Top Party Nuts	Food	
506	Tell Tale Almonds	Food	
516	Tell Tale Mixed Nuts	Food	
517	Tell Tale Canned Peanuts	Food	
522	Tell Tale Walnuts	Food	
525	Tell Tale Party Nuts	Food	

FIGURE 5.16 – Test results of a LOV

Finally, you can set which columns shall be visible. This is important especially for end users. If you are building a LOV to be used in a driver, and this driver shall be presented to end users as a filter for his document, it is important to choose which data shall be shown and which shall be hidden, for improved human readability.

Creating a validation rule

SpagoBI supports the validation of the document input parameters via validation rules. Validation rules can be defined in the **Behavioural model > Constraints Management** section.

A validation rule checks parameter values as given by LOVs, to verify that they comply with the defined constraints.

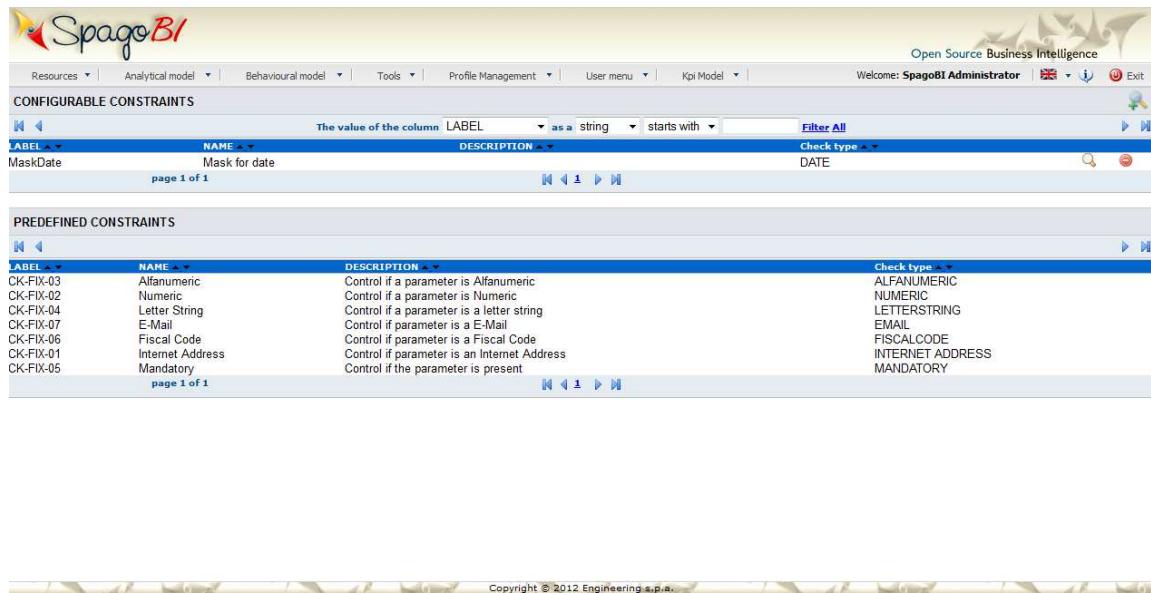


FIGURE 5.17 – Create a validation rule

SpagoBI default checks are:

- Alphanumeric. It checks if the parameter is Alphanumeric
- Numeric. It checks if the parameter is Numeric
- Letter String. It checks if the parameter is a letter string
- E-Mail. It checks if the parameter is an e-Mail
- Fiscal Code. It checks if the parameter is a Fiscal Code
- Internet Address. It checks if the parameter is an Internet Address
- Mandatory. It checks if the parameter has a value.

If the administrator needs to create additional validation rules, he can click on the icon to open the rule creation interface. Here he can define a customized validation rule using the available check options:

- Date: Check Date Format
- Regular Expression: Apply regular Expression
- Max/Min Length: Check

- Range
- Decimal: Decimal Places.

Creating an analytical driver

As explained at the beginning of this section, analytical drivers use information about users, their roles and profiles to filter data returned by their associated LOVs. Users, roles and profiles must have already been defined in the project context so that they are available to the driver.

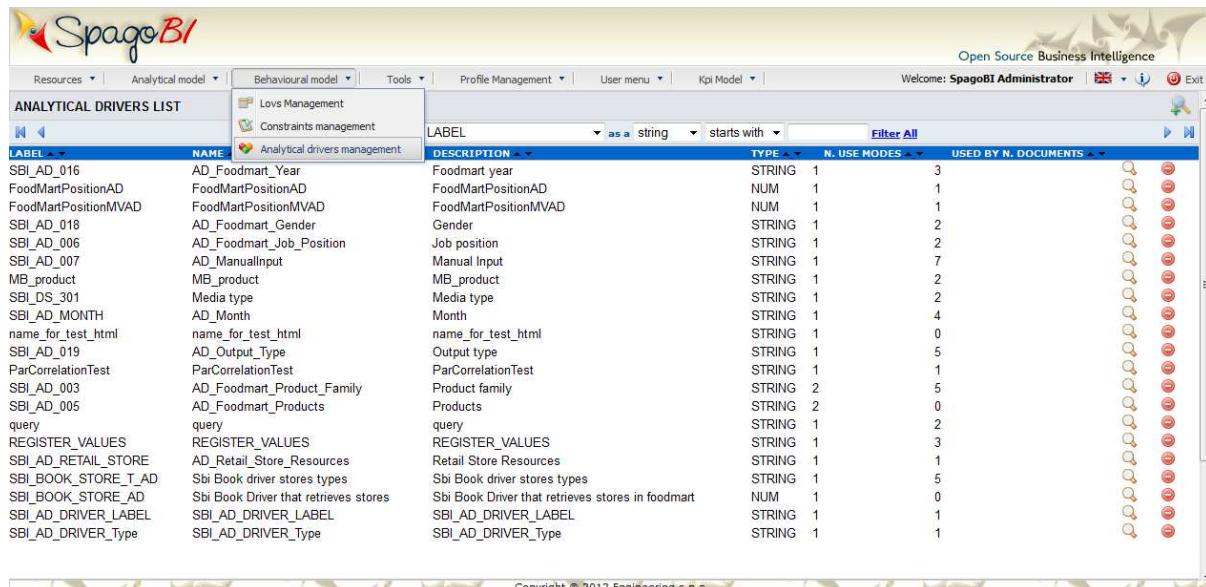


User and role management

Learn how to create and manage users, roles and profiles at section **The Behavioural Model, User and role configuration**, in this chapter.

To create a driver, select **Behavioral Model > Analytical Drivers Management** from the toolbar. You will see the entire list of available drivers.

For each driver, the list shows unique label, name, description, the type of driver and the number of documents currently associated to it. At the end of the row, click on the icon to see the details of the selected driver and on the icon to delete it. Notice that you cannot delete a driver if a document is currently using it.



The screenshot shows the SpagoBI Analytical Drivers List interface. At the top, there's a navigation bar with links like 'Resources', 'Analytical model', 'Behavioural model', 'Tools', 'Profile Management', 'User menu', 'Kpi Model', and 'Welcome: SpagoBI Administrator'. Below the navigation bar is a toolbar with icons for 'Lobs Management' and 'Constraints management'. The main area is titled 'ANALYTICAL DRIVERS LIST' and contains a table with columns: 'LABEL', 'NAME', 'DESCRIPTION', 'TYPE', 'N. USE MODES', and 'USED BY N. DOCUMENTS'. The table lists various drivers such as SBI_AD_016 (AD_Foodmart_Year), FoodMartPositionAD, FoodMartPositionMVAD, SBI_AD_018 (AD_Foodmart_Gender), SBI_AD_006 (AD_Foodmart_Job_Position), SBI_AD_007 (AD_ManualInput), MB_product, SBI_DS_301 (Media type), SBI_AD_MONTH (AD_Month), name_for_test_html, SBI_AD_019 (AD_Output_Type), ParCorrelationTest, SBI_AD_003 (AD_Foodmart_Product_Family), SBI_AD_005 (AD_Foodmart_Products), query, REGISTER_VALUES, SBI_AD_RETAIL_STORE, SBI_BOOK_STORE_T_AD, SBI_BOOK_STORE_AD, SBI_AD_DRIVER_LABEL, SBI_AD_DRIVER_Type, and SBI_AD_DRIVER_Type. Each row includes a 'Filter All' button and several small icons for editing, deleting, and viewing.

LABEL	NAME	DESCRIPTION	TYPE	N. USE MODES	USED BY N. DOCUMENTS
SBI_AD_016	AD_Foodmart_Year	Foodmart year	STRING	1	3
FoodMartPositionAD	FoodMartPositionAD	FoodMartPositionAD	NUM	1	1
FoodMartPositionMVAD	FoodMartPositionMVAD	FoodMartPositionMVAD	NUM	1	1
SBI_AD_018	AD_Foodmart_Gender	Gender	STRING	1	2
SBI_AD_006	AD_Foodmart_Job_Position	Job position	STRING	1	2
SBI_AD_007	AD_ManualInput	Manual Input	STRING	1	7
MB_product	MB_product	MB_product	STRING	1	2
SBI_DS_301	Media type	Media type	STRING	1	2
SBI_AD_MONTH	AD_Month	Month	STRING	1	4
name_for_test_html	name_for_test_html	name_for_test_html	STRING	1	0
SBI_AD_019	AD_Output_Type	Output type	STRING	1	5
ParCorrelationTest	ParCorrelationTest	ParCorrelationTest	STRING	1	1
SBI_AD_003	AD_Foodmart_Product_Family	Product family	STRING	2	5
SBI_AD_005	AD_Foodmart_Products	Products	STRING	2	0
query	query	query	STRING	1	2
REGISTER_VALUES	REGISTER_VALUES	REGISTER_VALUES	STRING	1	3
SBI_AD_RETAIL_STORE	AD_Retail_Store_Resources	Retail Store Resources	STRING	1	1
SBI_BOOK_STORE_T_AD	Sbi Book driver stores types	Sbi Book driver stores types	STRING	1	5
SBI_BOOK_STORE_AD	Sbi Book Driver that retrieves stores	Sbi Book Driver that retrieves stores in foodmart	NUM	1	0
SBI_AD_DRIVER_LABEL	SBI_AD_DRIVER_LABEL	SBI_AD_DRIVER_LABEL	STRING	1	1
SBI_AD_DRIVER_Type	SBI_AD_DRIVER_Type	SBI_AD_DRIVER_Type	STRING	1	1

FIGURE 5.18 – Analytical driver management

To create a new driver, click on the  icon at the top right corner. The driver creation interface will open. At first execution only the upper part of the window is shown (wrt. FIGURE 5.19).

The upper part is the **Detail** section, where you can set the label, name and description. Choose the type between Date, String or Number depending on the type of expected data. Select **Functional** or **Temporal** if the driver is used by an end-user or a scheduler, respectively. A click on the button  will save the driver and let the section below appear.

In the **Analytical Driver Use Mode Details** section, the driver is associated to LOVs and roles via use modes, and checks are defined, if any.

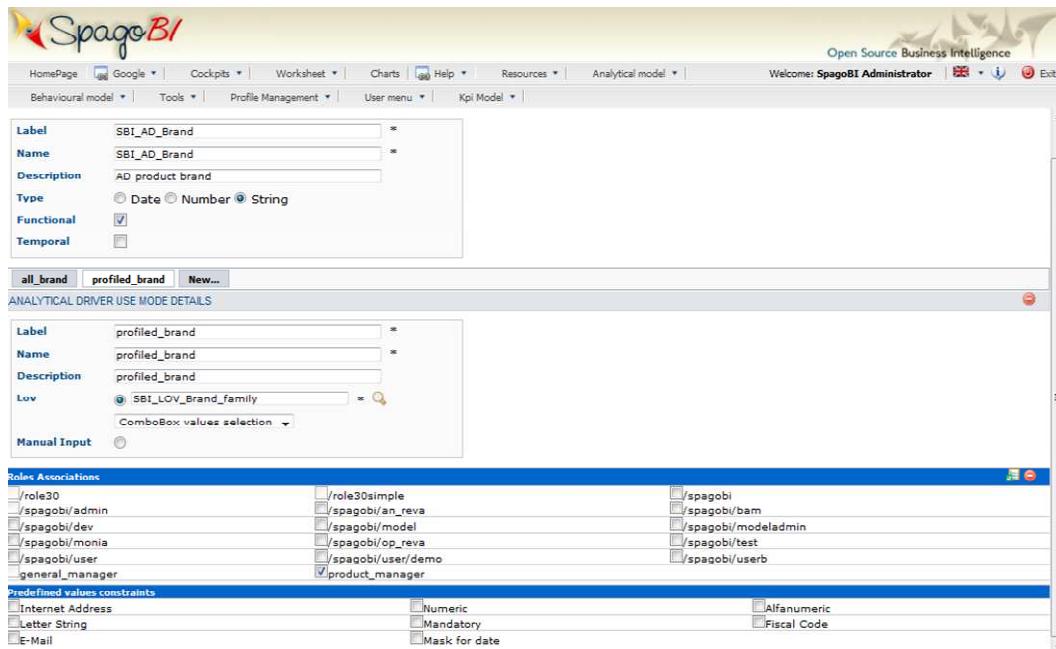


FIGURE 5.19 – Analytical driver creation

Each use mode is represented in a separate tab. Let us consider the example discussed at the beginning of this section. We create two use modes for the driver “Product Family”. The first use mode (called “All Brands”) is associated to the role of the product brand director (or general manager, in FIGURE 5.19) and to a LOV returning all product families for all brands.

The second use mode, called “Profiled_Brands” is associated to the role of the product family director (or product manager, in FIGURE 5.8) and to a LOV returning only those brands that belong to a specific family (see the code example in section Creating a list of values). The family is selected by checking the value of the family attribute in the user profile.

FIGURE 5.19 shows the use mode All Brands. In the upper part of the tab label, name and description are set. To associate a LOV to the use mode, click on the icon. The LOV Lookup List will appear, like the one shown in FIGURE 5.20. We search the LOV already defined “LOV_Brand_family” and

select it by clicking on the green tick icon. If you wish to leave the driver empty for manual input, select the corresponding option instead of the LOV.

The screenshot shows a SpagoBI interface titled "Lov lookup list". The main area is a grid of rows, each representing a LOV entry. The columns are labeled: "LABEL", "NAME", "DESCRIPTION", and "INPUT TYPE". The "LABEL" column contains names like "SBI_LOV_025", "SBI_LOV_030", etc. The "NAME" column contains descriptions like "LOV_Foodmart_User_Family", "LOV_Foodmart_Versioning_Col", etc. The "DESCRIPTION" column provides a brief description of each LOV. The "INPUT TYPE" column includes options like "SCRIPT", "QUERY", "FIX_LOV", and "REGISTER VALUES", each with a green checkmark icon indicating they are available. The interface has a standard Windows-style toolbar at the top and a footer with copyright information.

LABEL	NAME	DESCRIPTION	INPUT TYPE
SBI_LOV_025	LOV_Foodmart_User_Family	Foodmart user family profiled	SCRIPT ✓
SBI_LOV_030	LOV_Foodmart_Versioning_Col	Foodmart Version column	SCRIPT ✓
SBI_LOV_028	LOV_Foodmart_Versioning_Flag_Y	Foodmart Versioning flag Y	SCRIPT ✓
SBI_LOV_029	LOV_Foodmart_Versioning_Mode	Foodmart versioning mode	FIX_LOV ✓
SBI_LOV_006	LOV_Foodmart_Warehouse_Profitability	Foodmart Warehouse Profitability	QUERY ✓
SBI_LOV_004	LOV_Foodmart_Warehouses	Foodmart Warehouses	QUERY ✓
SBI_LOV_013	LOV_Foodmart_Year	Foodmart Year	FIX_LOV ✓
FoodMartPosition_lov	FoodMartPosition_lov	FoodMartPosition_lov	QUERY ✓
SBI_LOV_044	LOV_KPI_Behaviour	KPI behaviour	FIX_LOV ✓
LOV_FIX_DATE	LOV_FIX_DATE	LOV_FIX_DATE	FIX_LOV ✓
LOV_KPI_HIERARCHIES	LOV_KPI_HIERARCHIES	LOV_KPI_HIERARCHIES	QUERY ✓
LOV_KPI_OU	LOV_KPI_OU	LOV_KPI_OU	QUERY ✓
SBI_LOV_AM_001	LOV_ORDERING_TYPE	LOV_ORDERING_TYPE	FIX_LOV ✓
SBI_LOV_AM_002	LOV_ORDER_BY	LOV_ORDER_BY	FIX_LOV ✓
SBI_LOV_AM_003	LOV_SBI_DOC_LABEL	LOV_SBI_DOC_LABEL	QUERY ✓
SBI_LOV_AM_004	LOV_SBI_DOC_TYPE	LOV_SBI_DOC_TYPE	QUERY ✓
SBI_LOV_AM_005	LOV_SBI_ENGINE	LOV_SBI_ENGINE	QUERY ✓
SBI_LOV_AM_006	LOV_TYPE	LOV_TYPE	QUERY ✓
MB_product	MB_product	MB_product	QUERY ✓
SBI_LOV_MONTH	LOV_Month	Months in fact table	QUERY ✓
SBI_LOV_027	LOV_Foodmart_Num_Cluster	Number of clusters	FIX_LOV ✓
SBI_LOV_000	LOV_Output_Type	Output Type	FIX_LOV ✓
SBI_LOV_002	LOV_Foodmart_Products_PM	Products for Product Manager	QUERY ✓
SBI_LOV_003	LOV_Foodmart_Products_FD	Products for Family Director	QUERY ✓
REGISTER_VALUES	REGISTER_VALUES	REGISTER_VALUES	FIX LOV ✓

FIGURE 5.20 – LOV Lookup list

Now that the LOV has been associated, we return to the previous screen where we can select the display and data selection modality:

- List values selection. The filter will look like a lookup table, with one possible selection only.
- CheckList values selection. The filter will look like a lookup table, with multiple possible selections.
- Combo Box values selection. The filter will look like a drop down menu, with one selection only.

In our example we set a combo box for this use mode. Then we associate the use mode to the desired role, in our case the role Product_Manager, which we have defined in the previous sections. We can also set validation checks if needed, then save.

We repeat the same procedure for the second use mode, All_Brands: we associate it to the LOV returning all brand names and associate the role General Manager. Then we set a check list for this use mode.

At document execution time, we will see the filter on the left if we log in as the general manager, and the filter on the right if we log in as product manager (of food and drink, respectively).

BRAND_NAME	NUM_PRODUCT
1 ADJ	1
2 Akron	2
3 American	14
4 Amigo	2
5 Applause	2
6 Atomic	8
7 BBB Best	29
8 Best	4
9 Best Choice	36
10 Better	22
11 Big City	2
12 Big Time	31
13 Bird Call	19
14 Black Tie	2
15 Blue Label	22
16 Blue Medal	5
17 Booker	20
18 Bravo	22
19 Carlson	20
20 Carrington	31

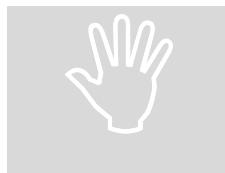
BRAND_NAME
High Top
Tell Tale
Ebony
Tri-State
Hermanos
Tip Top
Kiwi
Curlew
Dual City
Amigo
Green Ribbon
Swell
Applause
Big City
Washington
Excellent
Token
Fabulous
Skinner
Good
Walrus
Pearl
Portsmouth
Top Measure
Super
Landslide
BBB Best
Plato

FIGURE 5.21 – Profiled filters based on the use of analytical drivers: general managers (left) and product managers (right)

Publication environment

To personalize each user's interaction with the BI analysis, SpagoBI supports the creation of an analytical portal for end users. This allows the project administrator to build customized menus with specific features, to guide the end user through the analysis.

Menu configurations are associated to roles. This means that different users will see different menus according to their roles. So we can create different portals for a variety of end users, without changing the content of the analysis.



Menu configuration vs. BM visibility restrictions

Menu can be configured for improved end user interaction. However, documents and functionalities that are not visible to a user, either because of his role or due to his profile attributes, will remain not visible even within a menu.

Menus are visualized on the toolbar at the top of the window and can contain up to 4 levels. Each node in a menu can be empty or it can be associated to different types of resources:

- a static page
- an analytical document (with or without parameters)
- a functionality
- an external application.

Empty nodes are supposed to be the root for other children nodes, so they carry no particular content.

To start customizing a menu, select **Tools > Menu Configuration**. You will be shown the current structure of menu configuration. Click on **Insert child** to create a new menu item or **Detail** to modify an existing one.

The screenshot shows the SpagoBI Administration interface with the 'Menu configuration tree' tab selected. It displays a hierarchical menu structure across three panels:

- Top Panel:** Shows the main menu bar with 'Resources', 'Analytical model', 'Behavioural model', 'Tools', 'Profile Management', 'User menu', 'Kpi Model', and a 'Welcome' message for 'SpagoBI Administrator'. There are also language and profile selection buttons.
- Middle Panel:** Titled 'Menu configuration tree', it shows a tree structure starting with 'Menu tree'. Under 'Menu tree', there are nodes for 'HomePage', 'Examples', 'Funzioni', 'Google', and 'Browse'. The 'Google' node has a small red circle icon indicating it has children.
- Bottom Panel:** Titled 'Menu configuration tree', it shows a detailed view of the 'Google' node. It lists 'Insert child', 'Detail', 'Funzioni', 'Change-with-father', and 'Browse'. Below these are 'Move up' and 'Move down' buttons.

FIGURE 5.22 – Create or modify a menu item

After defining name and description, associate the role(s) that will see this menu item. To do this, check the boxes corresponding to the roles in the lower table.

Finally, select the type of menu item, i.e., the Menu node content. Depending on the selection, further customization options will appear, as we will show in the following. Empty nodes do not require further information.

Static page

Choosing a static page allows SpagoBI to associate an HTML page to the menu node. The page should be placed under the folder: <RESOURCES_PATH>\static_menu\. The home.html page is available by default: you can either modify it, or create a new static HTML page at this path.

Then the correct page can be selected from the drop down menu (see FIGURE 5.23). This option may be useful, for example, to create a welcome page for the end user.

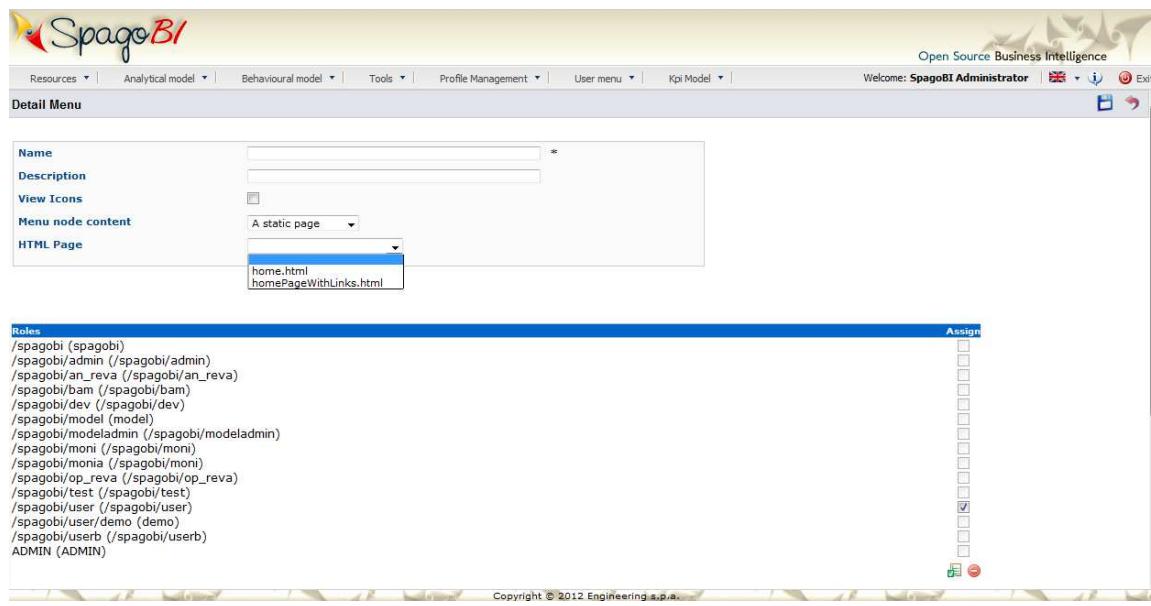


FIGURE 5.23 – Assign roles to a menu containing a static page

Analytical document

This option allows to link a document to a menu item. Choose the document using the search button  and browse existing documents.

Some analytical documents have parameters associated. If this is the case, in the **Document parameters** list you can define the list of parameters, according to the following syntax:

```
par1 = val1 & par2 = val2 & ...
```

If you include all document parameters in this list and assign them a value, the document will be launched with assigned parameter values. If you leave some parameters out of the list, they will be asked to the user right before document execution.

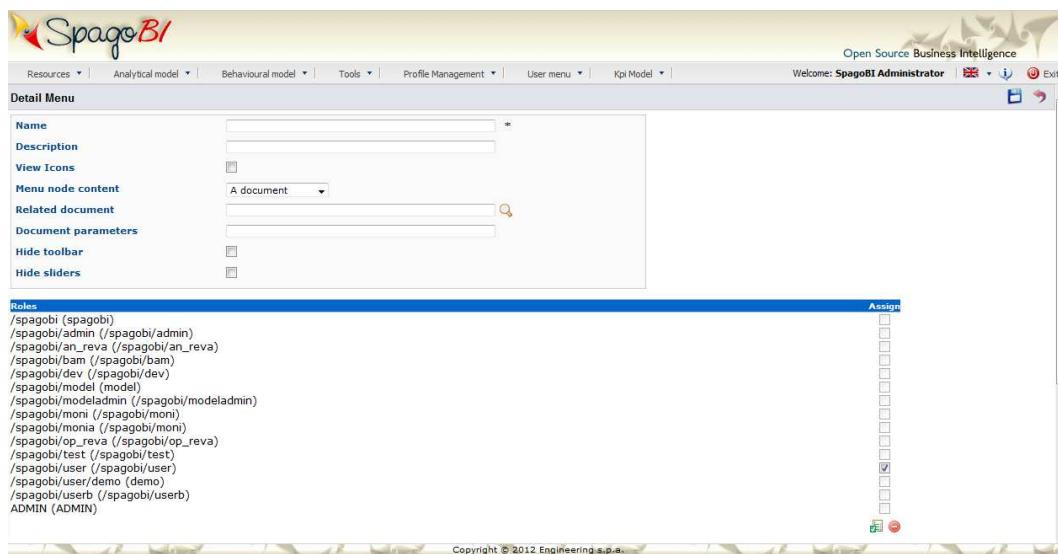


FIGURE 5.24 – Define a menu item that contains a document

There is a particular type of document that you can link to a functionality. QbE and OLAP documents contain what SpagoBI calls *subobjects*, i.e., sub-parts of a document. In particular, they are QbE queries and OLAP views, respectively. If you select one of those documents from the lookup window, you will be asked to choose the subobject you want to link to the functionality.

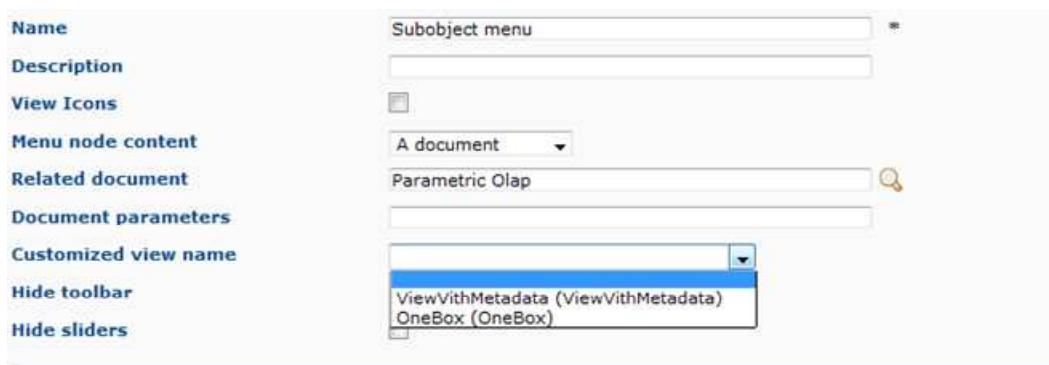


FIGURE 5.25 - Linking a subobject from a JPivot document to a menu item

The next items allow you to choose whether to hide the toolbar or the sliders.

Functionality

Functionalities can be defined by choosing among the options proposed in the default list, specifically:

- Document Browser
- Document tree
- Worklist
- Hot Link
- Subscription Lists
- Events.

If you chose the option Document tree, a pop-up window will open and you will be asked to select the path.



SpagoBI Functionalities

Read more about each of these functionalities at section Cross Services, in this chapter.

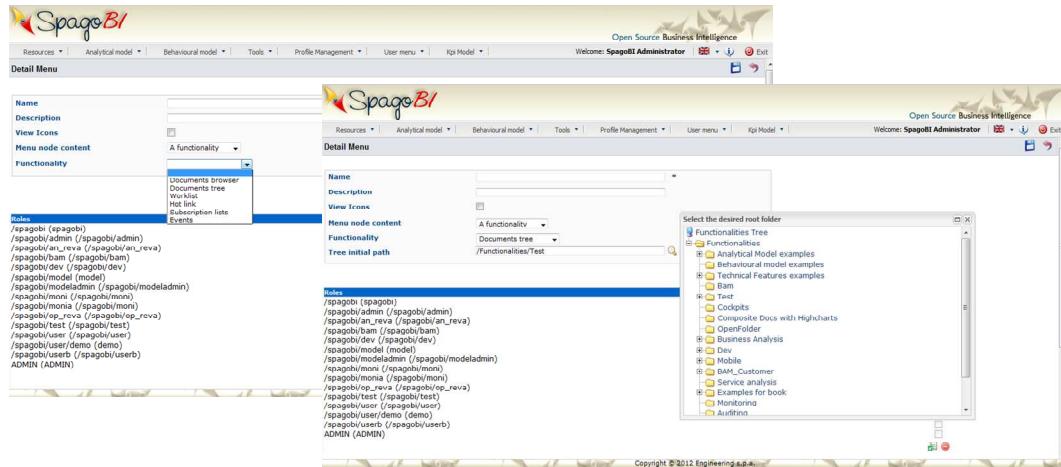


FIGURE 5.26 –Define a menu item that contains a functionality

External application

It is possible to link the menu item to an external application, e.g., an external web page. To do so, simply paste the URI of the external link into the **URI** field and save.

Analytical model

The *analytical model* represents the business intelligence analysis, in the strict sense of the word. For this reason it does not have its own box on the architectural schema, but it uses all its components and layers.

The main element of the model is the *analytical document*, which groups under a common concept all different types of documents that can be developed with SpagoBI (report, chart, cockpit, etc.) in a BI analysis.

Main concepts

The creation and management of analytical documents in SpagoBI involves different elements:

- **Template.** The template defines the standard layout of a document, including specific information on its appearance and the way contents should be displayed. Templates can be encoded by hand or using SpagoBI Studio designers, when available. For each analytical document the history of templates is maintained. Old templates can be restored if needed. A new version is saved at each deployment, either manual or from SpagoBI Studio.
- **Dataset.** Each document is associated to one or more datasets. The dataset provides actual data that will be represented according to the defined template. So we could say that the dataset provides the actual content of the analysis and the template is in charge of giving it a meaningful structure.
- **Data source.** In order to retrieve data via the dataset, a source of information must be defined. Depending on the type of document, the data source may be associated to the document either directly or implicitly (via the dataset).
- **Parameters.** Parameters allow the connection between the document and analytical drivers associated to it. In other words, at document execution time, each driver generates a value that is assigned to the corresponding parameter.

These elements are used in combination by each document engine to produce a specific type of document. The output of this process is by default HTML that can be navigated with a browser. Other output formats are possible, including XLS, CSV, PDF, XML.

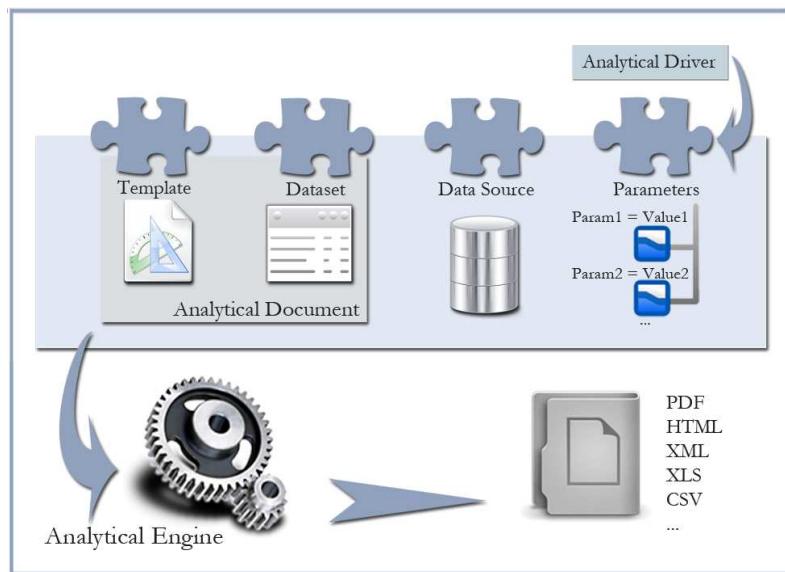


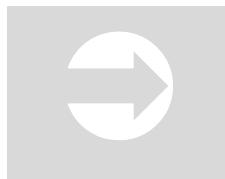
FIGURE 5.27 – The components of the analytical documents in SpagoBI

Document types

As briefly explained in the introduction of this chapter, SpagoBI supports many different document types, both analytical and operational ones, namely:

- Reporting
- Multidimensional analysis (OLAP)
- Charts
- Interactive cockpits
- KPIs
- Data Mining
- Free Inquiry
- Ad-hoc reporting
- Location Intelligence
- Real-time
- Mobile

- Office automation
- Collaborative tools
- ETL
- External processes
- Master data management



Analytical and operational engines

Each document type, both analytical and operational, is managed by one or more dedicated engines. Learn how to use the functionalities of each engine at chapter 6 – Analytical Engines, and chapter 7 – Operational Engines.

Regardless of their specific type, all documents are described in an abstract way as an *analytical document*, which interacts with all generic components (such as the behavioural model or cross-services) and the specific engine it refers to (such as the report engine, the OLAP engine, etc.).

Therefore, each *analytical document* is managed in the same way with respect to:

- document storage and versioning
- document life cycle, with an approval procedure, including different states (such as development, test, released or suspended)
- template versioning template
- multiple positioning on the repository and indirectly first visibility level
- rules to restrict document visibility to some end-user profiles only
- relation with analytical drivers and their possible dependency (correlation)
- multi-format exporters logic
- attribution of business metadata
- scheduled execution

- collection of auditing and monitoring data
- adding end-user notes
- adding bookmarks
- end-user evaluation
- sending the document by email
- on-line or off-line (scheduled) execution.

This means that the above mentioned features are also inherited by every new engine that is developed or integrated into SpagoBI.

Now let us see in detail how to create and manage analytical documents in SpagoBI.

Register an analytical document

There are two different ways to create a new document on SpagoBI Server. If you are using SpagoBI Studio, simply click on Deploy and the document window will open with pre-filled options.



Deploy a document from SpagoBI Studio

SpagoBI Studio is the tool that allows to design and upload documents onto SpagoBI Server. Please refer to chapter 4 - SpagoBI Studio for full details and examples.

The second option is to create the document by hand on the Server. This is the most general way since the Studio designer is not available for all documents yet. In the following we describe the complete procedure to create a document on the Server.

First of all select **Analytical model > Documents development** from the main toolbar. By default the tree list of all defined documents is shown. Clicking on the icon at the top right corner, the view will switch to a flat list, and vice-versa (see FIGURE 5.28).

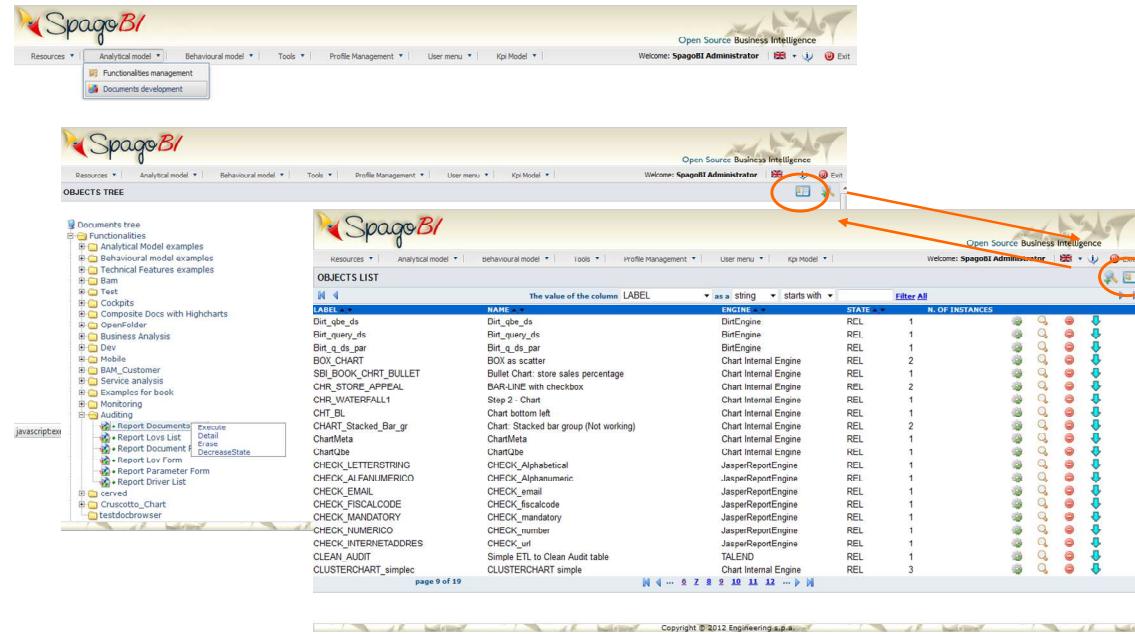


FIGURE 5.28 — Document development: tree view (left) and flat list view (right)

In both display modalities, if you click on a single document you will see the following options:

- **Execute.** To run the document.
- **Detail.** To see document details, such as name, dataset, template versions.
- **Erase.** To delete the document.
- **Increase/Decrease Document State.** To change the document current state in its life cycle.

FIGURE 5.29 shows the detail panel of a document. On the left, document details are shown, including name, type, dataset and state. On the right, you can alternatively see either the history of document templates or the functionality tree and the document position in it.

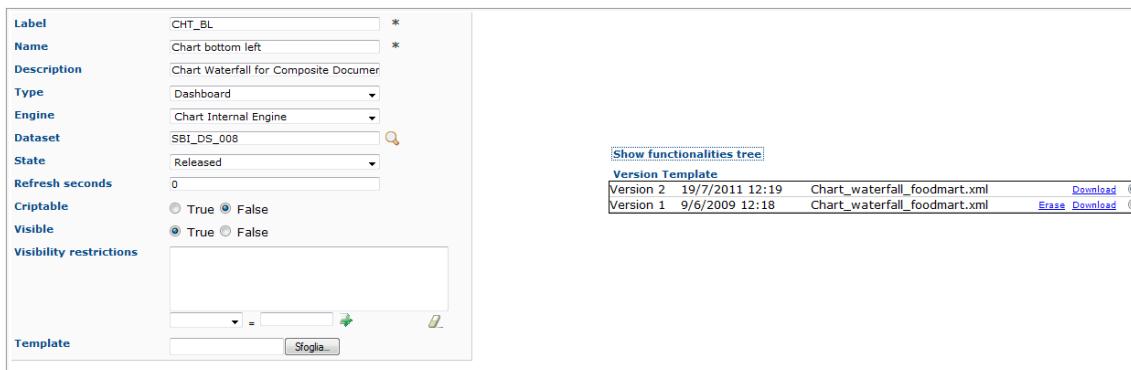


FIGURE 5.29 - Detail panel of SpagoBI analytical document

To create a new document click on the icon at the top right corner of the page. You will be shown a window like the one in FIGURE 5.29. This is very similar to the detail window seen above, except that we have not defined all details yet. Remember to save the document at each step by clicking on the icon at the top right corner, to avoid loss of data.

First of all, choose a label, a name and a description. The label is the unique identifier of the document in SpagoBI Server. Then, select the type of document and the appropriate engine from the drop down menus, according to the document you are developing (see FIGURE 5.30). Note that engines should have been registered by the administrator using the **Engines management** tool.

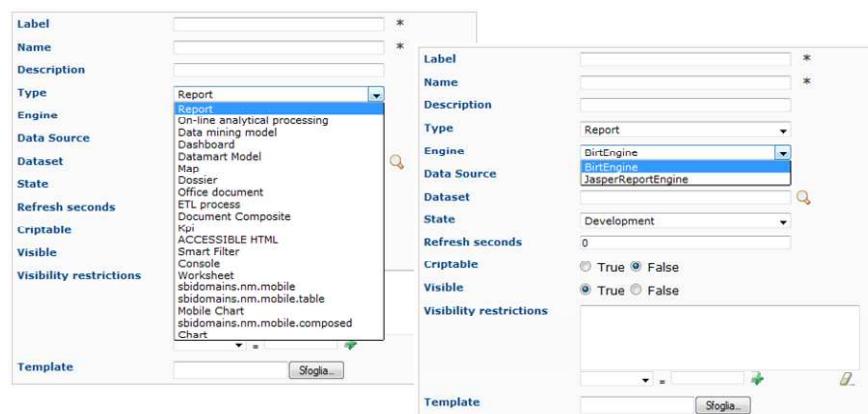


FIGURE 5.30 – Select type of document and engine for a new analytical document



SpagoBI Analytical Engines

If you are not sure about the type of document and engine you should select, or you want to learn how to register an engine, please check the section Administrative Tools > Engine management, in this chapter.

Now you have to select the dataset and data source that will feed your document with data. Both should have already been defined in the corresponding sections for SpagoBI to show them in the available options of the menus. Select the data source from the drop down menu. Then click on the icon and select the dataset from the lookup window.



Dataset and Data Source definition

To learn how to configure a data source and to create a dataset on SpagoBI Server, please refer to sections Administrative Tools > Data Source / Data Sets in this chapter.

Note that some types of document do not require the definition of a dataset at this point because they use embedded datasets. Depending on the type, it may also be necessary to set the data source.



Engine configuration

Dataset and data source checks are options that can be configured for each engine using the Engine Configuration functionality, described at section Administrative Tools, in this chapter.

The engines requiring dataset definition are:

- Chart

- Geo
- Dashboard
- KPI
- BIRT
- JasperReport.

The engines requiring data source definition are:

- BIRT
- JasperReport
- JPivot
- JPalo.

Full documentation about engine configuration can be found at the project's wiki page¹⁸. We also recommend to check the wiki for updates and more technical instructions.

Document lifecycle

The next step is to choose a state for it in the **State** drop down menu. At any time in fact, each document is associated to a state, which will typically change over time following the development of the project. Those states are:

- development
- test
- released
- suspended.

Upon creation, the document is by default in development state. Any time you upload a new template or make changes to the document, it is recommended that you choose the state based on the actual phase of the development.

¹⁸ https://wiki.spagobi.org/xwiki/bin/view/spagobi_server/Configure_v3

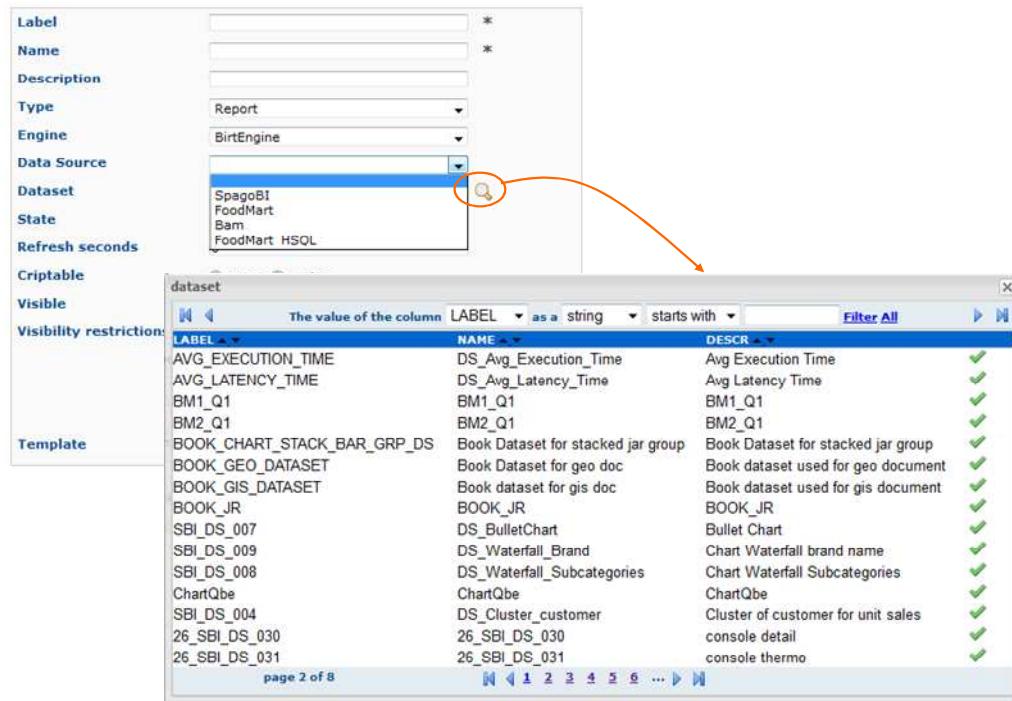


FIGURE 5.31 - Selecting a dataset for the new document

The main reason for this is that the state of the document has an impact on its accessibility. As discussed in the behavioural model, SpagoBI defines role types (administrator, developer, tester, user). States are compatible with the corresponding role type: a document in development state can be accessed by developers only; a document in test phase can be accessed by testers only; a document in released state can be accessed by users only. Administrators have full access to all documents at any phase.

Note that a tester may change the state of a document from test back to development. Administrators can change the state of documents at any time.

Template Versioning

When you register a document on the Server, you need to associate a template to it. Click on **Browse** and upload the template from your local file system. You may have edited the template by hand or using the Studio designer. Clearly you will not have to upload the template if you are using the automatic deploy from the Studio.

SpagoBI Server supports versioning of uploaded templates. To view them in the document detail window, click on **Show document templates** in the right panel. All templates are saved with their date and name, and can be easily restored or deleted. To restore a template, choose it in the list by clicking on the selector, then remember to save: the new template will be uploaded. Using the same list you can download or delete a template.



FIGURE 5.32 - Template versioning for analytical documents

Document Visibility

After defining all details, you need to choose where the analytical document shall be saved in the functionality tree. This choice has an impact on the visibility of the document. Because folders in the functionality tree are subject to different access policies, which can be set when creating the node, each document saved in that folder will inherit permissions accordingly.



Repository structure and rights

The Functionalities tree is SpagoBI document repository. It allows administrators to rule access to single documents via authorizations on the containing folders. It is part of SpagoBI Behavioural Model, which is explained at the corresponding section in this chapter.

Note that the same document can be saved in different points of the functionality tree. This allows the administrator to make the document

accessible to multiple roles based on visibility rules defined for the containing folder(s).

To save your document in the repository, switch the perspective on the right panel by clicking on **Show functionalities tree**. This operation is needed only if you moved to the template history view. Here you can choose where you wish to save the document, by ticking the corresponding folder in the tree. If you wish to save it at multiple locations, tick all of them before saving. Each user having access to the containing folder will see the document.

Visibility rules

In addition to the standard mechanism supported by the functionalities tree, it is possible to further customize access to a document based on user profile attributes. This allows administrators to rule access to documents at a very fine-grained level, beyond simple repository-based policies.

This can be done by editing conditions in the **Visibility** item of the detail panel. To add a new condition pick a profile attribute from the drop down menu, assign it a value, then click on . This will add a new condition that must be verified to allow a user to access the document. In the same way you can add further conditions, and possibly remove all of them by clicking on the eraser .

Finally, additional options that can be set for a document include:

- the refresh time for real time documents (console and dashboard);
- the possibility to encrypt it (currently not supported);
- and to make it visible/invisible to end users and executable as stand-alone.

The figure consists of two screenshots labeled 'a)' and 'b)'.

Screenshot a): A configuration interface for an analytical document. It includes fields for Label, Name, Description, Type (Report), Engine (BirtEngine), Data Source, Dataset, State (Development), Refresh seconds (0), Criptable (True), Visible (True), and Visibility restrictions (name = TEST_NAME AND surname = TEST_SURNAME). Below these, a 'Template' section shows a dropdown menu with options: surname, address, birth_date, email, family, name, and surname. The 'surname' option is selected.

Screenshot b): A 'Functionalities Tree' interface showing a hierarchical list of documents and folders. The tree includes categories like Functionalities, Analytical Model examples, Behavioural model examples, Technical Features examples, Bam, Test, Cockpits, Composite Docs with Highcharts, OpenFolder, Business Analysis, Dev, Mobile, BAM_Customer, Service analysis, Examples for book, Monitoring, Auditing, cerved, Cruscotto_Chart, testdocbrowser, Personal Folders, biuser, biadmin, Docs created by scheduler, mb, and 10416. Some items have checkmarks next to them.

FIGURE 5.33 - Managing visibility of an analytical document via a) visibility conditions and b) functionalities tree

Association with analytical drivers

We have already discussed the role of analytical drivers and how they are connected to analytical documents via parameters. In this section we show how to practically define this association.



Analytical drivers

The creation of analytical drivers is described in detail at section Behavioural Model > Analytical drivers, in this chapter.

We assume that the document template and datasets are correctly set in terms of parameter definition. In particular, they should be correctly referenced with their URI.

To add a new parameter, you can start editing the tab in the lower part of the document detail panel. Choose a human readable name for the **Title**. Then click on to choose the driver you wish to associate to the document. This will open the driver lookup window, where you can select the dataset by clicking on . You can also inspect or delete a driver from here.

Once you have selected the driver, you should write the exact **URI** of the corresponding parameter. By default the parameter is **visible**: if you want to hide it from the end user, untick the checkbox. Note that the parameter will still exist and receive values from the associated driver. However, this will be hidden and the user will not be able to see or choose any value for this parameter. Remember to save.

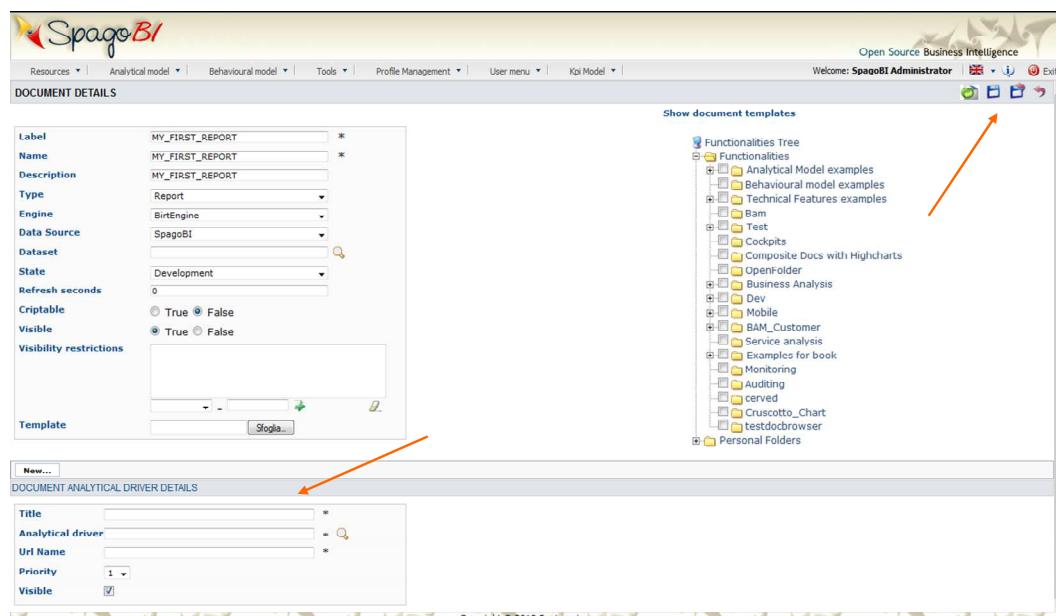
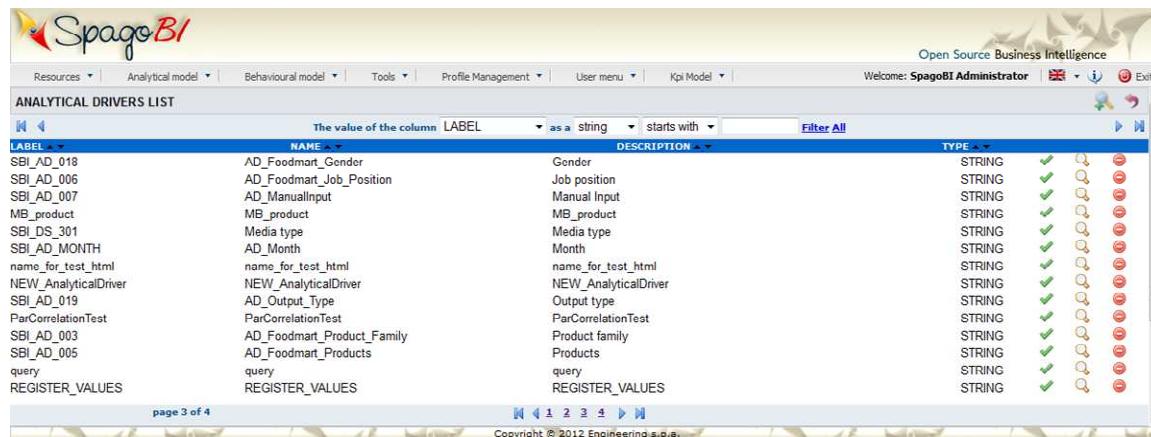


FIGURE 5.34 - Adding parameters to an analytical document

To add further parameters, click on **New**. Repeat the same procedure how many times you wish. At this point you may wish to change the order of parameters (i.e., how they are presented to the user). To do so, you can modify the **priority**.

In the following we will see some special operations that can be performed on drivers associated to a document.



The screenshot shows the SpagoBI interface with the title 'ANALYTICAL DRIVERS LIST'. The main area is a table with columns: LABEL, NAME, DESCRIPTION, and TYPE. The table contains 18 rows of driver definitions. The 'NAME' column includes entries like 'AD_Foodmart_Gender', 'AD_Foodmart_Job_Position', 'AD_ManualInput', etc. The 'DESCRIPTION' column provides a brief description of each driver. The 'TYPE' column indicates the data type for each driver, mostly STRING. The interface includes a toolbar at the top with various icons and a status bar at the bottom.

LABEL	NAME	DESCRIPTION	TYPE
SBI_AD_018	AD_Foodmart_Gender	Gender	STRING
SBI_AD_006	AD_Foodmart_Job_Position	Job position	STRING
SBI_AD_007	AD_ManualInput	Manual Input	STRING
MB_product	MB_product	MB_product	STRING
SBI_DS_301	Media_type	Media type	STRING
SBI_AD_MONTH	AD_Month	Month	STRING
name_for_test_html	name_for_test_html	name_for_test_html	STRING
NEW_AnalyticalDriver	NEW_AnalyticalDriver	NEW_AnalyticalDriver	STRING
SBI_AD_019	AD_Output_Type	Output type	STRING
ParCorrelationTest	ParCorrelationTest	ParCorrelationTest	STRING
SBI_AD_003	AD_Foodmart_Product_Family	Product family	STRING
SBI_AD_005	AD_Foodmart_Products	Products	STRING
query	query	query	STRING
REGISTER_VALUES	REGISTER_VALUES	REGISTER_VALUES	STRING

FIGURE 5.35 - Drivers lookup window

Correlation between analytical drivers

In the context of a document, two different analytical drivers may be connected to each other: this means that the possible values of a driver are limited by the value(s) of another driver.

This feature can be useful when two (or more) analytical drivers are logically related. For example, if we consider a driver retrieving all departments in a country and a driver retrieving all regions in a country, if the user selects a region, it is meaningless to show him all cities: he should only be enabled to choose among the cities in the selected region.

To configure a correlation within a document, you should first of all make sure that the parent driver and the child driver share at least one column in one LOV of a use mode. This column defines which value from the parent driver will be applied to the child, in order to constrain the results.

To set the correlation, click the tab of the driver dependent and displaying the details click on the **Correlation** button. Let us consider an example directly on the documents of SpagoBI: linking the document type to the engine (this is part of the A&M application).

A click on the correlation button  of the driver will open the correlation graphical editor. Here you can add a new correlation rule by clicking on the  icon. Here you need to define:

- the father driver
- the type of logical operator, in order to compare values of the father driver with values of the child driver
- the column, generated by the child driver, whose value will be compared with the value of the same column in the father driver.

In our case, the engine driver depends on the type driver that has DOC_TYPE as reference column.

If a driver depends on multiple father drivers, you can define multiple correlations. Create the needed correlations and choose how they are logically connected (via AND / OR operators) as shown in FIGURE 5.37. If needed, you can insert parenthesis at the extremes of each line clicking on the  and  icons.

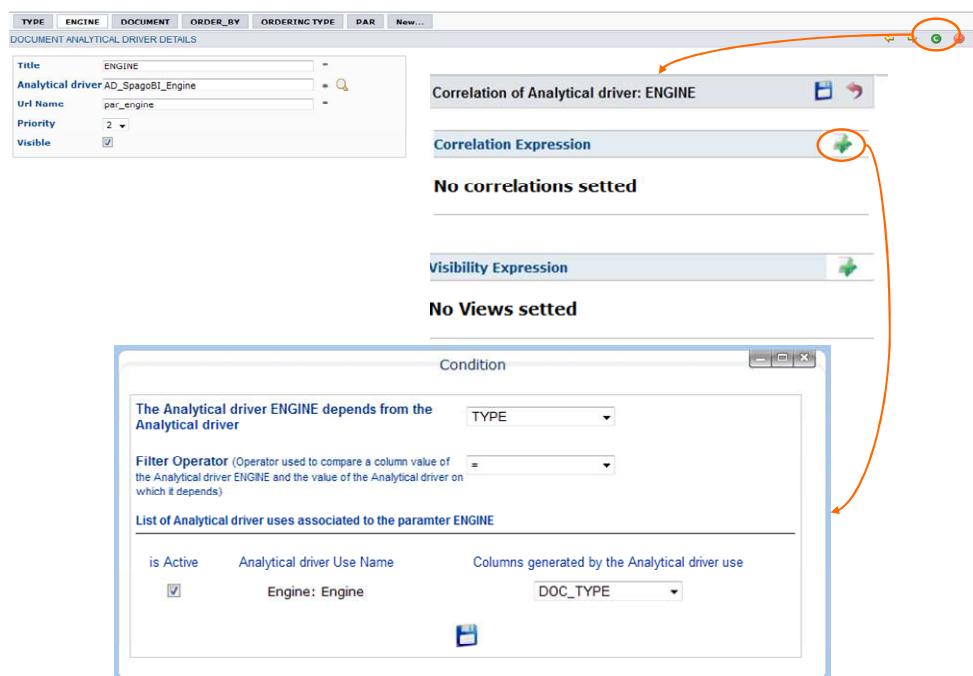


FIGURE 5.36 – Correlation between analytical drivers



FIGURE 5.37 – Multiple correlations between analytical drivers

Once defined the correlation, the child AD will display the labels during the runtime, as shown in italics:



FIGURE 5.38 – Labels displayed during the runtime

Controlled Visibility

Starting from version 3.3, another type of relation between analytical drivers has been supported by SpagoBI. It is possible to define values of a father driver that force the hiding or showing of a child driver in the parameters mask. Note that in the first case, the child driver is hidden by default, while in the second case the driver is shown by default.

To set a visibility expression, click on the button of the **Visibility Expression** area. In the graphical editor you can define visibility rules similarly to correlation ones.



Common tasks

In this section only the main common characteristics for all the document types have been explained. Others (notes, rate, bookmarks, email) are detailed in the ‘Document browser’ section.



Analytical engines and document types

In this section all the common characteristics for all the document types have been explained. However, each document type has its own peculiarities and, moreover, some of them depend on the single engine of that type.

The ‘Analytical Engines’ chapter explains all these peculiarities in detail.

Cross services

Document browser

The Document Browser is a standard functionality of SpagoBI Server. It enables users to access analytical documents, navigate document folders tree, search, sort and execute documents.

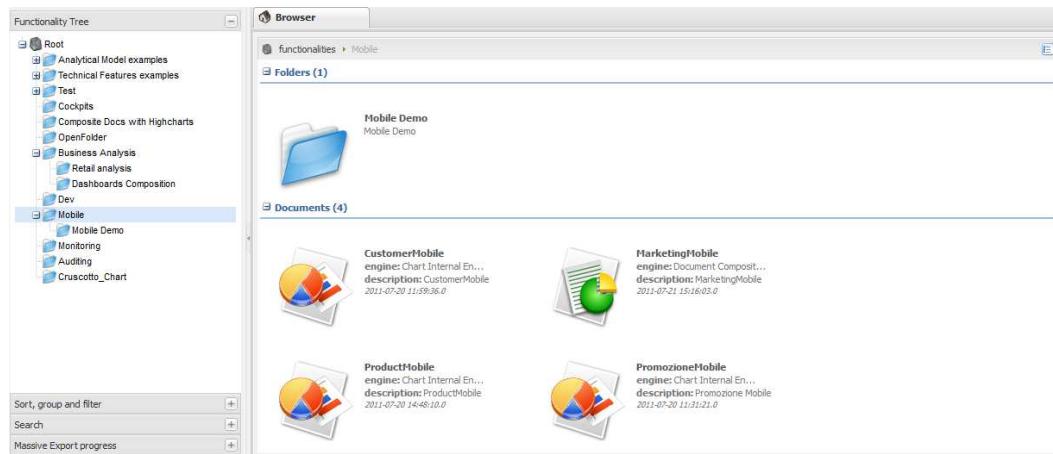


FIGURE 5.39 – Document browser

The page is composed of two vertical sections: the one on the left is collapsible and includes four expandable panels:

- **Functionality tree.** To browse documents.
- **Sort, group and filter.** To reorganize and select documents.

- **Search.** To search for documents.
- **Massive Export progress.** To monitor the export process of worksheet documents.
 - **Started Exports.** Shows documents whose export is in progress.
 - **Completed Exports.** Contains the output of export processes, i.e., exported documents.

The right panel contains the selected functionality content (documents and folders) or the search result.

Each document preview contains a small summary of the document's information such as:

- name
- engine
- description
- creation date
- an image representing the document type.

This is the primary access for document execution. Each user can see the documents which he has been granted access to, and can execute them just by clicking the document preview.

Left Panel

In the following we describe each part of the right panel.

Functionality Tree

The functionality tree allows the user to browse the hierarchical structure of functionalities where documents visible to him are stored. The tree is filtered on the user's role.



Filtered visibility on documents

SpagoBI filters visibility on documents based on user role and user profile attributes. Read more at section The behavioural Model, in this chapter.

To expand a tree node, click on the plus icon located to the left of the node name. When a functionality is expanded, all its sub-folders are shown below it. Selecting a functionality node, the contained documents and folders are shown in the right panel.

Sort, Group, Filter

This functionality is useful to inspect the right panel, where documents and folders appear.

The Sort options allow the user to order the documents shown on the right panel by name, engine or creation date.

The Group options can be applied to the groups shown on the right panel, by label, name, creation date or user documents.

The Filter options can be used to display only folders or documents on the right panel.

Search

This is a full text search functionality, which the user can perform on his documents, through a simple text search or a more complex search method.

On the **Query** field, the user shall write the text he is searching for. It is also possible to insert some wildcard symbols in the query. In detail:

- To perform a single character wildcard search use the "?" symbol.
- To perform a multiple character wildcard search use the "*" symbol.

- You cannot use the * and ? symbols as the first character of a search.

The Search in field allows to prompt an attribute of the document that must match the query text. It shows a list containing the following options:

- ALL. Categories, label, name, description of the BI documents and also over name and description of the BI object's customized views
- LABEL. Document label
- NAME. Document name
- DESCRIPTION. Document description
- List of available categories
- List of metadata categories.

The **Advanced options** field can be used to search for similar words, such as a fuzzy search using the tilde, "~", symbol at the end of the query text. No Lucene syntax symbols can be used on the Query field, when searching for similar words.

The result of the search is displayed on the right section, showing all the resulting document previews.

The search index is built the first time the web application starts. Each time a document is inserted or updated, the index gets updated. To recreate the whole index, just delete the "idx" folder on the server file system, under your application server /resources folder.

Massive Export Progress

This functionality allows the execution and the scheduling of export procedures for worksheet documents.



Massive Export

This functionality is described in detail at section Massive Export, later in this chapter.

Right panel

The right panel of the document browser shows the folders and documents contained in the current selection of the left panel.

At the top right corner of the panel, the  icon allows to reorganize the documents view as a list. To come back to the tree view, click on the  icon, which will have appeared in the meanwhile.

Each document has metadata associated: to see them, click on the small blue icon that appears when passing over a document icon. Each metadata category can be filled by double clicking on the proper value column.

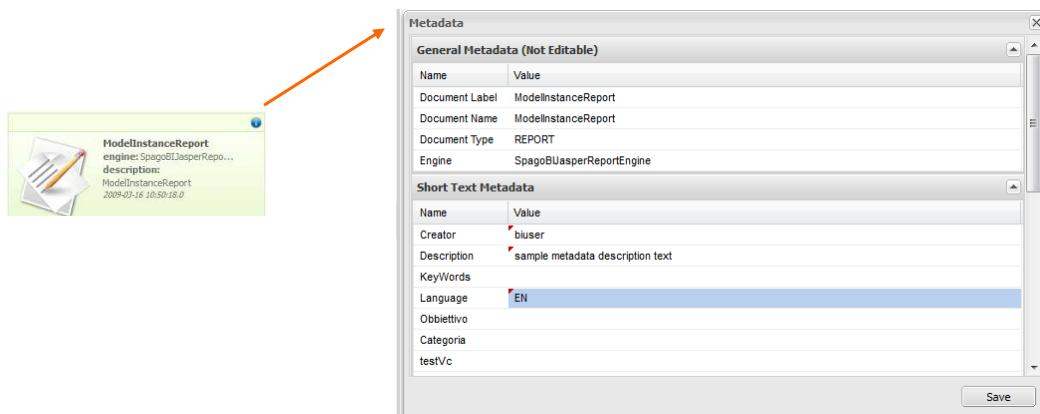


FIGURE 5.40 – View on document metadata



Document Metadata

Metadata visible in this window have been created before: learn how at section Business and Structural Metadata, later in this chapter.

Clicking on a document icon, the document will execute, showing a preview. A new tab is created in the right panel for each document executed. Documents preview tabs are kept active until the user makes a new selection on the left panel, or moves to another section of SpagoBI. In the following we provide details about the execution tab.

Execution Tab

The execution tab is composed of four sections, vertically collapsible:

- Execution section
- Customized views
- Saved parameters
- Scheduled executions.

The execution section is the area where the document is actually shown. If the document is associated to any parameter, a form to enter the requested parameters will be shown. If a parameter is mandatory, its label is drawn in bold. If a parameter depends on some other parameters, its label is drawn in italic.

After filling in parameters, click on the  icon in the top right toolbar to execute the document.

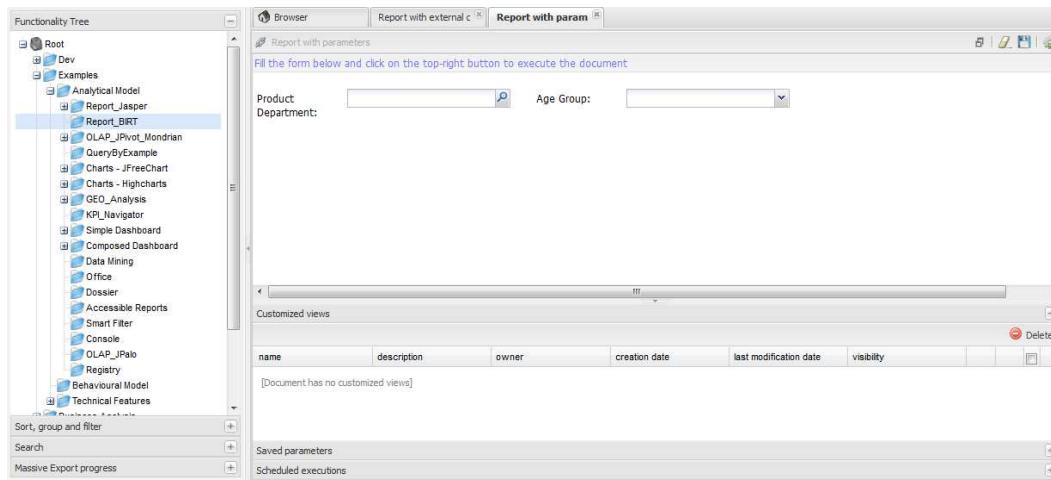


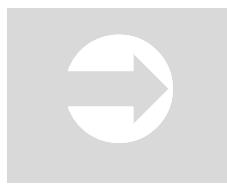
FIGURE 5.41 – Execution panel in SpagoBI document browser

Using the same toolbar, you can save the current selection of parameters form as a viewpoint, clicking on the  icon. This will save the current selection for later execution in the **Saved Parameters** section, so that they can be easily recalled without inserting values again.

If otherwise you wish to clear the current selection for parameters, click on the  icon: all values chosen for these parameters are cleared and you can start selecting new values.

The last icon on the toolbar (on the left) allows the resize of the parameters input form, hiding SpagoBI banner and leaving all space to parameters.

Section **Scheduled executions** refers to the scheduling functionality provided by SpagoBI. Here you can find the images of scheduled executions configured to produce snapshots.



Scheduled executions

SpagoBI provides a Scheduler, which supports several output options and scheduling configuration settings. Learn more at section Administrative Tools, chapter 5 – SpagoBI Server.

Once parameters are set, the document can be executed. The actual execution panel shows the result of the document execution and a richer top toolbar. Note that the functionalities of this toolbar, whose configuration depends on the authorizations granted to the user's role, are accessible whenever the document is executed: not only from the document browser but from the menu items as well.

In the following we describe all icons of the toolbar. We provide a short description, while in the next section we will go into details.

Toolbar

The full options of the toolbar are summarized in TABLE 5.2. In the following we provide details for each of them.

Rate document

The aim of this functionality is acquiring explicit data quality that can be further used within the internal SpagoBI audit and monitoring. Specifically, it allows

the administrator to identify anomalies about the use of an analytical document by end users.

Execution panel top toolbar		
	Back	Go back to the previous window.
	Expand/reduce	Make the panel larger, hiding all SpagoBI banners
	Rate document	Rate a document.
	Print document	Launch print document
	Send email	Open a window containing the form to fill the recipients of the mail, the object and the message.
	Save into personal folder	Save the current document into the personal folder.
	Bookmark	Save current document execution into user's Hot Links.
	Annotate	Add user notes to document execution
	See document metadata	See document metadata
	Exporters	Export document, depending on available export options configured for the specific engine
	Refresh/re-execute	Execute document with previously inserted parameters

TABLE 5.2 - Execution panel top toolbar

Rating a document means assigning it a value from the end user perspective. This brings additional information with respect to traditional audit and monitoring data, which can track number of executions but cannot interpret users' motivations. For example, we may find that a document is executed multiple times because it is very slow or gets stuck, which forces users to re-execute it. This is clearly a type of information that the administrator can only collect from end users.

To rate documents, click on the icon and the window in FIGURE 5.42 will appear. Choose the document and click **Vote**. The administrator can exploit the result of votes, in order to evaluate and improve the quality of the document.



FIGURE 5.42 –Document rating

Personal folder

Each user has a personal folder where he can save his documents. The folder is personal because other users (except the administrator) can not see his folder and vice-versa. It can be useful to store documents of interest or under development that are not ready for sharing yet.

The folder is located directly under the root of the functionalities tree and named after the username given at login. Note that this is the same folder where QbE documents associated to metamodels will be uploaded by SpagoBI Meta.

To save a document in a personal folder, execute a document and click on the  icon: the document will be automatically saved into your personal folder.

Bookmark (Hot links)

Bookmarks allow the user to save the current execution of a document in a list of preferred links, called **Hot links**.

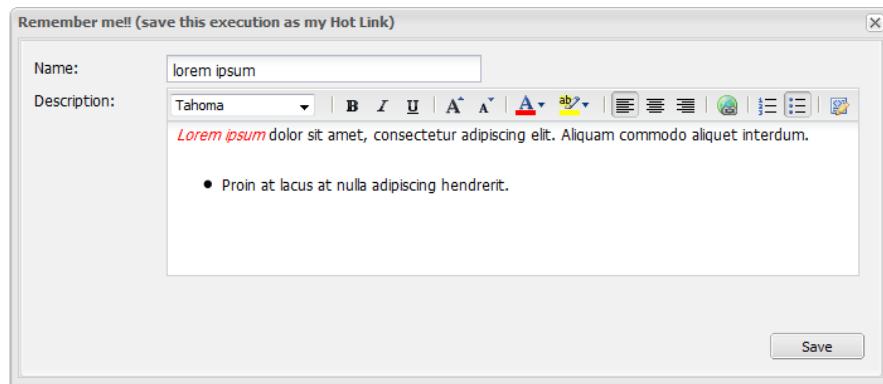


FIGURE 5.43 – “Remember me” section of the interface dedicated to *Hot links* in the user menu

The interface is divided into three main parts:

- **Remember me** - document executions that the user chose to save
- **Most Popular** - most executed documents by the user
- **My recently used sections** - recently executed documents.

If you want to add the execution of a document to your bookmarks, click on the icon. In the popup window you can give a name to the link and add comments. Click on **Save**. You can retrieve your bookmarks and links via the **User menu > Hot Links** menu item functionality.

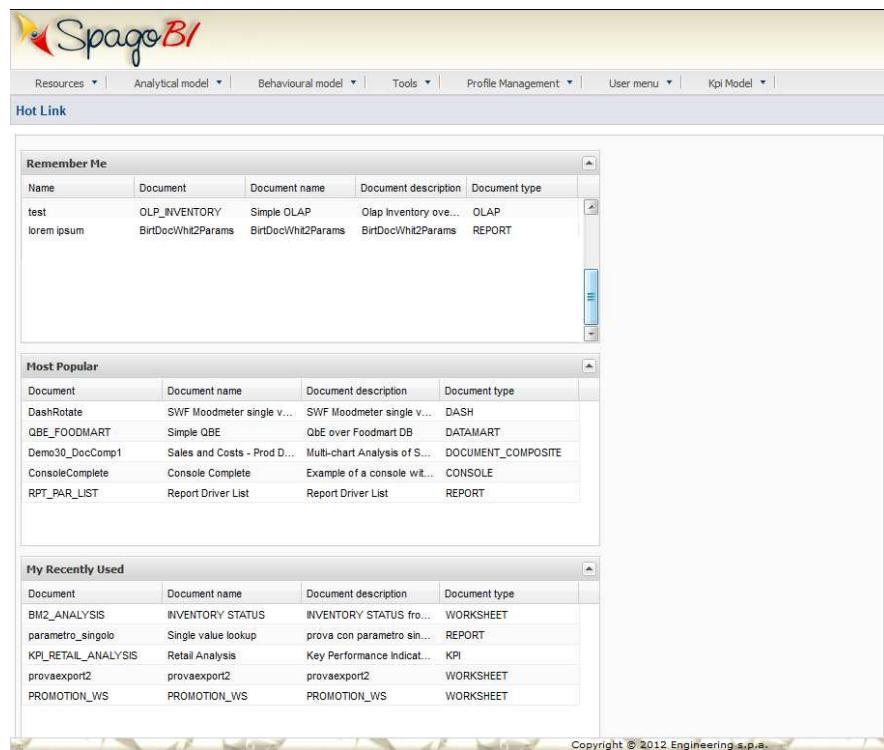


FIGURE 5.44 –Hot links visualization window

Notes

SpagoBI offers a simple collaborative tool to share notes and comments on documents, allowing users to share information and receive feedback. This may be useful, for example, to limit the number of exchanged e-mails: notes can be stored online and are accessible to all users sharing the same access rights.

Click on the icon and write your comment in the text editor. Make it private or public (i.e., accessible to users with the same rights as you) by selecting the preference in the appropriate box. Click **Save** to confirm.

All public comments from all users, as well as your private notes, will be shown the next time you open this window. If you want to edit or delete a note, click on the corresponding symbols in the selected row. Then click **Back to the list**. Selecting a row you can also export the note in PDF or RTF format.

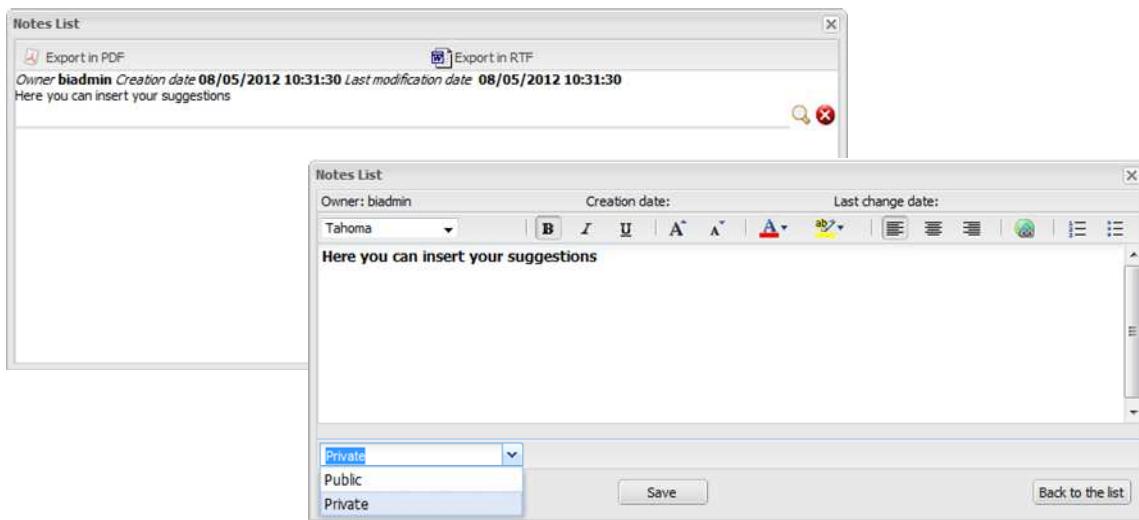


FIGURE 5.45- Note editor for analytical documents

Business and structural metadata

SpagoBI allows the definition of business metadata to describe an object, in our case a document. Business metadata are different from technical metadata used in SpagoBI Meta to build the metamodel. These are some business information associated to the document that is intended to help users to understand, access and classify it. As such, it has been mainly conceived for the end user understanding.

The user is allowed to see business metadata clicking on the icon. There are three types of business metadata, some of them are editable, while others can only be read:

- General Metadata: read-only
- Short text Metadata: editable
- Long text Metadata: editable.

General metadata contain basic information about the document, which cannot be altered because it is related to the structure of the document (e.g., type, engine, label). These can be useful since they provide a synthetic view on the document.

Short and long text metadata should be used to add relevant business information about the document: all users executing the document will see this information, which will help them understand the purpose and context of the document.

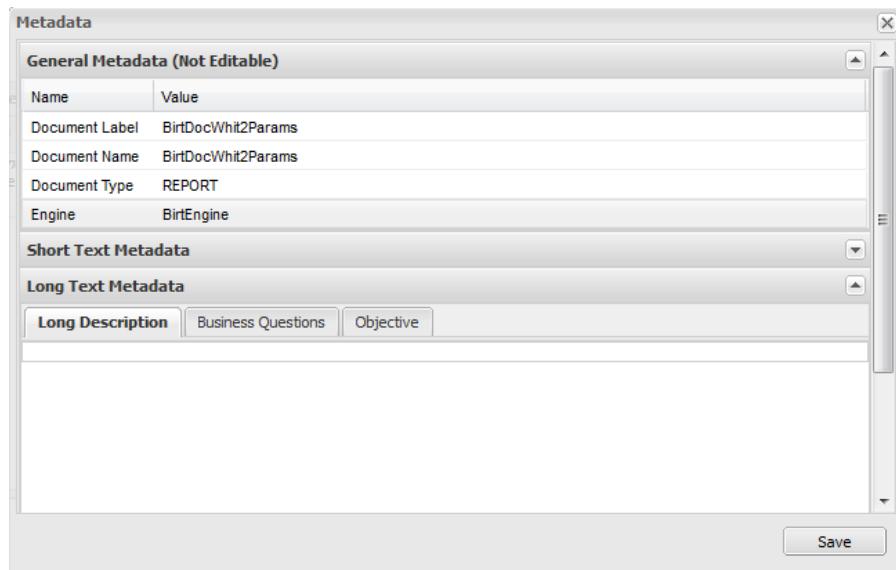
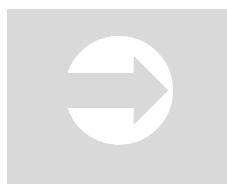


FIGURE 5.46 – Document business metadata

In general, metadata editing should be done by users with the expertise and authority to do so. Therefore, it is possible for the administrator to assign the right to save metadata via the **Profile Management > Roles Management** interface. The right is not specific to a profile, but it is part of the authorizations that can be granted to any role. This applies to bookmarks as well.



Role management

SpagoBI behavioural model includes roles, users and profiles. To understand how they relate to each other and how to assign a grant to a role, please refer to section Behavioural Model, in this chapter.



FIGURE 5.47 – Business metadata icon in the document thumbnail from the document browser

Metadata can be accessed from the execution panel, as seen above, or from the document browser before execution: just click on the little blue icon visible in each document thumbnail (see FIGURE 5.47).

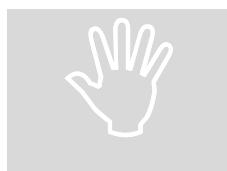
Exporters

Each SpagoBI document can be exported into several formats, depending on the options offered by the engine. In general, SpagoBI supports the following export formats:

- HTML
- XLS
- CSV
- PDF
- HTML.

Clicking on the icon, you will see the available formats for the current document. Select one and the document will be exported in the correct format.

If you want to add a new format (supported by SpagoBI) to a document type, you can do it manually by directly operating on SpagoBI metadata repository.



Modify SpagoBI metadata repository

Editing SpagoBI metadata database is a critical operation that requires both attention and expertise. The risk is to compromise the correct functioning of SpagoBI. Be sure that this operation is carried out with proper attention.

Exporters are stored as the association of an exporter type (e.g., PDF, XLS) and an engine type. So you need to create a new association and store it in the metadata repository.

The table to be modified is called `SBI_EXPORTERS`, which contains the correspondence between engines and export domains. Here you need to add a new row for each new association that is not already present.

First of all, look for the the export domain id in the `SBI_DOMAINS` table. Then look for the engine id in the `SBI_ENGINES` table. Finally, add a row to the `SBI_EXPORTERS` domain where the value for the `DEFAULT_VALUE` column is 1.

An example of code using a MySQL repository would be:

```
INSERT INTO sbi_exporters (`ENGINE_ID`, `DOMAIN_ID`,  
`DEFAULT_VALUE`) VALUES  
(<value_of_engine_id>,<value_of_domain_id>,1);
```

Refresh the Server and the list of exporters for the chosen document type will appear.

Subscriptions

Subscription lists allow the delivery of documents to users via user-defined mailing lists. This feature is used in combination with the scheduler. It can be useful, for example, when a document shall be seen by a set of users on a regular basis.



Scheduler

SpagoBI scheduler allows the execution of documents at pre-defined time. This tool is described in a dedicated section, later in this chapter.

To create a new subscription, click on **Tools > Subscription Lists**. You will see the list of all existing distribution lists. Click on the  icon: a new window will

open, where you can assign a name and a description to your new distribution list and save it.

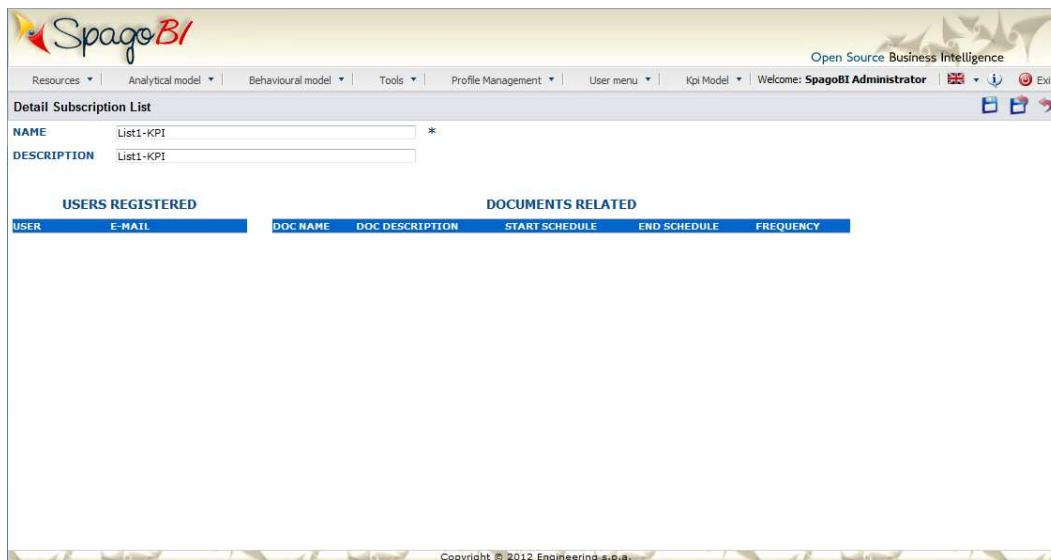


FIGURE 5.48 – Create a new distribution list

Now you can schedule one or more documents and send it to the subscription list that you have just created (see the corresponding section for details). This will create an association between the list and the document(s), which will be visible under the **Documents Related** panel, as shown in FIGURE 5.48.

To subscribe to a list, click on **User menu > Subscription List User**. Note that this menu is not visible by default in the user page, so it must be added by the administrator. The list of all existing **Distribution Lists**, which you are allowed to see, will appear. The visibility of a distribution list depends on the visibility of the analytical documents scheduled in it.

If you want to subscribe to a list, click on the icon: enter your email address and save. From now on (until you remove your subscription by clicking on) , you will periodically receive the result of the execution of the documents associated to the list.

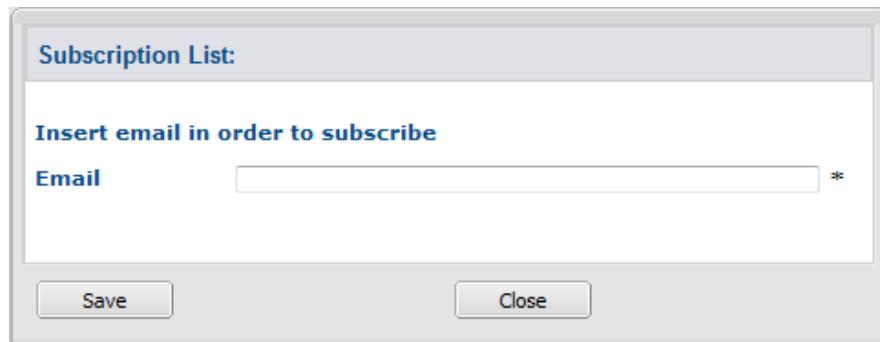


FIGURE 5.49 – Enter details to subscribe to a Distribution list

Massive export and scheduling

This feature allows the massive export of SpagoBI worksheet documents located in the same folder. This relieves the user from the burden of manually executing each document and exporting it. In addition, the export process can be immediately executed or scheduled.

Because massive export is a very resource-consuming operation, it is not allowed for every user by default. After careful evaluation, the administrator should assign this permission to the user's role via the **Profile Management --> Roles Management** interface: in the detail of a role, he should tick the **Do Massive Export** options.



Role management

SpagoBI behavioural model includes roles, users and profiles. To understand how they relate to each other and how to assign a grant to a role, please refer to section Behavioural Model, in this chapter.

The output of massive export is a .zip file containing one file for each exported document. Each file includes the name of the corresponding document; in case the process failed, the standard document is replaced by a text file explaining the reason for failure.



Export of worksheet documents

Up to SpagoBI 3.5 version, tables and crosstables included in worksheet documents are exported. The export of charts contained in a worksheet document is not supported yet.

To start a massive export process, open the document browser and select the folder of interest. If you click on the icon that appears on the top right, the export will immediately start; if you click on the icon, you can schedule the execution of the process.

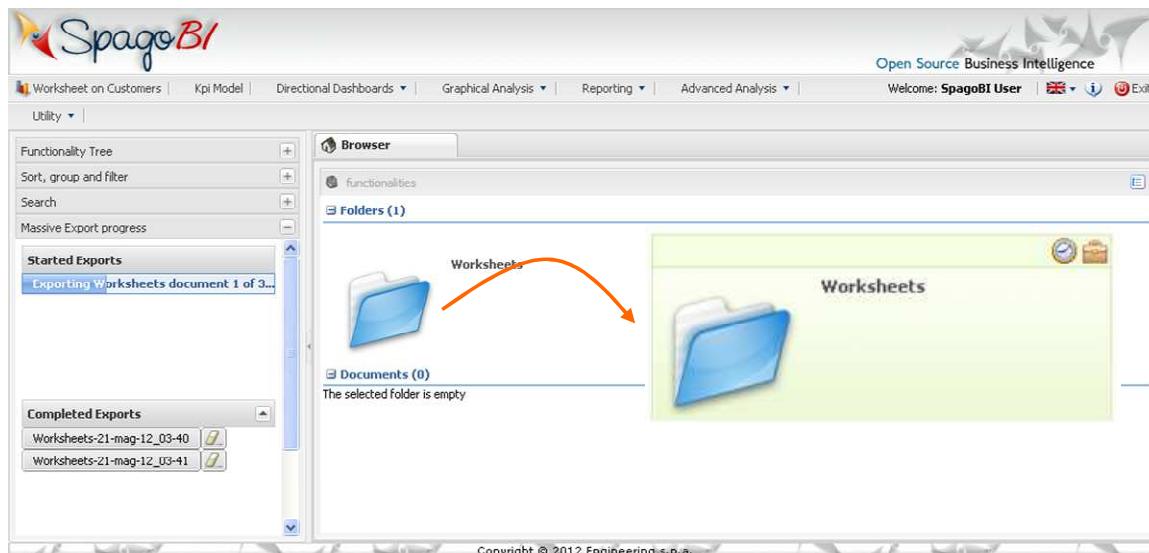


FIGURE 5.50 - Massive export of worksheet document

If you choose for immediate export, you will be prompted a wizard to:

- select the role of the user that is exporting documents. This applies to users with multiple roles only and has an impact on values retrieved from analytical drivers when executing for export.
- choose whether the export process should iterate on filters defined in the worksheet (if any)
- select the output format
- check the total number of documents to be exported.



Filters on worksheet documents

SpagoBIWorksheetEngine supports ad-hoc reporting by allowing users to build their own analytical documents. This includes the possibility to add various filters: please refer to section Ad-hoc reporting, in chapter 6, Analytical Engines.

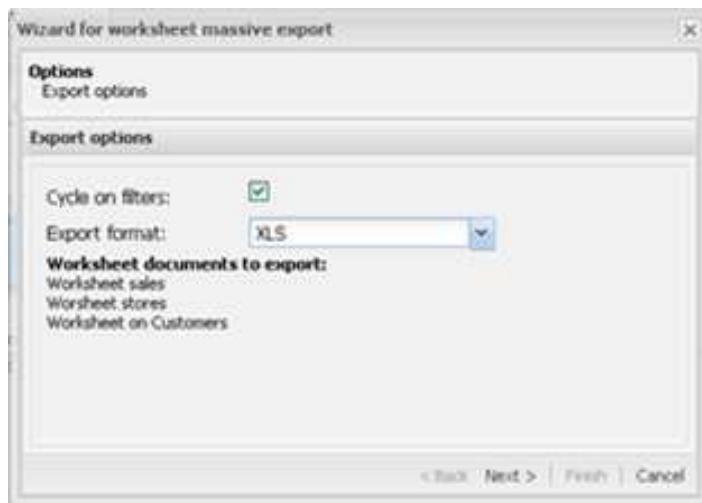


FIGURE 5.51 - Wizard for manual massive export

Once these options are set, you will have to set all analytical drivers for the documents contained in the folder. Click on Finish and the export process will start. This process is completely asynchronous, so the user can perform other activities or even terminate the session and come back later again.

You can track the state of both manual and scheduled export processes from the left panel in the document browser, under **Massive Export Progress** (see FIGURE 5.50). Under Started Exports you will find all processes either manually started or scheduled: if they are in progress, a bar will show the advancement of the process in terms of number of exported documents. In the Completed Exports section, terminated processes are shown. Click on the process to download its output and then to delete it from the list.

Now let us see the case of scheduled export. As we said above, click on and the scheduled export wizard will require the following information:

- date and time to start export
- date and time to stop export
- the frequency of the schedule: if you choose **Oneshot**, the execution will happen only once, otherwise a regular execution can be set.

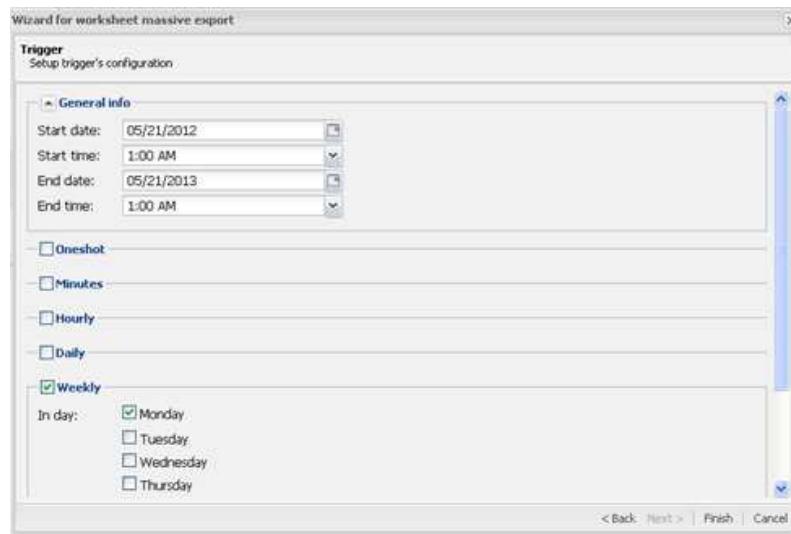


FIGURE 5.52 - Wizard for scheduled export of worksheet documents

Administrative Tools

Scheduler

SpagoBI scheduler allows to schedule the execution of one or more analytical documents published on the Server. Documents executed by the scheduler can then be distributed along different dispatching channels.

In order to define a new scheduled activity the administrator must specify which documents compose the activity and how to execute them. The list of all scheduled activities can be seen selecting **Tools > Scheduler**.

To create a new activity click on  in the up right corner. Give a name and a description to the new activity and then select the documents that compose it by clicking on  and selecting them from the pop up wizard.

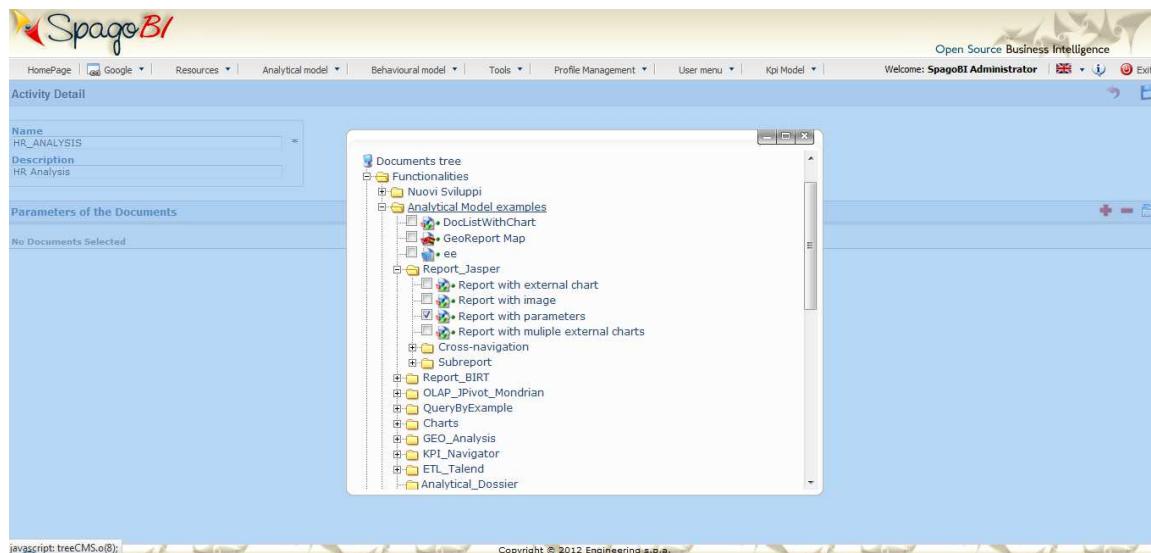


FIGURE 5.53 – Selecting documents for a scheduled activity

Now you need to specify how the scheduler must handle the analytical drivers of each selected document having parameters. There are two possibilities:

- selecting a value from those available for the specific analytical driver at definition time (like in the case of parameter **Output format** in FIGURE 5.54);
- executing the report one time for each possible values of the analytical driver (like in the case of parameter **Position** in FIGURE 5.54).

A scheduled activity can be composed by more than one report. It is also possible to add the same report to a scheduled activity more than once using the button .

Once all desired documents have been added and the management configuration of their parameters has been set up, save the activity by clicking on . The new activity is shown in the list and can be modified or deleted using respectively .

Activity Detail

Name: HR_ANALYSIS *

Description: HR Analysis

Parameters of the Documents

Report with parameters

Position: How do you want to assign values to the parameter?
Load values from a parameter modality just before execution

Role to be used to retrieve values: /spagobi/admin
Iterate for each value

Output Format: How do you want to assign values to the parameter?
Assign fixed values now

(Fill using role /spagobi/admin)
PDF

Don't iterate for each value

FIGURE 5.54 - Setting document parameters in a scheduled activity

You can associate one or more schedules to all defined activities. The number of schedules associated to one activity is shown in the column **Number of schedules**. To see and modify the list of the schedules associated to an activity, click on the . Similarly, click on the same button to associate schedules to newly created activities.

A schedule is composed of two types of configuration options:

- trigger configurations
- dispatch configurations.

Trigger configurations are located at the top part of the schedule detail page and are common for all the documents that compose the scheduled activity. By means of trigger configurations, it is possible to specify when the execution of the scheduled activity must be executed. It is possible to configure trigger options in order to have one single execution in a particular moment in time (see FIGURE 5.55 a) or to have multiple executions within a particular period (see FIGURE 5.55 b).

Schedule detail

Name: HR_ANALYSIS_194028

Description:

Start Date: 30/04/2012 *

Start Time: 12:00

End Date: 30/04/13

End Time: 12:00

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

Schedule detail

Name: HR_ANALYSIS_194028 *

Description:

Start Date: 30/04/2012

Start Time: 12:00

End Date: 30/04/13

End Time: 12:00

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

Every n months: 1
 In month: JAN FEB MAR APR MAY JUN JUL AUG
 SEP OCT NOV DIC

The day: 1
 The week: First Second Third Fourth Last
 In day: SUN MON TUE WED THU FRI SAT

a)

b)

FIGURE 5.55 - Schedule configuration: a) trigger and b) dispatch

Dispatch configuration instead are located in the bottom part of the schedule detail page and are different for all the documents that compose the scheduled activity. All documents that compose the activity have their own dispatch configuration and the same document can be distributed along multiple dispatch channels.

There are 6 possible dispatch channels that can be used to distribute the results of the execution of a scheduled activity:

- Save as snapshot
- Save as file
- Save as document
- Send to Java class
- Send mail
- Send to distribution lists.

In the following sections we explain them in detail.

Save as snapshot

The executed document can be saved as snapshots in cyclic buffers of configurable size. For example, it is possible to store in the buffer the last 12

snapshots (the **History Length** field) of one report, scheduled to be executed one per month, in order to have a one-year long history.

The list of all snapshots contained in the buffer can be accessed from the **Scheduled executions** panel contained in the bottom part of the document execution panel (see FIGURE 5.56). Each snapshot can be opened or deleted from this panel. A snapshot contains data queried from the database at the moment of its execution performed by the scheduler.

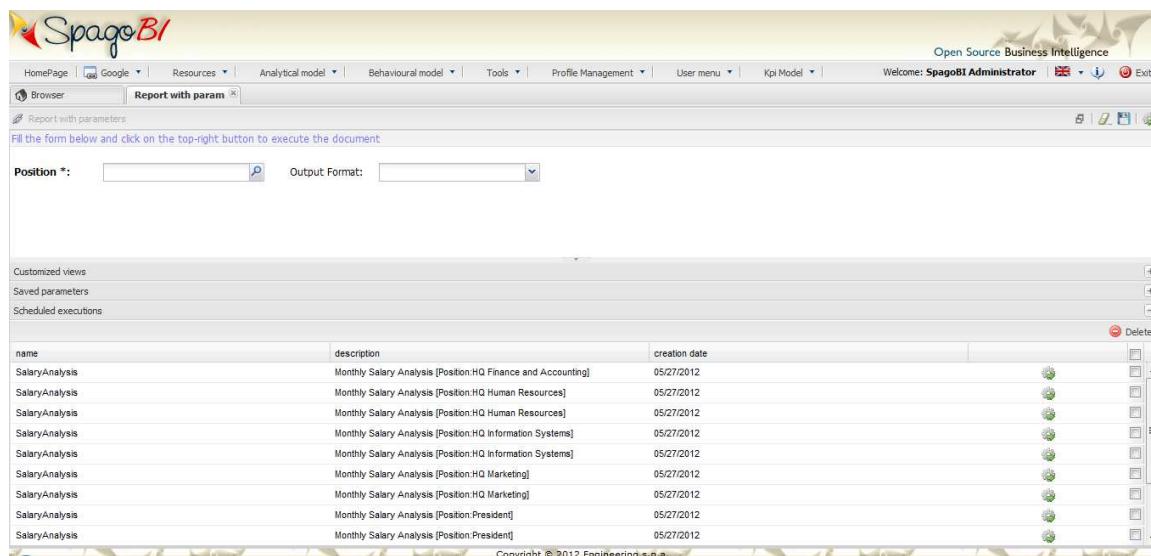


FIGURE 5.56 - Check snapshots on Scheduled executions panel

Save as file

The executed document can be saved as file on the server filesystem in a subfolder of SpagoBI /resources folder. The administrator can specify the relative path of this subfolder.

Save as document

The executed document can be saved as office document on the SpagoBI functionality tree. The document execution will be saved in the specified folder and will be visible to all the users that can access that particular folder.

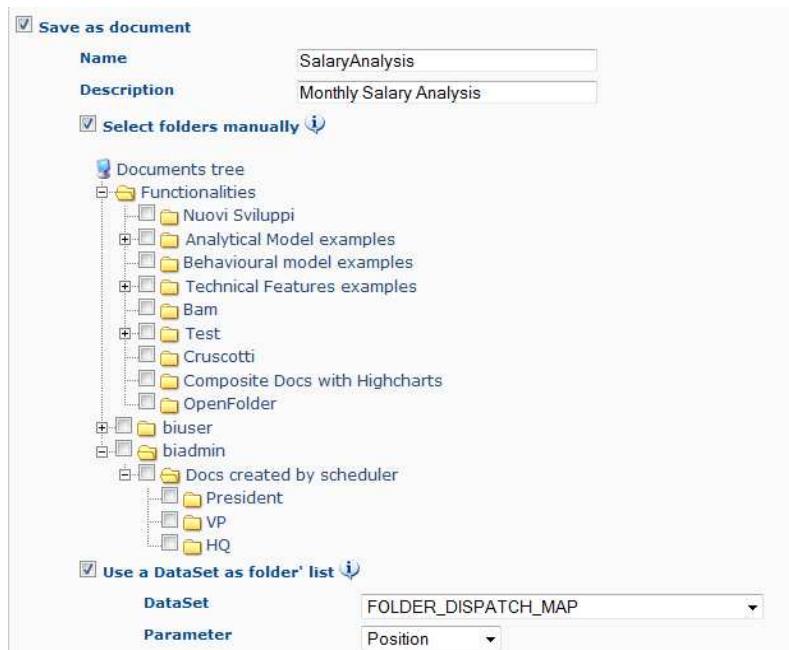
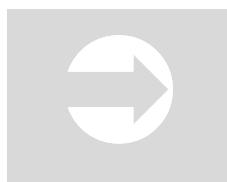


FIGURE 5.57 - Save scheduled execution as a document in SpagoBI functionalities tree

For those documents whose execution is iterated over a parameter value, it is also possible to use the value of the parameter to decide to which folder the document shall be dispatched. To do so, define a mapping dataset composed of two columns:

- the first containing a specific parameter value
- the second containing the label of the folder where the document shall be dispatched when the document is executed with the corresponding parameter value.



Dataset definition

To learn how to create a dataset, please refer to section Data Sets in this chapter.

Once you have defined the mapping dataset, you can use it in the configuration settings of the document dispatcher, as shown in FIGURE 5.57.

parameter_value	folder_label
1 President	President
2 VP Country Manager	VP
3 VP Information Systems	VP
4 VP Human Resources	VP
5 VP Finance	VP
6 HQ Information Systems	HQ
7 HQ Marketing	HQ
8 HQ Human Resources	HQ
9 HQ Finance and Accounting	HQ

FIGURE 5.58 - Dataset to dispatch execution results to different folders

Like in the previous case, the scheduler will execute the report one time for each possible value of the parameter. This time, however, execution results will be dispatched in different folders, according to the mapping defined in the dataset.

Send to Java class

The executed document can be sent to a Java class implementing a custom dispatch logic. The custom class must extend the abstract class `JavaClassDestination` that implements the method `execute`. This method is called by the scheduler after document execution.

Below an example of Java class.

```
package it.eng.spagobi.tools;

import it.eng.spagobi.analyticalmodel.document.bo.BIOBJECT;

public abstract class JavaClassDestination
    implements IJavaClassDestination {

    BIOBJECT biObj=null;
    byte[] documentByte=null;

    public abstract void execute();

    public byte[] getDocumentByte() {
        return documentByte;
    }
}
```

```

    }

    public void setDocumentByte(byte[] documentByte) {
        this.documentByte = documentByte;
    }

    public BIObject getBiObj() {
        return biObj;
    }

    public void setBiObj(BIObject biObj) {
        this.biObj = biObj;
    }

}

```

The method `getDocumentByte` can be used to get the executed document, while the method `getBiObj` can be used to get all metadata related to the executed document.

The following code snippet shows an example of a possible extension of class `JavaClassDestination`.

```

public class FileDestination extends JavaClassDestination {

    public static final String OUTPUT_FILE_DIR = "D:\\ScheduledRpts\\";
    public static final String OUTPUT_FILE_NAME = "output.dat";

    private static transient Logger logger =
Logger.getLogger(FileDestination.class);

    public void execute() {
        File outputDir;
        File outputFile;
        OutputStream out;
        byte[] content = this.getDocumentByte();
        String outputFileName;

        logger.debug("IN");

        outputFile = null;
        out = null;

try {
    outputFileName = getFileName();

```

```

        logger.debug("Output dir [" + OUTPUT_FILE_DIR + "]");
        logger.debug("Output filename [" + outputFileName +
" ]");

        outputDir = new File(OUTPUT_FILE_DIR);
        outputFile = new File(outputDir, outputFileName);

        if(!outputDir.exists()) {
            logger.debug("Output dir ["
+ OUTPUT_FILE_DIR
+ " ] does not exist");

            logger.debug("Creating output dir ["
+ OUTPUT_FILE_DIR
+ " ] ...");

        if(outputDir.mkdirs()) {
            logger.debug("Output dir ["
+ OUTPUT_FILE_DIR + "] succesfully created");
            } else {
                throw new SpagoBIRuntimeException(
"Impossible to create outputd dir ["
+ OUTPUT_FILE_DIR + "]");
            }
        } else {
            if(!outputDir.isDirectory()) {
                throw new SpagoBIRuntimeException(
"Outputd dir ["
+ OUTPUT_FILE_DIR
+ " ] is not a valid directory");
            }
        }

        try {
            out = new BufferedOutputStream(
new FileOutputStream(outputFile));
        } catch (FileNotFoundException e) {
            throw new SpagoBIRuntimeException(
"Impossible to open a byte stream to file ["
+ outputFile.getName() + "]", e);
        }

        try {
            out.write(content);
        } catch (IOException e) {
            throw new SpagoBIRuntimeException(
"Impossible to write on file ["
+ outputFile.getName() + "]", e);
        }
    }
}

```

```

        }

    } catch(Throwable t) {
        throw new SpagoBIRuntimeException(
"An unexpected error occurs while saving document"
+ " to file [" + outputFile.getName() + "]", t);
    } finally {
        if(out != null) {
            try {
                out.flush();
                out.close();
            } catch (IOException e) {
                throw new SpagoBIRuntimeException(
"Impossible to properly close file [" +
outputFile.getName() + "]", e);
            }
        }
        logger.debug("OUT");
    }
}

private String getFileName() {
    String filename = "";
    BIOBJECT analyticalDoc;
    List analyticalDrivers;
    BIOBJECTParameter analyticalDriver;
    String extension = "pdf";

    analyticalDoc = getBiObj();
    analyticalDrivers =
analyticalDoc.getBiObjectParameters();
    for(int i = 0; i < analyticalDrivers.size(); i++) {
        analyticalDriver =
(BIOBJECTParameter)analyticalDrivers.get(i);

        String parameterUrlName =
analyticalDriver.getParameterUrlName();
        List values =
analyticalDriver.getParameterValues();

        if(!parameterUrlName.equalsIgnoreCase("outputType")){
            filename += values.get(0);
        } else {
            extension = "" + values.get(0);
        }
    }
}

```

```
        filename = filename.replaceAll("[^a-zA-Z0-9]", "_");
        filename += "." + extension;

        return filename;
    }

}
```

The class `FileDestination` copies the executed documents to the local filesystem in a folder named `D:\ScheduledRpts`. The name of the report file is generated concatenating all the parameter values used by the scheduler during execution.

Once implemented and properly compiled, the Java class must be exposed to the classpath of SpagoBI web application. For example, you can pack the compiled class into a jar file, copy it into the lib folder of SpagoBI web application and restart the server.

As a last step, it is necessary to assign the fully qualified name of the new class, e.g., `it.eng.spagobi.tools.DestinationFile.`, to the configuration property `classpath`.

Send mail

The executed document can be sent to one or more mail recipients. The list of mail addresses to be used to forward the executed document can be defined in three different ways:

- statically
- dynamically, using a mapping dataset or
- dynamically, using a script.

Static list

If you want to choose a static list, check the option `Fixed list of recipients` and fill the configuration property `Mail to` with the list of desired mail

addresses separated by a comma, as shown in figure. An mail for each executed document will be sent to all the mail addresses contained in the list.

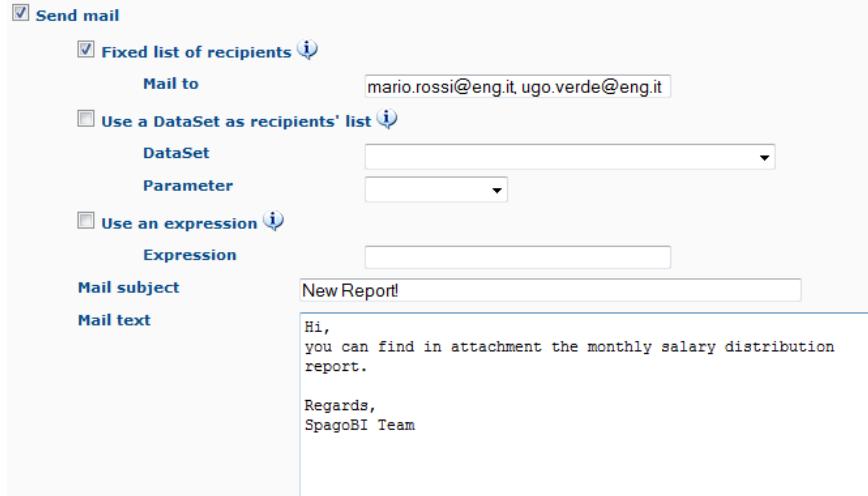


FIGURE 5.59 - Static configuration of email recipients of a scheduled execution

Dynamic list with mapping dataset

In this case, you have to define a two-column dataset:

- the first containing a specific parameter value
- the second containing each mail address the executed document should be dispatched to.

Basically when the parameter has a given value, the document will be sent to the corresponding email address.

Preview/Values for all Parameters	
parameter_value	mail_address
1 President	Alessandra.Toninelli@eng.it
2 VP Country Manager	monia.spinelli@eng.it
3 VP Information Systems	monia.spinelli@eng.it
4 VP Human Resources	monia.spinelli@eng.it
5 VP Finance	monia.spinelli@eng.it
6 HQ Information Systems	andrea.gioia@eng.it
7 HQ Marketing	andrea.gioia@eng.it
8 HQ Human Resources	andrea.gioia@eng.it
9 HQ Finance and Account...	andrea.gioia@eng.it

FIGURE 5.60 - Example of mapping dataset for dynamic distribution list

Once you have defined the mapping dataset, you can use it in the configuration settings of the document dispatcher, as shown in FIGURE 5.61 a.



a)



b)

FIGURE 5.61 - Dynamic configuration of distribution list: a) with mapping dataset and b) wth script

With this configuration, the scheduler will execute the report one time for each possible value of the parameter **Position**, then dispatching the results to different recipients. Specifically, all execution results passing a value of the **Position** parameter to the report starting with VP will be sent to Monia, the ones starting with HQ will sent to Andrea and the ones starting with President will be sent to Alessandra.

Dynamic list with script

Check the option **Use an expression** (see FIGURE 5.61 b) and assign a value to the configuration property **Expression** with a parameter-dependent expression like the following:

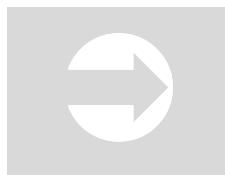
```
$P{dealer}@eng.it
```

Where:

dealer is a document parameter label (`$P{dealer}` will be replaced by the parameter value of the scheduled execution).

Send to distribution list

The executed documents can be sent to one or more pre-defined distribution lists. All users subscribed to one of the selected distribution lists will receive the executed documents by mail.



Subscription lists

To learn how to create a distribution (subscription) list, please refer to section Administrative Tools > Subscriptions, in this chapter.

A screenshot of the SpagoBI interface. At the top left, there is a checked checkbox labeled "Send to Distribution Lists". Below it is a table with two rows. The table has two columns: "NAME" and "DESCRIPTION". The first row contains "Test List" under NAME and "Test List" under DESCRIPTION. The second row contains "HRTeam" under NAME and "HR Team" under DESCRIPTION. Both rows have a small checkbox column to their left, with the second row's checkbox being checked.

	NAME	DESCRIPTION
<input type="checkbox"/>	Test List	Test List
<input checked="" type="checkbox"/>	HRTeam	HR Team

FIGURE 5.62 - Send execution results to distribution list

Set all schedule configurations (trigger and dispatch), then save the schedule. The new activity will be shown in the list. It can be modified or deleted clicking on the corresponding icons.

Import/export

This tool allows the export of all or a subset of SpagoBI documents currently registered on the Server, together with all SpagoBI objects connected to them, such as datasets or analytical drivers. It is a quick and easy way to re-create a business analysis on a new instance of SpagoBI Server, without having to manually perform copies of objects and documents.

In the import/export procedure most SpagoBI objects are involved, including documents, drivers, LOVs and roles. The following objects are not exported and should therefore be re-created in the target environment:

- Users
- Menu configuration
- Hot links and Remember me
- Static resources

To start an export procedure, click on the **Tools > Import/Export** to open the wizard, which is divided in two parts (as shown in FIGURE 5.63). The left panel allows to export documents, while the right panel imports them.

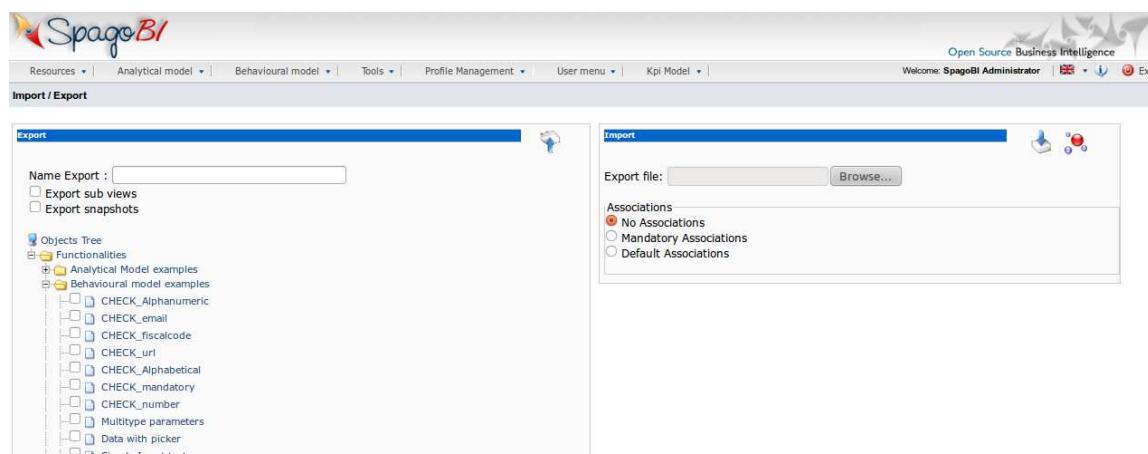


FIGURE 5.63 - Import/export graphical interface

Let us now start with the export procedure. In the **Name Export** field enter a name for the package that will contain the exported documents. Select the items you want to export on the **Objects Tree**. This tree represents the Functionalities tree of your Server. Note that you are not required to manually select drivers or datasets associated to documents, since this is automatically managed by the exporter.

Click on  at the right top corner of the Export session and wait for the process to complete. When the export is complete, click on **Download**, where you will find the package containing documents and related objects.

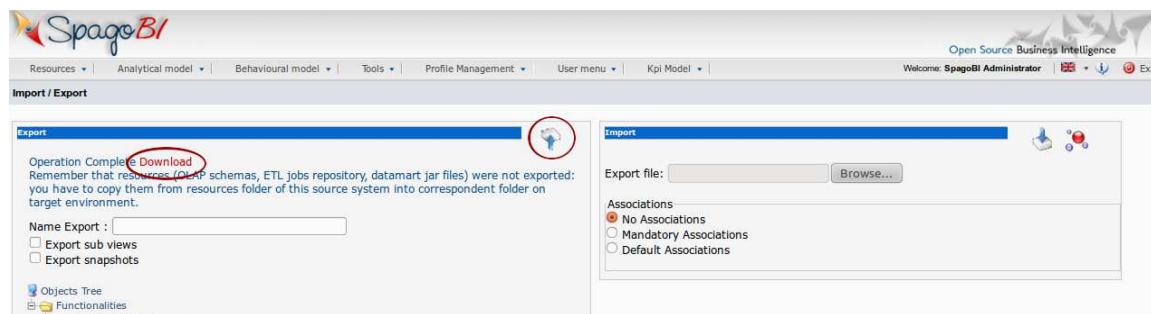


FIGURE 5.64 - Download exported packages

The package that you have downloaded through this procedure allows you to import your documents into another SpagoBI environment.

Move to the right panel of the interface. **Browse** your file system to select the package and load it. By default no association is selected, which means that you will have to manually map objects of the imported package with objects already defined on the destination environment (e.g., roles and drivers).

If you want to manage associations, click on  to open the file containing the list of associations, which you can manually edit. Note that this manual operation requires some expertise, so you may prefer to skip this step and manage associations via the guided procedure (see below).

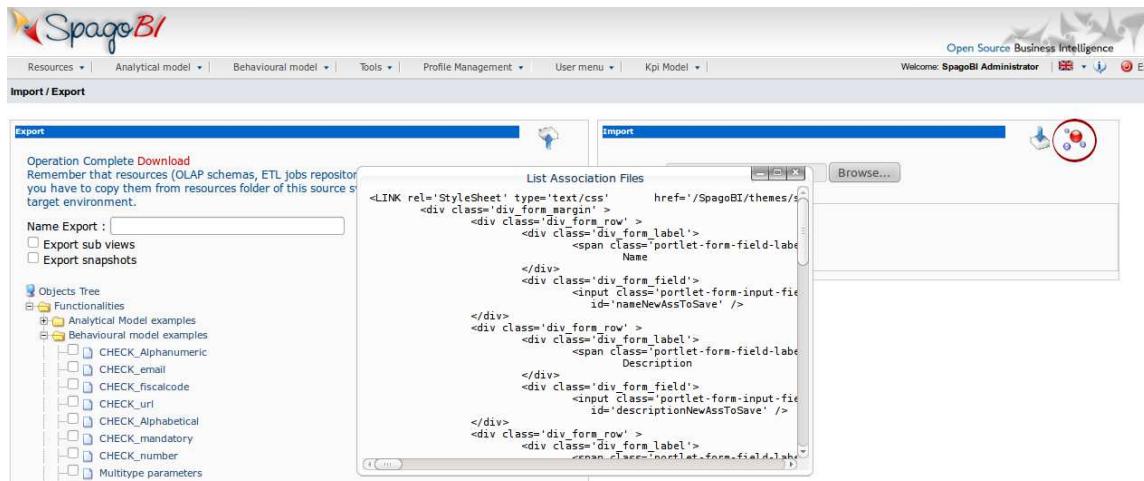


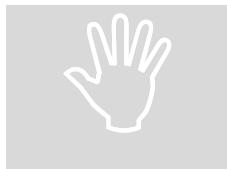
FIGURE 5.65 - Upload imported package

Click on and wait for the importer to complete its process.

On the first page you are shown **Role Associations**: check that roles already defined in the destination environment are properly mapped onto the roles defined in the imported package. Then click **Next** and proceed in the same way with **Engine Associations**, and with **Data Source Associations**.

The last page concerns possible **Metadata** conflicts. Here you have to choose if you wish to overwrite existing meta-data with the imported ones. When you are done, click on **Next** to complete the import procedure.

Imported documents will be included in the Functionalities tree under the same folder of the source environment. If not already in place, the folder will be created.



Overwriting metadata

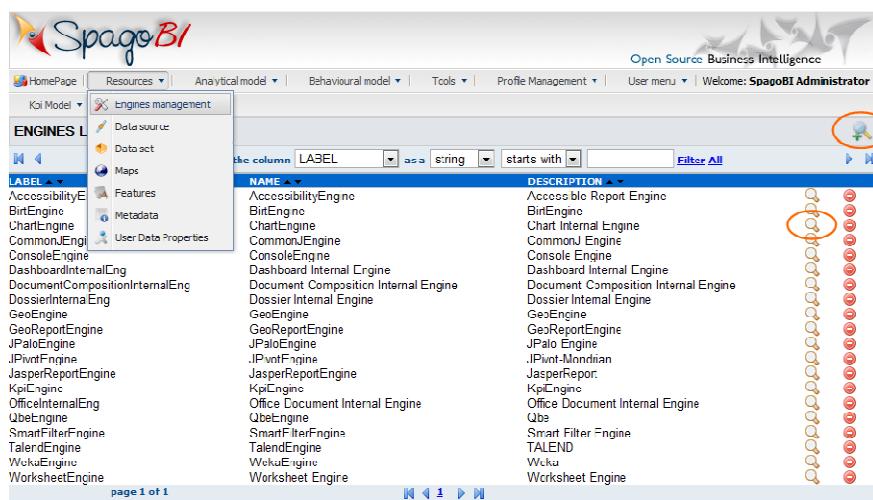
Overwriting metadata during an import procedure may be convenient when you are sure that the new environment and the one you are overwriting contain the same objects (e.g., roles, drivers). Otherwise, it may be safer not to overwrite and manually manage the merge of metadata.

Engine management

As explained in the *Analytical Model* section, each analytical domain (report, chart, cockpits, etc.) is supported by one or more engines, developed as external or internal ones. The external engines are single web applications related to SpagoBI core, the internal engines are simpler SpagoBI classes delegated to display the final results of an analytical document.

Engines can work on different servers and the same engine can even be instanced many times, so as to distribute the workloads according to the customer needs and usages. To this end, make sure that you have properly installed and configured the servers and the engines.

The **Engine Management** function is available for the administrator under the **Resources** menu.



NAME	DESCRIPTION			
AccessibilityEngine	Accessible Report Engine			
BirtEngine	Birt Engine			
ChartEngine	Chart Internal Engine			
CommonEngine	Common Engine			
ConsoleEngine	Console Engine			
DashboardInternalEng	Dashboard Internal Engine			
DocumentCompositionInternalEng	Document Composition Internal Engine			
DossierInternalEng	Dossier Internal Engine			
GeoEngine	Geo Engine			
GeoReportEngine	GeoReport Engine			
JPaloEngine	JPalo Engine			
JPivotEngine	JPivot Engine			
JasperReportEngine	JasperReport Engine			
KpiEngine	Kpi Engine			
OfficeInternalEng	Office Document Internal Engine			
QbeEngine	Qbe Engine			
SmartFilterEngine	Smart Filter Engine			
TalendEngine	TALEND			
WokuEngine	Woku Engine			
WorksheetEngine	Worksheet Engine			

FIGURE 5.66 – Engine list

Selecting the icon a new engine is added; selecting the , the administrator can access the detail page.

ENGINE DETAILS	
Label	QbeEngine *
Name	QbeEngine *
Description	Qbe
Document type	Datamart Model
Engine type	External Engine
Use Data Set	<input type="checkbox"/>
Use Data Source	<input checked="" type="checkbox"/>
Data Source	FoodMart
Url	/SpagoBIQbeEngine/servlet/AdapterHTTP *
Secondary Url	
Driver Name	it.eng.spagobi.engines.drivers.qbe.QbeDriver *

FIGURE 5.67 – Engine detail

Here the administrator can set relevant engine attributes:

- **Label:** the short identifier of the engine
- **Name:** the full name of the engine
- **Description:** a free description of the engine
- **Document type:** which analytical domain the engine refers to
- **Engine type:** internal or external engine
- **Use data set:** flag to set if the engine uses data set as data provider
- **Use data source:** flag to set if the engine uses a specific data source
- **Data source:** the data source used by the engine, when the previous flag is true
- **URL:** the main URL where the engine works. It is used for external engines only and can be set as an absolute path (`http://<server>:<port>/<path>`) to directly refer different server instances, or it can be set in a relative way and, in that case, the path refers the application server root.
- **Secondary URL:** the URL that will be used for scheduled execution. If not defined, the main URL will be used.
- **Driver name:** name of the engine main class.

The following table shows how to configure all SpagoBI Engines, with their relevant attributes, setting the URLs as relative paths.

SpagoBIBirtReportEngine			
Doc. type	Report	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIBirtReportEngine/BirtReportServlet		
Driver	it.eng.spagobi.engines.drivers.birt.BirtReportDriver		
SpagoBIJasperReportEngine			
Doc. type	Report	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIJasperReportEngine/JasperReportServlet		
Driver	it.eng.spagobi.engines.drivers.jasperreport.JasperReportDriver		
SpagoBIAccessibleReportEngine			
Doc. type	Report Accessible	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIAccessibilityEngine/servlet/AccessibilityServlet		
Driver	it.eng.spagobi.engines.drivers.accessibility.AccessibilityDriver		
SpagoBIJFreeChartEngine			
Doc. type	Dashboard	Engine type	Internal
Data set	YES	Data source	NO
Driver	it.eng.spagobi.engines.chart.SpagoBIChartInternalEngine		
SpagoBIJSChartsEngine			
Doc. type	Chart	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIChartEngine/servlet/AdapterHTTP?ACTION_NAME=CHART_ENGINE_EXTJS_START_ACTION		
Driver	it.eng.spagobi.engines.drivers.generic.GenericDriver		
SpagoBIJPaloEngine			
Doc. type	OLAP	Engine type	External
Data set	NO	Data source	NO
URL	/SpagoBIJPaloEngine/com.tensegrity.wpalo.SpagoBIJPaloEngine/JPaloEngineStartServlet		

Driver	it.eng.spagobi.engines.drivers.jpalo.JPaloDriver		
SpagoBIJPivotEngine			
Doc. type	OLAP	Engine type	External
Data set	NO	Data source	YES
URL	/SpagoBIJPivotEngine/JPivotServlet		
Driver	it.eng.spagobi.engines.drivers.jpivot.JPivotDriver		
SpagoBIJPXMLEngine			
Doc. type	OLAP	Engine type	External
Data set	NO	Data source	NO
URL	/SpagoBIJPXMLEngine/JPivotServlet		
Driver	it.eng.spagobi.engines.drivers.jpivot.JPivotDriver		
SpagoBIQbeEngine			
Doc. type	Free inquiry	Engine type	External
Data set	NO	Data source	YES
URL	/SpagoBIQbeEngine/servlet/AdapterHTTP		
Driver	it.eng.spagobi.engines.drivers.qbe.QbeDriver		
SpagoBISmartFilterEngine			
Doc. type	Free Inquiry - Smart Filter	Engine type	External
Data set	NO	Data source	YES
URL	/SpagoBIQbeEngine/servlet/AdapterHTTP		
Driver	it.eng.spagobi.engines.drivers.smartfilter.SmartFilterDriver		
SpagoBIWorksheetEngine			
Doc. type	Ad-hoc reporting	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIQbeEngine/servlet/AdapterHTTP		
Driver	it.eng.spagobi.engines.drivers.worksheet.WorksheetDriver		
SpagoBICConsoleEngine			
Doc. type	Real-time Console	Engine type	External
Data set	NO	Data source	YES
URL	/SpagoBICConsoleEngine/servlet/AdapterHTTP?ACTION_NAME=CONSOLE_ENGINES_START_ACTION		
Driver	it.eng.spagobi.engines.drivers.generic.GenericDriver		

SpagoBIDashboardEngine			
Doc. type	DashBoard	Engine type	Internal
Data set	YES	Data source	NO
Driver	it.eng.spagobi.engines.dashboard.SpagoBIDashboardInternalEngine		
SpagoBIGeoEngine			
Doc. type	Location Intelligence	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIGeoEngine/servlet/AdapterHTTP		
Driver	it.eng.spagobi.engines.drivers.geo.GeoDriver		
SpagoBIGisEngine			
Doc. type	Location Intelligence	Engine type	External
Data set	YES	Data source	YES
URL	/SpagoBIGeoReportEngine/GeoReportEngineStartAction		
Driver	it.eng.spagobi.engines.drivers.generic.GenericDriver		
SpagoBIKpiEngine			
Doc. type	KPI	Engine type	Internal
Data set	NO	Data source	NO
Driver	it.eng.spagobi.engines.kpi.SpagoBIKpiInternalEngine		
SpagoBICompositeDocEngine			
Doc. type	Cockpit	Engine type	Internal
Data set	NO	Data source	NO
Driver	it.eng.spagobi.engines.documentcomposition.SpagoBIDocumentCompositionInternalEngine		
SpagoBIWekaEngine			
Doc. type	Data Mining	Engine type	External
Data set	NO	Data source	YES
URL	/SpagoBIWekaEngine/WekaServlet		
Driver	it.eng.spagobi.engines.drivers.weka.WekaDriver		
SpagoBITableMobileEngine			
Doc. type	Mobile Report	Engine type	External
Data set	YES	Data source	NO
URL	/SpagoBIMobileEngine/servlet/AdapterHTTP?ACTION_NAME=MOBILE_ENGINE_START_ACTION		

Driver	it.eng.spagobi.engines.drivers.mobile.report.MobileReportDriver		
SpagoBIChartMobileEngine			
Doc. type	Mobile Chart	Engine type	External
Data set	YES	Data source	NO
URL	/SpagoBIMobileEngine/servlet/AdapterHTTP?ACTION_NAME=MOBILE_ENGINE_START_ACTION		
Driver	it.eng.spagobi.engines.drivers.mobile.chart.MobileChartDriver		
SpagoBICockpitMobileEngine			
Doc. type	Mobile Cockpit	Engine type	External
Data set	NO	Data source	NO
URL	/SpagoBIMobileEngine/servlet/AdapterHTTP?ACTION_NAME=MOBILE_ENGINE_START_ACTION		
Driver	it.eng.spagobi.engines.drivers.mobile.cockpit.MobileCockpitDriver		
SpagoBIDossierEngine			
Doc. type	Collaboration	Engine type	Internal
Data set	NO	Data source	NO
Driver	it.eng.spagobi.engines.dossier.SpagoBIDossierInternalEngine		
SpagoBIOfficeEngine			
Doc. type	Office document	Engine type	Internal
Data set	NO	Data source	NO
Driver	it.eng.spagobi.engines.officedocument.SpagoBIOfficeDocumentInternalEngine		
SpagoBITalendEngine			
Doc. type	ETL	Engine type	External
Data set	NO	Data source	YES
URL	/SpagoBITalendEngine/JobRunService		
Driver	it.eng.spagobi.engines.drivers.talend.TalendDriver		
SpagoBIProcessEngine			
Doc. type	External process	Engine type	External
Data set	NO	Data source	NO
URL	/SpagoBICommonJEngine/servlet/AdapterHTTP		
Driver	it.eng.spagobi.engines.drivers.commonj.CommonjDriver		

TABLE 5.3 – Configuration settings for all SpagoBI engines

Map catalogue

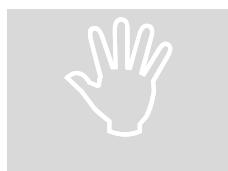
The map catalogue used by the GEO engine is centrally managed by the core of SpagoBI Server. The administrator of the platform can access the catalog through the **Resources > Maps** or **Resources > Features** options.

In the **Maps** menu, the administrator can visualize the complete list of maps registered in the catalog and modify it by deleting or adding maps. To add a new map, click the **Add Map** button, at the top right of the page. A new page will open.



FIGURE 5.68 – Add a new map

Specify the unique name of the map, add a description (optional), the reference to the file that includes the encoding and type of encoding (at present, the catalog supports only maps in SVG format).



SVG map

Because the GEO document template requires to define the name of the group to which the engine will apply thematization, any SVG map must contain at least one non empty object group.

Once the configuration parameters have been entered, users can save the new map by clicking the **Save** button, located at the top right of the page. When the document is saved, the map is parsed and all its layers are indexed, so that afterwards they can be referenced in the template.



FIGURE 5.69 –Save a new map

The list of layers of the various maps registered in the catalog appears by selecting the **Features** option in the **Resources** menu. The list of layers included in a map and the complete list of the layers defined in the registered maps can be edited as well. This option is useful in order to remove the layers of one or more maps, in order not to expose them to the engine.

On the other hand, the option allowing to produce new maps that are not actually present in one or more maps is a possible future development of the map catalogue. However, at present this does not have any impact on the real use and functioning of the GEO engine.

Data sources

A data source is a connection to a database available to SpagoBI objects. The purpose of the datasource is to allow access to data defined over that data source, without the need to redefine the connection to the database. Datasources can be used directly by some SpagoBI documents, datasets and LOV.

SpagoBI manages two types of datasources:

- JDBC
- JNDI.

If you define a datasource as JDBC, connections to the database are managed directly by SpagoBI. Therefore, each report accessing the same data source will open a new connection.

If the data source is defined as JNDI, the management of the database connections is delegated to the application server on which SpagoBI is working. The application server will autonomously manage its connection pool to optimize performances. This option is generally more appropriate in case the BI project has several documents and requires efficiency.



Data source configuration

Please refer to section Installation and Configuration in this chapter, and to the project's wiki for technical details about data source configuration.

<http://wiki.spagobi.org/xwiki/bin/view/Main/>

The list of available datasources can be found under Resources/Data source menu item.

DATA SOURCE LIST	
LABEL	DESCRIPTION
Bam	Bam
FoodMart	Foodmart
SpagoBI	SpagoBI

FIGURE 5.70 – Data Source List

Using the top filtering toolbar, the administrator can perform keyword-based searches among the list of available datasources. It is possible to remove the selected datasource by clicking on the small red button icon on the right of the corresponding row, or to edit the datasource detail by clicking on the detail icon.

To add a datasource, click the icon at the top right corner. FIGURE 5.71 shows the Detail panel that will open.

Detail Data Source

LABEL	FoodMart *
DESCRIPTION	Foodmart
DIALECT	HSQL
MULTISCHEMA	<input type="radio"/> Yes <input checked="" type="radio"/> No
SCHEMA ATTRIBUT	
TYPE	<input checked="" type="radio"/> Jndi <input type="radio"/> Jdbc
JNDI NAME	java:comp/env/jdbc/foodmart
URL	
USER	
PASSWORD	
DRIVER	

FIGURE 5.71 – Detail panel of a SpagoBI data source

Using the editor you can define and configure a new data source by setting the following configuration fields:

- **Label.** The unique identifier of the datasource (mandatory)
- **Description.** A brief description of the data source
- **Dialect.** An option list to choose the SQL dialect describing the database type. Supported dialects are:
 - Oracle
 - SQL Server
 - HSQL
 - MySQL
 - PostgreSQL
 - Ingres
 - DB2 AS400.
- **Multischema.** Available options are Yes or No. If a data source is multischema, then the schema name must be defined through the user's Profile Attribute defined in the field below.

- **Schema attribute.** In case the data source is multischema, this field contains the Profile Attribute name. The value of this profile attribute determines the schema name.
- **Type.** The available options are JDBC or JNDI.
- **JNDI NAME.** This field must be filled with the JNDI resource string already defined in the application server configuration file. Obviously the JNDI name must be defined only in case of a corresponding data source type. If the data source is JDBC, you can skip to the next field.

For Jboss and Tomcat, the JNDI string has the following format: `java:comp/env/jdbc/<resource_name>`, where the `resource_name` (in case of single schema) is the resource name. If the data source is multischema, then the string is `java:comp/env/jdbc/`. Depending on the user accessing the datasource, the resource name will be replaced with the value of the profile attribute defined in the Schema attribute field.

The following fields must be set only in case of data source id of type JDBC.

- **URL.** Database URL
- **User.** Database username
- **Password.** Database password
- **Driver.** The class of the proper Driver, depending on the database Dialect. For instance, for HSQL database it is: `org.hsqldb.jdbcDriver`



User Profile Attributes

Each SpagoBI user is assigned a profile with attributes. The user profile is part of the more general behavioural model, which allows tailored visibility and permissions on SpagoBI documents and functionalities. Read more at chapter 4, SpagoBI Sever, section The behavioural model.

Once the data source has been defined, it should be tested before saving. This can be done using the toolbar on the top, which allows the following operations (from left to right):

- Test (before saving)
- Save the datasource configuration
- Save and go back to the list of datasources
- Back to the list of datasources without saving.

Data sets

A data set is a generic data provider for analytical documents. SpagoBI manages several data set types:

- SQL statement from all the registered Data Sources
- Java class
- Web Services
- XML files
- Script (Groovy, Javascript or embedded Javascript)
- Query over the metamodel
- Custom data set.

All types of dataset share some common operations, while others are specific to each of them.

Generally speaking, the definition of a dataset in SpagoBI includes the following steps:

- Choose a name and a unique label
- Choose the type of dataset and the source, depending on the dataset type

- Write the “code” defining the dataset
- Associate parameters to the dataset, if any (optional)
- Apply transformations (optional)
- Test the dataset and save it.

Some of these steps depend on the specific type of dataset, as we will see.



Cross Navigation

Please read the sections dedicated to analytical documents/engines, if you want to learn how to associate a dataset to a SpagoBI document.

Dataset creation

To create a new dataset in SpagoBI Server, go to the **Resources > Dataset** menu. The graphical editor shown in FIGURE 5.72 will open.

The dataset graphical editor is divided into two sections:

- The left side showing the list of all available datasets
- The right side showing four tabs, each corresponding to a specific type of edit operation on the dataset.

Each item of the list in the left panel shows the dataset label (i.e., the dataset unique identifier), name and type, as well as the number of documents currently using it.

To create a new dataset, click on the **Add** icon. If your dataset is similar to another existing dataset, you can click on the **Clone** icon. This will create a copy of the dataset, except the label. All fields are pre-filled with values from the existing dataset but they can be modified and saved without affecting the original dataset.

To remove an existing dataset, click on the small red icon on the corresponding row in the left panel of the editor.

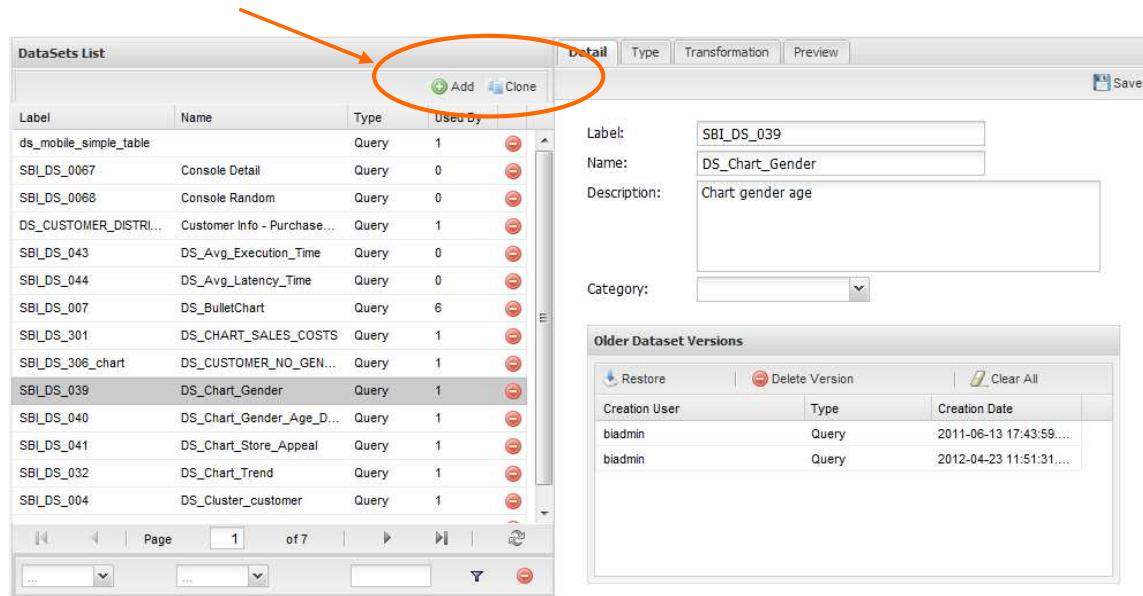


FIGURE 5.72 – Dataset editor in SpagoBI

Once created a new dataset in the left side of the editor, you should edit it on the right side. Each tab here corresponds to a step in dataset definition, namely:

- **Detail.** Here you define the name and the label of the dataset, and an optional description. In the lower part you can see a versioning system for the dataset.
- **Type.** Here you define the type of dataset, write the code for the dataset and add parameters to it, if any.
- **Transformation.** Here you can apply transformation to the dataset results, e.g., pivoting, if needed.
- **Preview.** Here you can see a preview of the dataset results. This is very useful to detect possible errors in the dataset code before associating it to a document.

Dataset name and type definition

When creating a new dataset, you must first define some information in the **Detail** tab:

- The **label** of the dataset, i.e., the unique identifier
- The **name** of the dataset, i.e., a user friendly name for dataset.

Optional fields are:

- The **description** of the dataset
- The **category** of the dataset. This field is not mandatory but it can be used to categorize datasets in your BI project. In the **Tools > Manage Domains** you can manage dataset categories: each domain with domain code set to **CATEGORY_TYPE** will be available for categorization.

Once you have set label and name (and optional fields), move to the **Type** tab to actually define the dataset.

Here you must choose the **Dataset Type**. As noticed at the beginning of the section, SpagoBI is able to manage different types of dataset: you can select the one you need in the drop down menu. Depending on the selected type, the interface changes to accept the proper configuration of the dataset.

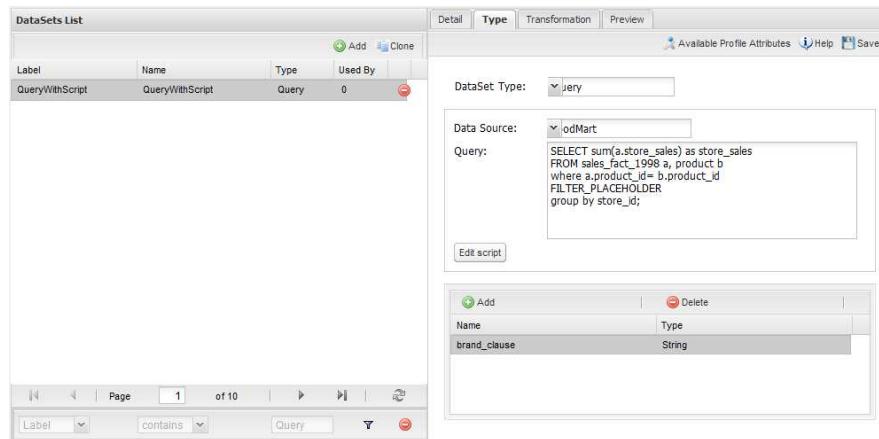


FIGURE 5.73 - Dataset of type query

Definition of the dataset code

The next step is the actual definition of the “code” defining the dataset. This will obviously change according to the chosen type. In the following we show how to define each type of dataset, based on the selected type.

Query

Selecting the **query** option requires the BI developer to write an SQL statement to retrieve data. The SQL dialect depends on the chosen data source. The SQL text must be written in the **Query** text area. For example:

```
SELECT s.customer_id as CUSTOMER,
sum(s.store_sales) as SALES,
c.yearly_income as INCOME,
p.media_type as MEDIA
FROM sales_fact_1998 s,
customer c, promotion p
WHERE s.customer_id=c.customer_id and
s.promotion_id=p.promotion_id
group by s.customer_id,
c.yearly_income,
p.media_type
```

Starting version 3.5, it is also possible to dynamically change the original text of the query at runtime. This can be done by defining a script (Groovy or Javascript) and associating it to the query. Click on the **Edit Script** button and the script editor will open.

Here you can write the script. The base query is bounded to the execution context of the script (variable **query**) together with its parameters (variable **parameters**) and all the profile attributes of the user that are executing the dataset (variable **attributes**).

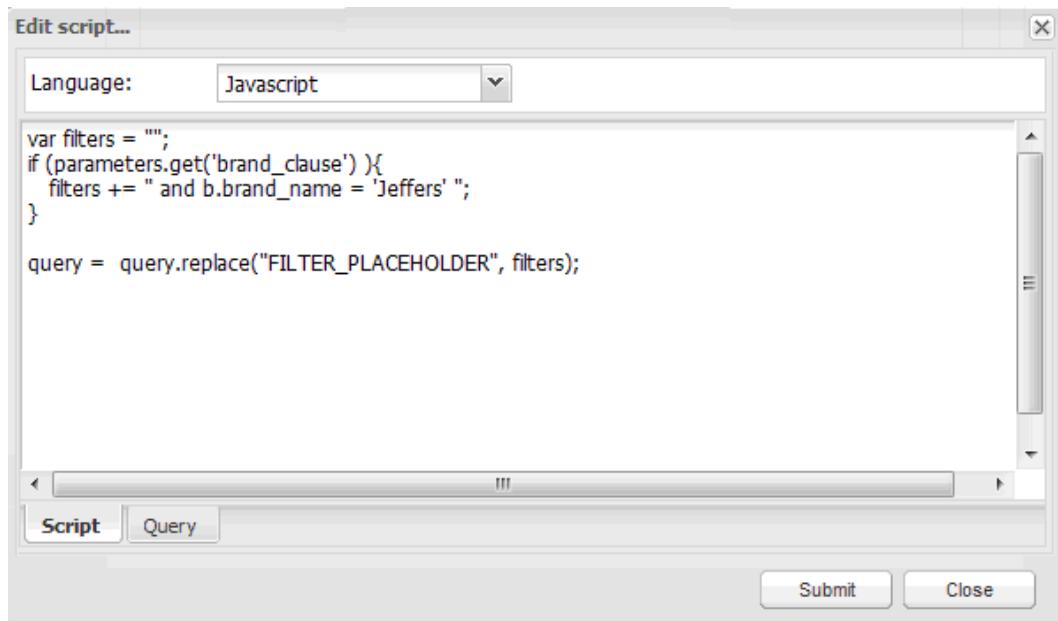


FIGURE 5.74 - Script editor of a dataset

The following example uses a Javascript to dynamically modify the FROM clause of the original query according to the value of the parameter *year* selected at runtime by the user:

```

if( parameters.get('year') == 1997 ) {
    query = query.replace("FROM sales_fact_1998",
                          "FROM sales_fact_1997");

} else {
    query = query; // do nothing
}

```

File

To define this type of dataset you first need to save the .xml file in the **resources\dataset\files** folder in your file system, then select the corresponding type in the dataset editor. The xml dataset file should look like the following example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ROWS>
    <ROW X="Supermarket" X0="0,5100" Y0="1,2360" X1="0,8900"/>
    <ROW X="Small Grocery" X0="0,6312" Y0="1,5000" X1="1,100"/>
...
</ROWS>
```

Where each row of the result set is defined by the tag `ROW`, and each column is an attribute of it.

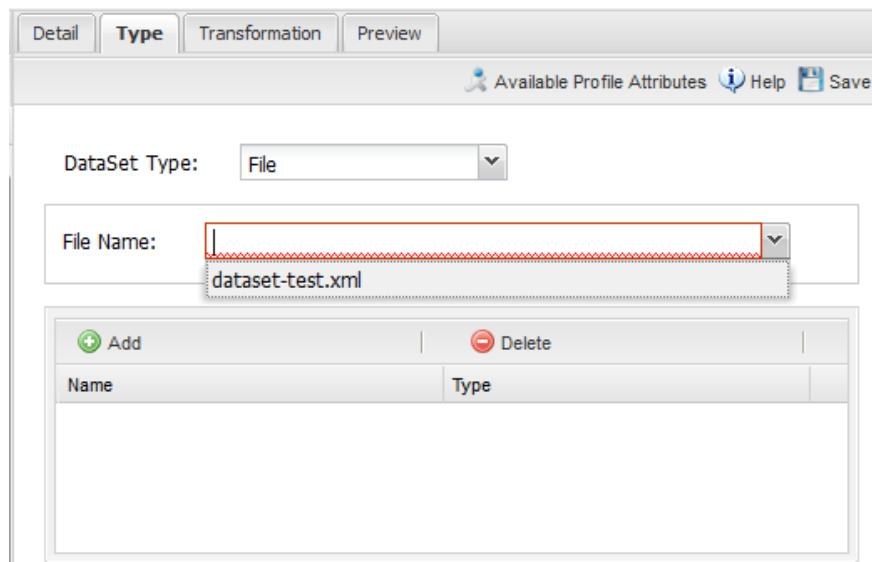


FIGURE 5.75 – File dataset creation

Script

If you select this option, the results of the dataset will be produced by a script. Therefore, the developer should write a script returning an XML string containing a list of values with the syntax shown below.

```
<ROWS>
    <ROW value="value1" .../>
    <ROW value="value2" .../>
...
</ROWS>
```

If the script returns a single value, this will be automatically encoded in the XML format above.

The script must be written in one of the following languages:

- Groovy
- Javascript.

SpagoBI already provides some Groovy and Javascript functions returning the value of a profile attribute:

- **returnValue('\${attribute_name}')**: Returns the unique value of the profile attribute with name: attribute_name

For example:

```
returnValue("${single_value_attribute}")
```

- **getListFromMultiValueProfileAttribute('\${attribute_name}')**: Returns the list of values of a multi value profile attribute.

For example:

```
getListFromMultiValueProfileAttribute("${multi_value_attribute}")
```

- **getMultiValueProfileAttribute('\${attribute_name}', 'prefix', 'newSplitter', 'suffix')**: Returns the list of values of a multi value profile attribute, preceded by a prefix, separated by the new splitter and followed by a suffix.

For example:

```
getMultiValueProfileAttribute('${multi_value_attribute}', "in (" , "," , ")" )
```

Java Class

Selecting a dataset of Java Class type allows the execution of complex data elaboration implemented by a Java class.

The compiled class must be available at \webapps\SpagoBI\WEB-INF\classes with the proper package. The class defined by the developer must implement the following interface:

it.eng.spagobi.tools.dataset.bo.IJavaClassDataSet

And the methods to implement are:

- public String **getValues**(Map profile, Map parameters);

This method provides the result set of the dataset using profile attributes and parameters. The String to return must be the XML result set representation of type:

```
<ROWS>
  <ROW value="value1" .../>
  <ROW value="value2" .../>
  ...
</ROWS>
```

- public List **getNamesOfProfileAttributeRequired**();

This method provides the names of profile attributes used by this dataset implementation class. This is a utility method, used during dataset execution.

QbE

The QbE dataset type option allows the definition of dataset results based on a query defined over a metamodel. To be available for dataset creation, the query must have already been created and uploaded on SpagoBI Server either via SpagoBI Meta or the Qbe Engine (on the Server).



Query over a metamodel

SpagoBI metamodel allows the high level representation of a data domain in terms of relevant business concepts. Queries over a metamodel can be built either with SpagoBI Meta (see chapter 3) or using SpagoBI QbeEngine (see chapter 5).

To create a Qbe dataset you should not only select the data source (as for all other dataset types), but also the Datamart from the list of available ones. Once selected a datamart, click on the small lens icon on the right (see FIGURE 5.76). The QbE query editor will open and allow you to dynamically create the query.

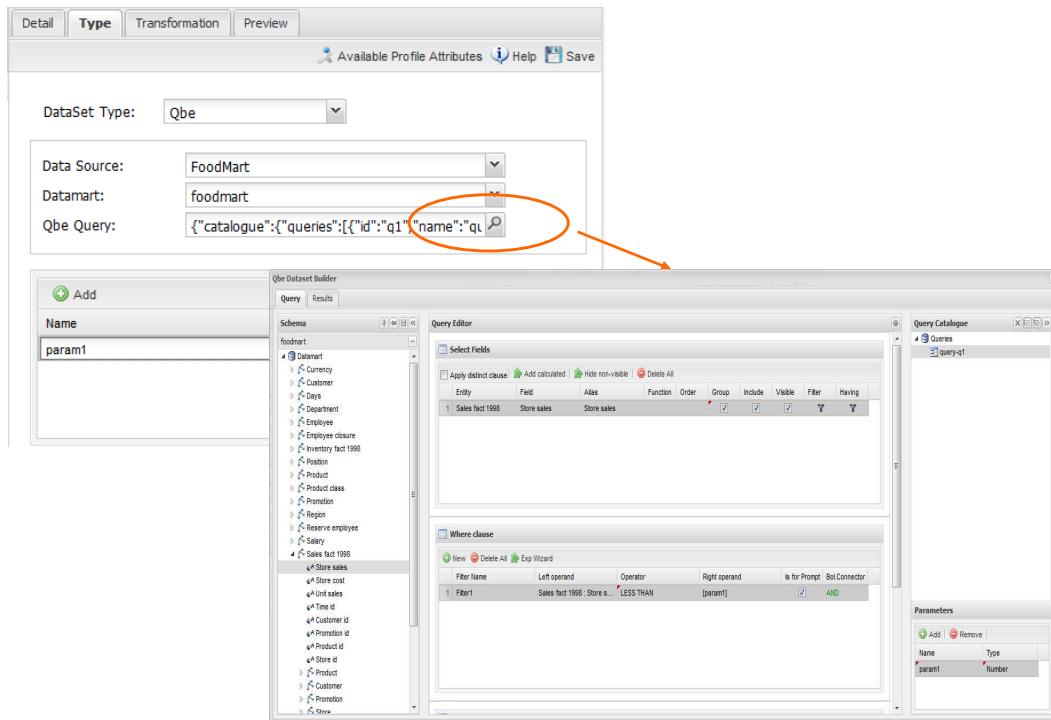


FIGURE 5.76 – QbE dataset creation

Custom

Selecting a Custom dataset type allows the developer to execute complex data elaboration by a custom Java dataset implementation.

There are two options to do this:

- either implement the **it.eng.spagobi.tools.dataset.bo.IDataSet** interface or
- extend the **it.eng.spagobi.tools.dataset.bo.AbstractCustomDataSet** class.

The methods executing the dataset that must be implemented are:

- void **loadData()**;
- void **loadData**(int offset, int fetchSize, int maxResults).

Using the `AbstractCustomDataset` class allows the developer to access predefined utility methods, such as:

- public void `setParamsMap(Map paramsMap)`;
- public `IDataSetTableDescriptor createTemporaryTable (String tableName, Connection connection)`;
- public `IDataStore decode(IDataStore datastore)`;
- private void `substituteCodeWithDescriptions(IDataStore datastore, Map<String, List<String>> codes, Map<String, List<String>> descriptions)`;
- private void `substituteCodeWithDescriptionsOnColumn(int columnIndex, IDataStore datastore, List<String> codes, List<String> descriptions, boolean append)`;
- private `Map<String, List<String>> getCodes(IDataStore datastore)`.

The full class name (package included) must be set on the Java class name field, while it is possible to add custom attributes for dataset execution and retrieve them via the following method of the `IDataSet` interface:

- `Map getProperties()`.

Add Parameters

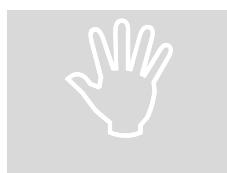
All dataset types but File allow the addition of parameters. This means that results can be customized according to the value of one or more parameters at execution time.

Parameters can be managed from within the `Type` tab. Two operations are needed to add a parameter to the dataset:

- Insert the parameter in the actual text of the dataset

- Create the parameter in the parameters list below the editor area.

The syntax to add a parameter in the dataset “code” text is `$P{parameter_name}`. At dataset execution time, the parameter will be replaced by its actual value.



Attention to parameters' names!

If the dataset is used by a SpagoBI document, then the document parameters' URL must match the parameter name set in the dataset Type tab, in order for the dataset to be passed correctly.

Any parameter added to your dataset must be added to the parameters list, too. To add a parameter in the list, click on the **Add** button. A new row will be created in the list: double click on the name and edit the parameter values.

There are three different types of parameters. For each of them the placeholder will be replaced according to a different pattern, as follows:

- **String:** the parameter value will be surrounded with single quotes if not already present.
- **Number:** the parameter value is treated as a number, with no quotes; an exception is thrown if the value passed is not a number.
- **Raw:** the parameter value is treated as a string containing a set of values; single quotes are removed from the “containing” string, not from the single strings composing it.
- **Generic:** the parameter is simply passed as it is, with no further processing.

In the following we show examples of parameterization for the different dataset types.

▪ Query

Parameters must be written in the query string. Here a simple example:

```
SELECT s.customer_id as CUSTOMER,
sum(s.store_sales) as SALES,
c.yearly_income as INCOME,
p.media_type as MEDIA
FROM sales_fact_1998 s,
customer c, promotion p
WHERE s.customer_id=c.customer_id and
s.promotion_id=p.promotion_id and
p.media_type in ($P{MediaType})
group by s.customer_id,
c.yearly_income,
p.media_type
```

The parameter `MediaType` is of type String.

▪ Script

An example of Groovy parameterized script:

```
double a = $P{ParSales};
double b = $P{ParCosts};

double toReturn = 0;
if (b != null && a != null){
toReturn = a - b;
}

returnValue(new Double(toReturn).toString());
```

The parameters `ParSales` and `ParCosts` are of type Raw.

▪ QbE

It is possible to add parameters using the **Parameters** list on the bottom right corner of the window (see FIGURE 5.76). The BI developer can use those parameters by dragging and dropping them on the right operand of a filter condition, Where clause or Having clause.

If the clause is set for prompting (i.e., the option `Is for prompt` is checked), then the parameter value is asked through a mask like the above one. Closing

the QbE dataset builder window, parameters will be automatically imported in the parameters list.

Datasets of type Query and Script can also use profile attributes. Differently from parameters, profile attributes do not need to be explicitly added to the parameter list since they have been defined elsewhere.

Clicking on the button  Available Profile Attributes you can see all profile attributes defined in the behavioral model and choose the one(s) you wish to insert in the dataset query/script text.

The syntax to include attributes into the dataset text is \${attribute_name}. Profile attributes can be single or multivalue.



User Profile Attributes

Each SpagoBI user is assigned a profile with attributes. The user profile is part of the more general behavioural model, which allows tailored visibility and permissions on SpagoBI documents and functionalities. Read more at chapter 5 - SpagoBI Sever, section The behavioural model.

Transformation

In some cases it is useful to perform transformations on the results of a dataset, to obtain data in the desired format. The most common operation is the pivot transformation, which allows the switch between rows and columns of the dataset results. SpagoBI supports this operation on any type of dataset.

To set a pivot transformation, select **Pivot Transformer** in the drop down menu of the **Transformation** tab. Then set the following fields:

- **Name of Category Column to be Pivoted.** Here you should write the name of the dataset column whose values will be mapped onto columns after pivoting.

- **Name of Value Column to be Pivoted.** Here you should write the name of the result set column, whose values should become values of the previous columns (category columns).
- **Name of the Column not to be Pivoted.** Here you should write the name of those columns that should not be altered during the transformation.
- In case you wish to add a number to category columns (e.g., `1_name_of_column`), you should check the option **Automatic Columns numeration**.

An example of usage is available in FIGURE 5.77, showing the result set of the dataset.

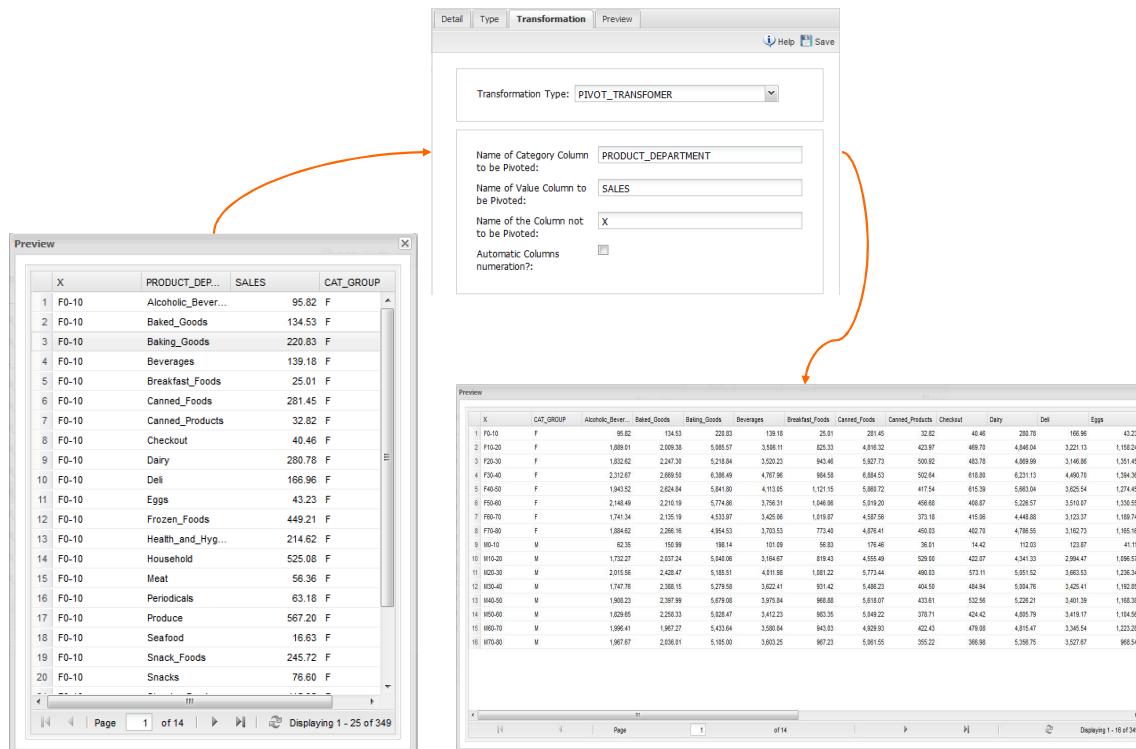


FIGURE 5.77 – Pivot transformation

Preview

Before actually using the dataset in a document, it is a good practice to test it. Clicking on the button **Preview** within the **Preview** tab, you can see a preview of the result set. This allows the developer to check any anomaly or possible error in the dataset definition, before using it.

If some parameters have been set, a window with their list will be shown: their values must be entered by double clicking on the corresponding value column. If a parameter type is set to String, just write the value you want to assign in the preview: quotes will be added automatically. On the other hand, if the type is raw or generic but you want to input text, then remember to add quotes to the test value.



FIGURE 5.78 – Parameter prompt window in the dataset preview

Manage existing datasets

In case you have already defined the dataset and you wish to restore an older version, the **Older Dataset Versions** list stores the history of all the previously saved dataset versions. The purpose of this versioning systems is to allow the BI developer to:

- Restore an older version of the dataset by selecting it from the list and clicking on the **Restore** button. This replaces the current dataset version with the selected one.
- **Delete** the selected version
- Remove all the history clicking on the **Clear All** button.

A filtering toolbar on the bottom toolbar can be used to search among the dataset list according to various keyword-based criteria. To remove filters and see all datasets, just click on the small red icon.

6. Analytical engines

In the previous chapter we learnt that the *analytical model* includes all the developed analysis of a BI project. We also learnt that SpagoBI supports several analysis domains (report, chart, cockpits, etc.) providing many *analytical engines* for each of them.

More in detail, SpagoBI Server provides:

- four engines for reporting
- three engines for multi-dimensional analysis
- three engines for charts
- one engine for interactive cockpits
- one engine for KPIs
- two engines for data mining
- two engines for free inquiry and driven data-selection
- one engine for ad hoc reporting
- two engines for location intelligence
- two engines for RT dashboards and consoles
- four engines for mobile
- one engine for office automation
- one engine for collaborative tools.

Then, we learnt that each *analytical document* concerns a single analytical domain. However, all of them are managed uniformly in SpagoBI Server.

This chapter explains the specific characteristics of each engine and the type of document it is able to manage.

Report

Reports represent the most common way to show data on structured and predefined views.

The main characteristics of a report are:

- combination of numerical data (tables, lists, cross tables), charts and images
- static and pixel-perfect layout
- multi-page and multi-format output (PDF, HTML, DOC, etc.)
- organization of data by groups and sections
- universal usage (summary, detail, analytical or operational)
- being suitable for off-line production and distribution (scheduled execution)
- ease of use.

For these reasons reports usually have a pervasive level of usage: they are used by everybody to perform both synthetic and detailed analysis, having a particularly low level of difficulty.

SpagoBI provides several report engines:

- SpagoBIBirtReportEngine, integrating the well-known report engine by Actuate
- SpagoBIJasperReportEngine, integrating the other well-known report engines by Jaspersoft

- SpagoBIAccessibleReportEngine: a specific engine allowing to produce tabular reports, which are accessible according to the international law WCAG 2.0 and the Italian law (n.4/2004, known as "Legge Stanca")
- SpagoBIBOEngine: a specific interface to be run into a SpagoBI Server, report already developed on Business Objects 6.5.



Business Objects

The integration of Business Objects reports satisfies the common need of saving the investments already done by customers, where new analytical needs or end-user targets go towards the usage of open source products. The SpagoBIBOEngine is not released as all the other engines, but it is available on the public SVN of the project because it must be compiled on the customer's environment.

SpagoBIBirtReportEngine

SpagoBIBirtReportEngine manages BIRT report templates. A report template for BIRT is a text file with the **.rptdesign** extension.

A template can be manually modified by very expert users by editing XML code. A graphical designer is provided for all developers who want to easily design a report for this engine. The graphical designer, which is integrated in SpagoBI Studio, allows the creation of BIRT templates.



Report designer for BIRT

The chapter 4 - SpagoBI Studio, Report with BIRT paragraph, provides an essential guide to develop a report for BIRT. Please refer to this section for further details.

The developer can use SpagoBI Studio deployment service to easily register the report with its template on SpagoBI Server. Alternatively, any valid BIRT

template (developed with or without SpagoBI Studio) can be directly uploaded to SpagoBI Server using the web interface for document management.



Set the connectionName parameter

Even if you are developing it without SpagoBI Studio, remember to add a parameter called `connectionName` to your report. This is required to correctly execute the report on the Server.



BIRT

All standard BIRT functionalities work with SpagoBIBirtReportEngine. For a full overview of BIRT reporting tool and a detailed developer guide, please refer to the official documentation at <http://www.eclipse.org/birt>

SpagoBIJasperReportEngine

SpagoBIJasperReportEngine manages Jasper report templates. A report template for Jasper is a text file with `.jrxml` extension.

A template can be manually modified by very expert users by editing XML code. A graphical designer is provided for all developers who want to easily design a report for this engine. The graphical designer, which is integrated in SpagoBI Studio, allows the creation of Jasper templates.



Report designer for Jasper

In chapter 4 - SpagoBI Studio an essential guide on how to develop a report for Jasper is available. Please refer to section *Designer for SpagoBIJasperReportEngine* for details.

Developers can use SpagoBI Studio deployment service to easily register the report with its template on SpagoBI Server. Alternatively, any valid Jasper

template (developed with or without SpagoBI Studio) can be directly uploaded to SpagoBI Server using the web interface for the document management.



Jasper Report

All Jasper standard functionalities work with SpagoBIJasperReportEngine. For a full overview of Jasper reporting tool and a detailed developer guide, please refer to the official documentation at
<http://jasperforge.org/projects/jasperreports> and
<http://jasperforge.org/projects/ireport>

SpagoBI Accessible Report Engine

The AccessibleReportEngine allows the creation of *accessible* reports, i.e., web reports that can be accessed by user with disabilities.

Accessible reports should be designed following specific guidelines to make them usable even in the presence of physical disabilities. For example, colors should be constrained within a certain range, in order to avoid excessive contrast. Similarly, any HTML document should be designed so that it can be browsed without using a mouse (e.g., with a keyboard). Links and images must always contain an alternative text and pages must be resizable without compromising their content.



W3C Web Accessibility Initiative

To read the full specifications of accessible web documents and further documentation please refer to
<http://www.w3.org/WAI/>

SpagoBI offers an engine for the development of accessible reports. An example is shown in FIGURE 6.1. Accessible reports can be executed within SpagoBI Server as any other analytical document, but the most frequent usage

is to embed it within a stand-alone application and execute it via SpagoBI SDK APIs.



Accessible reports

Although it is possible to execute accessible reports from SpagoBI Server, their usage is primarily intended for execution in stand-alone applications requiring the visualization of accessible data.

Percentage stores and costs for type store

Store type	X0	Y0	X1
Deluxe Supermarket	13	78	35
Gourmet Supermarket	19	76	37
Mid-Size Grocery	21	67	37
Small Grocery	16	71	36
Supermarket	21	78	37

FIGURE 6.1 - Example of SpagoBI accessible report

The steps to create a report with the Accessible Engine follow:

- Write an XSL template
- Create or modify the CSS style sheet (optional)
- Create the analytical document on the Server.

The BI developer should prepare an XSL template according to the accessibility laws in force in his country. There is currently no available designer in SpagoBI for accessible reports, so the template should be manually edited. Nevertheless, since the XSL template is rather simple and well documented, it can easily be written by hand.

An example of template is shown below:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" />

  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
        <link href="../main.css" rel="stylesheet" type="text/css"
/>
      </head>
      <body>
        <h2>Result:</h2>
        <table>
          <tr class="verde">
            <th>Store type</th>
            <th>Unit cost</th>
            <th>Store cost</th>
            <th>Cost sale</th>
          </tr>
          <xsl:for-each select="ROWS/ROW">
            <tr>
              <td><xsl:value-of select="@X"/></td>
              <td><xsl:value-of select="@X0"/></td>
              <td><xsl:value-of select="@Y0"/></td>
              <td><xsl:value-of select="@X1"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

```

When writing the XSL template, consider that data executed by the AccessibleEngine are formatted by SpagoBI according to the following XML syntax:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<ROWS>
  <ROW X="Supermarket" X0="0,5100" Y0="1,2360" X1="0,8900"/>
  <ROW X="Small Grocery" X0="0,6312" Y0="1,5000" X1="1,100"/>
...
</ROWS>

```

Once you are satisfied with the template, you can create a new style and associate it to the document by referencing the CSS file within the XSL template. This is an optional step, since the engine already has a default CSS file called `main.css` and stored under the main folder of the engine (`webapps\SpagoBIAccessibilityEngine`). If you wish to change the style, you can choose to extend the `main.css` or to create a new style sheet from scratch. In this case, you must save the new .css file under the above-mentioned location and change the reference in the XSL template.

Now you can register the document on SpagoBI Server by uploading the template and setting the options.



SpagoBI SDK

For more details on SpagoBI SDK, please refer to chapter 8 - SpagoBI SDK.

As discussed in the previous chapter, all SpagoBI analytical documents can be associated to a data source and a dataset. The Accessible Engine provides two different options to do this, based on the types of usage discussed above.

- If you are planning to execute the document from a stand-alone application via SpagoBI SDK APIs, you can associate a dataset/a data source to it using two special parameters: **query** and **connectionName**.
- If you are planning to execute the document directly on SpagoBI Server, you should associate a dataset/a data source to it using the document creation window.

If you select both data source and dataset when creating the document on the Server, the report will execute by retrieving data from the selected dataset and data source. Note that both dataset and data source should already have been defined on the Server.

If you only select the dataset when creating the document, the report will execute by retrieving data from the selected dataset. The use of the data source depends on whether the parameter **connectionName** has a value: if this is the case, its value is used as a data source. If not, the data source is the one associated to the dataset on the Server.

Finally, you can leave the dataset blank when creating the document. At execution time, you will need to store a string describing the SQL query in the **query** parameter. Note that this parameter only accepts SQL syntax. The data source will be selected with the same rule as above.

Setting the dataset and the data source using parameters allows to associate them dynamically to the document. It is the natural option for an accessible table that is accessed from a stand-alone application.



Associate parameters to analytical documents

For a full understanding of parameters and how to associate them to documents, please refer to the Behavioral Model section in chapter 5 – SpagoBI Server.

Multidimensional analysis

Multidimensional analysis (OLAP) allows the hierarchical inquiry of numerical measures, over predefined dimensions. The user can monitor data on different detail levels and from different perspectives, through drill-down, drill-across, slice-and-dice, drill-through processes.

The main characteristics of an OLAP are:

- the need for a specific data structure (logical or physical)
- analysis based on dimensions, hierarchies and measures
- interactive analysis
- freedom to re-orient analysis

- different levels of data analysis, through synthetic and detailed views
- drill-down, slice and dice, drill-through.

For these reasons, OLAP usually has a high level of usage, mainly by analysts. In fact, it allows the selective navigation over data, through synthetic and detailed views, with a medium level of difficulty.

SpagoBI provides different OLAP engines:

- SpagoBIJPivotEngine, integrating the well-known JPivot/Mondrian OLAP engine
- SpagoBIJPaloEngine, integrating JPalo client over the well-known Mondrian server
- SpagoBIJPXMLAEngine: a specific engine that uses the XMLA standard for a generic access to an OLAP server (such as MS Analysis services or SAP infocubes) using JPivot client.

SpagoBIJPivotEngine

SpagoBIJPivotEngine integrates Mondrian OLAP server and JPivot cube navigation client, to provide multi-dimensional analysis. These two modules are integrated within the same web application.

Mondrian is a *Relational Online Analytical Processing* (ROLAP) tool. It provides the back-end support for both SpagoBIJPivotEngine and SpagoBIJPaloEngine. OLAP structures, such as cubes, dimensions and attributes, are mapped directly onto tables and columns of the data warehouse. This way, Mondrian builds an OLAP cube in memory that can be accessed by client applications.

JPivot is the front-end tool to interact with Mondrian server and shows the results via the typical OLAP functionalities, e.g., drill down, slicing and dicing on a multi-dimensional table. JPivot translates user's navigation actions into MDX queries on the multi-dimensional cube, and shows query results on the table he is navigating.

In addition, JPivot allows the configuration of the navigation toolbar, the definition of profiling logics and the setting of cross navigation links, as we will see in the remainder of this section.

The creation of an OLAP analytical document with SpagoBIJPivotEngine requires the following steps:

- Cube modelling
- Catalogue configuration
- Template building
- Analytical document creation.

Note that the first two steps are common to both JPivot and JPalo Engine, and they are typical of OLAP engines.



FIGURE 6.2 - Architecture of : a) JPivot and b) JPalo OLAP Engines

Creating the analysis with Mondrian OLAP Server

In this section we will go through the steps performed on Mondrian, namely:

- Cube modelling
- Catalogue configuration.

These are characteristic of the OLAP engine and they are shared by both JPivot and JPalo Engines.

Cube modelling

The very first step for a multi-dimensional analysis is to identify essential information describing the process/event under analysis and to consider how it is stored and organized in the database. On the basis of these two elements, a mapping process should be performed to create the multi-dimensional model.



From the relational to the multi-dimensional model

The logical structure of the database has an impact on the mapping approach to be adopted when creating the multi-dimensional model, as well as on query performances.

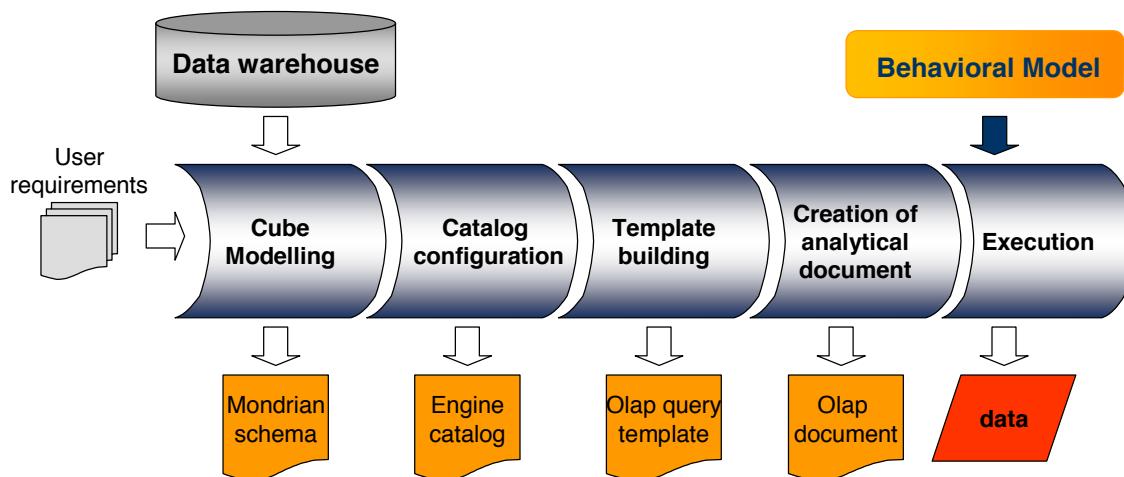


FIGURE 6.3 - Creation of an OLAP analytical document

If the structure of the relational schema complies with multi-dimensional logics, it will be easier to map the entities of the physical model onto the metadata used in Mondrian schemas. Otherwise, if the structure is highly normalized and scarcely dimensional, the mapping process will probably require to force and approximate the model to obtain a multi-dimensional model.

As said above, Mondrian is a ROLAP tool. As such, it maps OLAP structures, such as cubes, dimensions and attributes directly on tables and columns of a relational database via XML-based files, called Mondrian *schemas*. Mondrian schemas are treated by SpagoBI as resources and organized into catalogues.

Hereafter, an example of Mondrian schema.

```
<?xml version="1.0"?>
<Schema name="FoodMart">

    <!-- Shared dimensions -->
    <Dimension name="Customers">
        <Hierarchy hasAll="true" allMemberName="All Customers"
primaryKey="customer_id">
            <Table name="customer"/>
            <Level name="Country" column="country"
uniqueMembers="true"/>
            <Level name="State Province" column="state_province"
uniqueMembers="true"/>
            <Level name="City" column="city"
uniqueMembers="false"/>
        </Hierarchy>
        ...
    </Dimension>
    ...
    <Cube name="Sales">
        <Table name="sales_fact_1998"/>
        <DimensionUsage name="Customers" source="Customers"
foreignKey="customer_id"/>
        ...
    <!-- Private dimensions -->
    <Dimension name="Promotion Media" foreignKey="promotion_id">
        <Hierarchy hasAll="true" allMemberName="All Media"
primaryKey="promotion_id">
            <Table name="promotion"/>
            <Level name="Media Type" column="media_type"
uniqueMembers="true"/>
        </Hierarchy>
    </Dimension>
    ...

```

```

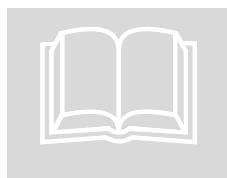
<!-- basic measures-->
<Measure name="Unit Sales"    column="unit_sales"
aggregator="sum"      formatString="#,###.00"/>
<Measure name="Store Cost"   column="store_cost"
aggregator="sum"      formatString="#,###.00"/>
<Measure name="Store Sales"  column="store_sales"
aggregator="sum"      formatString="#,###.00"/>

<!-- derived measures-->
<CalculatedMember  name="Profit" dimension="Measures">
  <Formula>
    [Measures].[Store Sales] - [Measures].[Store Cost]
  </Formula>
<CalculatedMemberProperty name="format_string"
value="$#,##0.00"/>
  </CalculatedMember>
</Cube>
  ...
</Schema>

```

Each mapping file contains one schema only, as well as multiple dimensions and cubes. Cubes include multiple dimensions and measures. Dimensions include multiple hierarchies and levels. Measures can be either primitive, i.e., bound to single columns of the fact table, or calculated, i.e., derived from calculation formulas that are defined in the schema.

The schema also contains links between the elements of the OLAP model and the entities of the physical model: for example, `<table>` sets a link between a cube and its dimensions, while the attributes `primaryKey` and `foreignKey` reference integrity constraints of the star schema.



Mondrian

For a detailed explanation of Mondrian schemas, please refer to the documentation available at the official project webpage: <http://mondrian.pentaho.com/>

Engine catalogue configuration

To reference an OLAP cube, first insert the corresponding Mondrian schema into the catalogue of schemas managed by the engine. When creating a new

template, you can choose among the available cubes using their registered schemas.

- Put a copy of the .XML schema file into the folder /resources/olap of SpagoBI Server.
- Open the configuration file engine-config.xml, located at <your_root_installation>\webapps\SpagoBIJPivotEngine\WEB-INF\classes and add your schema as in the example below.

```
<Engine_configuration>
    ...
    <Schemas>
        <schema catalogUri="/Olap/schema_name.xml"
               name="schema_logical_name"
        /
        </ Schemas >
    </ Engine_configuration >
```

Where:

- **catalogURI** is the name of the .XML file containing the schema definition that you have previously copied in /resources/olap
- **name** is the actual name of the schema.

Creating and executing the analytical document with JPivot Client

Template building

Once you have created the cube, you need to build the mapping template between the cube and the analytical document. There are two options for building an OLAP mapping template in SpagoBI JPivotEngine.

- Manually edit the template. This option is recommended to very expert users only.
- Use the template building wizard provided during the development of the OLAP analytical document.

In the following discussion we will adopt the second approach. Then we will shortly illustrate the structure of the mapping template and give hints for manual editing.

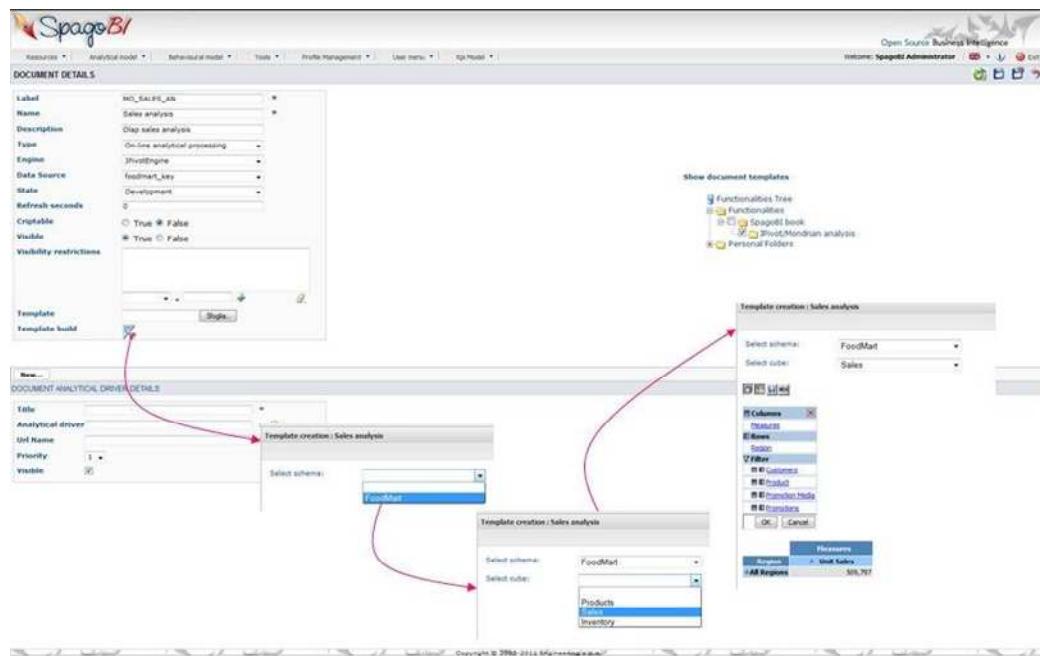


FIGURE 6.4 - JPivot template building wizard of an OLAP query

We are going to follow these steps:

- Open the wizard clicking on the small icon at the bottom
- Select the desired schema among those registered on the server
- Select a cube within the chosen schema.

Here the wizard will compose a default MDX query and draw the corresponding pivot table. Most likely you will need to modify the default visualization, so the last step is:

- Configure the default visualization until the searched result is reached.

The visualization can be modified in two ways:

- Edit the MDX query or
- Edit the pivot table using the **Cube Configurator**.

Any Change performed on a table will affect the other one and vice versa.

We will first go through the query editor. The features of the **Cube Configurator** are shown in section JPivot Client.

MDX Query Editor

When clicking on the MDX button, a window opens where you can both edit the query and add parameters.

The query is written using the Multi Dimensional eXpression (MDX) language. MDX is a specific language to operate on data structures managed by OLAP systems. MDX queries cannot be managed by traditional SQL clients: specific OLAP clients are needed to handle MDX queries and results.

Here is the simple MDX query from the example above:

```
select {[Measures].[Unit Sales]} on columns,  
       {[Region].[All Regions]} on rows  
  from [Sales]  
 where [Product].[All Products].[Drink]
```



MDX Syntax specification

To learn more about the MDX language specification and its usage, please refer to online documentation.

<http://msdn.microsoft.com/en-us/library/ms145506.aspx>

In the MDX editor window you can also associate parameters to the query. This is similar to the association of parameters to a dataset defined as a SQL query. If you insert a parameter with a given URI into the MDX query, remember to associate the same parameter URI to the OLAP analytical document on the Server. This way, the parameter can be associated to an analytical driver.



Analytical Drivers

Read how to associate analytical drivers to documents at section Behavioral Model, in chapter 5 - SpagoBI Server.

Once you are happy with the result, save the template by clicking on the corresponding icon.

Mapping template structure

The template generated by the wizard is an .XML file telling SpagoBI JPivotEngine how to navigate the OLAP cube.

As said above, very expert users can edit the template by hand. The following is an example of mapping template:

```
<?xml version="1.0" encoding="UTF-8"?>
<olap>
    /* schema configuration*/
    <cube reference="schema_name"/>
    /* query configuration*/
    <MDXquery>
        ...
    </MDXquery>
    <MDXMondrianQuery>
        ...
    </MDXMondrianQuery>
    /* toolbar configuration*/
    <TOOLBAR>
        ...
    </TOOLBAR>
    /* cross navigation configuration*/
    <CROSS_NAVIGATION>
        ...
    </CROSS_NAVIGATION>
    /* data profiling*/
    <DATA-ACCESS>
        ...
    </DATA-ACCESS>
</olap>
```

The different sections of the file follow:

- The `CUBE` section sets the Mondrian schema. It should reference the exact name of the schema, as registered in the catalogue on the Server
- The `MDXMondrianQuery` section contains the original MDX query (the one edited via the wizard)
- The `MDX` section contains a variation of the original MDX query, as used by SpagoBI Engine. This version includes parameters (if any).

Below the details of the MDX query sections. The name of the parameter will allow SpagoBI to link the analytical driver associated to the document via the parameter (on the Server).

```
<?xml version="1.0" encoding="UTF-8"?>
<olap>
/* schema configuration*/
<cube reference=" FoodMart"/>
/* query configuration*/
    <MDXquery>
        select      {[Measures].[Unit Sales]} ON COLUMNS,
                    {[Region].[All Regions]} ON ROWS
        from [Sales]
        where [Product].[All Products].[ ${prodFam} ]
            <parameter name="productFamily" as="prodFam"/>
    </MDXquery>

    <MDXMondrianQuery>
        select {[Measures].[Unit Sales]} ON COLUMNS,
                           {[Region].[All Regions]} ON ROWS
        from [Sales]
        where [Product].[All Products].[Drink]
    </MDXMondrianQuery>

</olap>
```

- The `toolbar` section is used to configure visibility options for the toolbar in the OLAP document.

Below an example of template excerpt, where the various buttons are listed. Some of them are visible (i.e., the attribute `visible` is set to true) while others are not (the attribute is set to false).

The exact meaning and functionalities of each toolbar button are explained in the next section, JPivot Client.

```
<?xml version="1.0" encoding="UTF-8"?>
<olap>
    ...
    /* toolbar configuration*/
    <TOOLBAR>
        <BUTTON_CUBE visible="true" />
        <BUTTON_MDX visible="false" />
        <BUTTON_ORDER visible="false" />
        <BUTTON_FATHER_MEMBERS visible="true" />
        <BUTTON_HIDE_SPANS visible="false" />
        <BUTTON_SHOW_PROPERTIES visible="false" />
        <BUTTON_HIDE_EMPTY visible="true" />
        <BUTTON_SHIFT_AXIS visible="true" />
        <BUTTON_DRILL_MEMBER visible="true" />
        <BUTTON_DRILL_POSITION visible="true" />
        <BUTTON_DRILL_REPLACE visible="false" />
        <BUTTON_DRILL_THROUGH visible="false" />
        <BUTTON_SHOW_CHART visible="false" />
        <BUTTON_CONFIGURE_CHART visible="false" />
        <BUTTON_CONFIGURE_PRINT visible="false" />
        <BUTTON_FLUSH_CACHE visible="false" />
        <BUTTON_SAVE visible="true" />
    </TOOLBAR>
    ...
</olap>
```

Creating the analytical document

As usual in SpagoBI, the analytical document should be created on the Server. Once you have the template ready (either produced via the wizard or another tool, or manually edited), you can create the OLAP document on the Server.

To create a new OLAP document, create a new document in the Analytical Documents area, then select **Online analytical processing > JPivot Engine**. Choose a name, a functionality, load the mapping template and save. You will see the document in the functionality (folder) you selected, displayed with the typical cube icon (see FIGURE 6.5).

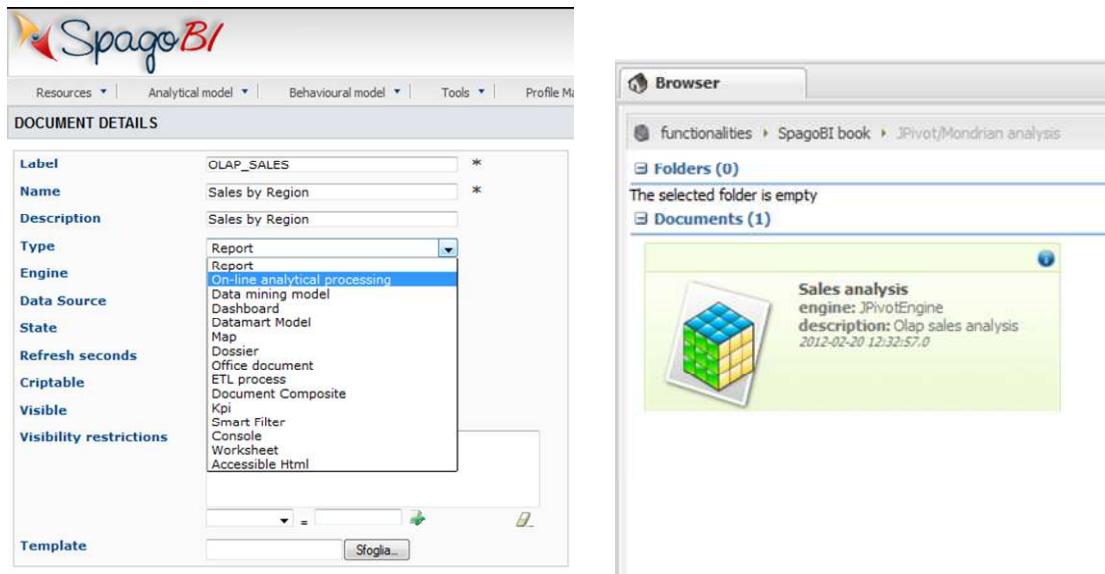


FIGURE 6.5 – Creation of an OLAP document

Executing the OLAP document

OLAP analytical documents created with SpagoBI JPivotEngine can be navigated thanks to the JPivot client. To run the JPivot client, just open the OLAP document.

JPivot provides a toolbar with all typical navigation functionalities of an OLAP navigation, such as drill (down, through and across), slicing & dicing, navigation view configuration, charts generation and export of results in various formats.

TABLE 6.1 summarizes the meaning of each item contained in the toolbar.



Toolbar customization

Since the navigation functionalities require expertise to be properly used, it may be wise to customize the toolbar according to the user's level of knowledge. This can be done by editing the mapping template (see above).

■ Cube configuration

Clicking on this icon the configuration editor will open. Here you can set and customize the layout of the pivot table, according to the following options:

Cube configuration options		
	Move to column	Move a dimension to a column of the pivot table
	Move to row	Move a dimension to a row of the pivot table
	Move to filter	Move a dimension into the filter section. The dimension is hidden from the pivot table.
	Move down	In case of multiple dimensions inside a row or a column, move down the selected dimension
	Move up	In case of multiple dimensions inside a row or a column, move up the selected dimension
	Select a member - add a slice condition	Click on a dimension in a row or column to select single items. Click on a dimension within a filter and set a slicing condition.

TABLE 6.1 Cube configuration options in the Mondrian/JPivot Engine

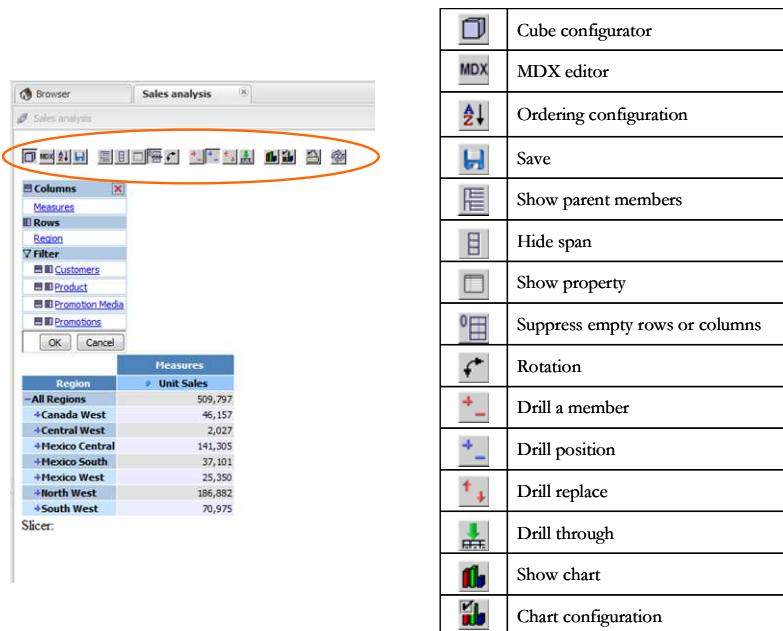


FIGURE 6.6 - JPivot configuration toolbar

- **MDX editor**

The MDX editor allows to directly modify the MDX query that implements the corresponding pivot table. Given the complexity of the MDX language, this option is recommended to very expert users only.

- **Ordering configuration**

This functionality allows the (re)ordering of data shown in the pivot table according to different criteria:

- Keep hierarchy. Ordering follows on hierarchy of shown dimensions
- Break hierarchy. Ordering does not follow hierarchy of shown dimensions
- Top and bottom count. Show only the number of items set in the ranking function.

Once set a default ordering criterion for the whole table, the specific order of columns can be changed, by using on the selectors on top of each column.

Single column ordering options		
	Natural order	Sort data using the natural order. Default behaviour.
	Ascending	Sort ascending. Switch to top count modality when bottom count has been set.
	Descending	Sort descending. Switch to bottom count modality when top count has been set.

TABLE 6.2 - Single column ordering options

- **Show parent members**

If this functionality is enabled, the pivot table shows the hierarchical levels of the dimensions that compose the pivot table as columns.

- **Hide span**

This affects the layout of the table by merging cells having the same value for dimensional elements.

- **Suppress empty rows or columns**

By selecting this button, you remove all empty cells (i.e., without a value) from the table. This will make the table easier and more pleasant to read. At the same time, the explicit visualization of empty cells highlights those dimensions that do not have values and may suggest further analysis.

- **Drill**

This is probably the main functionality of an OLAP analytical document. It consists in decomposing a bit of data by navigating the hierarchies that compose the analytical dimensions.

JPivot provides different drill modalities, namely:

- Drill member
- Drill position
- Drill replace
- Drill through

To these typical OLAP drill modalities, we add a specific functionality of SpagoBI, i.e.:

- Cross-navigation

Drill member

It navigates the document starting from a given hierarchy level, while maintaining the above levels visible. All occurrences of the selected member with respect to other dimensions in the table are exploded.

Drill position

It navigates the document starting from a given hierarchy level, while maintaining the above levels visible. Unlike the drill member, only the selected member is exploded. This is the default navigation modality.

Drill replace

It navigates the document starting from a given hierarchy level without keeping the above levels visible. Only members of the selected level are visible. This

modality is particularly useful in case of very deep navigation, as it keeps the table dimension constant at each level.

Drill through

As implemented by JPivot, it allows to navigate from a level of aggregated data to the maximum detail level available in the table. When you activate the functionality Drill through and you click on a cell, in the lower part of the window you will see a table with all details composing the result in the selected cell.

Now let us go back to the toolbar functionalities.

- **Show property**

It shows (if any) the columns defined in the OLAP as information attributes that are not drillable. If they have been defined in the OLAP model, those attributes are shown as additional columns.

- **Rotation**

It implements the pivoting of the table, i.e., switching rows and columns in the table.

- **Chart configuration**

It defines the type of chart to be shown (depending on the Show Chart option) and its properties, e.g., labels., background color, size and layout of the legend.

- **Show Chart**

It allows the visualization of the chart configured above.



JPivot

For a full overview of JPivot OLAP client and a detailed developer guide, please refer to the official documentation at <http://jpivot.sourceforge.net/>

Profiled access

As with any other analytical document, SpagoBI provides filtered access to data via its behavioural model.



The behavioral model

The behavioral model is a very important concept in SpagoBI. For a full understanding of its meaning and functionalities, please refer to chapter 5 - SpagoBI Server.

SpagoBI JPivot Engine provides two different ways to regulate data visibility based on user profiles. Data visibility can be profiled at the level of:

- the OLAP query
- the OLAP cube.

These two approaches are conceptually different. In the first case, filtering policies apply to the query: this means that the cube has all data visible and each query can be filtered according to different criteria. In the second case filtering policies directly apply to the cube: therefore, the cube itself is filtered and all queries over that cube share the same data visibility criteria.

To set the data profiling at the query level, you should edit the `data-access` section in the OLAP query template. The available access levels are:

- None. No member of this dimension will be accessible in the query.
- Custom. Access is granted to some members of the dimension only.
- All. All members of the dimension are accessible.

Below you can see an example where the visibility rule is applied to members of the dimension `[Product].[All Products].[${family}]`.

`-${family}` represents an attribute in the user profile. The rule states that each user will be allowed to see members of the `product` dimension having the `family` attribute value equal to the value of the profile attribute for that user.

For example, if the attribute has value Food for a user, he will be allowed to see only products whose category is Food. If the attribute has no value for a user, then he will be allowed to see all categories of the product.

```
<?xml version="1.0" encoding="UTF-8"?>
<olap>
    ...
/* data profiling*/
    <data-access>
        <granted-dimensions>
            <dimension name="Product" grantSource="ProfileAttributes">
                ...
                <rules access="custom" topLevel="" bottomLevel=""
rollupPolicy="PARTIAL" >
                    <members>
                        <MEMBER
name="[Product].[AllProducts].[${family}]" access="all" />
                    </members>
                </rules>
            </ dimension >
        </ granted-dimensions >
    </ data-access >
</olap>
```

The `topLevel` and `bottomLevel` attributes state the range of values that the user can see when navigating the cube (e.g., drilling down). If they are left blank, the user can access all hierarchy levels with no limitation.

The `rollupPolicy` attribute determines how Mondrian computes a member's total if the current role cannot see all children of that member. There are three possible values:

- Full. The total for that member includes all children. This is the default policy if you do not specify the attribute value.
- Partial. The total for that member includes only accessible children.
- Hidden. If any child is inaccessible, the total is hidden.

As said above, the second profiling modality takes place at the cube level. In this case you need to set the filter, which is based on an attribute in the user's profile (like the OLAP query case), within the Mondrian schema. This is the

result of the fact that data profiling is performed on the cube directly, not on the query.

Here you don't need to define a rollup policy because the filter acts directly the data retrieval logics on the Mondrian Server. So the user can only see the data that have been retrieved by the server according to the filter, with no further option.

```
<?xml version="1.0"?>
<Schema name="FoodMartProfiled">
...
    <Cube name="Sales_profiled">
        <Table name="sales_fact_1998"/>
        ...
        /*profiled dimension*/
        <Dimension name="Product" foreignKey="product_id">
            <Hierarchy hasAll="true" allMemberName="All
Products" primaryKey="product_id">
                <View alias="Product">
                    <SQL dialect="generic">
                        select
                            pc.product_family as
product_family,
                            p.product_id as product_id,
                            p.product_name as product_name,
                            p.brand_name as brand_name,
                            pc.product_subcategory as
product_subcategory,
                            pc.product_category as
product_category,
                            pc.product_department as
product_department
                        from
                            product as p, product_class as
pc
                        where
                            p.product_class_id =
pc.product_class_id
                            and pc.product_family =
'${family}'
                    </SQL>
                </View>
                <Level name="Product Family"
column="product_family" uniqueMembers="false"/>
                <Level name="Product Department"
column="product_department" uniqueMembers="false"/>
```

```

        <Level name="Product Category"
column="product_category" uniqueMembers="false"/>
            <Level name="Product Subcategory"
column="product_subcategory" uniqueMembers="false"/>
                <Level name="Brand Name" column="brand_name"
uniqueMembers="false"/>
                    <Level name="Product Name"
column="product_name" uniqueMembers="true"/>
                </Hierarchy>
            </Dimension>
        ...
</ Schema >
```

In the above example, the filter is implemented within the SQL query that defines the dimension:

```

<SQL>
...
AND PC.PRODUCT_FAMILY = '${FAMILY}'
</SQL>
```

Cross Navigation

Cross navigation is a characteristic feature of SpagoBI. The JPivot Engine supports cross navigation from OLAP documents to other analytical documents.



Cross Navigation

For further information on cross navigation and how it works, please refer to the corresponding section in chapter 5 – SpagoBI Server.

You can add a cross navigation link to the following elements of an OLAP document:

- A member of a dimension
- A cell of the pivot table

- The drill-through table. In this case you can either reach an analytical document or an external functionality.

Cross-navigation from a member of a dimension

The first two types of navigation can be set by editing the OLAP query template. In the first case you need to add a section called `clickable` inside the `MDX query` tag, as shown below:

```
<?xml version="1.0"?>
<olap>
    /* MDX query configuration*/
    <MDXquery>
        ...
        <clickable targetDocument="RPT_PROD_FAMILY"
uniqueName="[Product].[Product Family]" >
            <clickParameter name="family" value="{0}"/>
        </ clickable >
        </ MDXquery >
        ...
    </olap>
```

Where:

- The `uniqueName` attribute value is equal to the hierarchy level containing the member(s) that shall be clickable.
- The `clickParameter` element represents the parameter that will be passed to the destination document. The `name` attribute is the URI of the parameter that will be passed to the target document. The value `{0}` represents the currently selected member, as a convention: this value will be assigned to the parameter whose URI is `name`.
- The `targetDocument` attribute contains the exact name (label) of the destination document.

Once the cross navigation link is set, a small icon like the one in the toolbar will appear inside the cells of the pivot table. Clicking on a cell, the target document will be executed.

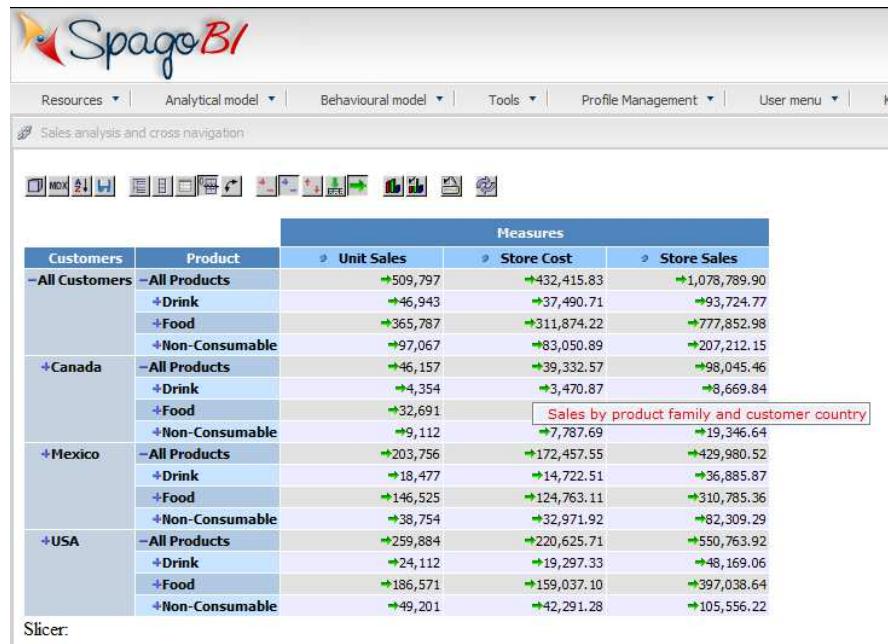


FIGURE 6.7 - Cross navigation from a member of a dimension in a JPivot document

Cross navigation from a cell of the pivot table

This case is similar to the one-dimension drill except that in this case values of all dimensions can be passed to the target document. In other words, the whole dimensional context of a cell can be passed.

To add a cross navigation link in a cell of the pivot table, add a section called `cross_navigation` to the OLAP query template, at the same level of the MDX query, as shown below. A green arrow will be visible in the JPivot toolbar to show that cross navigation is enabled.

```
<?xml version="1.0"?>
<olap>
/* cross navigation configuration*/
<cross_navigation>
    <target documentLabel="QBE_FOODMART" customizedView="Sales
by product family and country">
        <title>
            <![CDATA[ Sales by product family and customer
country ]]>
        </title>
        <description>
            <![CDATA[ Sales detail by product family and customer
country ]]>
    </cross_navigation>
```

```

</ description >
<parameters>
    <parameter name="family" scope="relative"
dimension="Product"
hierarchy="[Product]" level="[Product].[Product Family]" />
    <parameter name="country" scope="relative"
dimension="Customers" hierarchy="[Costumers]"
level="[Customers].[Country]" />
    </ parameters >
</target>
</cross_navigation> ...
</olap>
```

Where:

- The `documentLabel` attribute inside the `target` element contains the exact name (label) of the destination document.
- The `Parameters` element represents the set of parameters that will be passed to the destination document.
- The `name` attribute is the URI of the parameter that will be passed to the target document. Here it is possible to pass all visible dimensions by specifying their hierarchy level in the `dimension` attribute.



Cross navigation from a cell

Since you can transfer all visible dimensions of the OLAP document, cross-navigation starting from a cell allows you to transfer the whole execution context to the target document.

Cross navigation from the drill through table

This option allows you to add a column to the drill through table containing a link for cross navigation, like the one shown in FIGURE 6.8.

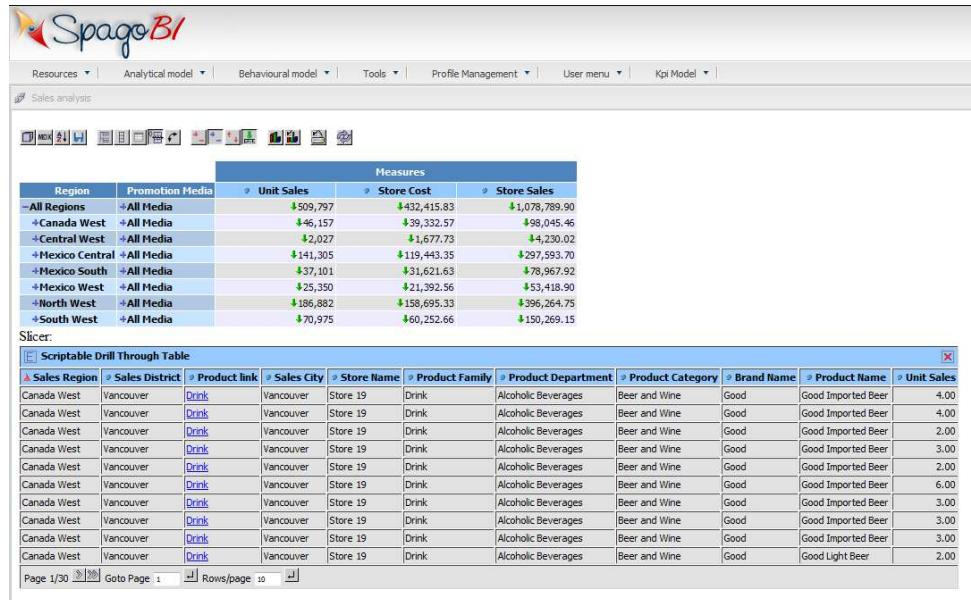


FIGURE 6.8 – Cross navigation with a drill through column in JPivot

First you need to 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 <script title="Product family link" position="3"
file="link.groovy"/>
</extension>
```

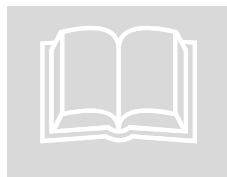
Then you need to edit the file SpagoBIJPivotEngine\WEB-INF\classes\groovy-scripts\link.groovy as follows:

```
output = [URL:"javascript:parent.execCrossNavigation(this.name",
Value:value['Family'])"]
```

SpagoBIJPaloEngine

SpagoBIJPaloEngine integrates Palo Pivot and Mondrian OLAP server in a single web application.

Both SpagoBIJPaloEngine and SpagoBIJPivotEngine rely on the Mondrian OLAP Server, but their client applications are JPalo and JPivot respectively (see FIGURE 6.2). The former has a nice user interface, developed via Google Web Toolkit, but does not support all functionalities provided by the latter. JPalo interacts via XMLA with the Mondrian server.



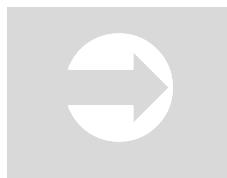
JPalo

For a full overview of the JPalo OLAP client and a detailed developer guide, please refer to the official documentation at <http://jpivot.sourceforge.net/>

Similarly to SpagoBIJPivotEngine, the creation of an OLAP analytical document with SpagoBIJPaloEngine requires the following steps:

- Cube modelling
- Catalogue configuration
- Template building
- Analytical document creation.

The first two steps are common to JPalo and JPivot engines. They consist in creating an OLAP cube and register it in the catalogue managed by Mondrian. Since we have previously explained them, we will move to the following ones.



Mondrian OLAP Server

Please refer to section SpagoBIJPivotEngine, in this chapter, to learn the functionalities of the Mondrian OLAP server, and the steps to create an OLAP model in SpagoBI.

Configuration of the JPalo client

Once you have registered the Mondrian schema, configure JPalo to access Mondrian structures.

To this end, you need to edit some configuration files, namely:

- SpagoBIJPaloEngine/WEB-INF/datasources.xml
- SpagoBIJPaloEngine/WEB-INF/classes/deploy_it.properties
- SpagoBIJPaloEngine/WEB-INF/classes/sql/mysql/credentials.

We will now show how to modify those files.

Edit SpagoBIJPaloEngine datasource file

```
<?xml version="1.0"?>
<DataSources>
    <DataSource>

        <DataSourceName>Provider=Mondrian;DataSource=MondrianFoodMart;</D
ataSourceName>
            <DataSourceDescription>Mondrian FoodMart Data
Warehouse</DataSourceDescription>
            <URL>http://localhost:8080/SpagoBIJPaloEngine/xmla</URL>

        <DataSourceInfo>Provider=mondrian;DataSource=java:comp/env/jdbc/f
oodmart;</DataSourceInfo>
            <ProviderName>Mondrian</ProviderName>
            <ProviderType>MDP</ProviderType>
            <AuthenticationMode>Unauthenticated</AuthenticationMode>
            <Catalogs>
                <Catalog name="FoodMart">

                    <Definition>file:../../resources/Olap/FoodMart.xml</Definition>
                        </Catalog>
                    </Catalogs>
                </DataSource>
            </DataSources>
```

where:

- **DataSource** is the JNDI name of the resource of type datasource used to connect to the data warehouse

- The `catalogs` tag contains the list of managed schemas/catalogues available for JPalo engine. Each name attribute (`Catalog`) represents the logical name associated to the schema.
- The `definition` tag sets the physical position of the OLAP Schema referenced by the containing catalogue.

Once these properties are set, Mondrian is able to connect to the data warehouse via the appropriate schema structures.

Edit SpagoBIJPaloEngine properties file

Now edit the SpagoBIJPaloEngine/WEB-INF/classes/deploy_it.properties. Here is an example of property file:

```
is.ssl=false
jpalo.admin.user=spagobi
jpalo.admin.password=spagobi
jpalo.mondrian.connection.url=localhost\,:8080
jpalo.mondrian.connection.service=SpagoBIJPaloEngine/xmla
jpalo.mondrian.connection.name=Mondrian-Spagobi
use.mysql=false
```

where:

- `jpalo.admin.user` is the JPalo web client username
- `jpalo.admin.password` is JPalo web client password
- `jpalo.mondrian.connection.url` is the connection URL to Mondrian server for XML communication. Do not write any protocol here, just the connection URL, such as **[MONDRIAN_HOST]\:[MONDRIAN_PORT]** (e.g., **localhost\:8080**).
- `jpalo.mondrian.connection.service` is the connection service name, whose default is **SpagoBIJPaloEngine/XMLA**. It must be changed only if the SpagoBIJPaloEngine context name changes.
- `jpalo.mondrian.connection.name` is the name of the Mondrian connection. It can be set on JPalo client interface.
- `use.mysql` must be set to true if you want to switch to MySql database to store JPalo client metadata, instead of the default HSQL

- `is.ssl` must be set to true if your protocol needs to be secure.

Edit SpagoBIJPaloEngine JDBC settings file

JPalo client stores its data on a database, which can be either HSQL or MySQL. The default DB is HSQL. For an HSQL database you need to have the following options set in the file SpagoBIJPaloEngine/WEB-INF/classes/sql/mysql/credentials.

```
jdbcURL = jdbc:hsqldb:file:[absolute_path_to_your_hsql_files]
jdbcPort = [port]
userName = database username
userPassword = database password
```

where `[absolute_path_to_your_hsql_files]` is the physical path of the HSQL files (e.g., `jdbc\hsqldb\file\..\..\database/MetaData`) and `[port]` is the port to connect (the default port is 9001).

INF/classes/sql/mysql/credentials.

On the other hand, if you wish to use MySQL DB, you will need to change the settings in the same file as follows:

```
jdbcDriver =com.mysql.jdbc.Driver
userPassword = database password
userName = database username
databaseName = schema name
jdbcURL = jdbc:mysql://[database_host]:[port]
```

where `[database_host]` is the IP address where the database is installed and `[port]` is the port to access it (the default port is 3306).

Once these properties have been set, JPalo will be able to connect via XMLA to Mondrian server and to store its runtime data in its database.

Creating and executing the analytical document with JPalo

The creation of an OLAP document is a complex task. Therefore, the task is divided into two different phases, which should be carried out by users with different levels of expertise.

First, an expert user creates a *view* on a data cube. Of course we assume that the Mondrian cube itself has been created by an expert user. The expert user will use his domain knowledge and technical competence on multi-dimensional analysis to select the dimensions and measures to be picked from the cube.

Once this is done, BI developers can create *customized views* starting from the above defined view. These customized views are called *sub-objects* in JPalo.

Now let us see the creation process step by step. To create a new OLAP document, create a new document in the Analytical Documents area, then select **Online analytical processing > JPivot Engine**. Choose a name, a functionality and save.

Each view in JPalo is based on an initial template, which states which connection and data cube are used in that document. This template is stored as a SpagoBI document template and can be created in two ways:

- Write it by hand and load it onto SpagoBI Server when creating the JPalo document. This will create an empty view on the selected cube.
- Create a document on SpagoBI Server and use the template building interface by clicking on the **Template Build** icon (see FIGURE 6.9).

The result of both choices is the creation of a base template, like the following:

```
<olap connection="Mondrian-SpagoBI" account="spagobi"
view="Inventory" cube="Inventory" ></olap>
```

where:

- **connection** is the name of the Mondrian connection (defined in `deploy_it.properties`)
- **account** is the username (defined in `deploy_it.properties`)

- view is the name of the view
- cube is the name of the chosen cube.

DOCUMENT DETAILS	
Label	jpalo-example *
Name	JPalo Analysis on Stores *
Description	JPalo Analysis on Stores
Type	On-line analytical processing
Engine	JPaloEngine
State	Released
Refresh seconds	0
Criptable	<input type="radio"/> True <input checked="" type="radio"/> False
Visible	<input checked="" type="radio"/> True <input type="radio"/> False
Visibility restrictions	(empty text area)
Template	<input type="button" value="Scegli file"/>
Template build	<input type="button" value=""/>

FIGURE 6.9 - Creation of an OLAP document with JPalo on SpagoBI Server

This is the only information that is stored in the document template. The actual content of the OLAP document (e.g., dimensions, measures, and so on) can be set via the JPalo interface only. They are stored by JPalo on its database (as configured above) and are not accessible via the document template.

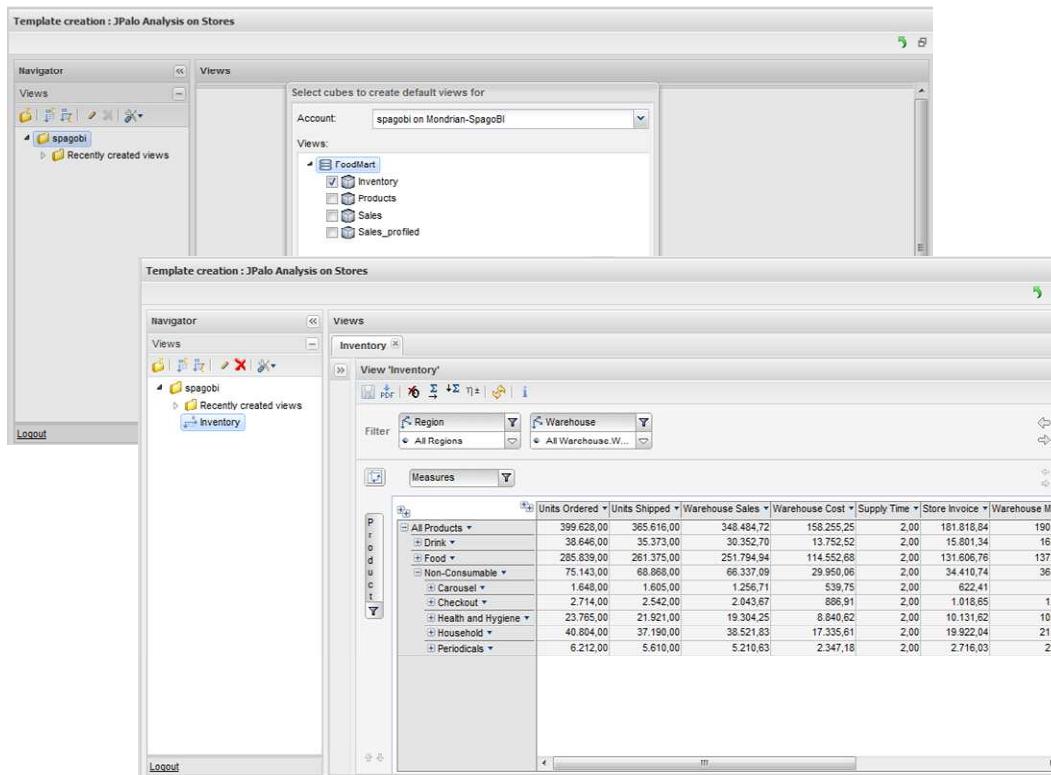


FIGURE 6.10 - Creation of a JPalo view using the graphical interface

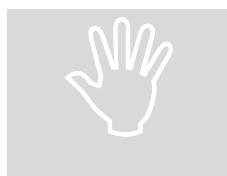
To create a view, you need to open JPalo graphical editor, after having set the base configuration (connection, cube), using the Template Build icon.

The steps to save an empty view are:

- click **Create new views** button
- choose the connection defined in deploy_it.properties for the user defined in the same file from the combo-box **Account**
- choose the cubes on which you wish to create the views (for each cube chosen a view will be created)
- click **Create** button.

In case you uploaded the base JPalo view template directly on the server, just open the empty JPalo document that you have just created and you will see an empty view.

Then select dimensions and measures from the cube by dragging and dropping them onto the appropriate axes. Save the view, come back to the Document Detail clicking on the green arrow on the top right of the screen and save the document detail. The view is ready for use.



JPalo document template

Dimensions and measures defining a view in JPalo are stored in the client's internal database. Unlike with JPivot, you can not modify the content of a JPalo view by editing the document template.

When a BI developer with the right credentials executes this document, he will see the view created by the expert user.

Now the developer can prepare his own analysis: modify dimensions and measures, filter them, explode them, invert axes and so on. At this point, the developer can save his customized view by clicking **Save SpagoBI subobject**. Then he can add further information such as business metadata, using the **Metadata** window.



Business and Structural Metadata

To learn what metadata are and how they are used in SpagoBI, please refer to section Cross Services in chapter 5 - SpagoBI Server.

Each developer can save his own customized views (called subobjects by SpagoBI) without overwriting the existing ones. Customized views are shown at the bottom of the page every time the JPalo document is executed.

Note that customized views are public by default, so there is no way to restrict their visibility: any user that is granted access to the JPalo view will also be entitled to see and modify all existing customized views.



Setting visibility on documents

To fully understand how SpagoBI manages visibility and rights on documents, please refer to chapter 5 – SpagoBI Server, section Behavioral Model, Repository Structure and Rights.

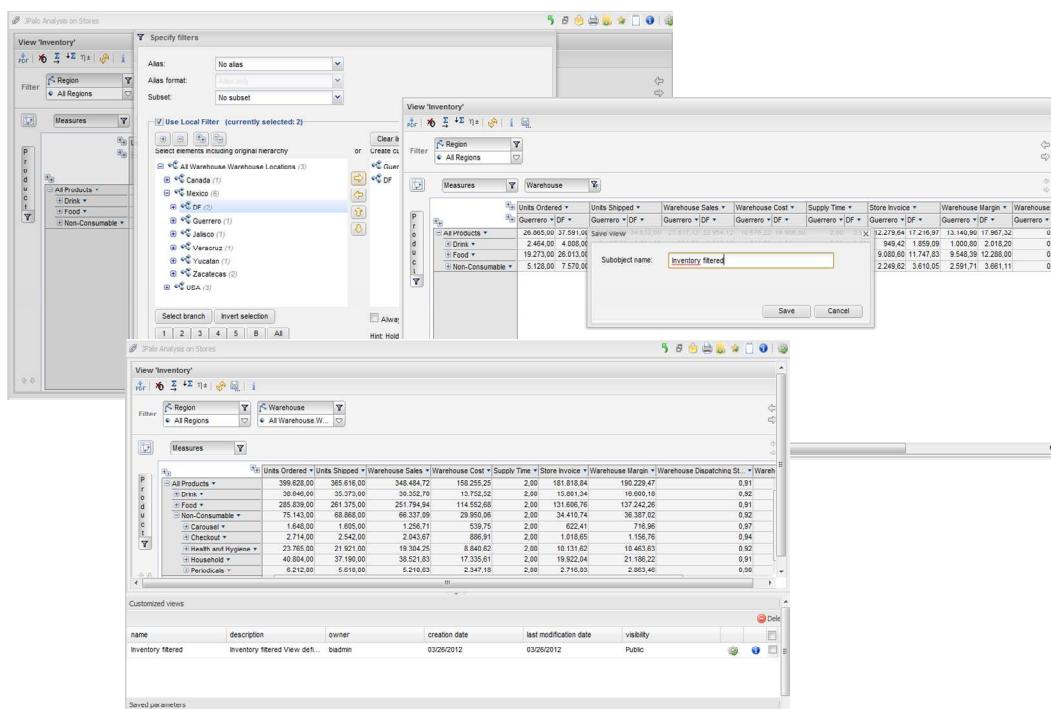


FIGURE 6.11 - Creation of a customized view in JPal

SpagoBIJPXMLAEngine

Unlike SpagoBIJPivotEngine and SpagoBIJPaloEngine, which both include a server component (i.e., the Mondrian OLAP server) and a client component (JPivot and JPalo, respectively), this engine is composed of a client component only.

The client is based on JPivot, while the OLAP server is external to the engine and can be reached via the XML Analysis (XMLA) standard. XMLA is a

SOAP-based protocol whose aim is to allow access to any standard multi-dimensional data source available on the Web.



XMLA

Read the official XMLA specifications at

<http://msdn.microsoft.com/en-us/library/ms977626.aspx>

The choice of using the JPXMLA engine rather than the Mondrian/JPivot engine typically depends on whether an OLAP server is already present and implemented in the considered system or it should be implemented from scratch. In the former case, JPXMLA is the best option because it allows the client to connect to an existing external server. On the other hand, if the OLAP needs to be implemented, it is preferable to use the Mondrian/JPivot engine that supports OLAP cube modelling.

As far as the final user is concerned, there is no substantial difference between the alternatives since the client is the same, i.e., JPivot.

In the following we will describe how to properly configure the engine. We will not discuss the design of the external OLAP server due to the huge number of available servers XMLA-compliant, each one having its own configuration procedure. We suggest to refer to the specific documentation for each server.



JPivot Client

For an overview of JPivot functionalities, please refer to chapter 6 – Analytical Engines, section SpagoBIJPivotEngine.

Engine configuration

In order to access an XMLA service on an external OLAP server, you first need to register the service endpoint in the engine configuration file at webapps\SpagoBIJPivotEngine\WEB-INF\classes.

In particular, you should add the endpoint definition to the CONNECTIONS-CONFIGURATION block, as shown below:

```
<CONNECTIONS-CONFIGURATION  
defaultConnectionName="YourServer_Name">  
  
    <CONNECTION  
        name="YourServerName"  
        type="xmla"  
        xmlaServerUrl="http://host:port/yourserver_xmlaservice"  
    />  
  
</CONNECTIONS-CONFIGURATION>
```

Where:

- The `name` attribute defines the logical name for the endpoint you want to define.
- The `xmlaServerUrl` attribute defines the URL of the endpoint used by the OLAP server to expose XMLA-based services (e.g., `http://localhost/xmla/msxisapi.dll`).

Several endpoints can be defined in the CONNECTIONS-CONFIGURATION block.

- The `defaultConnectionName` allows to specify the default endpoint for the engine.

Document creation

Before starting to create the document, it is recommended to check whether the engine is properly installed and configured.

In case the engine is not visible in the Engine Configuration list (**Resources > Engine Management**), you should check that the web application is active by invoking the following URL:

```
http://myhost:myport/SpagoBIJPXMLAEngine
```

If the application is working properly, you should see the following page:

Mondrian examples:

- [JPivot pivot table by XMLA](#)

Other links:

- [Mondrian home page](#)
- [Mondrian project page](#)
- [JPivot home page](#)
- [JPivot project page](#)

FIGURE 6.12 – SpagoBI JPXMLAEngine web application main page

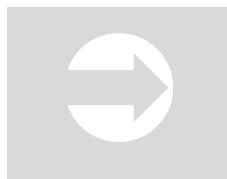
If the operation succeeds, you can go back to the Engine Management page and configure the engine.



Engine management

To check the configuration options for SpagoBI JPXMLAEngine, please refer to chapter 5 – SpagoBI Server, section Cross Services - Engine management.

The creation of an analytical document with SpagoBI JPXMLAEngine is exactly the same as with SpagoBI JPivotEngine. This is because the two engines have the same client component.



Creating and executing an OLAP document with JPivot

Please refer to chapter 5 – SpagoBI Server, section SpagoBI JPivotEngine, to learn how to create and execute an OLAP document with the JPivot client.

Chart

Charts are the most adopted method in presenting BI data, because they:

- allow an immediate perception of a phenomenon
- are easily understandable
- are focused on a visual impression more than a punctual lecture of values
- are specially suited to show trends and comparisons
- usually show limited domains
- are very easy to use
- are usually used to report official data in every kind of meeting.

For these reasons, charts have a pervasive level of usage, they are used by everybody to perform both synthetic and detailed analysis, having a particularly low level of difficulty.

SpagoBI provides many chart engines:

- SpagoBIJFreeChartEngine, integrating the well-known JFreeChart library
- SpagoBIHighchartEngine, integrating the HighChart library
- SpagoBIJSChartEngine: a specific engine using the ExtJS framework.



Highcharts

Since the Highcharts library will be discontinued starting 4.0 release, we omit the documentation, which will be out of date soon.

SpagoBIJFreeChartEngine

SpagoBIJFreeChartEngine allows the creation of several types of chart, including:

- Dial chart
- Bar chart
- Pie chart
- Cluster chart
- Box chart
- XY chart
- Scatter chart.

Each type of chart has its own design options. However, the steps to develop a chart are always the same, regardless of the type. In detail, they are:

- Define a dataset in the appropriate format for the specific chart
- Develop an XML template for the chart
- Create the analytical document on the server and execute it.



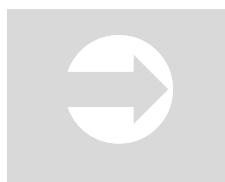
Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Model, chapter 5 – SpagoBI Server.

Chart Datasets

Each chart requires a specific type of dataset, depending on the type of columns and information required. In the next paragraphs we will provide details for each chart.

Datasets can either be queries or Business Inquiries, which are queries over a business model. In both cases the dataset must be created on the server before the chart and linked to the document at creation time.



Business Inquiry

Please refer to chapter 3 - SpagoBI Meta, to see how to define and inquire a business model.



Dataset Definition

To learn how to build a dataset of type query, please refer to chapter 5 - SpagoBI Server, section Data Sets.

Chart Template

When creating a chart directly on the Server, you must manually edit the template. The structure of all chart templates is quite regular, so the BI developer usually becomes familiar with template editing quickly, especially after the first development. For less expert users, the graphical designer for the JFreeChart Engine is an alternative solution.



Learning to design chart templates

To better understand the structure of a chart template, a viable option is to produce an initial template using SpagoBI Studio chart designer. Then proceed to make manual edits on the template and verify the results.

The chart template consists of different sections. In the following we briefly discuss their meaning, while we refer to the next sections for a detailed discussion.

Each chart template consists of the following sections (at least).

- **Category.**

The most external tag defines the type of chart and its title. An example is shown below:

```
<BARCHART type='simplebar' name="Sales and costs for month  
$P{month}">  
...  
</BARCHART>
```

Where

- the `type` attribute defines the sub-type of chart
- the `name` attribute defines the title of the chart.

In both title and subtitle you can insert parameters as well as profile attributes, according to the following syntax:

```
$P{parameter_name}  
${profile_attribute_name}
```

When the document is executed, the placeholder (as defined above) will be replaced by actual value of the parameter/profile attribute. For example, if your chart shows a historical analysis for a specific year (which is given as parameter), you can show the year in the title.



Profile attributes

To learn more on profile attributes, please refer to chapter 5 – SpagoBI Server, section The Behavioral Model.

- **Chart Style**

Some tags in the template are in charge of defining the style of the chart. Their name may vary but they usually contain the word style.

For example, the following template excerpt defines the style for the Y axis.

```
<STYLE_Y_AXIS_LABELS font='Arial' size='12' color='#0C2D83'  
orientation='horizontal' />
```

A style is characterized by the following attributes:

- `font`
- `size` in pixel
- RGB-based `color`
- vertical and/or horizontal `orientation` (optional).

▪ **Chart Configuration**

The `<conf>` tag contains a set of configuration parameters, specified in the `<name,value>` format.

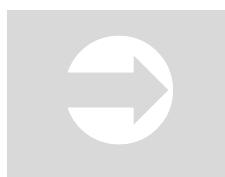
▪ **Dimension**

The `<dimension>` tag defines the width and height of the document, as shown below:

```
<DIMENSION width='250' height='250' />
```

In the remainder of the section, we will show for each type of chart:

- the structure of its template
- the required format for the dataset.



Designer for SpagoBI JFreeChartEngine

Non technical developers, who do not feel confident with template definition, may refer to the JFreeChart designer provided with SpagoBI Studio (see chapter 4).

Dial Chart

Dial charts show a value on a given scale, within an interval defined by a maximum and a minimum value and possible sub-intervals. They are useful to show performance indicators or any measure having reference values.

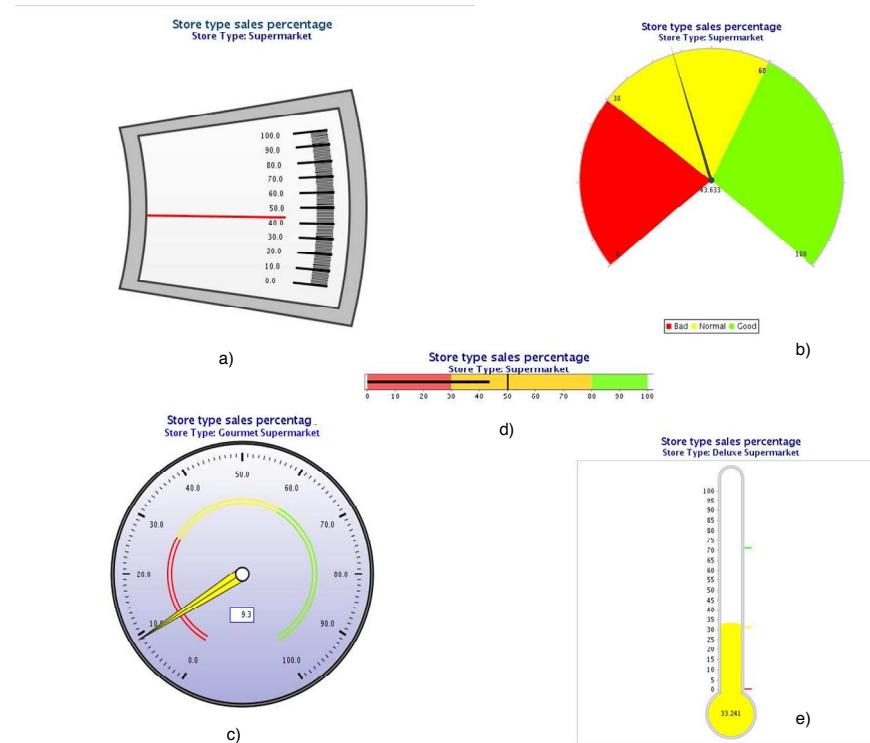


FIGURE 6.13 – Dial charts: a) Simple Dial b) Meter Dial c) Speedometer d) Bullet Dial and e) Thermometer

The JFreechart Engine offers different types of charts (as shown in FIGURE 6.13).

- Simple Dial
- Meter Dial
- Speedometer
- Bullet Dial
- Thermometer.

■ Dataset structure

The dataset for dial charts has always the following structure:

```
SELECT 30 as value from dual
```

Where the column returning the value to be drawn in the dial chart must be one and it must be called `value`.

■ Template structure

Style configuration for dial charts include: title and subtitle, as well as tick and value labels.

All dial charts include the following parameters in their Configuration template section:

- **Lower** – the minimum value of the scale
- **Upper** – the maximum value of the scale
- **Increment** – the interval between two ticks
- **Minor_tick** – the number of sub-ticks between two ticks
- **Legend** – if true, the legend will be shown.

Also, dial chart templates include a section to define the value interval. Each interval is characterized by:

- a label
- a minimum and a maximum value (for the considered interval)
- a color.

The next sections provide an overview of datasets and templates for each type of dial chart.

Simple dial

Here is an example of template.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DIALCHART type='simpaledial' name='Customer Capture'>
    <DIMENSION width='250' height='250' />
    <STYLE_TITLE font='Arial' size='16' color='#000000' />
    <STYLE_SUBTITLE font='Arial' size='12' color='#6699FF'
name='(Year 1997)' />
    <STYLE_VALUE_LABEL font='Arial' size='18' color='#00EEEE' />
    <STYLE_TICK_LABELS font='Arial' size='12' color='#FF0000' />
    <CONF>
        <PARAMETER name='orientation' value='vertical' />
        <PARAMETER name='lower' value='0' />
        <PARAMETER name='upper' value='100' />
        <PARAMETER name='increment' value='20' />
        <PARAMETER name='minor_tick' value='2' />
    </CONF>
    <INTERVALS>
        <INTERVAL min="0.0" max="20.0" color="#ff0000" />
        <INTERVAL min="20.0" max="70.0" color="#ffff00" />
        <INTERVAL min="70.0" max="100.0" color="#00ff00" />
    </INTERVALS>
</DIALCHART>
```

Let us analyze each element of the template.

```
<DIALCHART type='simpaledial' name='Customer Capture'>
```

Here we will see the type and sub-type of a chart, as already described in the commonalities section. The element \${P{month}} is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='250' height='250' />
```

As described in the common section, set the dimensions and style of the chart.

```
<STYLE_TITLE font='Arial' size='16' color='#000000' />
```

Specifically, we will set the style of the various elements such as:

- title
- subtitle. In this case you need to specify the text as follows:
name='(Year 1997)'
- value label
- tick labels

<CONF>

As specified in the common section, the configuration section includes the following parameters: lower, upper, increment, minor_tick and legend. Moreover, the simple dial has the parameter

- orientation to set the layout horizontal or vertical

<INTERVAL min="0.0" max="20.0" color="#ff0000" />

Here we define the interval: minimum, maximum and a color.

Style configuration options	
style_label_default	Default style for labels
style_x_axis_labels	Style for x axis labels
style_y_axis_labels	Style for y axis labels
style_title	Style for title
style_subtitle	Style of subtitle. This option requires that the text of subtitle is specified as value of the attribute “name”. E.g., name='Year 1998'
style_value_labels	Style for numbers representing values

TABLE 6.3 - Style configuration options for a simple dial chart

The template includes an additional configuration parameter, i.e., the orientation of the chart.

Speedometer

The template is nearly identical to the simple dial, except the type specification:

```
<DIALCHART type='speedometer' name='Customer Capture'>
```

Meter dial

The template is nearly identical to the simple dial, except the type specification:

```
<DIALCHART type='meter' name='Customer Capture'>
```

Thermometer

This chart differs from others in terms of interval definition. In the thermometer, it can only have three sub-intervals with fixed names, i.e.:

- Normal
- Warning
- Critical.

Therefore, the template is very similar to the simple dial, except the interval section.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DIALCHART type="thermomether" name="Store type sales
percentage">
<DIMENSION width='500' height='500' />
<COLORS background='#ffffff' />
<STYLE_TITLE font... />
...
<CONF>
...
</CONF>
<INTERVALS>
<INTERVAL label="normal" min="0.0" max="30.0" color="#ff0000" />
<INTERVAL label="warning" min="31.0" max="70.0" color="#ffff00"
/>
```

```
<INTERVAL label="critical" min="71.0" max="100.0" color="#00ff00">
/>
</INTERVALS>
</DIALCHART>
```

Bullet dial

The bullet dial chart allows the specification of a target value.

The template is very similar to the simple dial, except:

- The type
- The target parameter, which is defined within the configuration section.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DIALCHART type="bullet" name="Store type sales percentage">
<DIMENSION width='400' height='70' />
<COLORS background='#ffffff' />
<STYLE_TITLE ... />
<CONF>
...
<PARAMETER name="target" value="50.0" />
</CONF>
<INTERVALS>
...
</INTERVALS>
</DIALCHART>
```

Bar Charts

Bar charts are designed on two orthogonal axes. The x axis shows the *categories* of a chart, i.e., points on a given scale for which values have been calculated. For example, if a chart shows monthly values, categories are months.

Values distributed over categories belong to one or more *series*. For example, a chart showing daily store sales and costs has two different series. The y axis serves as a reference scale for actual values in a series.

All bar charts provide dynamic configuration filters for the following elements:

- Dynamic filter for categories, as a slider. Setting the appropriate configuration parameters, the slider can be shown or hidden. It is not visible unless the number of categories to show is less than the total number of categories.
- Dynamic filter for series. As shown in FIGURE 6.14, series can be shown or hidden by selecting them with a tick.
- Dynamic filter for layout: horizontal or vertical. The selection of one or of the other forces the redraw of the chart.

These options correspond to elements in the Configuration section of the chart template.

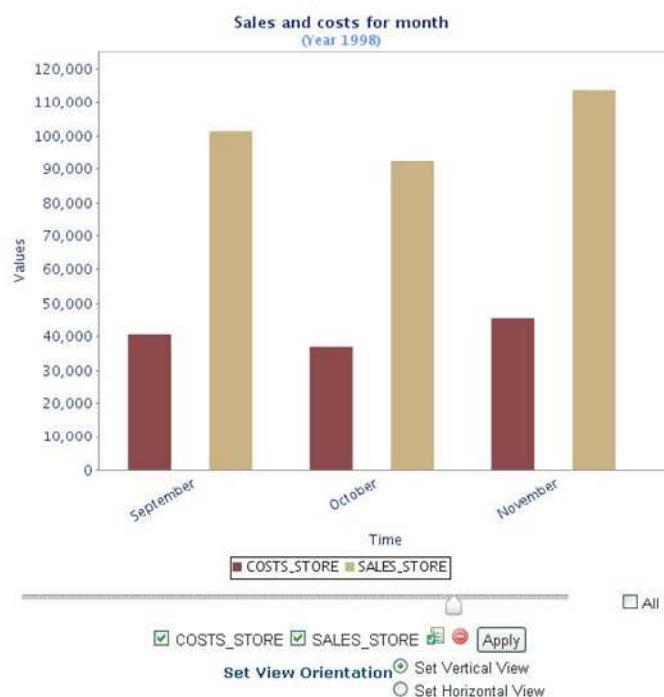


FIGURE 6.14 – Simple bar chart with months as categories and two value series

The engine offers four sub-types of bar charts, namely:

- Simple bar chart
- Linkable bar

- Stacked bar
- Overlaid bar line.

Since each of them has specific formats for datasets, we will discuss them separately afterwards.

Simple bar chart

This is the easiest type of bar chart, like the one shown in FIGURE 6.14.

■ Dataset structure

The dataset must have the following structure:

```
SELECT 'January' as x, 2000 as SALES_STORE, 1500 AS COSTS_STORE,  
500 as REVENUE FROM dual  
UNION  
SELECT 'February' as x, 2000 as SALES_STORE, 1700 AS  
COSTS_STORE, 300 as REVENUE FROM dual  
UNION  
SELECT 'February' as x, 2500 as SALES_STORE, 1700 AS  
COSTS_STORE, 800 as REVENUE FROM dual
```

Where:

- Column x represents categories (on the X axis)
- Other columns represent series
- The optional column cat_group allows the grouping of categories. If filters on group categories are active, setting this column enables group-based filtering.

■ Template

Below you can see an example of template.

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

```

<BARCHART type='simplebar' name="Sales and costs for month
$P{month}">
    <DIMENSION width='600' height='500' />
    <STYLE_LABELS_DEFAULT font='Arial' size='12'
color='#0C2D83' orientation='horizontal' />
    <STYLE_X_AXIS_LABELS font='Arial' size='12' color='#0C2D83'
orientation='horizontal' />
    <STYLE_Y_AXIS_LABELS font='Arial' size='12' color='#0C2D83'
orientation='horizontal' />
    <STYLE_TITLE font='Arial' size='14' color='#0C2D83' />
    <STYLE_SUBTITLE font='Arial' size='12' color='#6699FF'
name='(Year 1998)' />
    <STYLE_VALUE_LABELS font='Arial' size='10' color='#000000'
orientation='horizontal' />
    <COLORS background='#FFFFFF' />
    <CONF>
        <PARAMETER name='category_label' value='Time' />
        <PARAMETER name='value_label' value='Values' />
        <PARAMETER name='legend' value='true' />
        <PARAMETER name='legend_position' value='bottom' />
        <PARAMETER name='n_cat_visualization' value='3' />
        <PARAMETER name='dynamic_n_visualization' value='true' />
        <PARAMETER name='n_ser_visualization' value='1' />
        <PARAMETER name='filter_categories' value='true' />
        <PARAMETER name='filter_series' value='true' />
        <PARAMETER name='filter_series_buttons' value='true' />
        <PARAMETER name='view_slider' value='true' />
        <PARAMETER name='position_slider' value='bottom' />
        <PARAMETER name='slider_start_from_end' value='true' />
        <PARAMETER name='filter_cat_groups' value='true' />
        <PARAMETER name='maximum_bar_width' value='5.01' />
        <PARAMETER name='range_integer_values' value='false' />
        <PARAMETER name='range_axis_location'
value='BOTTOM_OR_LEFT' />
        <PARAMETER name='show_value_labels' value='true' />
        <PARAMETER name='enable_tooltips' value='true' />
        <PARAMETER name='orientation' value='horizontal' />
        <PARAMETER name='hidden_seriel' value='REVENUE' />
    </CONF>
    <SERIES_COLORS SALES_STORE='#C9B385' COSTS_STORE='#8C4A4A' />
    <DRILL />
</BARCHART>

```

Let us analyze all elements of the template.

```

<BARCHART type='simplebar' name="Sales and costs for month
$P{month}">

```

Here we see the type and sub-type of a chart, as already described in the commonalities section. The element \${P{month}} is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='600' height='500' />
<STYLE_LABELS_DEFAULT font='Arial' size='12' color='#0C2D83'
orientation='horizontal' />
```

As described in the common section, set dimensions and style for the chart.

Style options can be defined for a set of elements:

Style configuration options	
style_label_default	Default style for labels
style_x_axis_labels	Style for x axis labels
style_y_axis_labels	Style for y axis labels
style_title	Style for title
style_subtitle	Style of subtitle. This option requires that the text of subtitle is specified as value of the attribute “name”. E.g., name='Year 1998'
style_value_labels	Style for numbers representing values

TABLE 6.4 - Style configuration options for a simple bar chart

```
<COLORS background='#FFFFFF' />
```

Set the background color.

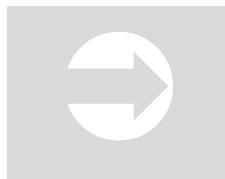
```
<CONF>
```

As we know, the tag `<CONF>` opens the section dedicated to configuration parameters. Configuration specifications for (simple) bar charts are listed below.

Filters	
filter_categories	If checked, enables to filter categories using a slider
filter_cat_groups	If checked, enables to filter category groups using a slider
filter_series	Allows the filtering of single series using a checkbox

filter_series_button	If true, the options “All” and “None” for filtering series will be shown
n_cat_visualization	If greater than the total number of categories, the slider is not shown
dynamic_n_cat_visualization	The number of visible categories can be modified on the chart by the user
n_ser_visualization	Number of series that will be shown
hidden_serie1	Hides the series referenced by the “name” attribute (in the example, “revenue”). To hide more than one series, add hidden_serie2, hidden_serie3, etc.
Legend	
legend	If true, the legend will be shown
legend_position	Possible options are: left, up, bottom, right
Slider	
view_slider	If true, the slider will be shown (provided that the number of visible categories is less than the total)
position_slider	Possible options: top, bottom
slider_start_from_end	If true, at first execution the slider is set to the end
Others	
category_label	Label for the category axis
value_label	Label for the value axis
maximum_bar_width	Sets a maximum width for bars (bar width is dynamically calculated)
range_integer_values	If true, only integer values will be shown
range_axis_location	Possible values: BOTTOM_OR_LEFT, BOTTOM_OR_RIGHT, TOP_OR_RIGHT, TOP_OR_LEFT
show_value_labels	If true, values will be shown beside bars
enable_tooltips	If true, tooltips are enabled. Tooltips must be returned from the dataset.
orientation	Possible options: vertical, horizontal

TABLE 6.5 - Configuration parameters for a simple bar chart



Designer for JFreechart

If you look at the Designer for JFreechart, at chapter 4 - SpagoBI Studio, you will notice that template tags are mapped onto configuration options of the graphical editor. This concerns all JFreechart documents.

```
<SERIES_COLORS SALES_STORE='#C9B385' COSTS_STORE='#8C4A4A' />
```

In this section we choose colors according to the RGB encoding, as already seen in the common section above. Series are referenced with the names that they have in the dataset.

Linkable bar chart

This is the linkable version of the simple bar, i.e., enabled for cross navigation. Clicking on a bar will redirect you to another SpagoBI document, passing the information related to the selected category and/or series.



Cross Navigation

Please refer to section Cross Navigation in chapter 5 – SpagoBI Server for a full explanation of this mechanism.

▪ Dataset structure

The dataset has the same structure as the simple bar (see above).

▪ Template

The template is very similar to the template of a simple bar chart, except two elements:

- The sub-type of chart
- The DRILL section, for cross navigation

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BARCHART type='linkablebar' name="Sales and costs for month
$P{month}">
    <DIMENSION ... />
    <STYLE_LABELS_DEFAULT ... />
    ...
    <COLORS background='#FFFFFF' />
    <CONF>
        ...
    </CONF>
    <SERIES_COLORS ... />

    <DRILL document="DocumentLabel">
        <PARAM name='categoryurlname' value='Month' />
        <PARAM name='seriesurlname' value='Sales' />
        <PARAM name="ParName_relative" type="RELATIVE"
value="ParameterPassed" />
        <PARAM name="ParName_absolute" type="ABSOLUTE"
value="AbsoluteValue" />
        <PARAM name="target" type="ABSOLUTE"
value="tab" />
        <PARAM name="title" type="ABSOLUTE"
value="Title from cross" />
    </DRILL>
</BARCHART>

```

Let us explain in detail the elements of the DRILL section.

```
<DRILL document="DocumentLabel">
```

The document attribute specifies the name (i.e., label of the target document in SpagoBI).

The parameters are summarized below:

Drill parameters	
categoryurlname	Name (URL) of the parameter corresponding to the category linked in cross navigation. Default is “category201D”
seriesurlname	Name (URL) of the parameter corresponding to the series linked in cross navigation. Default is “serie”
ParName_relative	Name (URL) of the parameter to be propagated in cross navigation. Type and value are fixed as shown in the template. If a parameter is relative, the value

	that will be passed to the target document is taken from the analytical driver associated to that parameter in the document.	
ParName_absolute	Name (URL) of the parameter to be propagated in cross navigation. Type is fixed as shown in the template. If a parameter is absolute, the value that will be passed to the target document is fixed and given by the “value” attribute.	
title (optional)	Title to be given to the destination document. Default is the name of the document	
target (optional)	value=“self”	The target document is opened within the same window and breadcrumbs are updated
	value=“tab”	If executing in a browser, the target document is opened in a new tab. Otherwise, same as Self.
	value=“update”	Useful when source and target documents are the same, to update the document with new values

TABLE 6.6 - Drill parameters for charts

Stacked bar chart and Cumulate Chart

This type of chart shows bars as stacked instead of side by side.

By setting configuration options properly, it is possible to define a cumulative series that will produce a chart known as Cumulative Chart (shown in FIGURE 6.15 b).

This chart type can be made linkable, by adding the drill section in the template.

▪ Dataset structure

```
SELECT 'January' as x, 2000 as SALES_STORE, 1500 AS COSTS_STORE,
500 as REVENUE FROM dual
UNION
SELECT 'February' as x, 2000 as SALES_STORE, 1700 AS
COSTS_STORE, 300 as REVENUE FROM dual
UNION
```

```
SELECT 'February' as x, 2500 as SALES_STORE, 1700 AS COSTS_STORE, 800 as REVENUE FROM dual
```

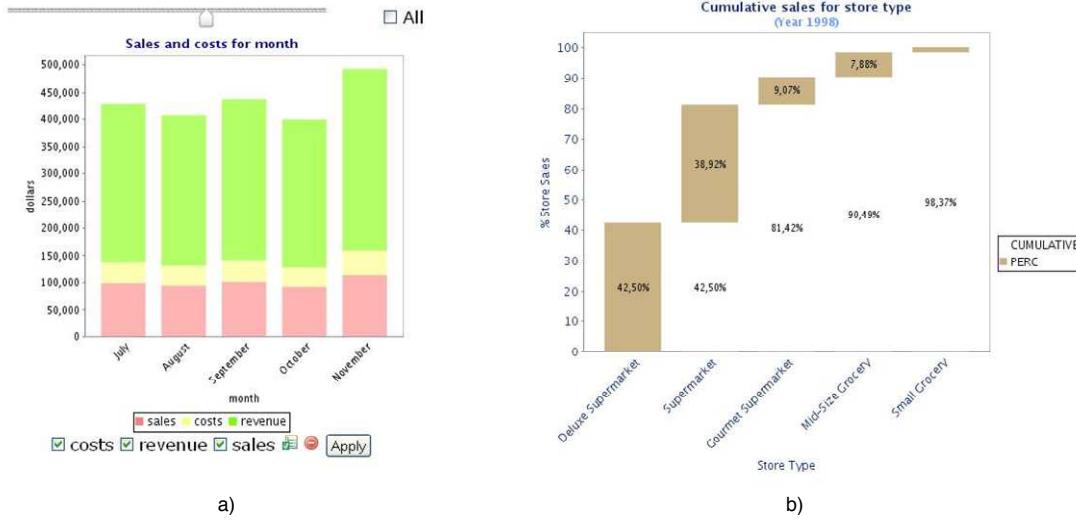


FIGURE 6.15 – Stacked bar charts: a) basic and b) with cumulate and percentage options

■ Template structure

The template is similar to the one of the linkable bar, with some relevant differences.

- The type of chart
- Some additional options for series colors
- Some additional configuration parameters.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BARCHART type="stacked_bar" name="Sales and costs for month">
<DIMENSION ... />
<COLORS background='#ffffff' />
<STYLE_Y_AXIS_LABELS font="ARIAL" size="10" color="#000000" orientation="HORIZONTAL" />
...
<CONF>
<PARAMETER name="range_axis_location" value="BOTTOM_OR_LEFT" />
<PARAMETER name="category_label" value="month" />
<PARAMETER name="legend_position" value="bottom" />
<PARAMETER name="value_label" value="dollars" />
```

```

<PARAMETER name="enable_tooltips" value="true" />
<PARAMETER name="n_ser_visualization" value="5" />
<PARAMETER name="n_cat_visualization" value="5" />
<PARAMETER name="filter_cat_groups" value="true" />
<PARAMETER name="show_value_labels" value="true" />
<PARAMETER name="filter_series_buttons" value="true" />
<PARAMETER name="filter_categories" value="true" />
<PARAMETER name="legend" value="true" />
<PARAMETER name="filter_series" value="true" />
<PARAMETER name="range_integer_values" value="true" />
<PARAMETER name="slider_start_from_end" value="true" />
<PARAMETER name="dynamic_n_visualization" value="false" />
<PARAMETER name="view_slider" value="true" />
<PARAMETER name="position_slider" value="top" />
<PARAMETER name="maximum_bar_width" value="5.01" />
<PARAMETER name="orientation" value="vertical" />
<PARAMETER name="add_labels" value="true" />
<PARAMETER name="value_labels_position" value="inside" />
<PARAMETER name="cumulative" value="false" />
<PARAMETER name="make_percentage" value="false" />
<PARAMETER name="percentage_value" value="false" />
</CONF>

<SERIES_COLORS REVENUE="#80ff00" COSTS_STORE="#ffff80"
SALES_STORE="#ff8080" />
<SERIES_LABELS REVENUE="revenue" COSTS_STORE="costs"
SALES_STORE="sales" />
<SERIES_ORDER_COLORS a_0="#ffffff" a_1="#008040" a_2="#0000ff"
/>

<DRILL ... />
</DRILL>
</BARCHART>

```

Additional configuration parameters are summarized below:

Stacked bar configuration options	
add_labels	Allows to add labels that must be returned by the dataset with the following name: ADD_<series_name>
cumulative	Adds a cumulative series
make_percentage	If selected, values are calculated in percentage
percentage_value	Shows values in percentage

Color configuration options	
series_labels	Defines a name for each series. Series are referenced by their name in the dataset and can be associated to a different label.
series_order_colors	Alternative to the series_colors tag. In case the names of the series are not known, it is possible to define an ordered set of colors that will be associated to series. Series are referenced as A_0, A_1, etc.

TABLE 6.7 - Additional configuration parameters for stacked bar charts

Overlaid bar line

This bar chart type can represent series as either bars or lines. To provide appropriate reference axis for both types of value representation, you can add a secondary Y axis and choose to which one each value series shall refer.

For each series you must specify the type of representation (bar o line) and the reference axis.

- **Dataset structure**

```
SELECT 'January' as x, 2000 as SALES_STORE, 1500 AS COSTS_STORE,
500 as REVENUE FROM dual
UNION
SELECT 'February' as x, 2000 as SALES_STORE, 1700 AS
COSTS_STORE, 300 as REVENUE FROM dual
UNION
SELECT 'February' as x, 2500 as SALES_STORE, 1700 AS
COSTS_STORE, 800 as REVENUE FROM dual
```

- **Template structure**

The template is similar to the one of the linkable bar, with some relevant differences.

- The type of chart
- Some additional configuration parameters.

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

```

<BARCHART type="overlaid_barline" name="Sales and costs for month">
<DIMENSION ... />
<COLORS background="#ffffff" />
<STYLE_... />
<CONF>
<PARAMETER name="range_axis_location" value="BOTTOM_OR_LEFT" />
<PARAMETER name="category_label" value="month" />
<PARAMETER name="legend_position" value="bottom" />
<PARAMETER name="value_label" value="dollars" />
<PARAMETER name="enable_tooltips" value="true" />
<PARAMETER name="n_ser_visualization" value="5" />
<PARAMETER name="n_cat_visualization" value="5" />
<PARAMETER name="filter_cat_groups" value="true" />
<PARAMETER name="show_value_labels" value="true" />
<PARAMETER name="filter_series_buttons" value="true" />
<PARAMETER name="filter_categories" value="true" />
<PARAMETER name="legend" value="true" />
<PARAMETER name="filter_series" value="true" />
<PARAMETER name="range_integer_values" value="true" />
<PARAMETER name="slider_start_from_end" value="true" />
<PARAMETER name="dynamic_n_visualization" value="false" />
<PARAMETER name="view_slider" value="true" />
<PARAMETER name="position_slider" value="top" />
<PARAMETER name="maximum_bar_width" value="5.01" />
<PARAMETER name="orientation" value="vertical" />
<PARAMETER name="add_labels" value="true" />
<PARAMETER name="value_labels_position" value="inside" />
</CONF>
<SERIES_COLORS REVENUE="#80ff00" COSTS_STORE="#ffff80"
SALES_STORE="#ff8080" />
<SERIES_LABELS REVENUE="revenue" COSTS_STORE="costs"
SALES_STORE="sales" />
<SERIES_ORDER_COLORS a_0="#ffffff" a_1="#008040" a_2="#0000ff" />
<SERIES_DRAW REVENUE="line" COSTS_STORE="line_no_shapes"
SALES_STORE="bar" />
<SERIES_SCALES REVENUE="1" COSTS_STORE="1" SALES_STORE="2" />
</BARCHART>

```

Additional configuration parameters for the overlaid bar chart are:

Overlaid bar line configuration options

second_axis_label	If this parameter has a value, a secondary axis with a label equal to the value will be created
add_label	Allows to add labels that must be returned by the dataset with the following name:

	ADD_<series_name>
stacked_bar_renderer_1	If true, bar series referring to the first axis will be shown as stacked
stacked_bar_renderer_2	If true, bar series referring to the second axis will be shown as stacked
value_label_position	Set value labels inside or outside bars. Values are “INSIDE” or “OUTSIDE”.

TABLE 6.8 - Configuration options for overlaid bar line chart

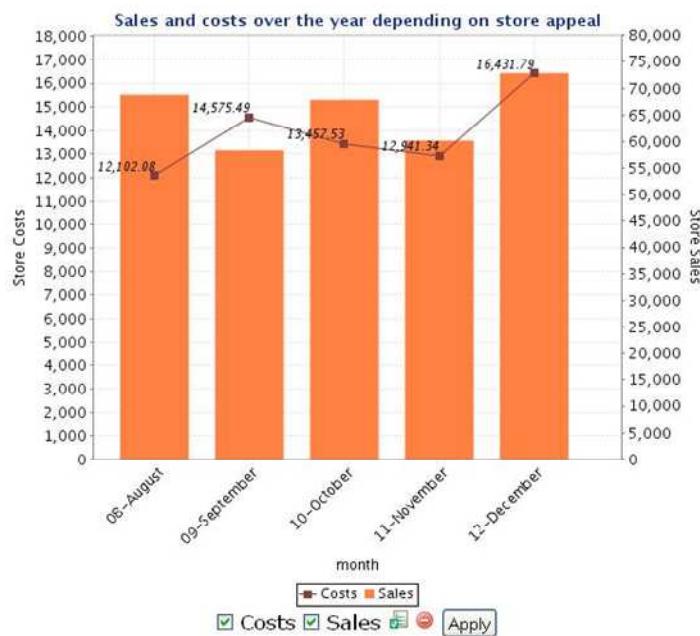


FIGURE 6.16 – Overlaid bar line chart with double Y axis

Pie Chart

Pie charts show values associated to a set of categories, according to the total value, represented by the whole pie.

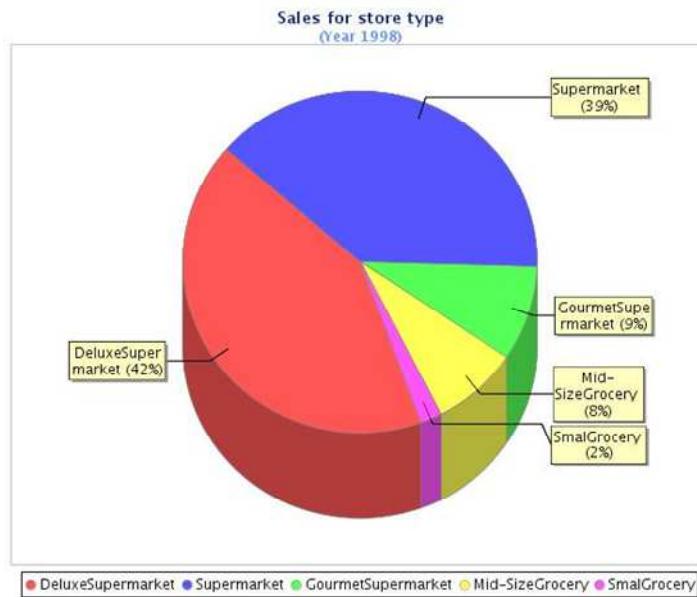


FIGURE 6.17 – Simple pie chart in 3D

▪ Dataset structure

The dataset must have the following structure:

```
select 30 as X, 20 as Y, 50 as Z from dual
```

where each column represents a category (i.e., a slice of pie).

There are two possible types of pie charts:

- Simple pie chart
- Linkable pie chart.

Simple pie chart

This is the basic pie. An example of template is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PIECHART type="simplepie" name="Sales for store type">
<DIMENSION width='600' height='500' />
<COLORS background='#ffffff' />
```

```
<STYLE_TITLE font="Arial" size="14" color="#0c2d83"
orientation="" />
<STYLE_SUBTITLE font="Arial" size="12" color="#6699ff"
orientation="" name="(Year 1998)" />
<STYLE_LABELS_DEFAULT font="Arial" size="12" color="#6699ff"
orientation="" />
<CONF>
<PARAMETER name="values" value="percentage" />
<PARAMETER name="dimensions" value="3d" />
</CONF>
</PIECHART>
```

Let us analyze all elements of the template:

```
<PIECHART type="simplepie" name="Sales for store type">
```

Here we see the type and sub-type of chart, as already described in the commonalities section. The element \$P{month} is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='600' height='500' />
<STYLE_TITLE font="Arial" size="14" color="#0c2d83"
orientation="" />
```

As described in the common section, set dimensions and style for the chart. Style can be set for title, subtitle and labels.

```
<CONF>
<PARAMETER name="values" value="percentage" />
<PARAMETER name="dimensions" value="3d" />
</CONF>
```

Here we set two configuration parameters:

Pie configuration options	
values	Possible values are “NORMAL” or “PERCENTAGE”. Normal means actual values.
dimension	Possible values are “2D” or “3D”

TABLE 6.9 - Pie configuration options

Linkable Pie

This is the linkable version of the simple pie, i.e., enabled for cross navigation.

The template is the same as the one of the simple pie, except the drill section:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
...
<CONF>
...
</CONF>
<DRILL document="Detail_report_for_store_type" >
    <PARAM name="categoryurlname" value="StoreType"/>
        <PARAM name="title" type="ABSOLUTE" value="Selected
type from cross" />
        <PARAM name="month" type="RELATIVE"
value="monthDriver" />
        <PARAM name="target" type="ABSOLUTE" value="tab" />
</DRILL>
</PIECHART>
```

The drill section:

```
<DRILL document="DocumentLabel">
```

The document attribute specifies the name (i.e., label of the target document in SpagoBI).

Drill parameters are the same as those of linkable bar charts and are summarized TABLE 6.6.

Cluster Chart

Cluster charts represent clusters, i.e., sets of discrete values that can be grouped together according to some criteria. Cluster charts have two axes: one for categories (X) and another for values (Y). Values can be divided into series. For

example, you can define clusters of customers based on the number of unit sales over time, for a specific type of good.

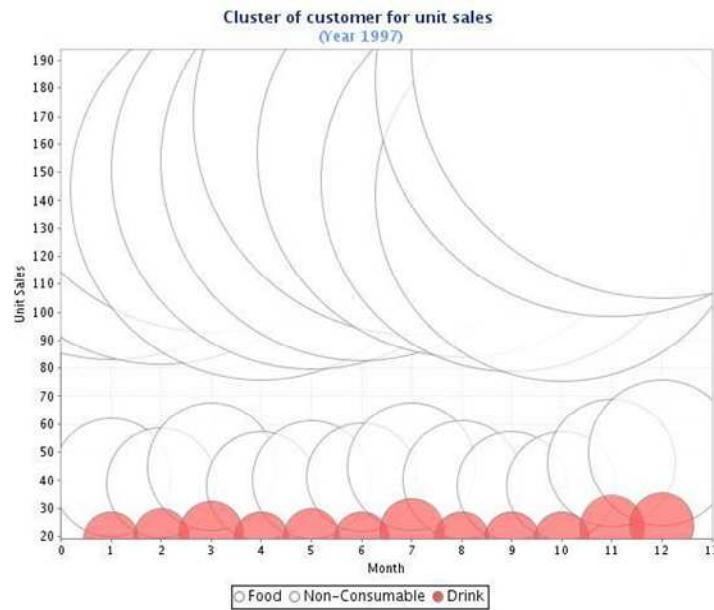


FIGURE 6.18 – Simple cluster chart with one series selected (in red)

There is only one type of cluster chart, called simple cluster chart.

▪ Dataset Structure

The dataset should have the following structure:

```
select 'TRUE' is_sel, 'Drink' series_name, 1 x, 15000 y, 0.8 z
from dual
union
select 'TRUE' is_sel,'Drink' series_name, 2 x, 12000 y, 0.7 z
from dual
union
select 'TRUE' is_sel,'Drink' series_name, 3 x, 14500 y, 0.6 z
from dual
union
select 'FALSE' is_sel, 'Food' series_name, 1 x, 15300 y, 0.1 z
from dual
union
select 'FALSE' is_sel, 'Food' series_name, 4 x, 11000 y, 0.3 z
from dual
union
```

```
select 'FALSE' is_sel, 'Food' series_name, 2 x, 12100 y, 0.5 z
from dual
```

Where:

- Column X defines a point on the X axis
- Column Y defines a point on the Y axis
- Column Z defines the size of clusters
- Column `series_name` defines the series (and the name can be freely chosen)
- There is also an optional boolean column, whose name is specified in the template (in our example it is called `is_sel`). This column states whether the series shall be selected or not: if true, the series is selected, if false it is not. If not specified, series will be assigned default colors.

- **Template structure**

The template has the following structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CLUSTERCHART type="simplecluster" name="Cluster of customer for
unit sales ">
<DIMENSION width='600' height='500' />
<COLORS background='#ffffff' />
<STYLE_X_AXIS_LABELS font="ARIAL" size="10" color="#000000"
orientation="HORIZONTAL" />
<STYLE_Y_AXIS_LABELS font="ARIAL" size="10" color="#000000"
orientation="HORIZONTAL" />
<STYLE_TITLE font="Arial" size="13" color="#0c2d83"
orientation="" />
<STYLE_SUBTITLE font="Arial" size="12" color="#6699ff"
orientation="" name="(1997)" />
<STYLE_LABELS_DEFAULT font="ARIAL" size="10" color="#000000"
orientation="HORIZONTAL" />
<CONF>
<PARAMETER name="column_sel" value="IS_SEL" />
<PARAMETER name="x_label" value="Month" />
<PARAMETER name="y_label" value="Unit Sales" />
<PARAMETER name="default_color" value="#FFFFFF" />
</CONF>
</CLUSTERCHART>
```

In detail:

```
<CLUSTERCHART type="simplecluster" name="Cluster of customer for  
unit sales ">
```

Here we see the type and sub-type of chart, as already described in the commonalities section. The element \$P{month} is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='600' height='500' />  
<STYLE_TITLE font="Arial" size="14" color="#0c2d83"  
orientation="" />
```

As described in the common section, set dimensions and style for the chart. Style can be set for title, subtitle and labels.

```
<COLORS background='#FFFFFF' />
```

Here we set the background color.

```
<CONF>
```

As we know, the tag <CONF> opens the section dedicated to configuration parameters. Configuration specifications for cluster charts are listed below.

Simple cluster chart configuration options	
x_label	The label for the category axis
y_label	The label for the value axis
column_sel	The dataset column used to define whether the series will be visible
default_color	Color for all non selected series

TABLE 6.10 - Simple cluster chart configuration option

Box Chart

A box chart depicts groups of numerical data through five-number summaries: the sample minimum, lower quartile, median, upper quartile, and the sample maximum.

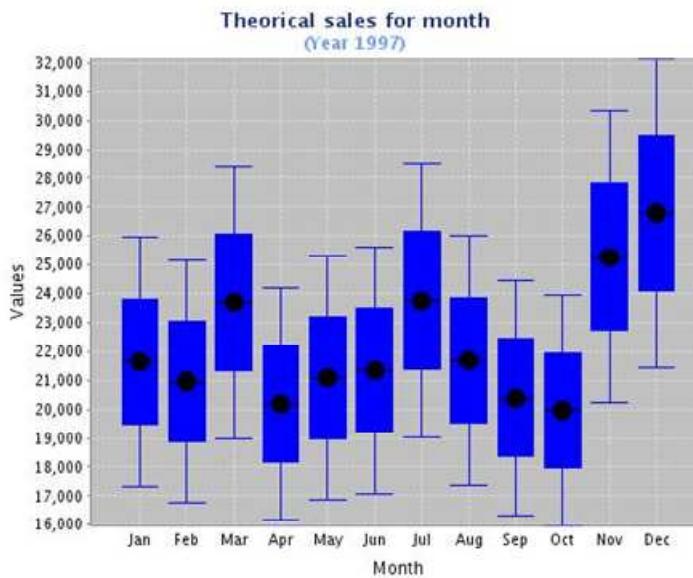


FIGURE 6.19 – Simple box chart

There is only one type of box chart, called Simple Box Chart, with the following configuration options.

▪ Dataset structure

The dataset must have the following structure:

```
select '0-10' as x, 2 as ser1, 3 as ser2, 4 as ser3, 2 as ser4, 5
as ser5 from dual
UNION
select '11-20' as x, 3 as ser1, 1 as ser2, 2 as ser3, 4 as ser4,
7 as ser5 from dual
```

Where:

- X is the category (mandatory)

- Other columns represent series and can have free names.

- **Template structure**

The template has the following structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOXCHART type="simplebox" name="Theoretical sales for month">
<DIMENSION width='500' height='400' />
<COLORS background='#ffffff' />
<STYLE_TITLE font="Arial" size="14" color="#0c2d83" />
<STYLE_VALUE_LABELS font="ARIAL" size="10" color="#ff0000"
orientation="HORIZONTAL" />
<STYLE_SUBTITLE font="Arial" size="12" color="#6699ff"
name="(Year 1997)" />
<STYLE_LABELS_DEFAULT font="ARIAL" size="10" color="#ff0000"
orientation="HORIZONTAL" />
<CONF>
<PARAMETER name="category_label" value="Month" />
<PARAMETER name="n_visualization" value="10" />
<PARAMETER name="value_label" value="Values" />
</CONF>
</BOXCHART>
```

In detail:

```
<BOXCHART type="simplebox" name="Theoretical sales for month">
```

Here we see the type and sub-type of chart, as already described in the commonalities section. The element \${month} is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='600' height='500' />
<STYLE_TITLE font="Arial" size="14" color="#0c2d83" />
```

As described in the common section, set dimensions and style for the chart. Style can be set for title, subtitle and labels.

```
<COLORS background='#FFFFFF' />
```

Here we set the background color.

<CONF>

As we know, the tag <CONF> opens the section dedicated to configuration parameters. Configuration specifications for cluster charts are listed below.

Simple box chart configuration options	
category_label	The label for the category axis
value_label	The label for the value axis
n_visualization	Number of series to be visualized

FIGURE 6.20 – Simple box chart configuration options

Scatter Chart

Scatter charts allow the visualization of values depending on two variables. Values are represented as points on a Cartesian space, where the position of a point on the x axis is given by the value of the first variable, and the position on the y axis is given by the second variable.

For example, in FIGURE 6.21 each point is a store: on the x axis we show the repartition of sales (in percentage) with respect to the average sales value (on the y axis).

There are two sub-types of scatter chart:

- Simple scatter chart
- Marker scatter chart.

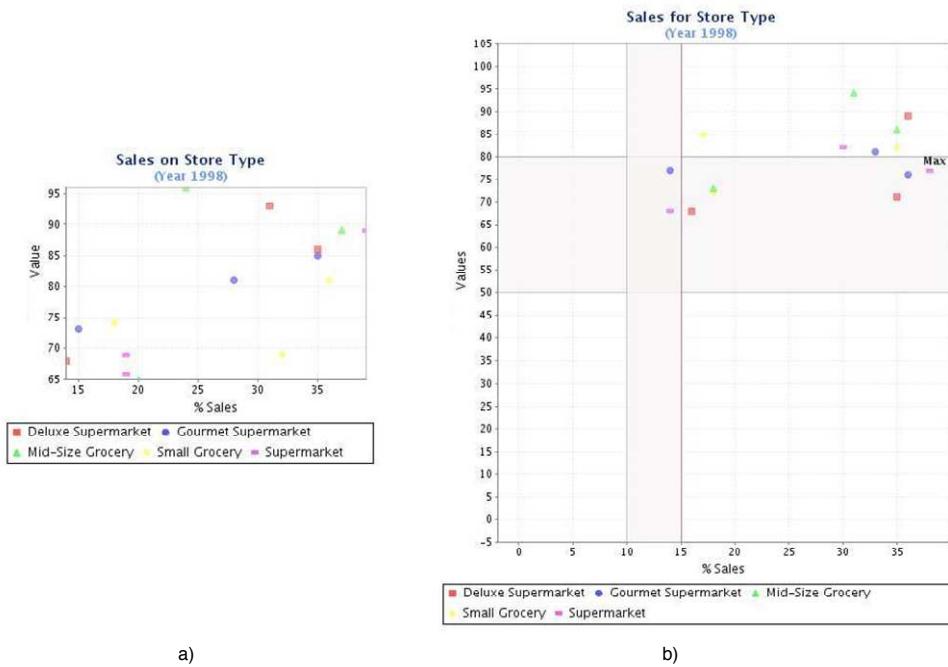


FIGURE 6.21 – Scatter charts: a) simple and b) with markers

▪ Dataset structure

The dataset is the same for both sub-types of scatter chart and must have the following structure:

```
select 'Deluxe Supermarket' as x, 3 as x0, 4 as y0, 10 as x1, 5
as y1 from dual
union
select 'Gourmet Supermarket' as x, 3 as x0, 4 as y0, 4 as x1, 2
as y1 from dual
union
select 'Supermarket' as x, 3 as x0, 4 as y0, 9 as x1, 7 as y1
from dual
```

Where:

- Column x defines the series
- The list of columns called X_i , with $i = 0, 1, 2, \dots$ and corresponding Y_i define positions on the chart.

Simple Scatter Chart

This is the basic version of a scatter chart.

- **Template structure**

Here is an example of template.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SCATTERCHART type="simplescatter" name="scatterProva">
<DIMENSION width='400' height='400' />
<COLORS background='#ffffff' />
<STYLE_Y_AXIS_LABELS font="" size="10" color="#ffffff"
orientation="" />
<STYLE_TITLE font="" size="10" color="#ffffff" orientation="" />
<STYLE_VALUE_LABELS font="" size="10" color="#ffffff"
orientation="" />
<STYLE_SUBTITLE font="" size="10" color="#ffffff" orientation=""
name="" />
<STYLE_LABELS_DEFAULT font="" size="10" color="#ffffff"
orientation="" />
<STYLE_X_AXIS_LABELS font="" size="10" color="#ffffff"
orientation="" />
<CONF>
<PARAMETER name="x_label" value="label x" />
<PARAMETER name="y_label" value="label y" />
<PARAMETER name="default_color" value="RGB {0, 0, 0}" />
<PARAMETER name="view_annotation" value="true" />
<PARAMETER name="legend" value="true" />
<PARAMETER name="legend_position" value="bottom" />
</CONF>
<SERIES_COLORS supermarket='#C9B385' grocery='#8C4A4A' />
</SCATTERCHART>
```

In detail:

```
<SCATTERCHART type="simplescatter" name="scatterProva">
```

Here we see the type and sub-type of chart, as already described in the commonalities section. The element \${month} is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='600' height='500' />
<STYLE_TITLE font="Arial" size="14" color="#0c2d83" />
```

```
...
<STYLE_LABELS_DEFAULT font="" size="10" color="#ffffff"
orientation="" />
<STYLE_X_AXIS_LABELS font="" size="10" color="#ffffff"
orientation="" />
```

As described in the common section, set dimensions and style for the chart. Style can be set for title, subtitle, default labels, x axis labels and y axis labels.

```
<COLORS background="#FFFFFF" />
```

Here we set the background color.

```
<CONF>
```

As we know, the tag `<CONF>` opens the section dedicated to configuration parameters. Configuration specific for simple scatter charts are listed below.

Scatter chart configuration options	
x_label	The label for the category axis
y_label	The label for the value axis
default_colour	Color for all non selected series
view_annotation	If true axes labels are visible
legend	If true, the legend will be shown
legend_position	Position of the legend

TABLE 6.11 - Configuration parameters of simple scatter chart template

```
<SERIES_COLORS supermarket="#C9B385" grocery="#8C4A4A" />
```

defines the color of each series. Series are referenced with the name they are assigned in the dataset.

Marker Scatter Chart

In addition to the simple scatter chart, here you can define range and markers, to inscribe values within pre-defined regions.

■ Template structure

The template is the same as the simple scatter, except parameters defining markers.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SCATTERCHART type="markerscatter" name="Sales for Store Type">
...
<CONF>
<PARAMETER name="x_range" value_low="10.0" value_high="30.0"  />
<PARAMETER name='y_range' value_low='50.0' value_high='100.0'  />
<PARAMETER name="x_marker" label="" value_start_int="10"
value_end_int="15" color_int="0" color="#cc0033"
value_marker="15.0"  />
<PARAMETER name="y_marker" label="Max" value_start_int="50"
value_end_int="80" color_int="0" color="#ffffff"
value_marker="80.0"  />
</SCATTERCHART>
```

In detail:

```
<PARAMETER name="x_range" value_low="10.0" value_high="30.0"  />
<PARAMETER name='y_range' value_low='50.0' value_high='100.0'  />
```

defines the range on the x and y axes.

```
<PARAMETER name="x_marker" label="" value_start_int="10"
value_end_int="15" color_int="0" color="#cc0033"
value_marker="15.0"  />

<PARAMETER name="y_marker" label="Max" value_start_int="50"
value_end_int="80" color_int="0" color="#ffffff"
value_marker="80.0"  />
```

These elements define the marker on the x and y axes. The marker has a name and an optional label. It may be: an interval or a value.

In the first case you need to set minimum and maximum interval values, while in the second case you need to set a value and an optional label. Note that the two options are not alternatives as they can co-exist (see example above).

Parameters for marker definition are:

Marker definition parameters	
name	Values are “x_marker” or “y_marker”
label	Label for the marker
value_marker	Value of the punctual marker
color	Color of the punctual marker
value_start_int	Minimum value for the interval marker
value_end_int	Maximum value for the interval marker
color_int	Color for the interval marker

TABLE 6.12 - Marker definition parameters

XY Block Chart

The Block Chart represents blocks whose color depends on a third dimension (Z). It is also called XY because it is based on a Cartesian representation.

There is currently only one type of XY chart, i.e., the block chart.

▪ Dataset structure

The dataset must return one column for each dimension (x, y, z) as follows:

```
select 5 as x, 20000 as y, 125 as z from dual
union
select 10 as x, 20000 as y, 249 as z from dual
```

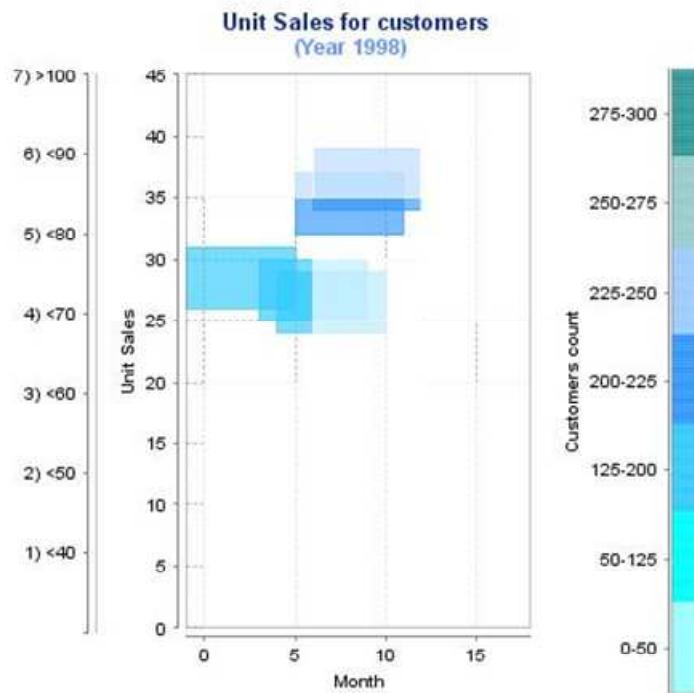


FIGURE 6.22 – Block chart

▪ Template structure

Here is an example of template:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XYCHART type="blockchart" name="Unit Sales for customers">
<DIMENSION width='400' height='400' />
<COLORS background='#ffffff' />
<STYLE_Y_AXIS_LABELS font="" size="10" color="#ffffffff"
orientation="" />
<STYLE_TITLE font="Arial" size="13" color="#0c2d83"
orientation="" />
<STYLE_LABELS font="Arial" size="10" color="#000000"
orientation="horizontal" />
<STYLE_SUBTITLE font="Arial" size="12" color="#6699ff"
orientation="" name="(Year 1998) " />
<STYLE_LABELS_DEFAULT font="Arial" size="10" color="#000000"
orientation="horizontal" />
<STYLE_X_AXIS_LABELS font="" size="10" color="#ffffffff"
orientation="" />
<CONF>
<PARAMETER name="n_visualization" value="20" />
```

```

<PARAMETER name="yrange_value_low" value="1" />
<PARAMETER name="xrange_value_high" value="20" />
<PARAMETER name="grid_height" value="5" />
<PARAMETER name="xrange_value_low" value="1" />
<PARAMETER name="x_label" value="Month" />
<PARAMETER name="yrange_value_high" value="50" />
<PARAMETER name="y_label" value="Unit Sales" />
<PARAMETER name="z_label" value="Customers count" />
<PARAMETER name="grid_width" value="6" />
</CONF>
    <ZRANGES name='zrange' type='static'>
        <RANGE value_low='1' value_high='50' colour='#99FFFF'
label='0-50' />
        <RANGE value_low='50' value_high='125' colour='#00FFFF'
label='50-125' />
        <RANGE value_low='125' value_high='250' colour='#33CCFF'
label='125-200' />
        <RANGE value_low='250' value_high='500' colour='#3399FF'
label='200-225' />
        <RANGE value_low='500' value_high='1000' colour='#99CCFF'
label='225-250' />
        <RANGE value_low='1000' value_high='2000' colour='#99CCCC'
label='250-275' />
        <RANGE value_low='2000' value_high='3000' colour='#339999'
label='275-300' />
        <RANGE value='300' colour='#283B64' label='outbound' />
    </ZRANGES>
    <YRANGES name='yrange' type='static'>
        <RANGE label='1) &lt;40' />
        <RANGE label='2) &lt;50' />
        <RANGE label='3) &lt;60' />
        <RANGE label='4) &lt;70' />
        <RANGE label='5) &lt;80' />
        <RANGE label='6) &lt;90' />
        <RANGE label='7) &gt;100' />
    </YRANGES>
</XYCHART>

```

In detail:

```
<XYCHART type="blockchart" name="Unit Sales for customers">
```

Here we see the type and sub-type of chart, as already described in the commonalities section. The element `$P{month}` is a placeholder for the parameter “month” (see above).

```
<DIMENSION width='400' height='400' />
<STYLE_TITLE font="Arial" size="14" color="#0c2d83" />
```

As described in the common section, set dimensions and style for the chart. Style can be set for title, subtitle, default labels, x and y axes labels.

```
<COLORS background='#FFFFFF' />
```

Here we set the background color.

```
<CONF>
```

As we know, the tag `<CONF>` opens the section dedicated to configuration parameters. Configuration specifications for block charts are listed below.

Block chart configuration options	
x_label	The label for the X axis
y_label	The label for the Y axis
z_label	The label for the Z axis (represented with colors)
X range	
xrange_value_low	Minimum for X axis range
xrange_value_high	Maximum for X axis range
Y range	
yrange_value_low	Minimum for Y axis range
yrange_value_high	Maximum for Y axis range
grid_width	Width of blocks
grid_height	Height of blocks

TABLE 6.13 - Block chart configuration options

```
<ZRANGES name='zrange' type='static'>
<RANGE value_low='1' value_high='50' colour='#99FFFF' label='0-50'
/>
```

This section contains the definitoion of all ranges on the z axis (they can be multiple).

For each range:

```
<YRANGES name='yrange' type='static'>
<RANGE label='1' &lt;40' />
```

Additional ranges on the Y axis, which will appear in the legend. An y range is characterized by a label only.

SpagoBIJSChartEngine

Since version 3.4 of SpagoBI, it has been possible to use a graphics engine called external SpagoBIChartEngine. It allows to display information on predefined graphics and interactive widgets.

In its first version, the engine uses the open source graphics library provided by Sencha¹⁹, which allows the definition of different types of graphs. The list of available charts includes:

- area chart
- bar chart
- line chart
- pie chart
- radar chart
- scatter chart
- gauge chart.

All widgets can assume different levels of complexity, depending on how they are configured and the context in which they are used. Among others, they are particularly valuable inside dashboards, where they can dynamically interact.

The configuration of these documents is realized in XML type templates.

¹⁹ <http://docs.sencha.com/ext-js/4-0/>

The next sections focus on the specific aspects of this type of document.

Dataset structure

As most engines of SpagoBI suite, SpagoBIJSChartEngine uses the dataset to retrieve the information to be displayed. There is no need for defaults columns/aliases. It is fully customizable according to the results you wish to obtain.

Below an example of dataset that can be used in a chart.

```
SELECT x,
       SALES_STORE,
       COSTS_STORE,
       REVENUE,
       ROUND((SALES_STORE*100)/TOT_SALES_STORE,2) AS PERC_SALES,
       ROUND((COSTS_STORE*100)/TOT_COSTS_STORE,2) AS PERC_COSTS ,
       ROUND((REVENUE*100)/TOT_REVENUE,2) AS PERC_REVENUE
FROM
  (SELECT d.the_month x,
          round(sum(s.store_sales)) AS SALES_STORE,
          round(sum(s.store_cost)) AS COSTS_STORE,
          round(sum(s.store_sales*s.unit_sales-s.store_cost)) AS
REVENUE
   FROM sales_fact_1998 s,
        time_by_day d
  WHERE s.time_id = d.time_id
    AND d.the_year = 1998
  GROUP BY d.the_month
  ORDER BY month_of_year)dati,
  (SELECT sum(s.store_sales) AS tot_SALES_STORE,
         sum(s.store_cost) AS tot_COSTS_STORE,
         sum(s.store_sales*s.unit_sales-s.store_cost) AS
tot_REVENUE
   FROM sales_fact_1998 s,
        time_by_day d
  WHERE s.time_id = d.time_id
    AND d.the_year = 1998) tot
```

Template

All documents of this engine need an XML template.

The SpagoBI JSChart engine implements a functionality allowing to obtain a template in JSON format from the XML file associated with the document

itself. In fact ExtJs JSON objects are mapped onto XML templates for SpagoBI JSChartEngine.

The rules for decoding the template from XML to JSON are listed below:

- every tag in the template becomes a simple JSON object, defined as {}, where each attribute corresponds to a property:

XML	JSON
<LEGEND position="bottom"/>	legend: {position:'bottom'}

- Each tag with suffix _LIST becomes an array [] in JSON format. Each tag in this group defines the single element of the array:

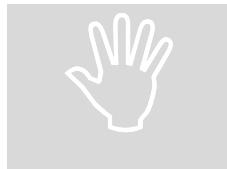
XML	JSON
<AXES_LIST> <AXES title='Number of Hits' /> <AXES title='Names' /> </AXES_LIST>	axes: [{title:' Number of Hits'}, {title: 'Names'}]

- Each property with the suffix _list converts the content into a simple list of values:

XML	JSON
<AXES title='Number of Hits' fields_list="data1,data2,data3" />	{ title: 'Number of Hits', fields: [data1,data2,data3] }

- To allow decoding of an uppercase letter - for example into compound nouns - use the character '_' (underscore):

XML	JSON
<pre><LABELS_STYLE color="#6D869F" fontWeight='bold' fontFamily='Arial' /></pre>	<pre>labelsStyle: { color: '#6D869F', fontWeight: 'bold', fontSize: '14px', fontFamily: 'Arial' }</pre>



JSON-XML decoding

The system is key-sensitive. Therefore, check that tags are uppercase and the properties are lowercase.

These few rules allow you to customize the templates in the most appropriate way, within the limits of what the graphical library allows you to do.

If some objects defined/created in the template cannot be properly decoded and managed, they will be simply ignored by the engine, without generating errors.

In the next sections we will see in detail how to configure a chart for each document. Further configuration options are possible and they are related to the features offered by Sencha Jext library, which was used to implement this engine.



ExtJS chart

The ExtJS library adopts JSON objects to represent chart configuration. To discover all available options for ExtJS chart configuration, please refer to <http://docs.sencha.com/ext-js/4-0/#/>

Generic configuration settings

Most configuration properties are common to all charts. Let us see them in detail:

```
<EXTCHART height='500' width='600' animate='true' shadow='true'
refreshTime='5'>
    <!-- Legend object properties -->
    <LEGEND position='bottom' />
    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>
    <!-- subtitle object properties -->
    <SUBTITLE text='Monthly detail'>
        <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
    </SUBTITLE>
    <COLORS color="#C9B385, #8C4A4A, #0C2D83" />

    <AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
    <LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
    <!-- Axis properties -->
    <AXES_LIST>
        <AXES title='Values' type='Numeric' grid='true' minimum='0'
position='left' fields_list="SALES_STORE,COSTS_STORE,REVENUE" />
            <AXES title='Month' type='Category' grid='true'
position='bottom' fields='x'>
                <LABEL>
                    <ROTATE degrees='315' />
                </LABEL>
                <GRID>
                    <ODD opacity='1' fill='#ddd' stroke='#bbb'
stroke-width='0.5' />
                </GRID>
            </AXES>
        </AXES_LIST>
    <!-- Series properties -->
    <SERIES_LIST>
        <SERIES type='area' axis='left' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
            <STYLE opacity='0.6' />
            <TIPS trackMouse='true' width='140' height='28' />
        </SERIES>
    </SERIES_LIST>
```

```
</EXTCHART>
```

▪ EXTCHART

This is the main tag and identifies the type of library to be used, in this case Sencha ExtJS 4.

Within this tag we can specify:

- *Width / height*: numeric. It specifies the size of the chart in pixels. Default '500x500 px'.
- *Animate*: true / false. It specifies whether the animation shall be enabled at plotting time. Default 'true'.
- *Shadow*: true / false. It specifies whether a light shadow effect shall be applied to the components. Default 'true'.
- *RefreshTime*: Numeric. It specifies the number of seconds after which data shall be refreshed. Default '0 ', according to which data are loaded only once at the beginning.

```
<EXTCHART height='500' width='600' animate='true' shadow='true'  
refreshTime='5'>
```

▪ LEGEND

This is the tag for the legend element. It is optional for all types of charts, except gauge. Within this tag we can specify:

- *Position*: top, bottom, left, right. It specifies where the legend shall be displayed according to the chart.

```
<LEGEND position='bottom' />
```

▪ TITLE

This is the title tag for a specific item. It is optional for all types of charts. Within this tag we can specify:

- *Text*: string. The text to be displayed. It is possible to include the parameter values after the following syntax: \$ {P} <param_url>

- <STYLE> is a sub-element that specifies the style properties of the title. You can specify color, font size and font.

```
<TITLE text='Sales, costs and revenues over the year'>
    <STYLE color='#003366' fontWeight='bold' fontSize='16px'
fontFamily='Arial' />
</TITLE>
```

▪ **SUBTITLE**

This is the tag for the element subtitle. It is optional for all types of charts. Within this tag we can specify:

- **Text**: string. The text to be displayed. It is possible to include the parameter values after the following syntax: \$ {P} <param_url>
- <STYLE> is a sub-element that specifies the style properties of the title. You can specify color, font size and font.

```
<SUBTITLE text='Monthly detail'>
    <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
</SUBTITLE>
```

▪ **COLORS**

This is the tag that allows the customization of the color palette used by the graphical library. It is optional for all types of charts. Within this tag we can specify:

- **Color**: string. Code(s) of the desired color(s).

```
<COLORS color="#C9B385, #8C4A4A, #0C2D83" />
```

▪ **AXES_STYLE**

This is the tag concerning the definition of the layout of the axes. It is optional and available for all the charts that display axes, such as lines, bars, radars, scatters. Within this tag we can specify:

- **color**: string. Code(s) of the desired color(s).
- **FontWeight**, **fontSize**, **fontFamily**: strings. It specifies the font size and font.

```
<AXES_STYLE color="#C6DBEF" fontWeight='bold' fontSize='14px'  
fontFamily='Arial' />
```

▪ **LABELS_STYLE**

This is the tag concerning the definition of the label shown in the graphic. It is optional. Within this tag we can specify:

- **color**: string. Code(s) of the desired color(s).
- **FontWeight**, **fontSize**, **fontFamily**: strings. It specifies the font size and font.

```
<LABELS_STYLE color="#6D869F" fontWeight='bold' fontSize='14px'  
fontFamily='Arial' />
```

▪ **AXES_LIST**

This is the tag concerning the definition of the axes. It is provided for all charts that display axes, such as lines, bars, radars, scatters and gauges. Within this tag we can specify an AXIS element with the details of a single axle:

- **Title**: string. The description that we want to see on the axis.
- **Type**: string. The axis type. It can assume the following values:
 - **Numeric**: to specify numerical values (series)
 - **Category**: to specify categories
 - **Gauge**: only for the gauge chart type.
- **Grid**: true / false. It enables the grid display between axes. Optional. Default 'false'.
- **Minimum**: numeric. It specifies the minimum value of axis. Optional.
- **Maximum**: numeric. It specifies the maximum value of axis. Optional.
- **Position**: top, bottom, left, right. It specifies where the axis shall be placed. Default: 'bottom'.
- **Fields_list**: List of values. It specifies the columns of the dataset (alias) to be displayed on the axis.

- <LABEL>: it is specific of labels of axis. It indicates the level of rotation.
- <Grid> specifies whether the configuration of the grid is active. In the following example we define a mat, two-tone (white and gray) grid.

```
<AXES_LIST>
    <AXES title='Values' type='Numeric' grid='true' minimum='0'
maximum='200' position='left'
fields_list="SALES_STORE,COSTS_STORE,REVENUE" />
    <AXES title='Month' type='Category' grid='true'
position='bottom' fields='x'>
        <LABEL>
            <ROTATE degrees='315' />
        </LABEL>
        <GRID>
            <ODD opacity='1' fill='#ddd' stroke='#bbb' stroke-
width='0.5' />
        </GRID>
    </AXES>
</AXES_LIST>
```

▪ SERIES_LIST

This is the tag concerning the definition of each series. It is provided for all charts. Within this tag we can specify an element SERIES, including the details of each single series:

- **Type:** area, bar, column, line, pie, radar, scatter, gauge. It specifies the type of series and its visualization.
- **Axis:** axis specification to be referenced. The value depends on the elements defined in the previous tag AXES.
- **xField / xField_list:** it specifies the columns of the dataset to be considered as values on the x axis.
- **yField / yField_list:** it specifies the columns of the dataset to be considered as the y-axis values.
- **Highlight:** true / false. It allows to light the series when the mouse passes over them. Default: 'false'.
- **<STYLE>:** it specifies the style of the series (e.g., opacity). Optional.

- <TIPS>: It enables the display of tooltips on the series. It allows to specify the size of the tooltip and related text. By default, the text displayed is {CATEGORY}: {SERIES}.

Now let us have an overview on the different types of charts and their templates.

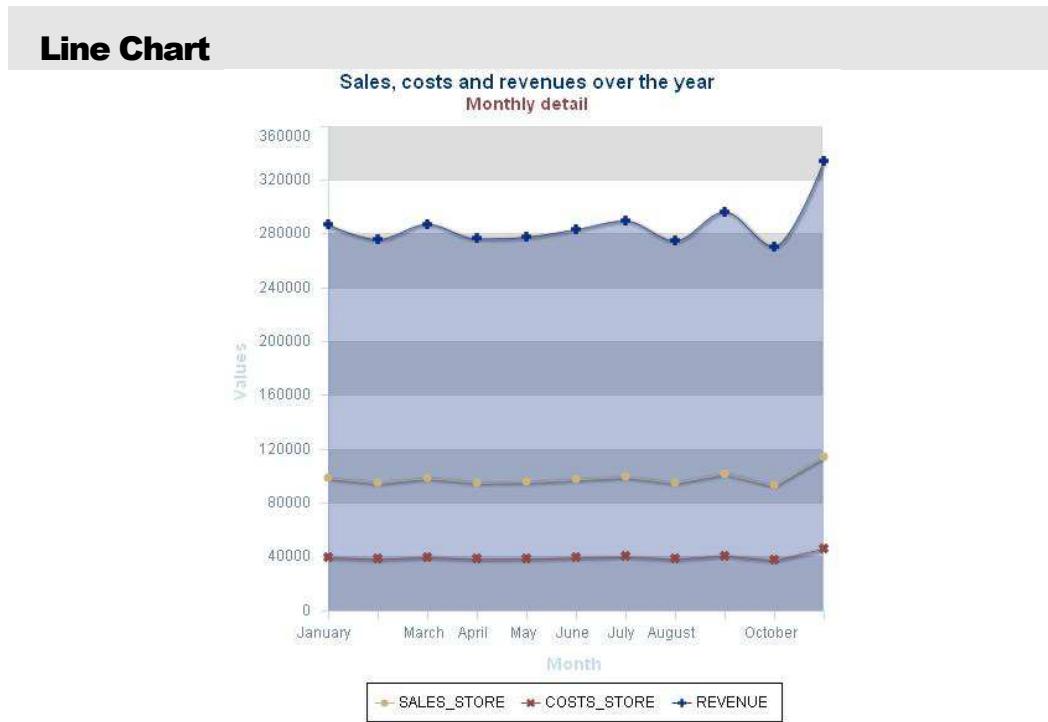


FIGURE 6.23 – SpagoBI ExtJS line chart

Template

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- see ExtJS 4' documentation to view all possible options
(http://docs.sencha.com/ext-js/4-0/#!/guide/drawing_and_chaining)
--&gt;

&lt;EXTCHART height='500' width='500' animate='true' shadow='true'&gt;
    &lt;!-- Legend object properties --&gt;
    &lt;LEGEND position='bottom' /&gt;

    &lt;!-- title object properties --&gt;</pre>

```

```

<TITLE text='Sales, costs and revenues over the year'>
    <STYLE color='#003366' fontWeight='bold' fontSize='16px'
/>
</TITLE>

<!-- subtitle object properties -->
<SUBTITLE text='Monthly detail'>
    <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px'
/>
</SUBTITLE>

<COLORS color="#C9B385, #8C4A4A, #0C2D83" />
<AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
<LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

<!-- Axis properties -->
<AXES_LIST>
    <AXES title='Values' type='Numeric' minimum='0'
position='left' fields_list="SALES_STORE, COSTS_STORE,REVENUE">
        <GRID>
            <ODD opacity='1' fill='#ddd' stroke='#bbb'
stroke-width='0.5' />
        </GRID>
    </AXES>
    <AXES title='Month' type='Category' position='bottom'
fields='x' />
</AXES_LIST>

<!-- Series properties -->
<SERIES_LIST>
    <SERIES type='line' axis='left' xField='x'
yField='SALES_STORE' >
        <TIPS trackMouse='true' width='140' height='28' />
        <MARKER_CONFIG type='cross' size='4' radius='4' />
    </SERIES>
    <SERIES type='line' axis='left' xField='x'
yField='COSTS_STORE' >
        <TIPS trackMouse='true' width='140'
height='28' />
        <MARKER_CONFIG type='circle' size='4' radius='4' />
    </SERIES>
    <SERIES type='line' axis='left' fill='true' xField='x'
yField='REVENUE' smooth='true' >
        <TIPS trackMouse='true' width='140' height='28' />
        <MARKER_CONFIG type='circle' size='4' radius='4' />
    </SERIES>
</SERIES_LIST>
</EXTCHART>

```

Most of the properties are defined in the general section. Below we focus on the properties related to this specific type of graph, as well as on the settings of the individual series.

```
<SERIES_LIST>
    <SERIES type='line' axis='left' xField='x'
yField='SALES_STORE' >
        <TIPS trackMouse='true' width='140' height='28' />
        <MARKER_CONFIG type='cross' size='4' radius='4' />
    </SERIES>
...
<SERIES_LIST>
```

Where:

- **type: line**
- **axis: left**, because the series refers to the numerical axis previously defined in the tag AXES.
- **xField** is the alias of the column in the dataset that returns the category (the month, in this example)
- **yField** is the alias of the column in the dataset that returns the configured series.
- **<TIPS>**: it activates the default tooltip (Category: Value series).
- **<MARKER_CONFIG>**: it defines the style of the marker value:
 - **type: cross, circle**
 - **size: total dimension**
 - **radius: radius eventual size.**

Bar Chart and Line Chart

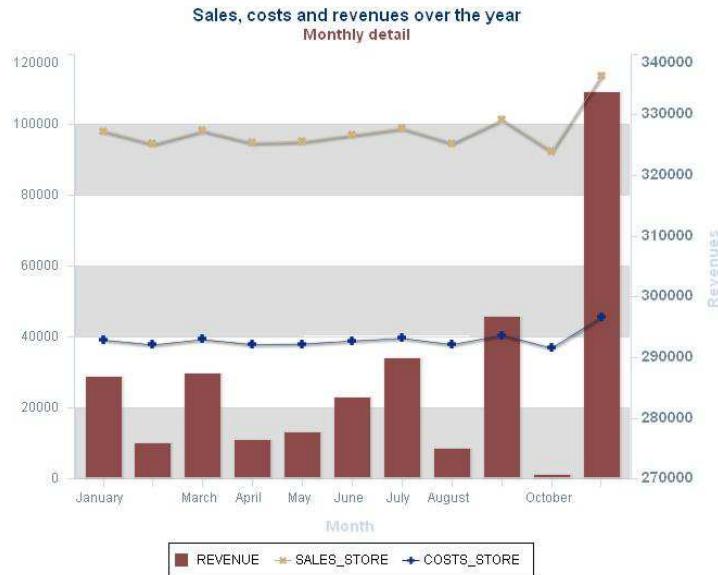


FIGURE 6.24 - SpagoBI ExtJS bar and line chart

Template

```
<EXTCHART height='500' width='700' animate='true' shadow='true'>
    <!-- Legend object properties -->
    <LEGEND position='bottom' />

    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- subtitle object properties -->
    <SUBTITLE text='Monthly detail'>
        <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
    </SUBTITLE>

    <COLORS color="#8C4A4A,#C9B385, #0C2D83" />
    <AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
    fontFamily='Arial' />
    <LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
    fontFamily='Arial' />

    <!-- Axis properties -->
    <AXES_LIST>
```

```

        <AXES title='Values' type='Numeric' minimum='0'
position='left' fields_list = "SALES_STORE,COSTS_STORE">
            <GRID>
                <ODD opacity='1' fill='#ddd' stroke='#bbb' stroke-
width='0.5' />
            </GRID>
        </AXES>
        <AXES title='Revenues' type='Numeric' position='right'
fields='REVENUE' />
        <AXES title='Month' type='Category' position='bottom'
fields='x' />
    </AXES_LIST>

    <!-- Series properties -->
    <SERIES_LIST>
        <SERIES type='column' axis='right' fill='true' xField='x'
yField='REVENUE' smooth='true' >
            <TIPS trackMouse='true' width='140' height='28' />
            <MARKER_CONFIG type='circle' size='4' radius='4' />
        </SERIES>
        <SERIES type='line' axis='left' xField='x'
yField='SALES_STORE' >
            <TIPS trackMouse='true' width='140' height='28' />
            <MARKER_CONFIG type='cross' size='4' radius='4' />
        </SERIES>
        <SERIES type='line' axis='left' xField='x'
yField='COSTS_STORE' >
            <TIPS trackMouse='true' width='140' height='28' />
            <MARKER_CONFIG type='circle' size='4' radius='4' />
        </SERIES>
    </SERIES_LIST>
</EXTCHART>
```

Below we focus on the properties related to this specific type of graph.

```

<SERIES_LIST>
    <SERIES type='line' axis='left' xField='x'
yField='SALES_STORE' >
        <TIPS trackMouse='true' width='140' height='28' />
        <MARKER_CONFIG type='cross' size='4' radius='4' />
    </SERIES>
...
<SERIES_LIST>
```

Where:

- **type:** line or column. The first one assigns the *line* type while the second one assigns the *bar* type.
- **axis:** left or right. It assigns the series to a specific axis according to the previous definition of axes.
- **xField** is the alias of the column in the dataset that returns the category (the month, in this example)
- **yField** is the alias of the dataset column returning the series.
- **<TIPS>** activates the default tooltip (Category: Value series)
- **<MARKER_CONFIG>** defines the style of the marker value:
 - **type:** cross, circle
 - **size:** total dimension
 - **radius:** radius size (optional).

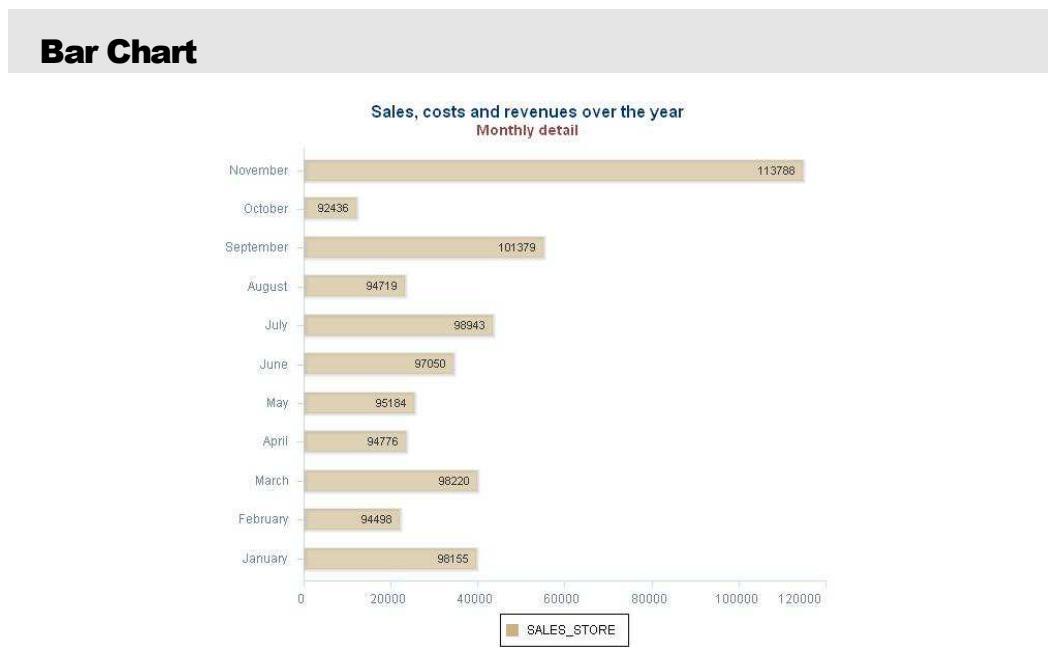


FIGURE 6.25 - SpagoBI ExtJS bar chart

Template

```

<EXTCHART height='500' width='600' animate='true' shadow='true'>
    <!-- Legend object properties -->
    <LEGEND position='bottom' />

    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- subtitle object properties -->
    <SUBTITLE text='Monthly detail'>
        <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
    </SUBTITLE>

    <COLORS color="#C9B385, #8C4A4A, #0C2D83" />
    <AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
    <LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

    <!-- Axis properties -->
    <AXES_LIST>
        <AXES title='false' type='Numeric' position='bottom'
minimum='0' fields_list="SALES_STORE" />
            <AXES title='false' type='Category' position='left'
fields='x' />
        </AXES_LIST>

    <!-- Series properties -->
    <SERIES_LIST>
        <SERIES type='bar' axis='left' gutter='80' highlight='true'
xField='x' yField_list='SALES_STORE' >
            <LABEL display='insideEnd' field='SALES_STORE'
orientation= 'horizontal' color='#333' text-anchor='middle' />
            <STYLE opacity='0.6' />
        </SERIES>
    </SERIES_LIST>
</EXTCHART>

```

Most of the properties are defined in the general section. Below we focus on the properties related to this specific type of graph, as well as on the settings of the individual series.

```
<SERIES_LIST>
    <SERIES type='bar' axis='left' gutter='80' highlight='true'
xField='x' yField_list='SALES_STORE' >
        <LABEL display='insideEnd' field='SALES_STORE'
orientation= 'horizontal' color='#333' text-anchor='middle' />
        <STYLE opacity='0.6' />
    </SERIES>
</SERIES_LIST>
```

Where:

- **type**: bar or column. The first one assigns the *horizontal bar* type, while the second one assigns the *vertical bar* type.
- **axis** assigns the series to a specific axis according to the previous definition of the axes.
- **gutter** specifies the space among bars. Default: 38.2.
- **highlight**: If true, it allows to light the series when the mouse passes over them.
- **xField** is the alias of the column in the dataset that returns the category (the month, in this example).
- **yField** is the alias of the column in the dataset that returns the configured series.
- **<LABEL>** defines the style of the labels on the chart:
 - **display** specifies the presence and location of the label on the bar. It can assume the following values: "rotate", "middle", "insideStart", "insideEnd", "outside", "over", "under", or "none". Default 'none'.
 - **field** is the column to be displayed.
 - **orientation** specifies the orientation of the label. Possible values: horizontal or vertical. Default: 'horizontal'.
 - **color** is the color code of the label. Default value: '#000' (black).
 - **anchor** specifies the type of vertical anchor/centering.

Grouped Bar Chart

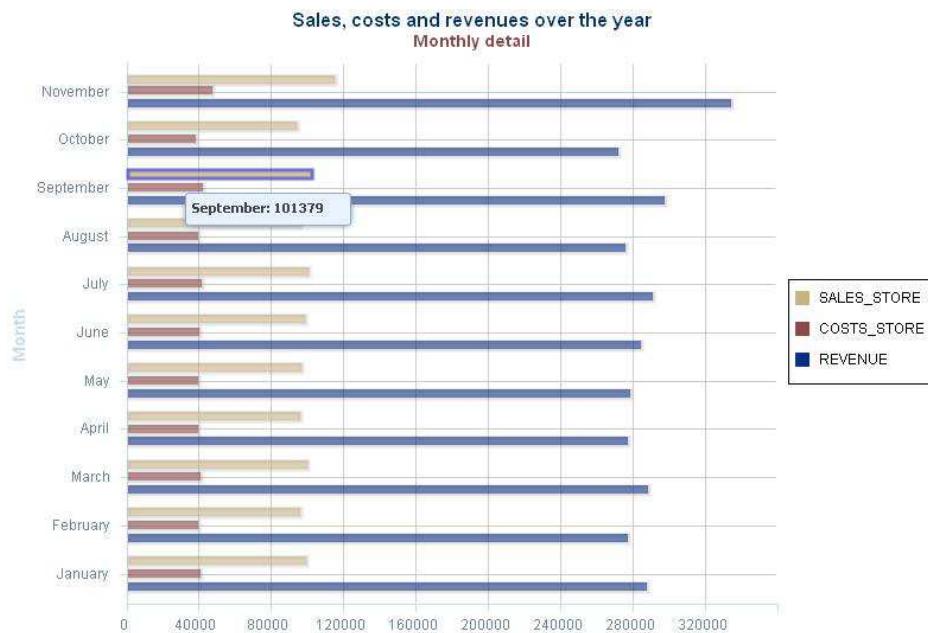


FIGURE 6.26 - SpagoBI ExtJs grouped bar chart

Template

```
<EXTCHART height='500' width='800' animate='true' shadow='true'>
  <!-- Legend object properties -->
  <LEGEND position='right' />

  <!-- title object properties -->
  <TITLE text='Sales, costs and revenues over the year'>
    <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
  </TITLE>

  <!-- subtitle object properties -->
  <SUBTITLE text='Monthly detail'>
    <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
  </SUBTITLE>

  <COLORS color="#C9B385, #8C4A4A, #0C2D83" />
  <AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
  fontFamily='Arial' />
```

```

<LABELS_STYLE color="#6D869F" fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

<!-- Axis properties -->
<AXES_LIST>
    <AXES title='false' type='Numeric' position='bottom'
grid='true' minimum='0'
fields_list="SALES_STORE,COSTS_STORE,REVENUE" />
    <AXES title='Month' type='Category' position='left'
grid='true' fields='x' />
</AXES_LIST>

<!-- Series properties -->
<SERIES_LIST>
    <SERIES type='bar' axis='bottom' highlight='true' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
        <TIPS trackMouse='true' width='140' height='28' />
        <STYLE opacity='0.6' />
    </SERIES>
</SERIES_LIST>
</EXTCHART>

```

Below we focus on the properties related to this specific type of graph.

```

<SERIES_LIST>
    <SERIES type='bar' axis='bottom' highlight='true' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
        <TIPS trackMouse='true' width='140' height='28' />
        <STYLE opacity='0.6' />
    </SERIES>
</SERIES_LIST>

```

Where:

- **type:** bar or column. The first one assigns the horizontal bar type, while the second one assigns the vertical bar type.
- **axis** assigns the series to a specific axis according to the previous definition of the axes.
- **highlight:** If true, it allows to light the series when the mouse passes over them.
- **xField** is the alias of the column in the dataset that returns the category (the month, in this example).

- `yField` is the alias of the column in the dataset that returns the configured grouping series.
- `<TIPS>` activates the default tooltip (Category: Value series)
- `<style>` defines the style (opacity) of the specified value.

Stacked Bar Chart

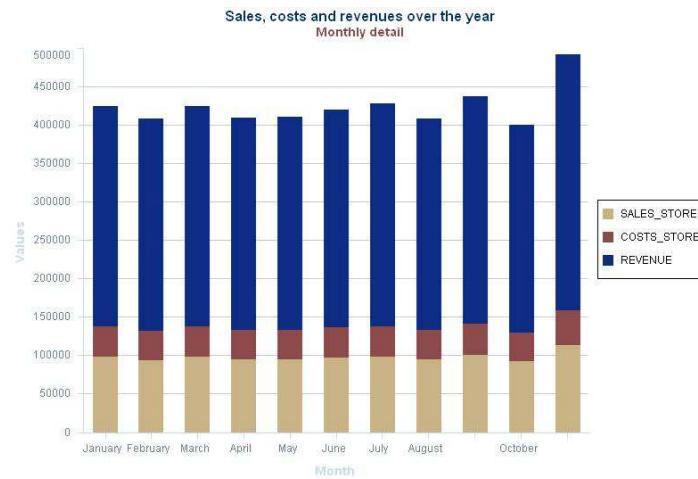


FIGURE 6.27 - SpagoBI ExtJS stacked bar chart

Template

```
<EXTCHART height='500' width='800' animate='true' shadow='true'>
    <!-- Legend object properties -->
    <LEGEND position='right' />

    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- subtitle object properties -->
    <SUBTITLE text='Monthly detail'>
        <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
    </SUBTITLE>
```

```

<COLORS color="#C9B385, #8C4A4A, #0C2D83" />
<AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
<LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

<!-- Axis properties -->
<AXES_LIST>
    <AXES title='Values' type='Numeric' position='left'
grid='true' fields_list="SALES_STORE,COSTS_STORE,REVENUE" />
    <AXES title='Month' type='Category' position='bottom'
fields_list='x' />
</AXES_LIST>

<!-- Series properties -->
<SERIES_LIST>
    <SERIES type='column' gutter='80' axis='left'
highlight='true' stacked='true' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
    </SERIES>
</SERIES_LIST>
</EXTCHART>

```

Below we focus on the properties related to this specific type of graph.

```

<SERIES_LIST>
    <SERIES type='column' gutter='80' axis='left'
highlight='true' stacked='true' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
    </SERIES>
</SERIES_LIST>

```

Where:

- **type:** **bar or column.** The first one assigns the horizontal bar type while the second one assigns the vertical bar type.
- **axis** assigns the series to a specific axis according to the previous definition of axes.
- **gutter** specifies the space between individual bars. Default: 38.2.
- **highlight.** If true, it allows to light the series when the mouse passes over them.

- `xField` is the alias of the column in the dataset that returns the category (the month, in this example)
- `yField` is the alias of the column in the dataset that returns the configured series.
- `stacked`, If true, it sets the type stacked bar.

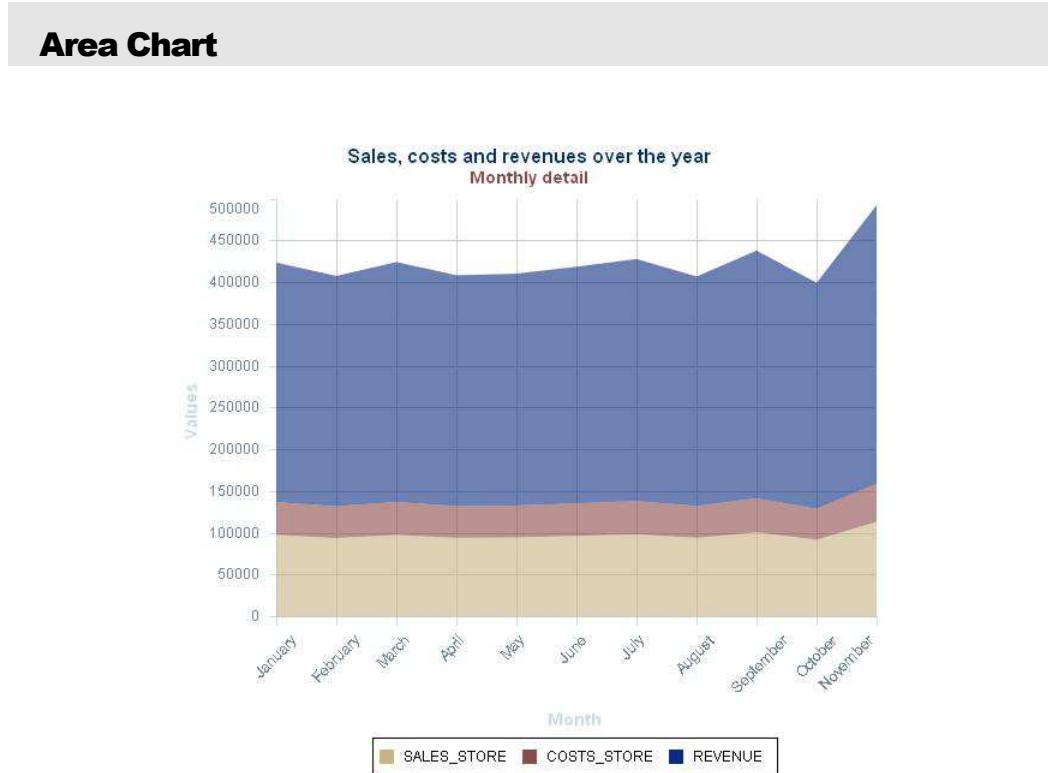


FIGURE 6.28 - SpagoBI ExtJS area chart

Template

```
<EXTCHART height='500' width='600' animate='true' shadow='true'
refreshTime='5'>
  <!-- Legend object properties -->
  <LEGEND position='bottom' />

  <!-- title object properties -->
  <TITLE text='Sales, costs and revenues over the year'>
    <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
```

```

</TITLE>

<!-- subtitle object properties -->
<SUBTITLE text='Monthly detail'>
    <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px'
/>
</SUBTITLE>

<COLORS color="#C9B385, #8C4A4A, #0C2D83" />
<AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
<LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

<!-- Axis properties -->
<AXES_LIST>
    <AXES title='Values' type='Numeric' grid='true' minimum='0'
position='left' fields_list="SALES_STORE,COSTS_STORE,REVENUE" />
    <AXES title='Month' type='Category' grid='true'
position='bottom' fields='x'>
        <LABEL>
            <ROTATE degrees='315' />
        </LABEL>
    </AXES>
</AXES_LIST>

<!-- Series properties -->
<SERIES_LIST>
    <SERIES type='area' axis='left' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
        <STYLE opacity='0.6' />
        <TIPS trackMouse='true' width='140' height='28' />
    </SERIES>
</SERIES_LIST>
</EXTCHART>

```

Below we focus on the properties related to this specific type of graph.

```

<SERIES_LIST>
    <SERIES type='area' axis='left' xField='x'
yField_list='SALES_STORE,COSTS_STORE,REVENUE' >
        <STYLE opacity='0.6' />
        <TIPS trackMouse='true' width='140' height='28' />
    </SERIES>
</SERIES_LIST>

```

Where:

- **type: area.** It sets the chart type.
- *Axis* assigns the series to a specific axis according to the previous definition of axes.
- **highlight.** If true, it allows to light the series when the mouse passes over them.
- **xField** is the alias of the column in the dataset that returns the category (the month, in this example)
- **yField** is the alias of the column in the dataset that returns the series (one or more) of the chart
- **<TIPS>** activates the default tooltip (Category: Value series)
- **<STYLE>** defines the style (opacity) of the specified value.

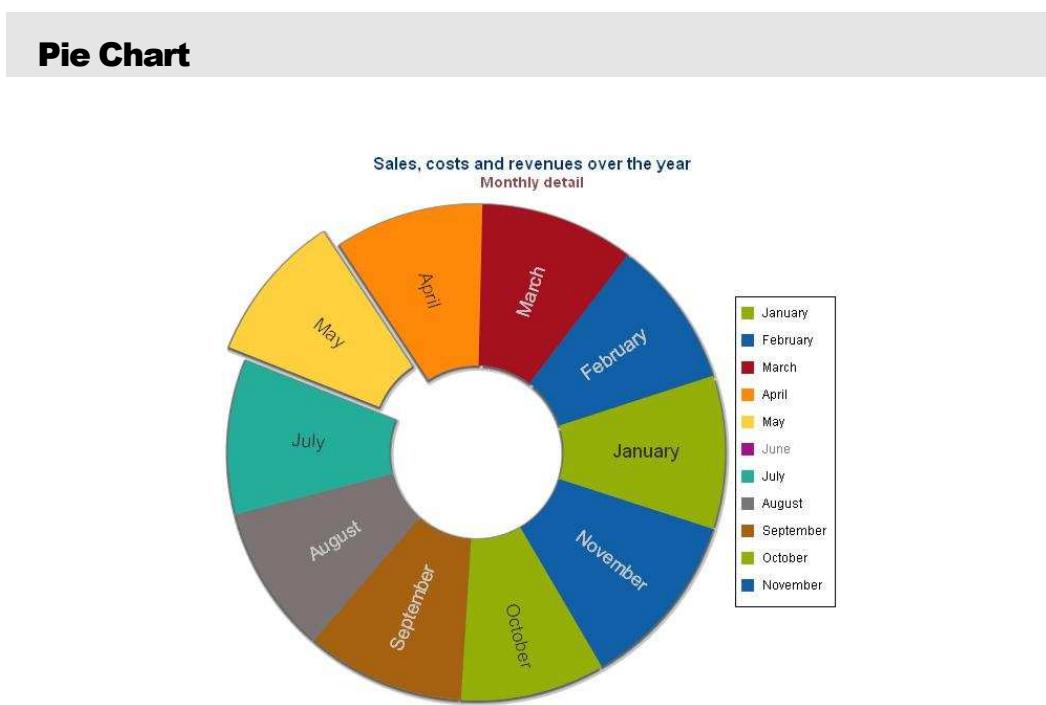


FIGURE 6.29 - SpagoBI ExtJS pie chart

Template

```
<EXTCHART height='500' width='600' animate='true' shadow='true'
insetPadding='60'>
    <!-- Legend object properties -->
    <LEGEND position='right' />

    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- subtitle object properties -->
    <SUBTITLE text='Monthly detail'>
        <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px'
/>
    </SUBTITLE>

    <!-- Series properties -->
    <SERIES_LIST>
        <SERIES type='pie' field='PERC_SALES' showInLegend='true'
donut='35'>
            <TIPS trackMouse='true' width='140'
height='28' />
            <HIGHLIGHT>
                <SEGMENT margin='20' />
            </HIGHLIGHT>
            <LABEL field='x' display='rotate' contrast='true'
font='18px Arial' />
        </SERIES>
    </SERIES_LIST>
</EXTCHART>
```

Below we focus on the properties related to this specific type of graph.

```
<SERIES_LIST>
    <SERIES type='pie' field='PERC_SALES' showInLegend='true'
donut='35'>
        <TIPS trackMouse='true' width='140'
height='28' />
        <HIGHLIGHT>
            <SEGMENT margin='20' />
        </HIGHLIGHT>
        <LABEL field='x' display='rotate' contrast='true'
font='18px Arial' />
    </SERIES>
</SERIES_LIST>
```

Where:

- **type:** 'pie' defines the type of chart.
- **field** is the alias of the column in the dataset that returns the series (the percentage of sales, in this example)
- **showInLegend**. If true, it adds an item to the legend.
- **donut** specifies the percentage of the radius to define the ring. It is 'false' if the ring does not exist. Default: 'false'.
- **<TIPS>** activates the default tooltip (Category: Value series)
- **<HIGHLIGHT>** specifies the action in reaction to the mouse moving over the chart. In this case, it defines a segment with a margin of 20.
- **<LABEL>:** it defines the style of labels on individual slices:
 - **field** specifies the column of the dataset to be displayed on the slices.
 - **display** specifies the type of displaying slices. It can assume the following values: "rotate", "middle", "insideStart", "insideEnd", "outside", "over", "under", or "none". Default: 'none'.
 - **contrast**. If true, it defines the label that shall be in contrast with the background color of the single slice. Default: 'false'.
 - **font** specifies the font of the labels. Default value: 11px Helvetica, sans-serif.

Gauge Chart

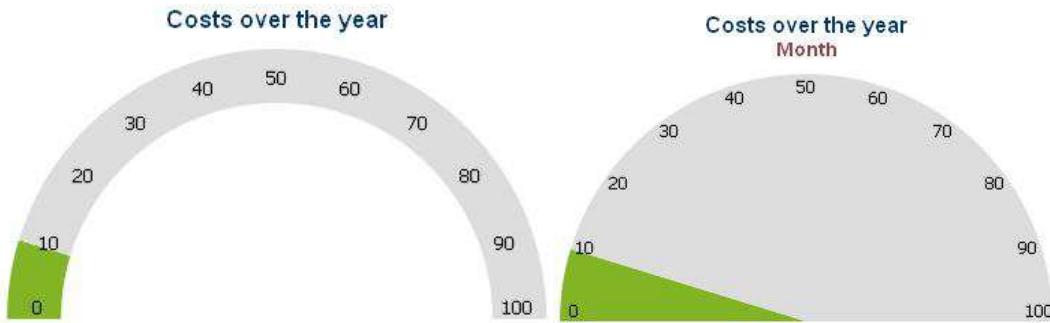


FIGURE 6.30 - SpagoBI ExtJS gauge chart

Template

```
<EXTCHART height='200' width='400' animate='true' shadow='true'
insetPadding='20' flex='1'>

    <!-- title object properties -->
    <TITLE text='Costs over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- theme properties -->
    <COLORS color="#058DC7, #50B432, #ED561B, #DDDF00, #24CBE5,
#64E572, #FF9655, #FFF263, #6AF9C4" />
        <LABELS_STYLE color'#6D869F' fontWeight='bold'
fontSize='16px' fontFamily='Arial' />

    <AXES_LIST>
        <AXES type='gauge' position='gauge' minimum='0'
maximum='100' steps='10' margin='10' />
    </AXES_LIST>

    <!-- Series properties -->
    <SERIES_LIST>
        <SERIES type='gauge' field='PERC_COSTS' donut='80'
colorSet="['#82B525', '#ddd']"><!--donut='false' -->
            <TIPS trackMouse='true' width='140' height='28'
text='{SERIE}%'/>
        </SERIES>
    </SERIES_LIST>
</EXTCHART>
```

Below we focus on the properties related to this specific type of graph.

```
<AXES_LIST>
    <AXES type='gauge' position='gauge' minimum='0'
maximum='100' steps='10' margin='-10' />
</AXES_LIST>
```

Where:

- **Type='gauge'**. Constant. It defines the gauge typical axis.
- **position='gauge'**. Constant.
- **minimum**: the minimum value of the scale.
- **maximum**: the maximum value of the scale.
- **steps**: number of steps.
- **margin**: the space between step and label. Default value: '10'.

```
<SERIES_LIST>
    <SERIES type='gauge' field='PERC_COSTS' donut='80'
colorSet="[ '#82B525', '#ddd' ]"><!--donut='false'  -->
        <TIPS trackMouse='true' width='140' height='28'
text='{SERIE}%' />
    </SERIES>
</SERIES_LIST>
```

Where:

- **Type='gauge'**. Constant. It defines the gauge chart.
- **field** is the alias of the column in the dataset that returns the number (the percentage of costs, in this example).
- **donut** specifies the percentage of the radius to define the ring. It is 'false' if the ring does not exist. Default: 'false'.
- **<TIPS>** activates the tooltip with a custom text (e.g., {SERIE}% is the series name)

Radar Chart

The radar chart represents data on a radial set of axes: categories are plotted on a circular line, while series are plotted on diameters on this circles, as shown in FIGURE 6.31.

Template

```
<EXTCHART height='600' width='650' animate='true' shadow='true'
insetPadding='20'>
    <!-- Legend object properties -->
    <LEGEND position='bottom' />

    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- subtitle object properties -->
    <SUBTITLE text='Monthly detail'>
        <STYLE color='#8C4A4A' fontWeight='bold' fontSize='14px' />
    </SUBTITLE>

    <COLORS color="#C9B385, #8C4A4A, #0C2D83" />
    <AXES_STYLE color='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
    <LABELS_STYLE color='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

    <!-- Axis properties -->
    <AXES_LIST>
        <AXES type='Radial' position='radial' >
            <LABEL display='true' />
        </AXES>
    </AXES_LIST>

    <!-- Series properties -->
    <SERIES_LIST>
        <SERIES type='radar' showInLegend='true' showMarkers='true'
xField='x' yField='SALES_STORE' >
            <MARKER_CONFIG radius='5' size='5' />
            <STYLE stroke-width='2' fill='none' />
            <!--<STYLE opacity='0.4' />-->
        </SERIES>
    </SERIES_LIST>
</EXTCHART>
```

```

<SERIES type='radar' showInLegend='true' showMarkers='true'
xField='x' yField='COSTS_STORE' >
    <MARKER_CONFIG radius='5' size='5' />
    <STYLE stroke-width='2' fill='none' />
    <!--<STYLE opacity='0.4' /-->
</SERIES>
<SERIES type='radar' showInLegend='true' showMarkers='true'
xField='x' yField='REVENUE' >
    <MARKER_CONFIG radius='5' size='5' />
    <STYLE stroke-width='2' fill='none' />
    <!--<STYLE opacity='0.4' /-->
</SERIES>
</SERIES_LIST>
</EXTCHART>

```

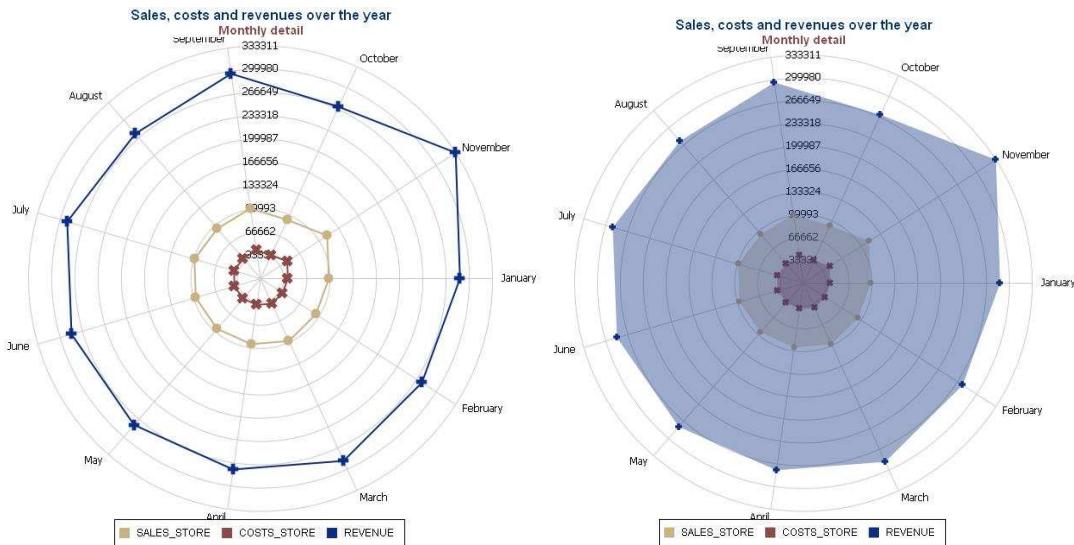


FIGURE 6.31 - SpagoBI ExtJS radar chart

Below we focus on the properties related to this specific type of graph.

```

<AXES_LIST>
    <AXES type='Radial' position='radial' >
        <LABEL display='true' />
    </AXES>
</AXES_LIST>

```

Where:

- **type:** 'Radial'. Constant. It defines the radar typical axis.
- **position:** 'radial'. Constant.
- **<LABEL>** If true, it enables the display of labels on the axis.

```
<SERIES_LIST>
    <SERIES type='radar' showInLegend='true' showMarkers='true'
xField='x' yField='SALES_STORE'
        <MARKER_CONFIG radius='5' size='5' />
        <STYLE stroke-width='2' fill='none' />
        <!--<STYLE opacity='0.4' /-->
    </SERIES>
    ...
</SERIES_LIST>
```

- **type:** 'radar'. It define the chart type.
- **showInLegend:** If true, it adds the series to the legend.
- **ShowMarkers:** If true, it enables the display of a specific marker.
- **xField** is the alias of the column in the dataset that returns the category (the month, in this example).
- **yField_list** is the alias of the column in the dataset that returns the series.
- **<MARKER_CONFIG>** defines the style of the marker value. It has the following attributes:**size** and **radius**.
- **<STYLE>** defines the general style:
 - **stroke-width** specifies the line size.
 - **fill:** 'none' or a color code to be applied on the area of the series.
 - **opacity:** the percentage of series color opacification.

Scatter Chart

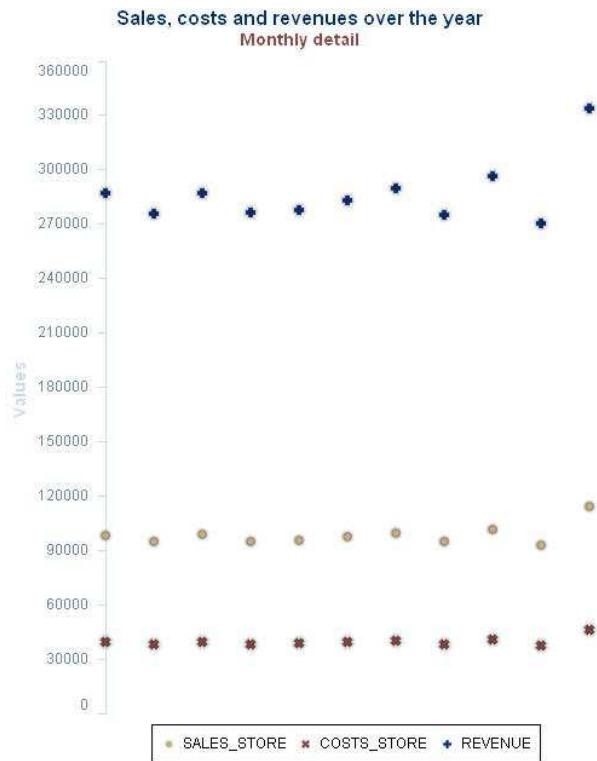


FIGURE 6.32 - SpagoBI ExtJs scatter chart

Template

```
<EXTCHART height='600' width='500' animate='true' shadow='true' >

    <!-- Legend object properties -->
    <LEGEND position='bottom' />

    <!-- title object properties -->
    <TITLE text='Sales, costs and revenues over the year'>
        <STYLE color='#003366' fontWeight='bold' fontSize='16px' />
    </TITLE>

    <!-- subtitle object properties -->
```

```

<SUBTITLE text='Monthly detail'>
    <STYLE color ='#8C4A4A' fontWeight='bold' fontSize='14px' />
</SUBTITLE>

<COLORS color="#C9B385, #8C4A4A, #0C2D83" />
<AXES_STYLE color ='#C6DBEF' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />
<LABELS_STYLE color ='#6D869F' fontWeight='bold' fontSize='14px'
fontFamily='Arial' />

<!-- Axis properties -->
<AXES_LIST>
    <AXES title='Values' type='Numeric' position='left'
fields_list = "SALES_STORE,COSTS_STORE,REVENUE"/>
</AXES_LIST>

<!-- Series properties -->
<SERIES_LIST>
    <SERIES type='scatter' axis='left' xField='x'
yField='SALES_STORE' color ='#a00'>
        <TIPS trackMouse='true' width='140' height='28'
/>
        <MARKER_CONFIG size='5' radius='5' />
    </SERIES>
    <SERIES type='scatter' axis='left' xField='x'
yField='COSTS_STORE' >
        <TIPS trackMouse='true' width='140' height='28'
/>
        <MARKER_CONFIG size='5' radius='5' />
    </SERIES>
    <SERIES type='scatter' axis='left' xField='x'
yField='REVENUE' >
        <TIPS trackMouse='true' width='140' height='28'
/>
        <MARKER_CONFIG size='5' radius='5' />
    </SERIES>
</SERIES_LIST>
</EXTCHART>
```

Below we focus on the properties related to this specific type of graph.

```

<SERIES_LIST>
    <SERIES type='scatter' axis='left' xField='x'
yField='SALES_STORE' color ='#a00'>
        <TIPS trackMouse='true' width='140' height='28'
/>
        <MARKER_CONFIG size='5' radius='5' />
    </SERIES>
    ...

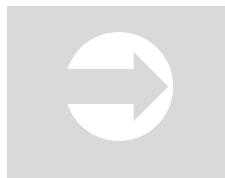
```

```
</SERIES_LIST>
```

- `type: 'scatter'` defines the chart type.
- `axis` assigns the series to the specific axis according to the previous definition of the axes.
- `xField` is the alias of the column in the dataset that returns the category (the month, in this example).
- `yField_list` is the alias of the column in the dataset that returns the series.
- `<MARKER_CONFIG>` defines the style of the marker value. It has the following attributes: `size` and `radius`.
- `<TIPS>` activates the default tooltip (Category: Value series)

Cross navigation

The ExtJSChart Engine allows the definition of cross-navigation paths between charts and other SpagoBI documents as well. Clicking on the different parts of a chart, users can navigate through detail documents.



Cross Navigation

To understand what cross navigation is and how it works, please refer to the corresponding section in chapter 4, SpagoBI Server.



FIGURE 6.33 - Cross navigation between two SpagoBI ExtJS charts

To take advantage of this function, the following configuration must be defined in your document template:

```
<!-- cross navigation definition -->
<DRILL document="BookExtGaugeFilled">
  <PARAM_LIST>
    <PARAM name="month" type="CATEGORY"/>
    <PARAM name='title' type="ABSOLUTE" value='Gender chart' />
    <PARAM name="value" type="SERIE" />
    <PARAM name="data" type="RELATIVE" />
  </PARAM_LIST>
</DRILL>
```

Where:

- <DRILL> is the tag that enables the cross navigation. Add it to the master document. Within this tag we can specify:
 - document : it specifies the label (defined in SpagoBI) of the document to be retrieved.
- <PARAM_LIST>: the list of parameters to be passed to the target document, defined as follows:

Cross navigation parameters definition		
name		String that defines the parameter label
type	CATEGORY	The parameter value that will be passed is the selected category
	SERIE	The parameter value that will be passed is the selected series
	ABSOLUTE	The parameter value that will be passed is a fixed value, defined in the “value” attribute
	RELATIVE	The parameter value that will be passed is the value of the parameter with the same name in the source document

FIGURE 6.34 - Cross navigation parameters definition for SpagoBI ExtJS charts

Export

The ExtJS Chart Engine supports export of documents in two formats: PDF and JPG. The export feature is available in the chart execution window, with no need for further template configuration.



FIGURE 6.35 - Export feature for SpagoBI ExtJS charts



Exporters

Read more about Export in SpagoBI at section Cross Services, chapter 5 – SpagoBI Server.

Analytical cockpit

Analytical cockpits are usually oriented to the immediate perception of multiple contexts. Their main characteristics are:

- a compound layout (including charts, maps, speedometers, KPI, etc.)
- visualization of data coming from different domains
- semantic relations among data
- progressive selections
- easy to use and understand
- supported by interactive tools.

For these reasons, analytical cockpits have a pervasive level of usage and they are usually very appreciated by high-level users and analysts for both synthetic and detailed analysis, without any particular difficulty.

SpagoBI offers a specific engine (SpagoBICompositeDocEngine) for the realization of complex cockpits which allow to aggregate several documents into a single view, connecting them with one another, fostering their interactive and intuitive usage.

Analytical cockpits frequently use dashboard components, specifically oriented to the synthetic control on performances. These widgets follow the metaphor of avionic cockpit, showing performance indicators by means of symbolic graphics on a fixed scale with thresholds to highlight the dangerous areas.



Dashboard

A dashboard component, usable in every cockpit to show punctual or real-time data, is also provided by SpagoBI. Please refers to section Real time, chapter 6 – Analytical Engines.

SpagoBICompositeDocEngine

The CompositeDocEngine allows the execution and visualization of multiple SpagoBI documents in the same page. These documents can be independent or connected by one or more internal navigation paths that allow the user to explore data across documents, following an increasing level of analytical detail.

Technically speaking, any SpagoBI document can be part of a composed document. Most often, composing documents are charts, reports, geo-referenced analysis or monitoring documents.

The steps to create a composed document on the Server are similar to any other SpagoBI document. In particular:

- Choose the composing documents (this is peculiar to this type of document)
- Develop an XML template for the composed document
- Create the analytical document on the Server and add analytical drivers, if any.



Designer for SpagoBIComposedDocEngine

In the following we describe how to manually create a composed document on SpagoBI Server. Composed documents can also be designed using SpagoBI Studio, as described in chapter 4 - SpagoBI Studio, section Cockpit.

Creating the template

Let us consider the document example shown in FIGURE 6.36. This document shows sales and costs by brand (table on the left), and by single product (bar chart on the right). It also supports an internal navigation path: clicking on a row of the table (i.e., a brand), data in the bar chart are refreshed to show only products belonging to that brand.

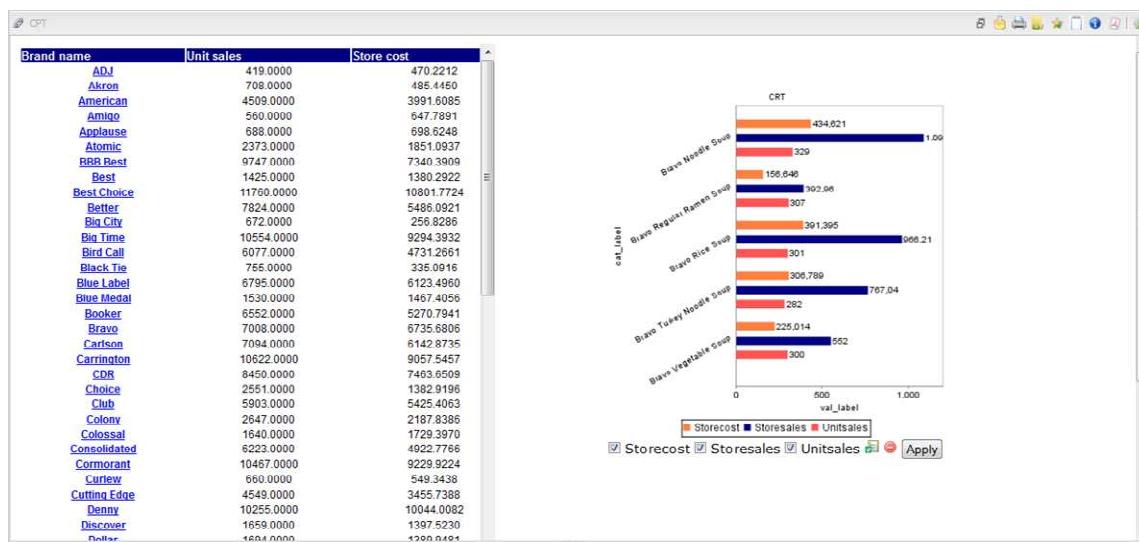


FIGURE 6.36 - Example of SpagoBI composed document

The example document uses the following template:

```
<DOCUMENTS_COMPOSITION
template_value="/jsp/engines/documentcomposition/template/dynamic
Template.jsp">
    <DOCUMENTS_CONFIGURATION video_width="1200"
video_height="1064">
        <DOCUMENT sbi_obj_label="RPT" >
            <PARAMETERS>
                <PARAMETER type="OUT" sbi_par_label="PrBrand"
default_value="Bravo">
                    <REFRESH>
                        <REFRESH_DOC_LINKED labelDoc="CRT"
labelParam="PrBrand" idParam="0"/>
                    </REFRESH>
                </PARAMETER>
            </PARAMETERS>
```

```

<STYLE
style="position:absolute;margin:0px;left:0px;top:0px;width:570px;
height:810px;" />
</DOCUMENT>
<DOCUMENT sbi_obj_label="CRT" >
<PARAMETERS>
    <PARAMETER type="IN" sbi_par_label="PrBrand"
default_value="Bravo" />
</PARAMETERS>
<STYLE
style="position:absolute;margin:0px;left:570px;top:0px;width:630px;
height:774px;" mode="auto"/>
</DOCUMENT>
</DOCUMENTS_CONFIGURATION>
</DOCUMENTS_COMPOSITION>

```

Where:

- The `<DOCUMENTS_COMPOSITION>` tag contains the configuration to properly execute the document and should not be modified.
- The `<DOCUMENTS_CONFIGURATION>` tag contains configuration information about the document. Here you can define the size of the default display to optimize the layout.
- The tag `<DOCUMENT>` defines each composing document. There will be one element of this type for each composing document. The attribute `sbi_obj_label` defines the label of SpagoBI document.
- The `<PARAMETERS>` section contains the definition of parameters either taken as input or sent as output for internal navigation by the specific document. Each `<PARAMETER>` block defines one parameter, as follows:

Parameter attributes in the composing document

type	A string containing the possible values: IN or OUT. If IN, the parameter is mandatory for the composing document. If OUT, the parameter is sent out for internal refresh of other documents.
sbi_obj_label	A string defining the URI of the parameter.
default_value	Value of the parameter, if no value is received at document execution time.

typeCross	Specifies the type of navigation associated to the parameter. Possible values: INTERNAL or EXTERNAL. If INTERNAL, one or more documents of the same composed document are refreshed. If external, a new document is opened in cross navigation.
-----------	---

TABLE 6.14 - Parameter definition for a composing document

- The <REFRESH> tag, contained in each <PARAMETER> section, defines the configuration for internal refresh of documents. In particular, it contains a <REFRESH_DOC_LINKED> tag with the following parameters:
 - **labelDoc**: a string defining the label of the SpagoBI document to be refreshed.
 - **LabelParam**: the URI of the parameter defined in the target document that will collect the value passed by the source document.
- The <STYLE> tag defined styles of the HTML <div> element that will contain the document. It can be modified using standard HTML configuration (e.g., borders).

KPI

KPI stands for Key Performance Indicator. It is a set of metrics, usually derived from simple measures, which allow managers to take a snapshot on key aspects of their business. KPI main characteristics are:

- summary indicators
- complex calculations
- characterized by thresholds supporting results evaluation
- reference target goals
- well-structured and hierarchical models, with weights
- easy to use but possibly complex to model
- associated to alarms

- may be visualized as dashboards
- not necessarily real time
- may refer to a specific time frame.

For these reasons, KPI models are always defined by expert analysts and then used to analyze performances through synthetic views that select and outline only meaningful information.

SpagoBI offers all necessary tools to create, manage, visualize and browse KPI hierarchical models.

Creating a KPI model with SpagoBI consists of three main phases. First, a KPI model is defined and the KPI analytical document is built. Then, KPI values defined in the model are calculated according to some temporal logics. Finally, those values are made visible to end users via appropriate reporting and visualization interfaces.

These phases correspond in SpagoBI to three distinct modules (as shown in FIGURE 6.37):

- A set of tools for metadata configuration. Metadata define the KPI model and how it is calculated.
- The actual engine that calculates KPI values, keeping the history of the calculated values
- The interface for KPI reporting and monitoring. By allowing the visualization of synthetic KPI values with respect to user-defined thresholds, it is the access point for end users to browse and analyze KPIs.

In the following sections, we discuss each phase and how to perform it using the corresponding SpagoBI module.

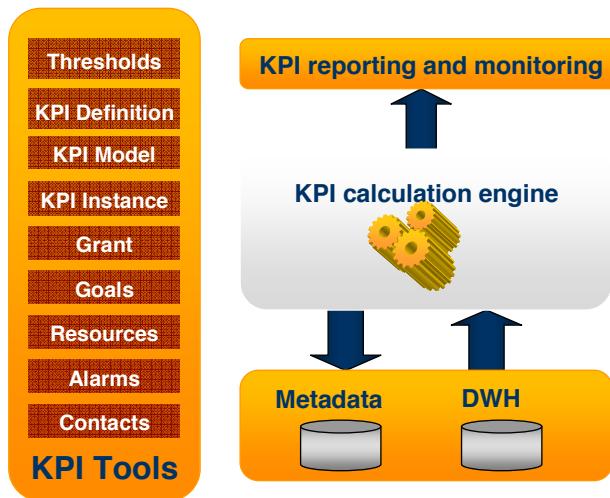


FIGURE 6.37 – SpagoBI KPI Engine modules

Build a KPI model

Building a KPI model requires the definition of different types of metadata specific to this analytical area. Some of them are essential to the KPI definition process and must be defined in any case, while others are optional and can be used to support the implementation of complex and sophisticated KPI scenarios.

The mandatory steps involving the metadata definition to build a KPI model are:

- **KPI definition (with calculation rules).** This is the basic building block of a KPI model. To define a single KPI, set its name and features, and define how to calculate it (via a SpagoBI dataset).
- **Model definition.** Once the single indicators are defined, put them together according to a hierarchical structure that reflects the underlying performance evaluation model.
- **Model instance.** Finally, create a specific instance of an abstract model, which will be used to build the actual KPI analytical document.

Other steps are optional, but they are recommended to give enhanced expressivity and flexibility to the KPI model:

- **Thresholds definition.** Indicators will be more meaningful if referred to a specific threshold: this gives a quantitative measure of their goodness.
- **Resources definition.** Sometimes you need to calculate the same KPI for different sectors of a business sharing the same characteristics (e.g., divisions or types of processes). So you can define just one KPI and apply it to different “resources” in your model.

Further functionalities are available to create more complex models and provide end users with advanced support tools:

- **Grant definition.** Very often organizations are structured in hierarchies with different functional roles. Granting a KPI means to enable the calculation of that KPI for a specific organization unit inside the organization hierarchy
- **Goals definition.** A goal consists of a set of qualitative objectives (with thresholds and weights) associated to a KPI model. As such, a goal does not impact KPI calculation but adds qualitative information to it.

KPI-specific functionalities are accessible via the **KPI Model** menu item, as shown in FIGURE 6.38. In addition to the functionalities shown in this menu, calculation rules are to be defined by creating one or more SpagoBI datasets via the **Resources > Data set** menu tool.



FIGURE 6.38 - SpagoBI KPI Model functionalities

To create a KPI model you can either choose to define the basic components and link them later to the KPI model, or to define them while defining the

model itself. KPI tools allow both modalities. For the sake of clarity, in this chapter we will adopt the first approach.

In the following, we discuss each of the above listed steps.

KPI and calculation rules definition

The first step to build a KPI model is to define its building blocks, i.e., single indicators. An indicator is calculated according to some logics, which may involve *simple* and *derived* measures. Simple measures typically correspond to information already stored in the data warehouse, while derived ones involve more complex formulas that use one or more simple variables, logical/mathematical operators and/or statistical functions.

Calculation Rules

KPI calculation rules are defined through a SpagoBI dataset. In case of simple measures, datasets are typically SQL queries, while for derived measure it is more common to use scripting or Java classes. This is not, however, a strict rule, rather a common practice.

In the example of FIGURE 6.39, we define a derived measure for the logistics area via a Groovy script: the inventory turnover rate. Basic variables, defined as parameters of the dataset, are the cost of goods and the value of the average inventory and their ratio is computed to have a derived measure.

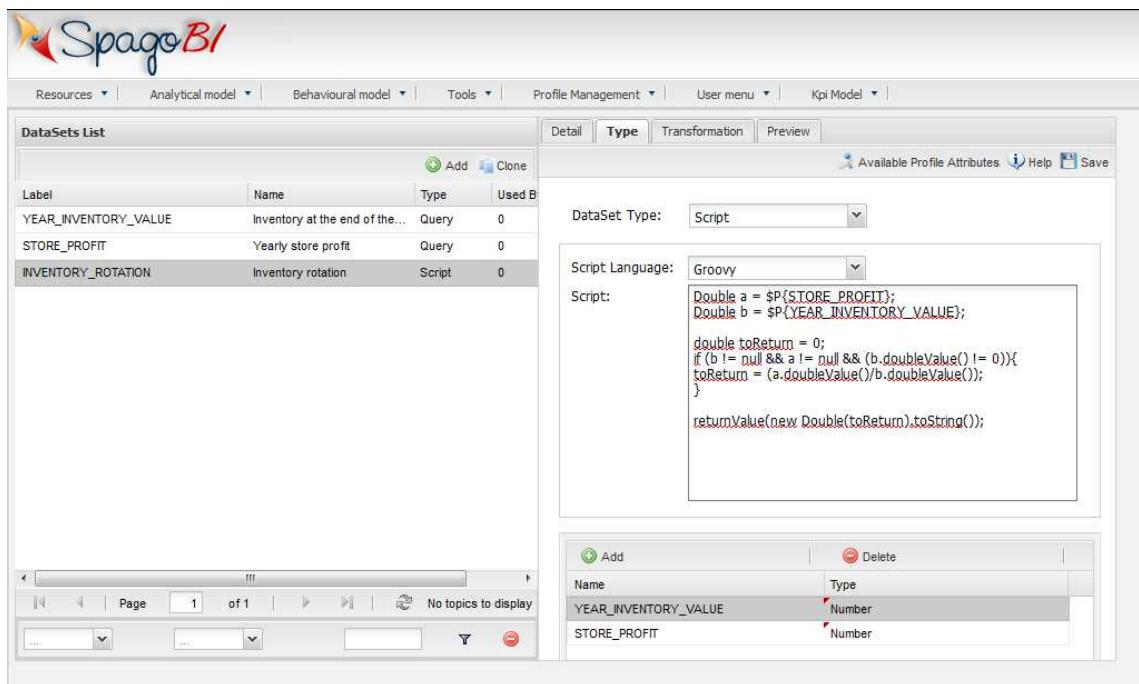
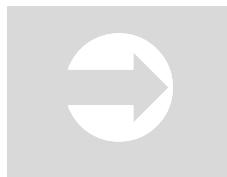


FIGURE 6.39 - Example of derived measure for KPI calculation



Dataset Definition

To learn how to define a dataset in SpagoBI, using different syntaxes and possibly adding parameters, please refer to section Data Sets in chapter 5 - SpagoBI Server.

A KPI calculation formula, as any other SpagoBI dataset, may contain parameters. Parameters may be user-defined, as in our example above.

A special set of parameters is specific for KPI definition: those parameters will be interpreted by the KPI Engine in datasets used by KPIs and will influence the calculation logics. Note that the user does not need to add those parameters in the **Parameters** tab below the dataset editor.

KPI-specific system parameters	
ParKpiResources	The resource name (from the SBI_RESOURCES tables) that is used for the KPI value computation.
ParKpiDate	The reference date to be used for the KPI value computation.
ParKpiDateFrom	The reference start date to be used for the KPI value computation.
ParKpiDateTo	The reference end date to be used for the KPI value computation.

TABLE 6.15 - KPI-specific system parameters

For example, we may define a SQL query that calculates sales based on month and brand. The month is calculated via the `ParKpiDate` parameter, while the brand has been defined as a resource and can thus be referenced via the `ParKpiResource` system parameter. If ten different brands have been defined as resources, then the KPI will be calculated for each of them. Resources will be explained in a later paragraph.

```
Select sum(store_sale) from sales_fact_1998 a, time_by_day b ,
product c
Where a.time_id = b.time_id and
      a.product_id = c.product_id and
      b.month = STR_TO_DATE($P{ParKpiDate}, '%d,%m,%Y') and
      c.brand_name = $P{ParKpiResource}
```

KPI definition

Once calculation rules have been set, we can define the indicator itself. KPIs may be single calculation items or they can depend on other KPIs, combined in a hierarchy. The KPI engine will take dependencies into account when calculating values for each KPI.

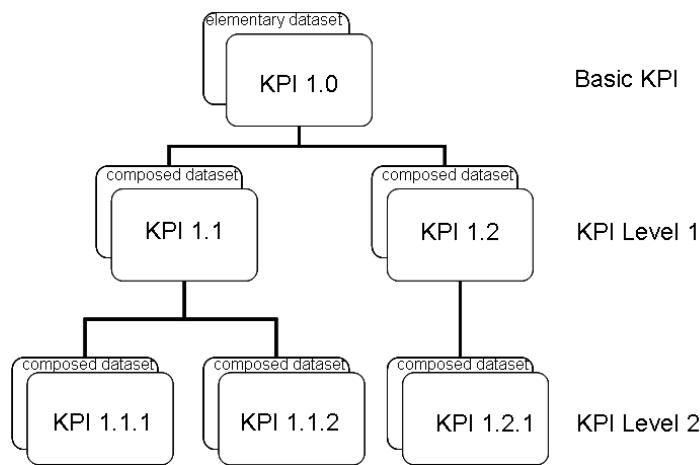


FIGURE 6.40 - KPI hierarchy

A KPI is associated to the following basic elements:

- a name, a unique code and a description
- a dataset defining its calculation rule
- a threshold.

These are the base building blocks of a KPI. Additional elements can be defined, such as a link to a document and a weight, as we will see in the next paragraphs.

To create a new KPI, select **KPI Model > KPI Definition** in the main menu. The KPI editor (shown in FIGURE 6.41) will open. On the left, the list of already defined KPIs. On the right, the KPI editor for viewing/modifying details of a KPI: the one selected in the left panel, or a new one when you click on **Add**.

The right editor panel consists of four tabs:

- **Detail**
- **Advanced**
- **Udp values**
- **KPI links**.

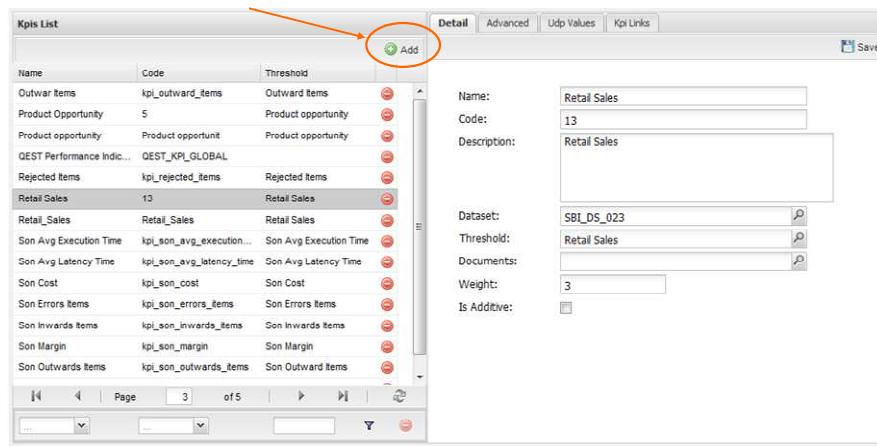


FIGURE 6.41 - KPI editor panel

In the **Details** tab, we set the basic information: name, code, description; dataset; and threshold. Note that the threshold is not mandatory but it is commonly associated to a KPI to increase expressivity.

We can also add to the KPI:

- A **weight** that reflects the relative importance of this KPI compared to the other KPIs related to it. The weight is meaningful whenever KPIs are put into a hierarchy and simple KPIs are used to calculate a complex one.
- A document. A link to this document will be shown in the KPI analytical document: clicking on it, the user will be able to navigate from the KPI to the document.

The little symbol shows that datasets, thresholds and documents can be searched among the existing ones and selected. Alternatively, you can create each of those items at the same time: clicking on the same symbol, you will add a new element (e.g., dataset, threshold), define it and associate it to the KPI.



The steps to build a KPI

We suggest you first create the dataset and the threshold; then create the KPI. This will make the process clearer and easier to manage.

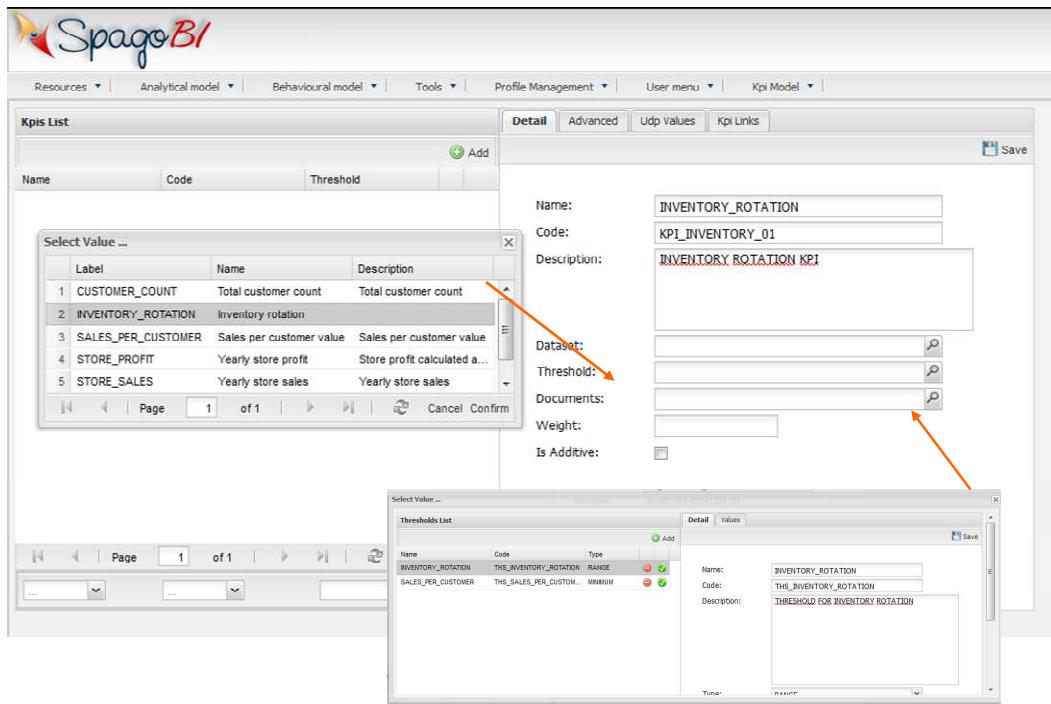


FIGURE 6.42 - Creating a new KPI – dataset and threshold definition

In the **KPI Links** tab, the list of parameters associated to the dataset will appear. In case the KPI is a derived one, its dataset must be composite. In the **KPI Links** section, you should consequently configure the listed parameters by assigning each of them to their original dataset.

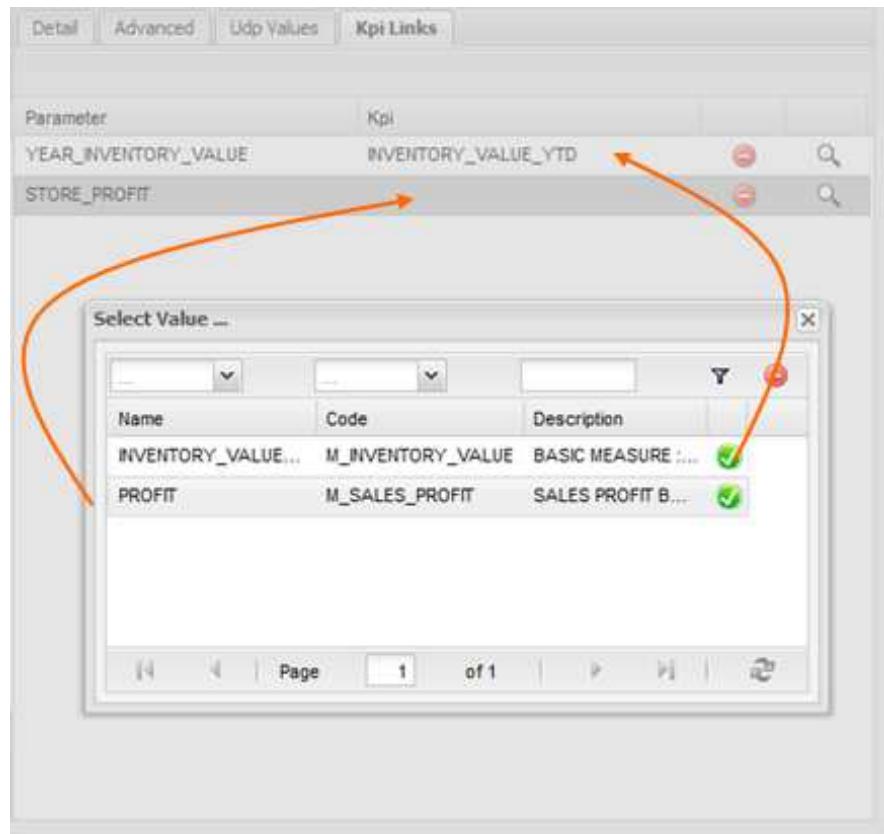


FIGURE 6.43 - Composed KPI configuration – KPI links

Model definition

A KPI model is a hierarchical structure that assembles and organizes KPIs according to a measurement model of processes of a specific business area. The model will be used as a basis for KPI reporting, as we will see shortly. For this reason it is important to build a proper model, which reflects the specific domain to be modelled.

To create a new KPI model, select **KPI Model > Model definition**. You must first have your individual KPIs ready so that you can put them together into a model. We have seen in the previous section how to build a single KPI, either a simple one or a derived one.

In the KPI model editor you can create a model. The editor has three sections:

- The list of existing models (on the left)

- The model configuration editor (in the center)
- The list of available KPIs. Note that by default the list may be hidden. If this is the case, click on the small arrow on the right border of the window (as shown in FIGURE 6.44, where the KPI list is hidden).

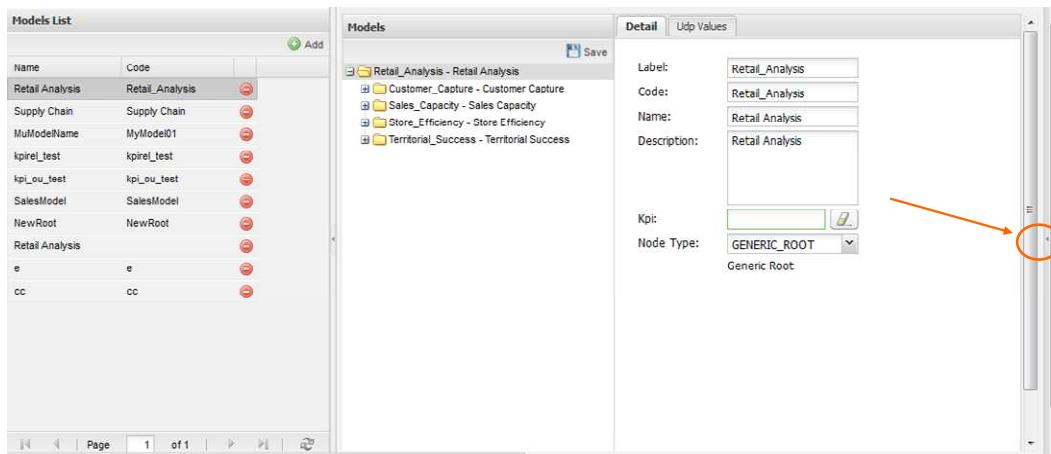


FIGURE 6.44 - KPI Model editor – KPI model list and model configuration editor

Similarly, clicking on the same arrows on the internal window separators, the KPI model panel can be minimized, to optimize the working area.

To visualize more details on an existing model, select the model on the left panel: details will be shown in the model configuration editor. To add a new model click on **Add**: the configuration panel will be shown, ready to be filled.

In a SpagoBI KPI model there are two possible types of nodes according to their position in the model tree:

- Root node (GENERIC_ROOT). There is only one root for each model, i.e., the root of all other nodes.
- Generic node (GENERIC_NODE). All other nodes.

In addition, there are two types of nodes, depending on whether they are linked to a KPI or they simply act as a folder for children nodes. For example, we may define a folder node that represents a business process and contains three KPI

nodes for that process. Note that this distinction is orthogonal to the previous one.

Now let us build a new KPI model. First of all, we define the root node by giving this node a name, a code and selecting GENERIC_ROOT in the **Node Type** input box.

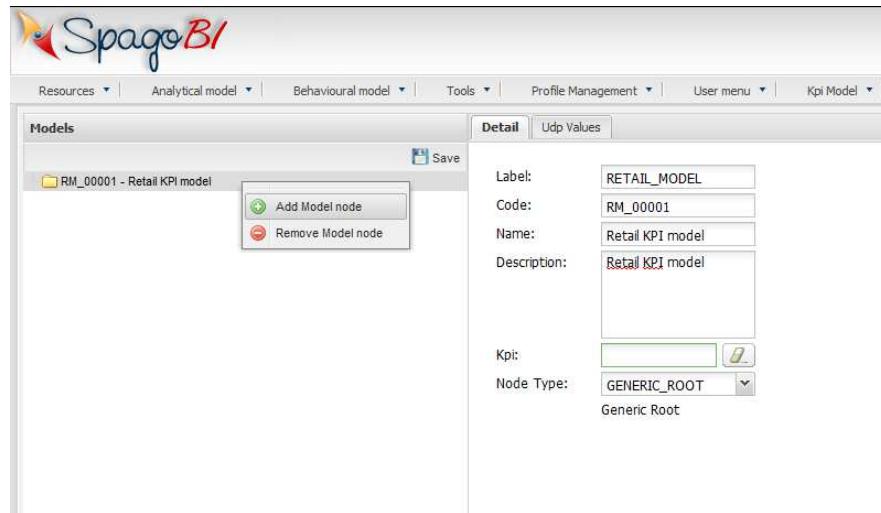


FIGURE 6.45 - Create the root node of a KPI model

Although you can assign a KPI to the root node, in our example we proceed on creating a child node and leave the root node as a folder. To add a new node, right click on the node you just created and select the **Add Model node**: a new node will be added as a child. Using the same contextual menu you can also delete a node.

We select GENERIC_NODE for the second node (because it is not the root). This time we want a KPI node: so we drag the selected KPI from the right panel and drop it into the **Kpi** input box. The new KPI node will appear in the tree model representation as a semaphore (see FIGURE 6.46).

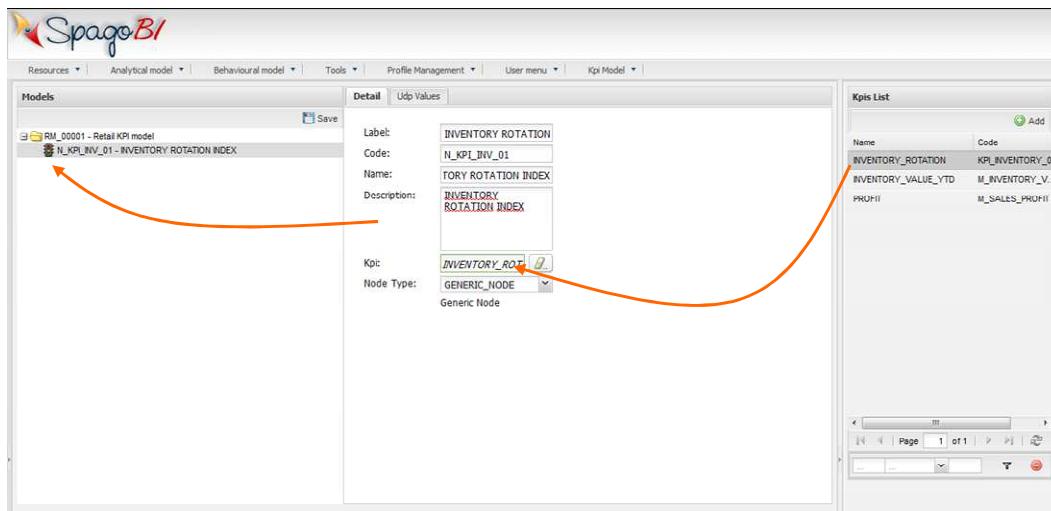


FIGURE 6.46 - Creating a KPI node in the model

In the same way you can add as many nodes as your model requires and build a nested hierarchy.



Building a KPI model

Consider that a KPI node may be the father of further KPI nodes, but does not generally have folder child nodes. This is because folder nodes just act as containers that organize and classify KPIs.

A KPI model represents logical model, a sort of a template of the actual KPI document. It is still defined at a high level of abstraction and does not include, for example, the definition of resources. The executable document is the instantiation of the KPI model, as discussed in the next section.

Model instance

Instantiating a model allows the KPI engine to derive an executable instance from the logical (KPI) model. It is possible to obtain several different instances from the same KPI model, which will differ because of their configuration dimensions.

Those configurable dimensions are:

- Thresholds
- Associations to SpagoBI resources
- Association with an organizational hierarchy.

To create a new KPI instance, select **KPI Model > Model Instance**. The editor shown in FIGURE 6.47 will be shown. This editor consists of three sections:

- The list of existing model instances (on the left)
- The model instance editor panel (at the center)
- The KPI model the current instance is based on (on the right).

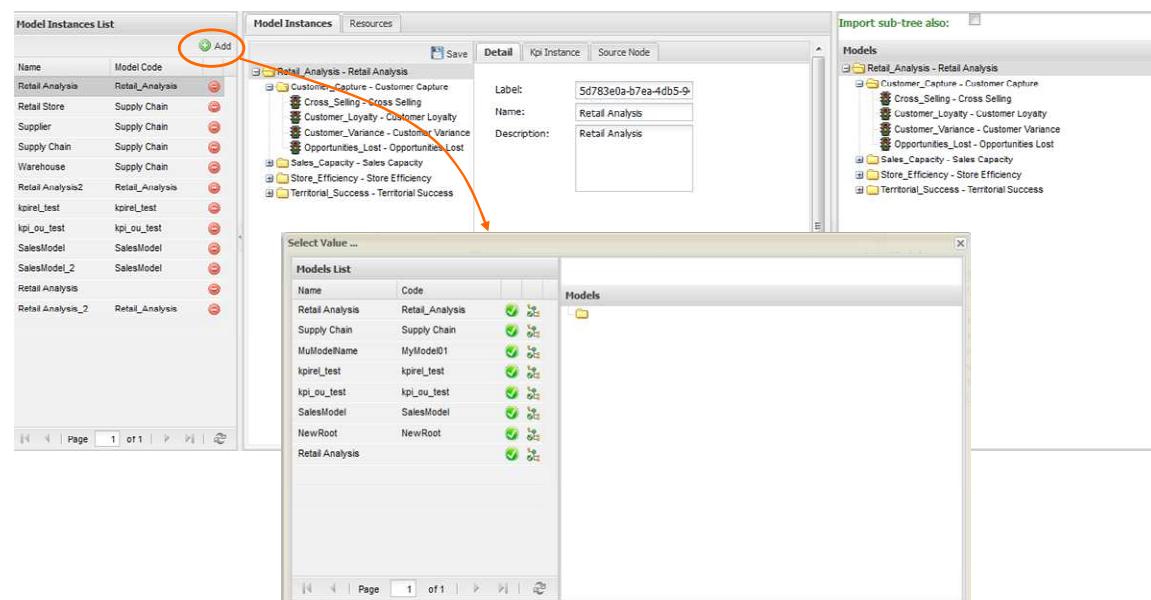


FIGURE 6.47 - KPI instance definition editor

There are two possible ways to create a model instance:

- Clone a whole existing KPI model
- Create an empty instance and fill it with levels and KPIs of interest picked from a model.

In both cases, you have to select  on the left panel and a pop up window with all available models will open. On each row there is a model and beside this two icons:

- Select  to clone the entire model
- Select  to build an empty instance with only the root node of the model.

In the latter case you can then drag and drop nodes (both KPIs and folders) from the model on the right panel onto the instance tree.

Now you can set several options in your instance model. In the **KPI Instance** tab, you can:

- change the association of a node to a KPI that was originally defined in the model. Use the  button to clean out the KPI value of the node. Then you either set the option **Kpi Instance**, and drag the new KPI into the cell you just cleaned; or you set the option **UUID** and you write the unique identifier of the KPI (the code);
- define thresholds and weights for the specific instance, overriding those defined for single KPIs. Note that these operations will only affect the current instance: neither the model nor single KPIs;
- set a target value for the KPI instance;
- set the type of chart that the KPI document will display for this node via the Chart Type option;
- check the **Save History** box if you wish to enable versioning of the KPI instance .

So far we described the basic operations. If you wish to read more advanced topics about KPI building, go on reading below. Otherwise jump directly to section Calculate a KPI.

Thresholds definition

Thresholds are a crucial element in KPI building as they support the benchmarking of calculation results compared to reference values. Their

definition is not mandatory in SpagoBI KPI engine; nevertheless, it is a good practice to set them when building a KPI model.

We have already seen that thresholds can be associated to a single KPI (see KPI definition) or directly linked to a node of a model instance. The value defined for the model instance will anyway override the one defined for the single KPI.

A threshold is characterized by the following elements:

- Name, unique code and description
- Type, which can be one of:
 - Range. An interval of values, divided in sub-intervals.
 - Minimum. A value representing the lowest admissible threshold for the KPI
 - Maximum. A value representing the highest admissible threshold for the KPI
- Values, depending on the type of threshold.

The KPI Engine offers a tool to define thresholds. Select **KPI Model > Threshold Definition**: the editor will open. It is composed of two sections:

- the list of already defined thresholds (on the left). This section also contains a filtering toolbar at the bottom.
- the configuration editor (on the right). This is composed of two tabs: **Details** and **Values**.

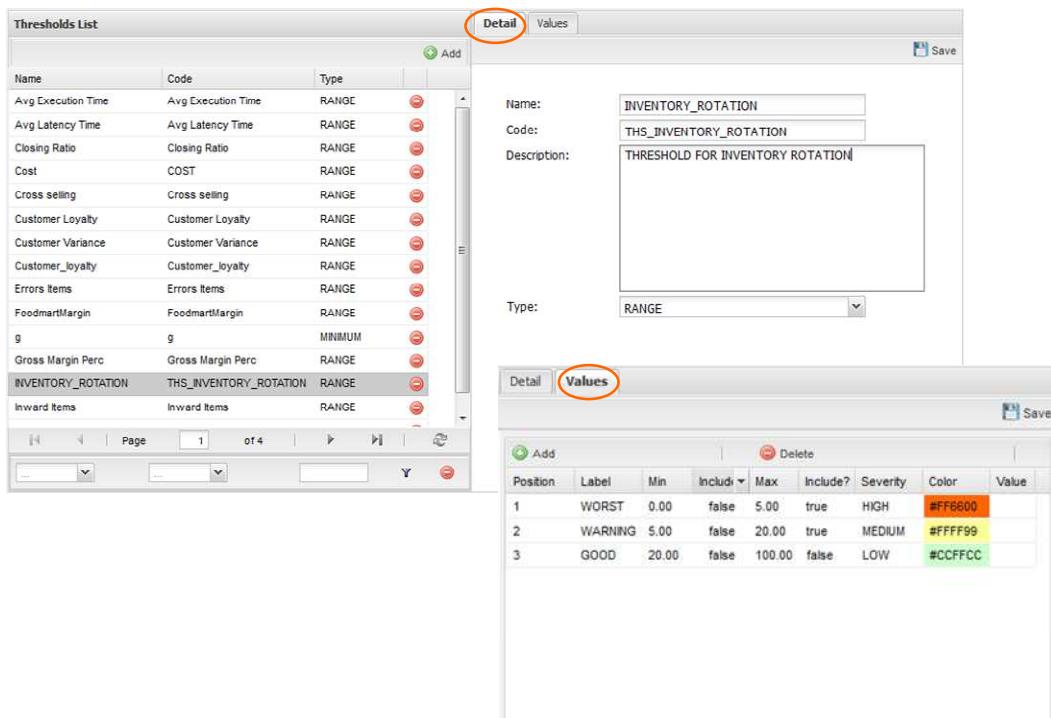


FIGURE 6.48 - KPI threshold configuration panel

To add a new threshold, click on Add. On the right you will be shown an empty editor, where you should define a name, a code and a description for the threshold. Then choose the type in the drop down menu.

In the **Values** tab, you will enter values for the threshold and visualization criteria for KPI reporting. For example, for a threshold of type Range, you need to define various sub-intervals within this range and assign them meaningful colors and names. If you defined a threshold of type maximum/minimum, you will define just one value.

Resources definition

As briefly discussed above, resources are an optional means to customize and iterate KPI calculation over the values of an external variable. This variable typically assumes the values of an analytical dimension in the data warehouse. The same KPI is therefore calculated for different values of the external variable.

For example, we may wish to calculate the inventory turnover ratio for each region: the same KPI should be calculated separately for stores located in each region (that is, the external variable).

To create a new resource, select **KPI Model > Resource Definition**. The editor is similar to editors for KPI metadata that we have already seen: on the left the list of already defined resources, on the right the resource detail panel.

To add a new resource, click on  . On the right you will be shown an empty editor, where you should define a name, a code and a description for the resource. Table and column name refer to the data warehouse representing the resource.

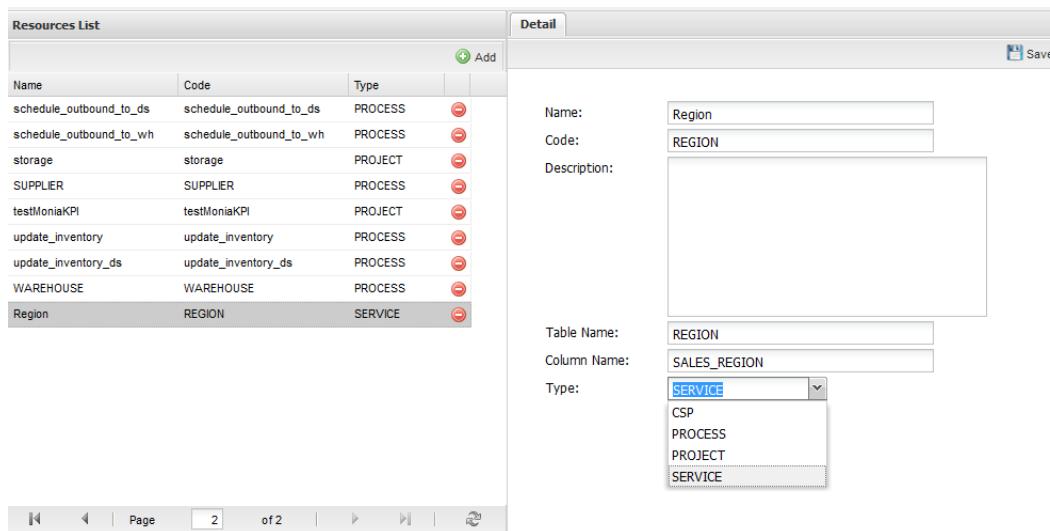


FIGURE 6.49 - Resource definition panel

Once the resource is defined, we need to associate it to the KPI so that calculation is performed according to the current value of the resource. This can be performed by using the resource as a parameter in the query defining the KPI calculation (see Calculation Rules).

Grant definition

A company might want to analyze its business data grouping them by organization units (e.g., company organization, sales force organization).

Organization hierarchies are usually represented by a tree of fixed or variable depth.

Granting means enabling a KPI calculation for a specific organization unit inside the organization hierarchy.

Organization entities must be defined inside the data warehouse: the leaves of each tree are directly connected to the data you want to analyze and the other nodes are located in special entities (organization dimension or bridge table). Many hierarchies could be defined on the same set of organization units.

To configure grants for a KPI model, you need to perform the following steps:

- Import organizational structures from source systems via a dedicated procedure
- Define grants using the web interface and assign KPIs of an instance model to organizational units
- Add special parameters to SpagoBI dataset that calculates the KPI included in the instance model.

Import organizational units

SpagoBI provides a Java API called `Loader Interface` to automatically import organizational structures from the data warehouse directly into the repository of the BI tool. Since the modelling logics of these hierarchies may vary depending on the source system, the implementation of the Loader Interface is typically part of the integration project that uses SpagoBI.

Grant definition

To add new grants, select **KPI Model > Grant definition**. At first execution, you need to synchronize with the organizational hierarchies contained in the data warehouse, clicking on  **Synchronize organizational units**. The same operation can be done whenever an update is needed.

After synchronization you can add a new grant clicking on . On the right you will be shown an empty editor, where you should define:

- a label, a name and a description for the grant
- the hierarchy from the imported ones
- the KPI instance from the available ones.

When you click **Save** for the first time, KPI calculation is enabled for each node of the associated hierarchy. You can see this from in the **Tree** tab, which shows on the left the organizational hierarchy with all its nodes, and on the right the KPI model instance.

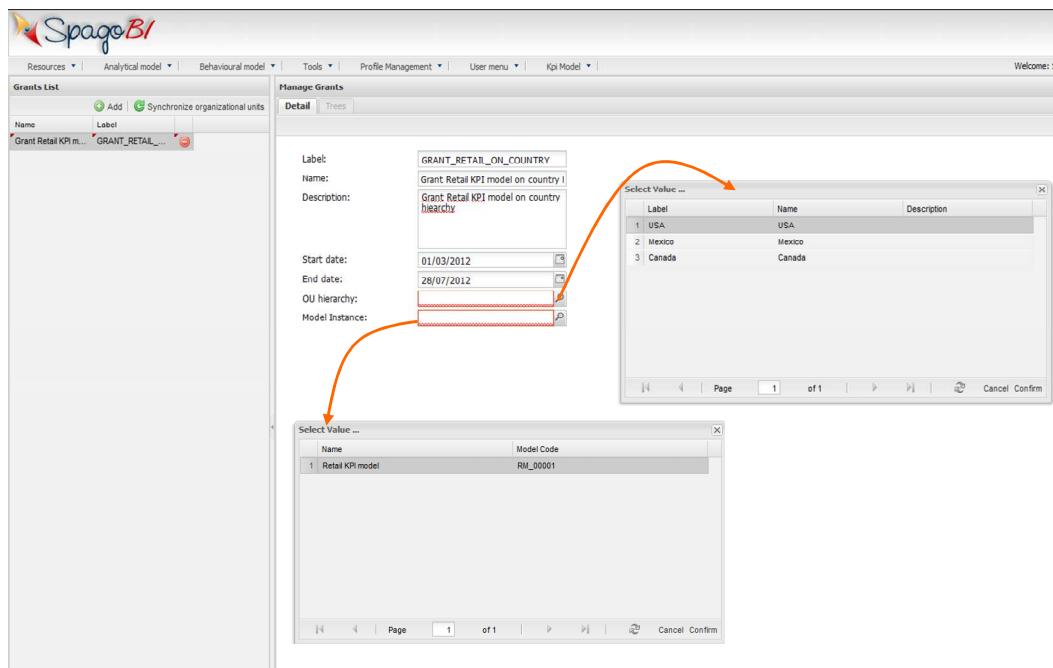


FIGURE 6.50 – KPI grants configuration panel

What you need to do is set the correspondence between the nodes in the hierarchy and the KPI nodes in the model instance. To do so:

- select a node (organization unit) in the hierarchy on the left tree
- select a model instance node on the right tree
- check one or more KPI(s) within the selected node.

The selected KPI(s) will be calculated for the corresponding organizational units.

You may act massively both at organization and KPI model level using the functionalities of the contextual menu:

- disable all descendants
- enable all descendants (only on KPI model tree).

These functionalities can be activated right clicking on a hierarchy node.

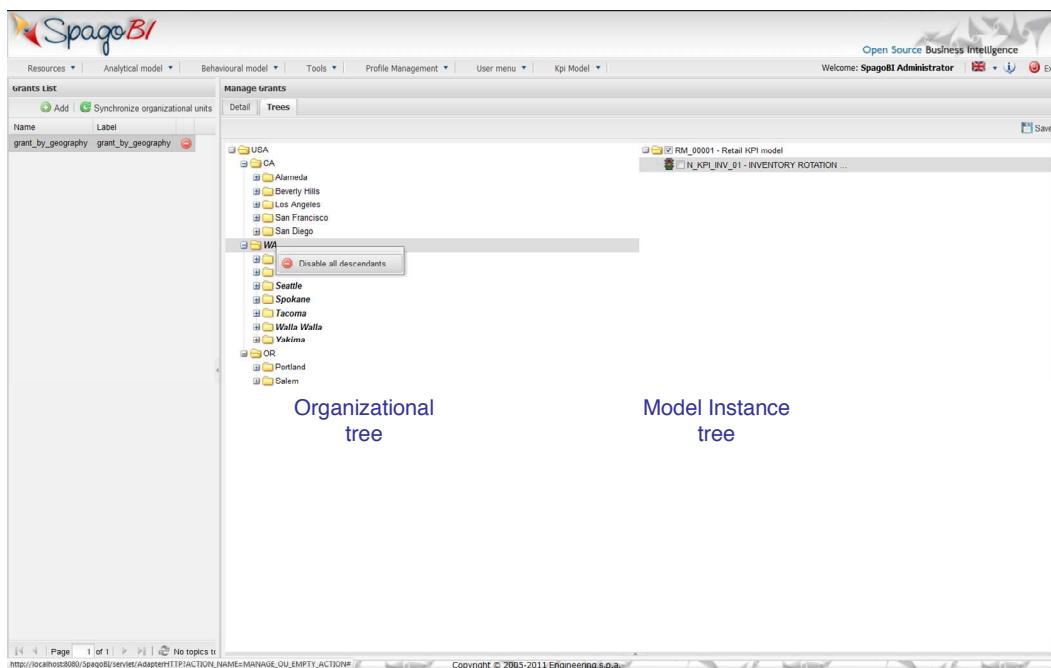


FIGURE 6.51 - Associate organizational units to KPI execution

KPIs calculated on a granted model require the addition of special dataset parameters:

- ParKpiHierarchy
- ParKpiOu.

In addition, grants require to set the `use_ou` property in the document template. Other useful parameters are `dateIntervalFrom` and `dateIntervalTo` that

constrain the dataset result on a time interval. All these parameters are described in TABLE 6.15 and TABLE 6.16.

Goals definition

Goals represent a functional extension of the KPI model and their definition does not impact KPI calculation nor document execution.

A goal consists of a set of qualitative objectives that are associated to a KPI model, together with thresholds and weights defining the extent to which each node contributes to goal reaching. Each objective is assigned to a KPI node and may be specialized in secondary objectives.

To model a new goal, select **KPI Model > Goal definition**. On the left you will see the list of defined goals, on the right the detail configuration editor. You can add a new goal clicking on . On the right you will be shown an empty editor, where you should define:

- a label, a name and a description for the grant
- the grant to which this goal is associated.

In the Goal Definition tab, you define the structure of the goal, with respect to its foundational elements: organizational units (on the left tree), KPI model instance (at the bottom) and goal hierarchy itself.

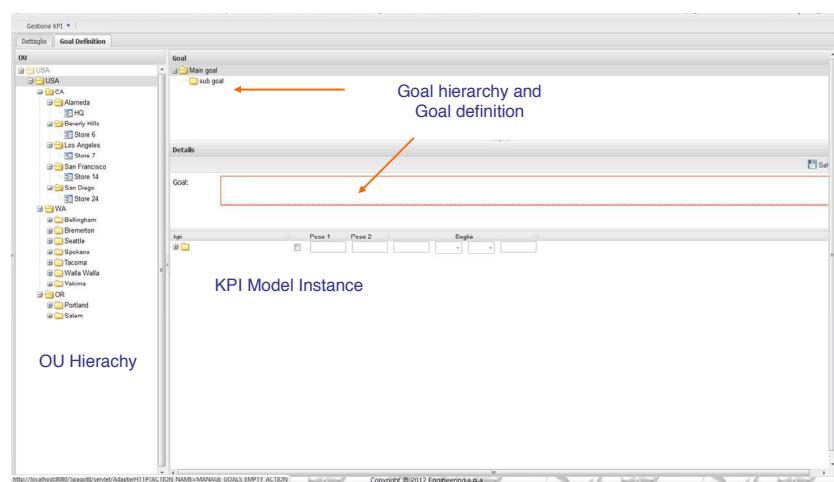


FIGURE 6.52 - Goal model definition

Calculate a KPI model

The KPI Engine can perform its calculation according to different logics, which can be configured via system variables. Their value is determined by their semantic, as described below, and will be assigned by the engine via the associated analytical drivers. The association with analytical drivers is explained in the next section.

In particular, the calculation logic is determined by the following parameters:

- **Behaviour.** This is the main parameter that drives KPI calculation logic.

The behaviour parameter may assume the following values.

- **Display.** The KPI engine will not calculate any new KPI values but it will only show the ones already available in the database. According to the required date parameter, the engine will show the most recent KPI value (KPI begin date \leq req. date).
- **Default.** For each KPI associated to the model instance, the KPI engine will check if at the requested date it is possible to find an already calculated value (KPI begin date \leq req. date $<$ KPI end date). If no valid value exists, a new value will be calculated at the requested date. If it finds more than one value, it will act like in the Display behaviour and it will show the most recent one from the requested date (KPI begin date \leq req. date).
- **Recalculate.** The behaviour is the same as the Default one, except that all the KPIs that do not have a defined periodicity attribute will be recalculated, even if a valid value already exists.
- **Force Calculation.** All KPIs are recalculated at the requested date, even if valid values already exist.

Additional variables that may influence calculation are related to the functionalities we discussed above: resources and grants, as well as time. These parameters are available for usage in datasets defining KPI calculation: using them will make the KPI parametric with respect to the given dimension:

- **ParKpiDate.** It represents at any time the current date of execution of the KPI. It can be inserted in the dataset used for KPI calculation to make KPI execution parametric with respect to time.
- **ParKpiResource.** It represents an attribute of the analysis model that can be used to aggregate KPI results. It can be inserted in the dataset used for KPI calculation to make KPI execution parametric with respect to a dimension.
- **ParKpiHierarchy.** It represents the hierarchy used to assign grants. It can be inserted in the dataset used for KPI calculation to make KPI execution parametric with respect to organizational hierarchies.
- **ParKpiOU.** It represents the node of a hierarchy used to assign grants. It can be inserted in the dataset used for KPI calculation to make KPI execution parametric with respect to nodes of organizational hierarchies.

If the KPI document is executed from the document browser or via scheduled execution (not in Display modality), the engine will calculate KPIs according to possible dependencies in the model. Values will be stored in dedicated repositories for later retrieval by the KPI analytical document. In other words, KPIs are calculated according to the logics set via system parameters and values are stored for subsequent visualization.

On the other hand, if KPI calculation is started online (i.e., not scheduled) values will be calculated at execution time and the report will be shown to the user immediately after execution. In other words, calculation logics defined by system variables will be ignored in this execution.

Create a KPI analytical document

Creating the KPI analytical document requires the following steps:

- Define a KPI model and instantiate it
- Define (if not already configured) analytical drivers
- Create the document.

When defining the model, you may add optional elements such as grants or thresholds, as we have seen until now.

In the previous section we showed how KPI calculation is driven by system parameters (e.g., behaviour, ParKpiResource, etc). These parameters must be associated to the KPI document and assigned a value at document execution time, like any other parameter in a SpagoBI document. This is done by linking the parameter to an analytical driver when associating it to the KPI document.



Analytical Drivers

Read how to associate analytical drivers to documents at section Register an analytical document, chapter 5 - SpagoBI Server.

The set of drivers shown in **Errore. L'origine riferimento non è stata trovata.** must be properly configured to make the KPI document working. If they are not already configured in the installation, they should be added via the Behavioural Model.



The behavioral model

The behavioral model rules visibility and rights on all analytical documents. For a full understanding of its meaning and functionalities, please refer to the corresponding section at chapter 5 - SpagoBI Server.

System parameter configuration for KPI documents				
System parameter	Description	Analytical Driver URL	LOV	Type
behaviour	This parameter allows end users to select the Behaviour type, for the computation rules, when they execute the KPI	behaviour	Fixed values : default display force_recalculation recalculate	Mandatory

	document. The behaviour defines when the computation engine has to update the KPI values.			
register values	This parameter allows the user to select whether the calculated KPI values are stored in the database or not. “true” is the default value.	register_values	Fixed values : true false	Optional
resource	This parameter allows users to select the resources on which they want to perform the analysis. One or more resources can be selected at the same time.	ParKpiResources	Query on SBI_RESOURCES tables	Mandatory on resource based calculation
Kpi date	This parameter allows end users to select the execution date. By selecting a date different from the current date, the KPI values will be presented at the moment of the selected Date.	ParKpiDate	n.a.	Optional
Hierarchy	This parameter allows end users to select an organisational	ParKpiHierarchy	Query on grant metadata tables	Mandatory on granted models

	hierarchy. A standard scenario uses these parameters to schedule massively KPI calculation based on the grant definition.			
Organisationa l Unit	This parameter allows end users to select a specific level inside the organization. A standard scenario uses these parameters to schedule massively KPI calculation based on the grant definition.	ParKpiOu	Query on	Mandatory on granted models

TABLE 6.16 - KPI system variables

Additional parameters (and the corresponding drivers) can be freely added to the document, as with any other SpagoBI document.

Once you have done this, you can create the document following the standard procedure:



Create analytical documents on SpagoBI Server

The process to create and register a document on the Server is described in detail at section Analytical Document, chapter 5 – SpagoBI Server.

Following the above described procedure, the creation of the document is automatic, with no need to manually edit the template. For finer grained configuration of the visualization document, you may also modify the KPI document template, as explained below.

KPI Document template

The template of a KPI document is an XML file. It sets the layout configuration of the visualization report and the link to the model instance.

An example of template is shown below. It corresponds to the report that will be shown in the next section, Viewing a KPI Model.

```
<?xml version="1.0" encoding="windows-1250"?>
<KPI model_node_instance='aaf505fd-6529-4b9a-bf54-92618061f8dd'
      name='Retail KPI Analysis' >
    <STYLE_SUBTITLE font='Arial' size='10' color='#6699FF'
      name='Reference date is $P{ParKpiDate}'/>
    <STYLE_TITLE font='Arial' size='12' color='#000000' />
    <CONF>
      <PARAMETER name='display_semaphore' value='true' />
      <PARAMETER name='display_bullet_chart' value='true' />
      <PARAMETER name='display_weight' value='true' />
      <PARAMETER name='display_alarm' value='true' />
      <PARAMETER name='use_ou' value='false' />
    </CONF>
</KPI>
```

Where:

- The `model_node_instance` attribute sets the association of the document with a model instance. The value must be the unique label automatically generated when the model was instantiated.

Some configuration attributes concern the report layout. Some others concern the KPI calculation modalities:

Attribute	Description
<code>display_semaphore</code>	When true, the KPI report displays a status icon representing the actual KPI performance
<code>display_bullet_chart</code>	When true, the KPI report displays a Bullet Chart showing thresholds, targets and actual values
<code>display_weight</code>	When true, the KPI report displays the KPI weight
<code>display_alarm</code>	When true, the KPI report displays an icon to view alarms

use_ou	This property is mandatory with a granted KPI model. When true, the KPI engine will take the grant specification for the calculation into account.
--------	--

TABLE 6.17 - Configuration parameter attributes

View a KPI model

Once the KPI document has been created, it can be executed in two ways:

- After a calculation in online modality or
- Executing the document with the behaviour parameter set to Display. In this case, values will not be re-calculated but retrieved from the last calculation.

The standard document is shown in the example of FIGURE 6.53. It consists of two sections:

- The KPI model instance tree, with synthetic values for each KPI (on the left)
- The navigator section, showing details about the selected KPI (on the right)

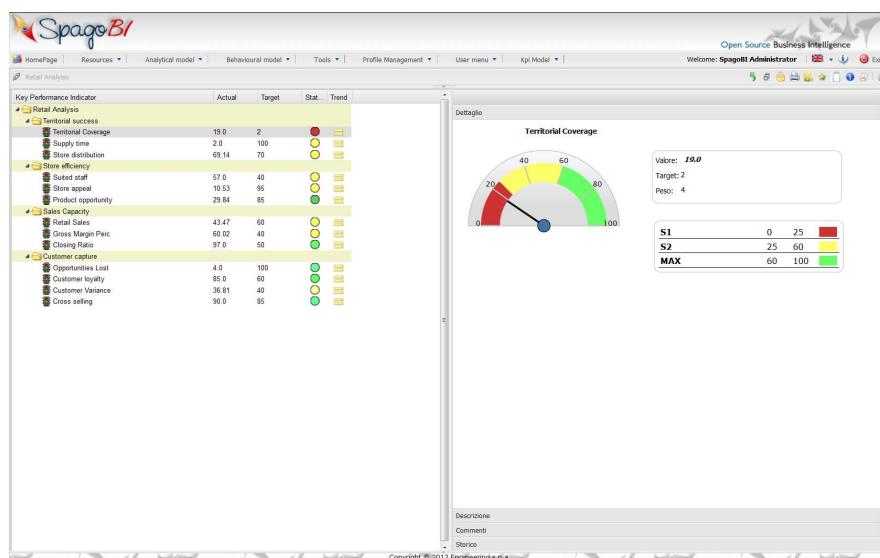


FIGURE 6.53 - KPI visualization report

The section on the left shows all nodes of the KPI model and the following values for each of them:

- **Actual.** Last calculated KPI value
- **Target.** Target value defined during model creation
- **Status.** Icon showing the status of the KPI based on its value
- **Trend.** Icon showing the trend of the KPI, given the actual and the historical values of the KPI.

The detail panel on the right is composed of four sub-sections:

- **KPI Detail.** A speedometer showing the KPI value with respect to defined thresholds, a legend for thresholds, the value, target and weight of the KPI.
- **Description.** This section shows the description of the KPI.
- **Notes.** Here the user can write comments, which will be associated to the specific instance.
- **History.** A chart showing the historical trend of the KPI over time. The interval can be varied using the toolbar.

Data Mining

Data Mining (DM) is the non trivial extraction of implicit, previously unknown and potentially useful knowledge from massive amount of data. The main characteristics of a data mining process are:

- the use of validated inductive algorithms
- to be dedicated to one-off studies
- the management of complex requests (segmentation, pattern identification, forecasts, associations, relationship networks, etc.)
- the usage of advanced algorithms (machine learning, neural networks, decision trees, fuzzy logics, etc.)

- to require the right choice and tuning of the algorithm
- to obtain results with reliability degrees
- to leave results open to interpretation of experts.

For these reasons, data mining is usually thought for seldom usage by very expert people or statisticians.

Basic concepts

The term DM does not simply refer to the functionalities allowing to re-organize data for research purposes (*information retrieval*) or analytical purposes (*data analysis – reporting, OLAP, etc.*). The key purpose of DM is to discover information implicitly available in huge amount of data (patterns, trends, relationships, etc.).

The term DM can refer to two different concepts: either the process leading from rough data to the analysis of the extracted knowledge; or the single task of this process leading to data extraction. In the following, the term DM is used according to this second meaning, while the term Knowledge Discovery in Database (KDD)²⁰ refers to the overall process.

²⁰ “From Data Mining to Knowledge Discovery in Databases” - Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth (<http://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1996-Fayyad.pdf>)

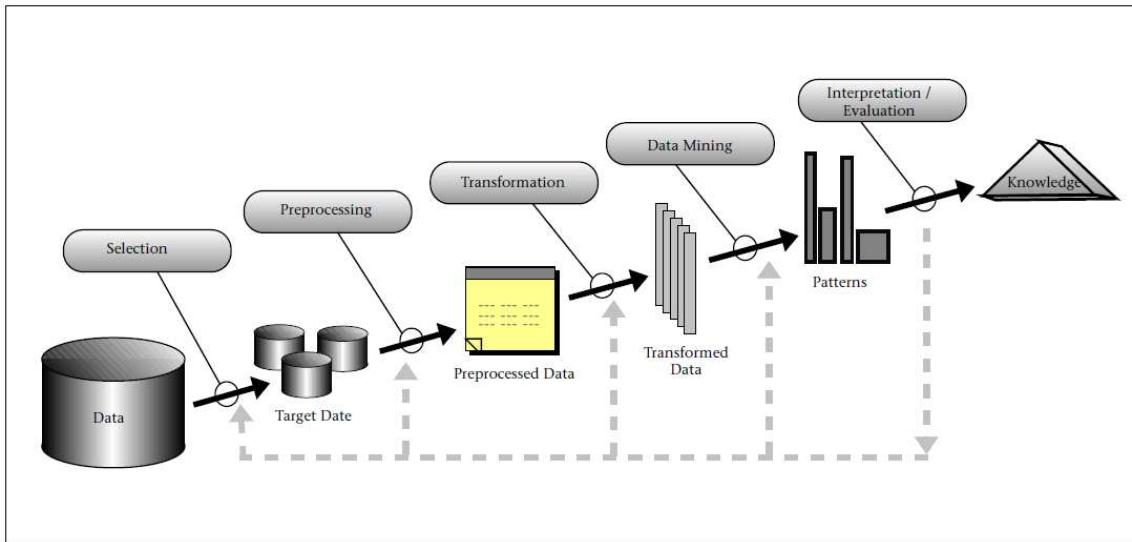


FIGURE 6.54 - Steps of a Knowledge Discovery in Database process

As shown in FIGURE 6.54, there are five basic stages in the KDD process:

- data selection and assessment
- pre-processing
- filtering and transformation
- data mining
- analysis and evaluation of the results.

The entire process is iterative, even within a single stage. Moreover, it is highly interactive, since it requires supervision by expert users, who also drive its evolution. The DM stage transforms consolidated, processed and filtered data into knowledge, using specific algorithms.

SpagoBI provides the SpagoBIWekaEngine, integrating the well-known Weka tool for data mining.

SpagoBIWekaEngine

SpagoBI includes a DM engine based on Weka. Weka is a DM algorithm library written in Java, developed by the University of Waikato in New Zealand and released under the GPL license. On top of this library, a set of tools has been built, which supports non-developing users with mining algorithms.



Weka

For a full overview of Weka data mining tool and a detailed developer guide, please refer to the official documentation at <http://www.cs.waikato.ac.nz/ml/weka/> or buy the official book: Ian H. Witten, Eibe Frank, Mark A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques* (3rd edition), Morgan Kaufmann, January 2011.

In particular, SpagoBI integrates the **Weka Knowledge Flow** tool. This tool allows the test and comparison of different data mining methods. It represents mining processes through a workflow, whose components represent the single operations to be executed on data (reading, pre-processing, filter, application of a mining method and drawing the results). The mining process designed using the Weka Knowledge Flow can be saved in a binary format or in an xml-based format called *kmfl*.

SpagoBI Weka-based engine is able to execute any mining process defined with the Weka Knowledge Flow, version 3.5. The steps to create a data mining document using the Weka engine are:

- Produce a data mining process using Weka Knowledge Flow and save it as a kmfl file
- Create the analytical document in SpagoBI and load the kmfl file as template
- In case parameters are needed, manually modify the kmf template before loading it. Then, create and associate the corresponding analytical drivers to the SpagoBI document.

In the next paragraphs, we will show how to produce the template using Weka Knowledge Flow, how to create the SpagoBI document. Section Advanced Functionalities, at the end of this section, provides details about parameterization of data mining documents.



SpagoBI support to Weka Knowledge Flow

SpagoBI does not support all elementary operators provided by the Weka Knowledge Flow for the creation of mining processes. The engine supports the reading and writing operators on database, all filters, the clusterization and association algorithms. On the other hand, no classification, evaluation and visualization algorithms are supported.

Installation and configuration of Weka

First of all, install Weka. SpagoBI current release supports Weka version 3.5. Weka can be easily installed, by launching the installer available in the archives, to be downloaded from the project forge (Weka can be launched both on Windows version²¹ and on Linux version²²).

Once the installation procedure is completed successfully, Weka can be launched executing the `RunWeka` command, located in the folder in which the application has been installed.

²¹ Weka 3.5 for Windows :

http://sourceforge.net/project/downloading.php?groupname=weka&filename=weka-3-5-5.exe&use_mirror=dfn

²² Weka 3.5 for Linux :

http://sourceforge.net/project/downloading.php?groupname=weka&filename=weka-3-5-5.zip&use_mirror=heanet

Weka's main window allows you to access the settings of the environment that welcomes the development tools (Weka CLI, Experimenter, Explorer and Knowledge Flow).

Before launching the Knowledge Flow, select **Help > SystemInfo** and check that the `java.class.path` system property includes the reference to the JDBC driver of the database. This allows Weka to read the input data before starting the process as well as to write the output data after completion of the process (in this example: MySQL).

Now select **Applications > KnowledgeFlow**: you are ready to start.

Creating the kmfl template

In the following we will show how the engine actually works by presenting an example of mining process. The final result will be the definition of a data mining process as a kmfl file, which will be used as the template for the SpagoBI analytical document.

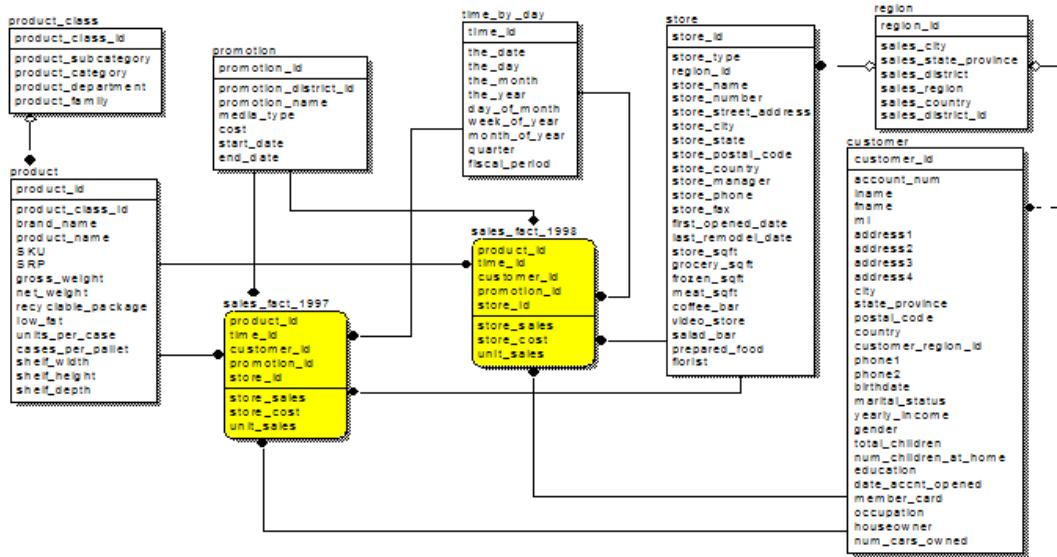


FIGURE 6.55 - DWH used in the KDD example with Weka

In our example, we aim at segmenting the set of users of a hypothetical supermarket chain through clusterization. Let us suppose to have already built a

DWH (see FIGURE 6.55) and that we do not have to perform the preliminary selection and consolidation stages of the KDD process.

Customers are segmented according to the number of purchases made in 1997 (`sales_fact_1997`). The results are stored in a dedicated customer-department relational table (`cust_to_dept_clustered`).

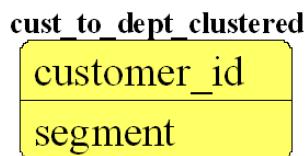


FIGURE 6.56 - Customer-department relational table

To perform this segmentation on the data of the `sales_fact_1997` table, you first have to pre-process data to be segmented. Pre-processed data can be used by the chosen clustering algorithm. The selection of the algorithm is then performed through a specific mining process, build through the Weka Knowledge Flow.

Pre-processing

During the pre-processing phase, you choose the customers' attributes on which the clustering operations should be performed. In fact customers are clustered according to their compliance with these attributes.

Considering that customers shall be grouped on the basis of their purchase habits, the attribute will be referred to the amount of purchases made by each customer in each department of the supermarket (e.g. deliverables, beverages, frozen foods, etc.). However, the `sales_fact_1997` table refers to the purchases of each single product, made by each customer in 1997.

As a result, the pre-processing stage aggregates this data according to the department in which the products have been purchased, in order to generate a new table, called `cust_to_dept_agg`, which includes the desired clusterization attributes.

SQL Query Area		
* SELECT * FROM cust_to_dept_agg c;		
customer_id	product_department	amount
3	Starchy Foods	2.7600
3	Baking Goods	4.2600
3	Household	5.3200
3	Produce	11.6100
3	Deli	11.7000
3	Beverages	12.4000
3	Frozen Foods	15.6700
3	Health and Hygiene	16.6100
3	Canned Foods	18.2100
5	Deli	1.0800
6	Canned Foods	5.4400
6	Baked Goods	6.9200
6	Periodicals	10.8300
6	Baking Goods	11.2000
6	Snacks	13.3200
6	Dairy	18.0500
10	Beverages	2.2000
10	Dairy	5.3200
10	Snacks	6.6600
10	Household	6.7800
10	Periodicals	8.6000
10	Snack Foods	8.7600
10	Canned Foods	15.5000
14	Produce	1.8400
14	Baking Goods	2.9200

FIGURE 6.57 - New table after pre-processing

Attribute selection is a key issue in clusterization. This process involves several important factors, which are even contrastive in some cases: attribute evenness, attribute statistical significance, attribute cardinality, etc.

Generally speaking, you can use any other attribute in the user segmentation process, such as the total amount of sales per product, in which case you do not need to perform the pre-processing phase. However, note that our choice allows us to limit the number of significant attributes to be clustered, which improves the response time.

Transformation

The transformation stage consists in re-organizing pre-processed data in such a way that they can be used by the chosen clustering algorithm.

During this phase, the developer acts not only on the data structure but also on their content. In other words, no data is modified, added or deleted. Specifically, a new `cust_to_dept` table shall be created, pivoting data included in the recently created `cust_to_dept_agg` table according to the `product_department` column. The new table will consequently include various columns, one for each value of the `product_department` column of the `cust_to_dept_agg` table.

SQL Query Area											
1 SELECT * FROM cust_to_dept c;											
customer_id	Periodicals	Breakfast_Foods	Eggs	Alcoholic_Beve...	Household	Beverages	Frozen_Foods	Dairy	Health_and_Hy...	Seafood	Baked_
2508	0.00000	0.00000	0.00000	0.00000	7.12000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1701	0.00000	8.22000	0.00000	0.00000	12.10000	0.00000	6.72000	0.00000	0.00000	0.00000	0.00000
6459	0.00000	8.22000	0.00000	0.00000	12.10000	0.00000	22.88000	11.84000	10.93000	0.00000	0.00000
7165	0.00000	8.22000	0.00000	1.95000	8.92000	0.00000	22.88000	11.84000	10.93000	0.00000	0.00000
4210	0.00000	8.22000	0.00000	1.95000	8.92000	5.92000	22.88000	11.84000	10.93000	0.00000	0.00000
9300	0.00000	8.22000	0.00000	1.95000	17.09000	20.67000	7.36000	5.55000	10.93000	0.00000	0.00000
1575	0.00000	5.48000	0.00000	9.48000	14.40000	20.67000	9.44000	5.55000	10.93000	2.37000	2.37000
5193	0.00000	5.48000	0.00000	9.48000	14.40000	20.67000	9.44000	5.55000	10.93000	2.37000	2.37000
10190	0.00000	5.48000	0.00000	9.48000	14.40000	14.25000	3.72000	5.55000	4.84000	2.37000	2.37000
7962	0.00000	5.48000	0.00000	9.48000	14.40000	14.25000	4.32000	5.55000	0.67000	2.37000	2.37000
1839	1.80000	7.05000	2.25000	22.69000	31.92000	2.80000	8.89000	18.78000	0.67000	2.37000	2.37000
4514	1.80000	7.05000	2.25000	22.69000	7.69000	2.80000	10.25000	18.78000	2.60000	2.37000	2.37000
8787	1.80000	7.05000	2.25000	22.69000	7.29000	2.80000	6.70000	5.52000	3.63000	2.37000	2.37000
3638	1.80000	7.05000	2.25000	16.50000	39.56000	5.16000	14.42000	27.75000	13.62000	2.37000	2.37000
9890	1.80000	7.05000	2.25000	16.50000	39.56000	5.16000	14.42000	9.24000	13.62000	2.37000	2.37000
9787	8.10000	7.05000	6.87000	13.29000	26.78000	9.12000	8.98000	2.88000	17.33000	2.37000	2.37000
5847	8.10000	7.05000	6.87000	13.29000	9.50000	4.75000	5.48000	3.58000	20.04000	2.37000	2.37000
3944	8.10000	7.05000	6.87000	13.29000	5.87000	2.47000	3.66000	3.56000	20.04000	2.37000	2.37000
500	8.10000	7.05000	6.87000	13.29000	4.41000	2.47000	3.66000	3.56000	20.04000	2.37000	2.37000
9421	8.10000	7.05000	6.87000	13.29000	4.41000	1.66000	3.66000	2.50000	2.28000	2.37000	2.37000
6870	8.10000	10.41000	6.87000	13.29000	4.41000	1.66000	3.66000	2.50000	8.61000	2.37000	2.37000
623	8.10000	10.41000	6.87000	13.29000	4.41000	1.66000	3.66000	14.64000	8.61000	2.37000	2.37000
602	8.10000	10.41000	6.57000	2.22000	13.10000	4.38000	16.72000	1.14000	9.96000	2.37000	2.37000
4824	10.80000	10.41000	6.78000	2.22000	52.14000	25.20000	57.45000	21.73000	17.17000	2.37000	2.37000

FIGURE 6.58 - New table after transformation

This transformation is needed since in these clustering algorithms the similarity among clients is calculated in terms of distance from one client to another in a multidimensional space.

Knowledge flow

Once you have completed the preliminary stages of the KDD process, you can go ahead with the definition and configuration of the mining algorithm, in order to extract knowledge from the input data. As previously mentioned, the extracted knowledge allows you to group users into new classes, according to their purchase habits registered in 1997. To this end, we will use clustering techniques.

The whole DM step is defined through the Weka Knowledge Flow tool. In particular, we will create a mining process specifying how the acquisition of the input data shall be performed, which clustering algorithm shall be applied and, finally, how the output data shall be managed.

Now we can launch the Knowledge Flow and start drawing the mining process. First of all, add the **DatabaseLoader** operator to the process, by selecting the **DataSources** panel.

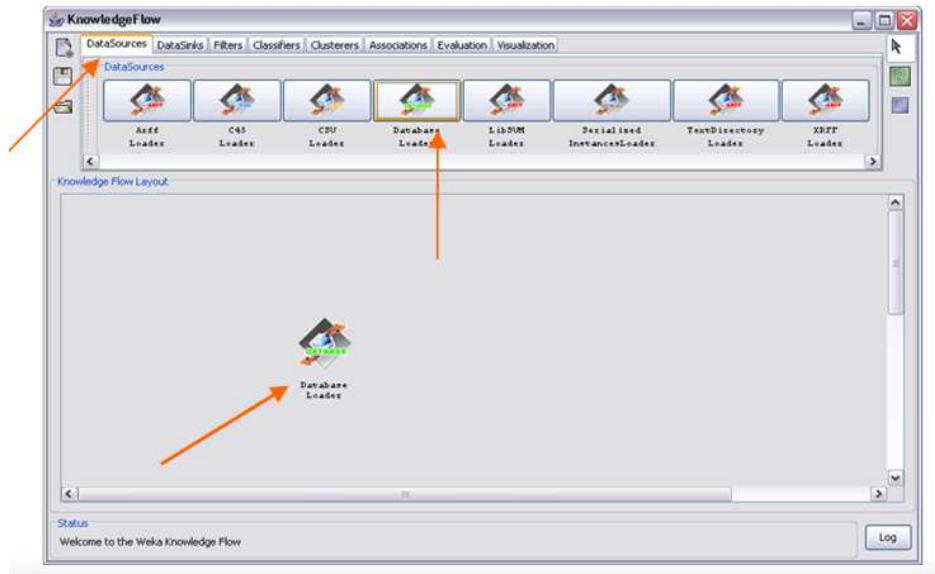


FIGURE 6.59 - Adding the database loader operator

Set this operator so that it can read the content of the `cust_to_dept` table, as shown in FIGURE 6.60.

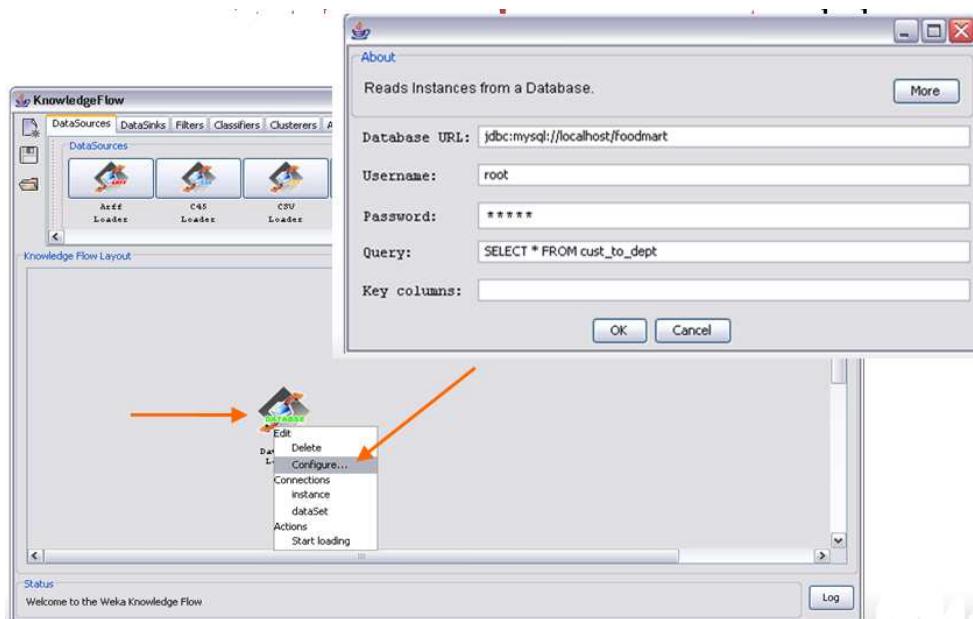


FIGURE 6.60 – Configuration of the database loader

Add the **DatabaseSaver** operator to the process by selecting it in the **DataSinks** panel. Configure this operator, so that it can write the output data of the process into the `cust_to_dept_clustered` table, as shown in FIGURE 6.61.

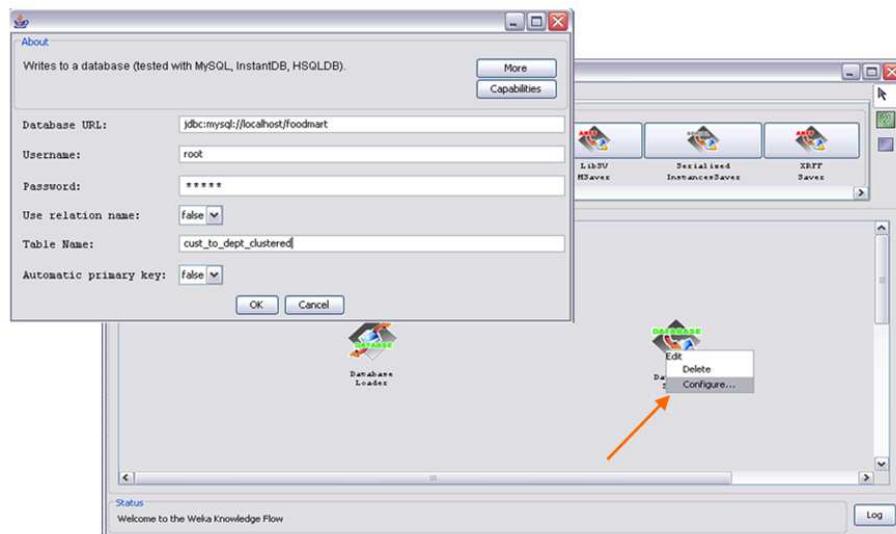


FIGURE 6.61 – Configuration of the database saver operator

Now you can define the clustering algorithm to be applied on the input data, which the **DatabaseLoader** operator reads from the database. This will allow the generation of the output data, to be written by the **DatabaseSaver** operator into the database. Add the **AddCluster** operator to this process, by selecting it in the **Filters** panel.

To configure the operator, choose the `simpleKMeans`²³ clustering algorithm. Specify that the first column of the input dataset shall be ignored by the clustering process, since it is not a descriptive attribute of the client object, but an identifying one.

²³ http://en.wikipedia.org/wiki/K-means_clustering

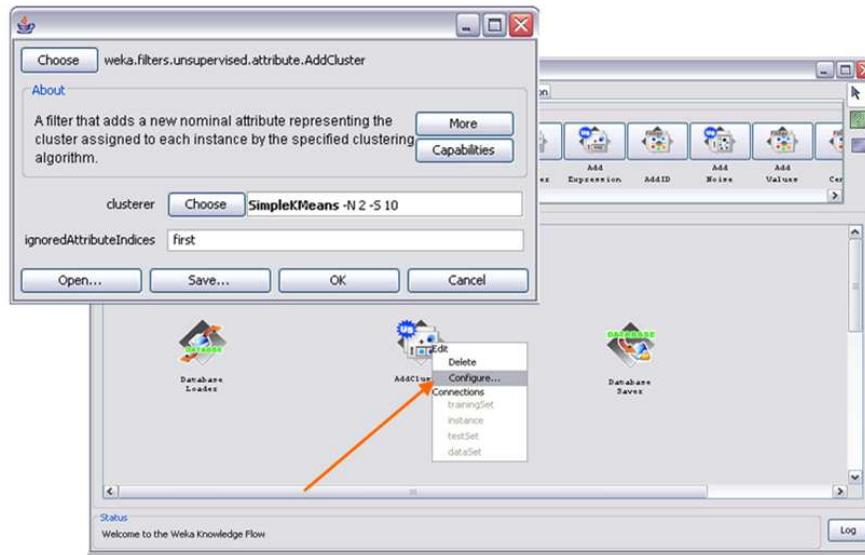


FIGURE 6.62 - Configuration of the addcluster operator

The clustering operator produces an output dataset which, compared to the input dataset, includes an additional column which defines records and related clusters. Before writing the output onto the database, you have to filter it, so as to remove all columns, except the first one and the last one. In other words, remove all columns that describe the customer (attributes).

To perform this action, use the **Remove** operator, by selecting it in the **Filters** panel. This is the consequence of the fact that we are dealing with an output table that is a relational one, including just customer IDs and their related clusters. Remove columns as shown in FIGURE 6.63.



FIGURE 6.63 - Remove unnecessary columns

To complete the process definition, all operators located on the left shall be connected to those located on the right via a dataset connection. The result of the entire process is shown in FIGURE 6.64.

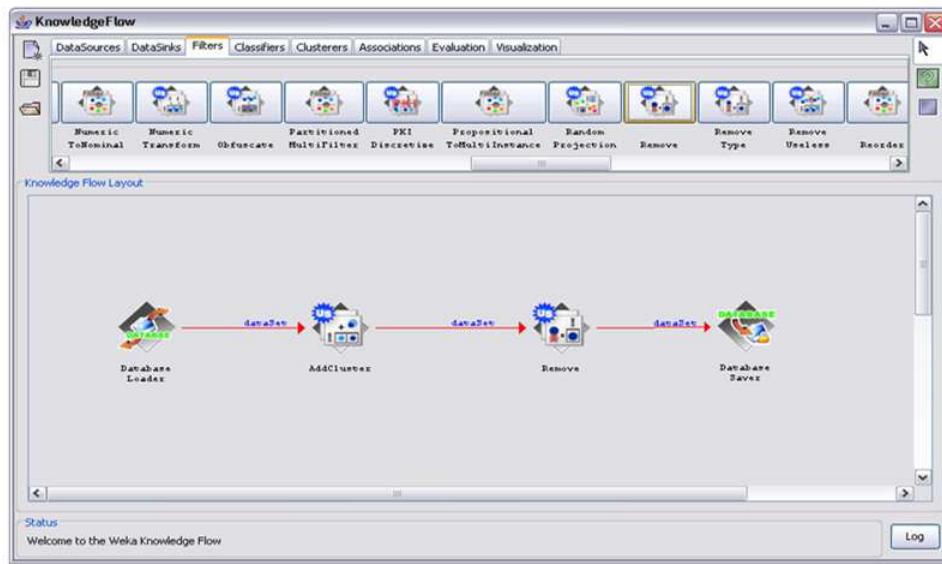


FIGURE 6.64 - Data mining process once completed

Save the defined process as a kmfl file. We will focus on it in the next paragraph, where it is used as the template of the analytical document to be registered into SpagoBI.

Create the analytical document

Once we have created the template, we can create a new analytical document.

Before starting to create the document, it is recommended to check whether the engine is properly installed and configured.

In case the engine is not visible in the Engine Configuration list (**Resources > Engine Management**), you should check that the web application is active by invoking the following URL:

```
http://myhost:myport/SpagoBIWekaEngine
```

If the application is working properly, you should see the following page:

SpagoBI Weka Engine

Menu

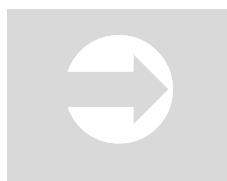
- [API doc](#)
- [Performance monitor \(JAMon\)](#)

External links:

- [SpagoBI's Home Page](#)
- [Wiki](#)
- [Forum](#)
- [Bug Tracker](#)

FIGURE 6.65 – Access page to SpagoBI Weka engine

If the operation succeeds you can go back to the Engine Management page and configure the engine.



Engine management

Please refer to chapter 5 – SpagoBI Server, section Cross Services- Engine management to check configuration options for SpagoBI Weka Engine

Now you can create the analytical document on Server, following the standard procedure. The template for this document is the **kfm1** file, generated after saving the mining process (see the previous paragraph).



Create analytical documents on SpagoBI Server

The process to create and register a document on the Server is described in detail at chapter 5 – SpagoBI Server, section Analytical Model.

```

1 customer_segmentation.kfml
2 <?xml version="1.0" encoding="utf-8"?>
3 - <!DOCTYPE object
4 [
5   <!ELEMENT object (#PCDATA|object)*>
6   <!ATTLIST object name CDATA #REQUIRED>
7   <!ATTLIST object class CDATA #REQUIRED>
8   <!ATTLIST object primitive CDATA "no">
9   <!ATTLIST object array CDATA "no"> <!-- the dimensions of the array; no=0, yes=1 -->
10  <!ATTLIST object null CDATA "no">
11  <!ATTLIST object version CDATA "3.6.1">
12 ]
13 >
14
15 <object class="java.util.Vector" name="__root__" version="3.6.1">
16 - <object class="java.util.Vector" name="0">
17 -   <object class="weka.gui.beans.BeanInstance" name="0">
18     <object class="int" name="id" primitive="yes">0</object>
19     <object class="int" name="x" primitive="yes">160</object>
20     <object class="int" name="y" primitive="yes">105</object>
21     <object class="java.lang.String" name="custom_name">DatabaseLoader</object>
22   <object class="weka.gui.beans.Loader" name="bean">
23     <object class="weka.core.converters.DatabaseLoader" name="loader">
24       <object array="yes" class="java.lang.String" name="options">
25         <object class="java.lang.String" name="0">url</object>
26         <object class="java.lang.String" name="1">jdbc:mysql://localhost/foodmart</object>
27         <object class="java.lang.String" name="2">user</object>
28         <object class="java.lang.String" name="3">root</object>
29         <object class="java.lang.String" name="4">password</object>
30         <object class="java.lang.String" name="5">admin</object>

```

FIGURE 6.66 – Creating an analytical document using the kfml file

The result of this operation is the new document shown in FIGURE 6.67.



FIGURE 6.67 – Analytical document correctly created.

Since the document execution is asynchronous, this notification only informs users on the correct start-up of the data mining process. On the other hand, the completion of this process is notified through the Event Monitor (User menu > Events).

Events list			
Event id	Event date	User	Event description
100	14-03-2007	staff_gen	Started execution of Weka data mining process. Exe...
101	14-03-2007	staff_gen	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"/> <input type="button" value="starts with"/> <input type="button" value="Filter All"/>			

FIGURE 6.68 – Event Monitor

Advanced functionalities

The following paragraphs aim to give you an overview on some advanced functionalities provided by the WEKA engine, such as the visualization of results, the parameterization of processes and the management of the results versioning.

Visualization

Each WEKA document can be associated to other documents deployed in SpagoBI. These documents, usually employed to analyze the results of the mining process, are associated to the WEKA document through the **Link** button located at the top right of the document detail page.

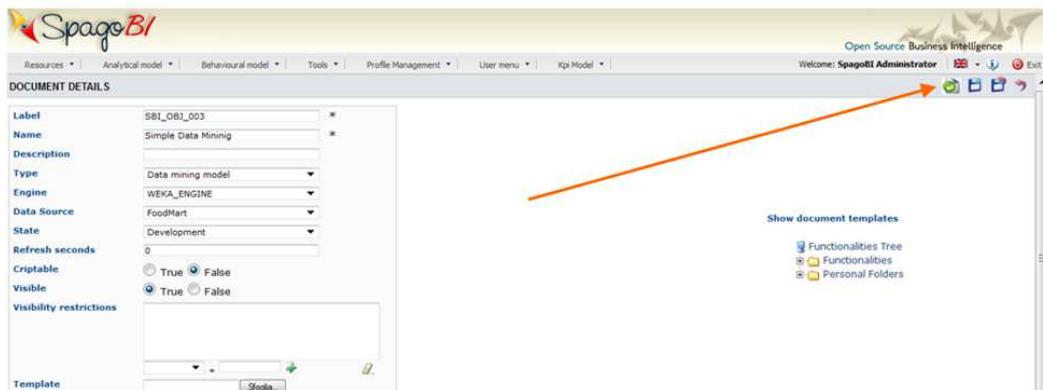


FIGURE 6.69 – Associating the WEKA document to other documents

Each type of document deployed in the platform can be freely associated to a WEKA document. In our specific case, we want to associate two reports that show the structure of the clusters generated by the algorithm.

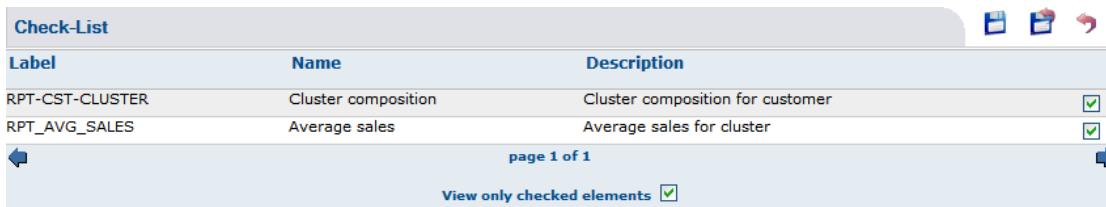


FIGURE 6.70 – Selecting the documents to be associated to the WEKA document

Given a specific department, the *Cluster composition* report shows the amount of the purchases made by customers, which belong to different clusters. This way, you can identify the clusters of those customers who most contributed to the sales of the pre-defined department. This information can be valuable in various contexts, such as to manage focused marketing campaigns.

Alcoholic Beverages				
(version: v-1.1)				
Big Spenders				
Num.	Cluster	Amount	Perc.	Agg. Perc.
1.	cluster1	981,46 €	7,00 %	7,00 %
2.	cluster3	956,87 €	6,82 %	13,82 %
3.	cluster17	910,19 €	6,49 %	20,30 %
4.	cluster5	835,25 €	5,95 %	26,26 %
5.	cluster18	772,57 €	5,51 %	31,77 %
6.	cluster7	744,67 €	5,31 %	37,07 %
7.	cluster4	714,18 €	5,09 %	42,16 %
Others				
Num.	Cluster	Amount	Perc.	Agg. Perc.
8.	cluster21	668,26 €	4,76 %	46,93 %
9.	cluster19	658,04 €	4,69 %	51,62 %
10.	cluster22	573,72 €	4,09 %	55,71 %
11.	cluster24	555,03 €	3,96 %	59,66 %
12.	cluster8	532,57 €	3,80 %	63,46 %
13.	cluster20	514,49 €	3,67 %	67,13 %
14.	cluster11	447,00 €	3,19 %	70,31 %
15.	cluster23	446,79 €	3,18 %	73,50 %
16.	cluster9	418,77 €	2,99 %	76,48 %
17.	cluster10	413,24 €	2,95 %	79,43 %
18.	cluster14	407,81 €	2,91 %	82,34 %
19.	cluster16	406,27 €	2,90 %	85,23 %
20.	cluster15	399,42 €	2,85 %	88,08 %
21.	cluster6	355,75 €	2,54 %	90,61 %
22.	cluster13	346,49 €	2,47 %	93,08 %
23.	cluster12	340,72 €	2,43 %	95,51 %

FIGURE 6.71 – Cluster composition report

On the other hand, given a cluster, the *Average Sales* report shows the average annual expenses incurred by the customers belonging to this cluster. This report shows the internal structure of the generated clusters. Moreover, it allows to define the purchase habits of each cluster. This information may be used to organize promotional campaigns or cross-selling activities.

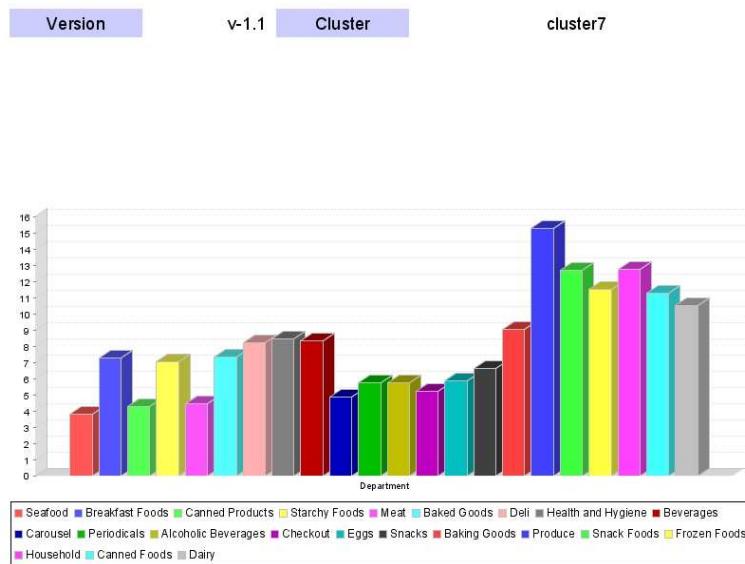


FIGURE 6.72 – Average sales report

Analysis associated to the WEKA document will be shown in the detail panel of the process termination event. This event is generated once the process has terminated and notified in the **User Menu > Events** menu item.

Parameterization

The Weka Knowledge Flow does not allow the parameterization of the flow. In other words, developers cannot set the various operators using the same parameters, instead of static values, which will be populated by the user at the document execution time.

SpagoBI exploits the XML-based coding of the kmfl template to manage the application parameterization. In particular, the developer should modify the template produced by Weka Knowledge Flow by manually adding parameters, using the following syntax:

`$P{ParameterName}`

The engine will replace all parameters defined as above with the value of the `ParameterName` input parameter, before parsing the template.

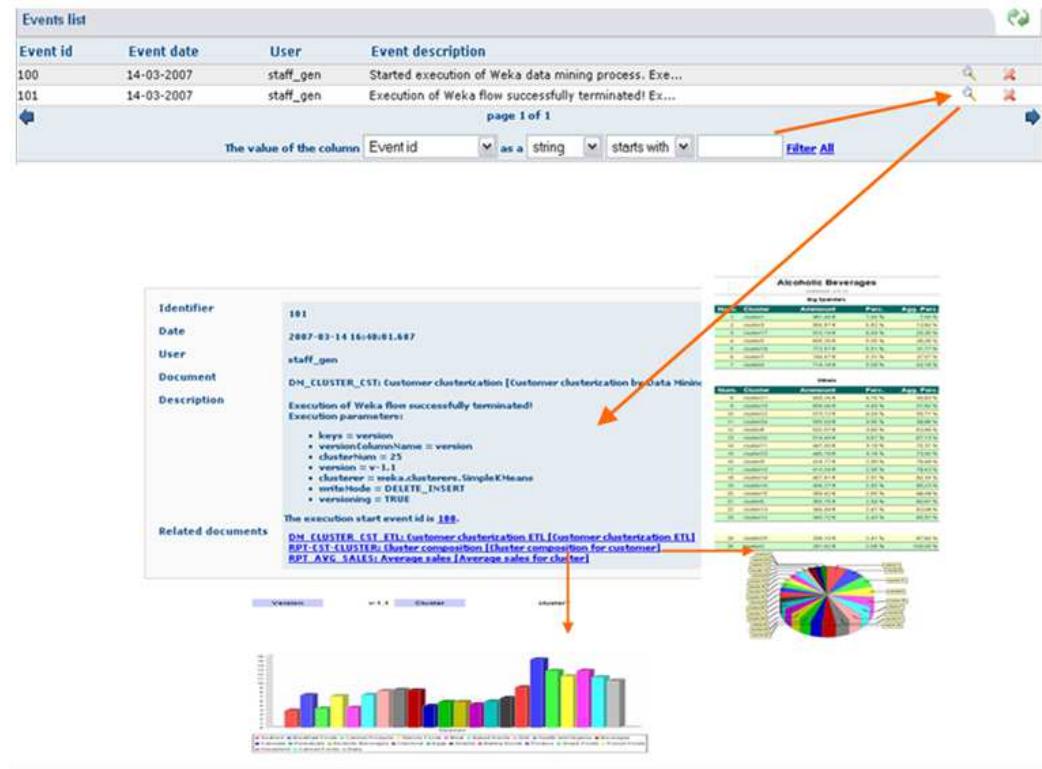
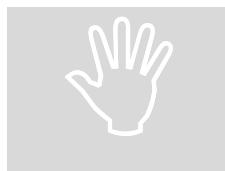


FIGURE 6.73 – Results of the document association



Editing the kmlf file

Weka Knowledge Flow does not natively support the parameterization of the process. Therefore, once the `kmlf` file has been modified to support parameterization in SpagoBI, it cannot be used in the Knowledge Flow again. We suggest to parameterize on a copy of the original `kmlf` file, before deploying it into SpagoBI.

The parameterization technique described above can also be used to allow the selection of the following elements at document execution time:

- the clustering algorithm
- the number of clusters to be generated.

To this end, you should modify the part of the template shown below:

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

```
</object>
```

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

```
</object>
```

Once the template is modified and re-deployed, you have to properly define two new analytical drivers and associate them to the WEKA document.

- The first one (**clusterer**) includes the selectable cluster algorithms.

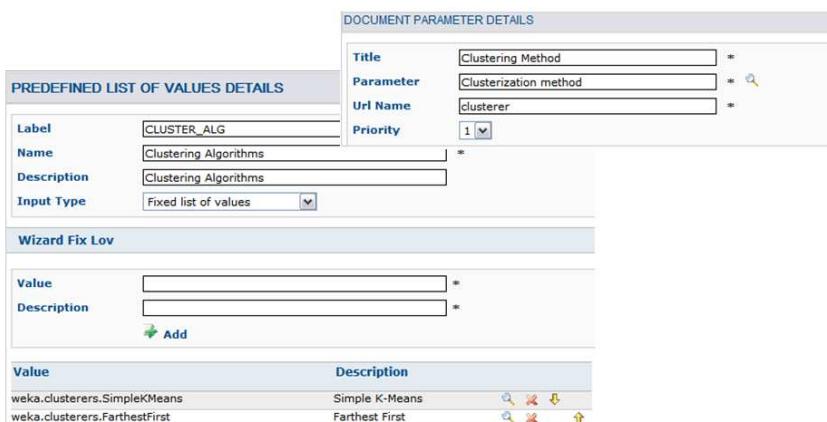


FIGURE 6.74 – Clusterer analytical driver

- The second one (`clusterNum`) includes the admissible values, defining the number of clusters to be generated.

The screenshot shows the 'DOCUMENT PARAMETER DETAILS' screen. At the top, there's a section for 'PREDEFINED LIST OF VALUES DETAILS' with fields for 'Label' (NUM_CLUSTER_FIX), 'Name' (Number of cluster), 'Description' (Number of cluster), and 'Input Type' (Fixed list of values). Below this is a 'Wizard Fix Lov' section with 'Value' and 'Description' fields, and a 'Add' button. At the bottom is a table of values:

Value	Description
10	10
15	15
20	20
25	25
30	30

FIGURE 6.75 – ClusterNum analytical driver

Versioning

The KDD process is a highly iterative process. The WEKA document deployed in SpagoBI can be executed several times, in order to find out the most effective result. The significance of this result is evaluated by the user through the analytical documents associated to the WEKA document.

This typically generates iterations between the mining stage and the evaluation of the KDD process. In SpagoBI, these iterations correspond to the repeated execution of the following processes: parameterization, execution, evaluation of the results.

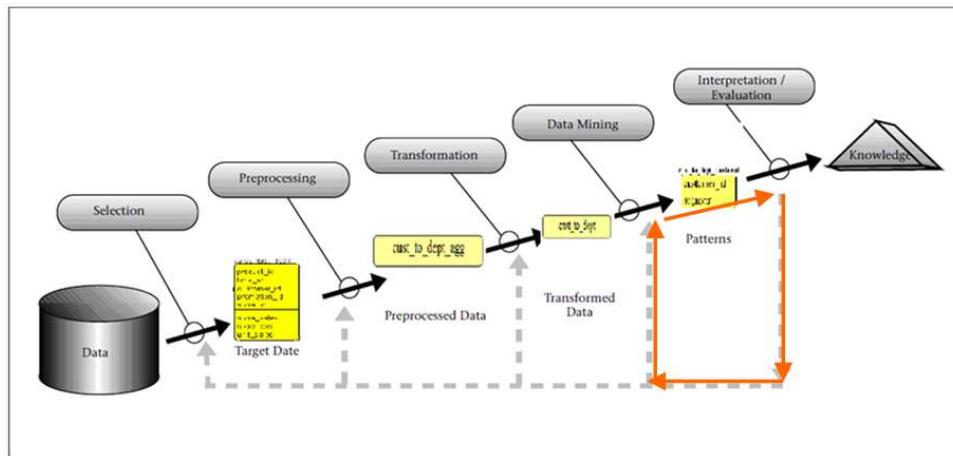


FIGURE 6.76 – KDD process in data mining

The WEKA engine allows you to manage the versioning of the results once the document is executed, through the analytical drivers described below.

Url name	Values	Description
versioning	TRUE or FALSE (default FALSE)	It specifies whether the versioning of the results shall be performed or not.
writeMode	DROP_INSERT DELETE_INSERT INSERT UPDATE_INSERT (default DELETE_INSERT)	It specifies which writing mode shall be used to insert the new results into the database.
versionColumnName	Any string, provided that it corresponds to the name of one of the columns included in the output table.	It is the column of the output table to be used to store the version number of a result. As for our example, we can add a column called “version” to the table, which specifies that a certain customer-cluster relation corresponds to a specific version of the document execution.

version	Any string, usually freely specified by the user.	It is the name or the number of the version associated to the current execution of the document.
keys	A list of strings, separated by a comma. Each string shall correspond to the name of a column included in the output table.	It specifies which columns are the key of the output column. This parameter is needed when the writeMode is set as UPDATE_INSERT.

TABLE 6.18 - Analytical drivers for Weka document versioning

Free Inquiry

Free inquiry empowers non-technical users with easy and free access to information via graphical interfaces. The main characteristics of a tool in this area are:

- it has a rich end-user GUI
- it allows to select attributes and set filters
- it does not require any knowledge of data structures
- it requires a semantic knowledge of data
- it is useful every time the free inquiry on data is more important than their graphical layout
- it leaves the management of results free
- it supports export capabilities
- it allows the repeatable execution of inquiries
- it works on a data domain with limitations.

Free inquiry is usually widely used, but mainly by technical people, analysts and operational users that are looking for a list of results that is not pre-arranged. It has a medium level of difficulty.

SpagoBI offers two solutions for Free Inquiry:

- QbE - Query by Example engine (SpagoBIQbeEngine), where users can define their own query through an entirely graphical modality. Moreover, they can execute the query, check the results, export them and save the query for further use
- Smart Filter (SpagoBISmartFilterEngine), to create simpler inquiry forms, whose data domain is predefined. The SmartFilter uses the easier and more intuitive approach of selection criteria, instead of representing a query.

SpagoBIQbeEngine

The QbE Engine allows users to query (a subset of) a database through a high level representation of the underlying entities and relations. Building a QbE query does not require any technical knowledge, but data domain knowledge: technical aspects, such as creating filters, aggregation and ordering criteria, are managed by a user-friendly graphical interface.

To create QbE queries you first need to define the document representing the underlying business model. Then queries can be defined and saved as customized views on the model. Once saved, QbE queries can be used by themselves, e.g., to execute queries on data, but also as base query for Smart Filter and Worksheet documents. This makes the QbE a truly central engine for SpagoBI.



Smart Filter

The next section of the current paragraph, Free Inquiry, is dedicated to the Smart Filter Engine.



Worksheet Engine

The Worksheet Engine allows the ad-hoc creation of analytical documents via a user-friendly graphical interface. See section Ad-hoc Reporting, in this chapter.

The creation of a QbE document and query involves the following steps:

- Datamart generation
- Template building
- Analytical document building
- Query design and execution.

The first three steps can be performed either manually or automatically, relying on SpagoBI functionalities. In the following we discuss each step in detail, showing basic and advanced functionalities of the QbE Engine.

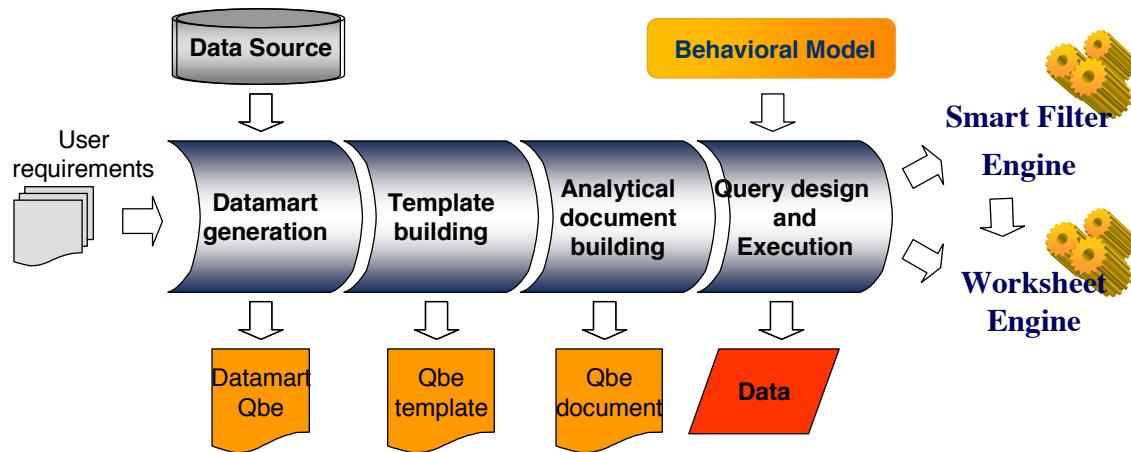
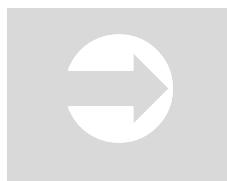


FIGURE 6.77 - QbE document lifecycle

Datamart generation

QbE queries are built over a high level representation of data, modelled in terms of end users' lexicon and concepts. Starting from 3.0 release, this datamart is automatically generated from the business model built and deployed with SpagoBI Meta.

When a new model is deployed from SpagoBI Meta onto the Server, the model is stored in a SpagoBI catalogue and saved as a QbE datamart. The QbE datamart will be visible as a document under the user's personal folder. Packages corresponding to the deployed metamodels are saved in subfolders of the /resources folder of SpagoBI Server.



SpagoBI Meta

SpagoBI Meta is SpagoBI module for technical metadata management and inquiry. Learn how to create and upload on Server a business model at chapter 3 - SpagoBI Meta.

To sum up, datamart generation via SpagoBI Meta is an automatic process that takes place whenever a metamodel is deployed from the Meta to the Server.

Template building

The template of a QbE document is automatically created when the corresponding metamodel is loaded from SpagoBI Meta (and the datamart is generated).

This initial template is the base version that sets the link between the QbE document and the datamart, but lacks of advanced functionalities, such as visibility and profiled access rules. These additional features can be added by hand afterwards, as we will discuss in the later section Advanced QbE functionalities.



Creating multiple datamarts

Because data profiling and visibility rules are added after datamart creation, it is possible to create several QbE datamarts derived from the base one, depending on functional or organizational criteria.

Analytical document building

This step needs to be performed only when the datamart has been manually created. This may happen, for example:

- When a new business model (datamart) has been created from an existing one
- When profiling rules have been added to an existing one
- When entities coming from different business models have been put together.

In those and similar cases, the procedure to build a QbE document is the same as for other SpagoBI documents.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at chapter 5 - SpagoBI Server, section Analytical Model.



Analytical Drivers

Read how to associate analytical drivers to documents at section Register an analytical document, chapter 5 - SpagoBI Server.

In all other cases, the document (QbE datamart) will be created automatically when you deploy a model from SpagoBI Meta. So there is no further need to create it: by default, you will find the QbE document under your personal folder.

Query design and execution

The QbE analytical model we just created (either automatically or manually) represents the model that can be queried - not the query itself. Queries can only be created and saved via the graphical editor.

In this paragraph we will show how to build a simple query with the QbE editor. Open the QbE document by double clicking on it, either from the document browser or from the menu **Analytical Model > Documents Development**. If the document requires parameters, fill the values and execute the document. The query editor will open as the result of the QbE document execution.

As shown in FIGURE 6.78, the window has four tabs:

- **Query designer**
- **Results area**
- **Worksheet designer**
- **Worksheet preview.**

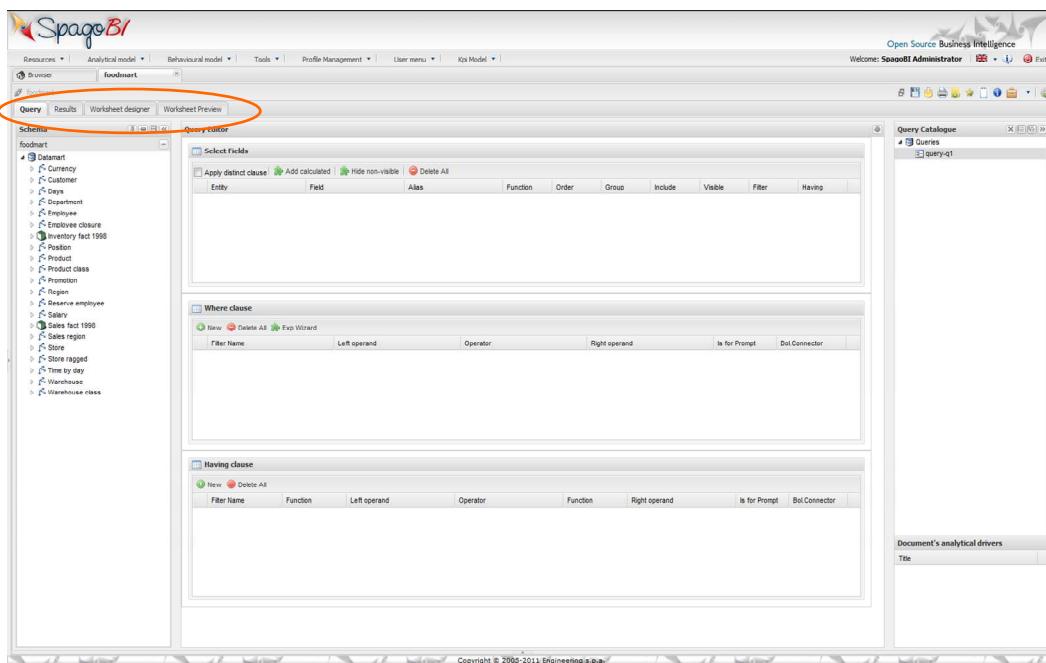


FIGURE 6.78 - QbE user interface

Here we focus on the first two tabs only: worksheet functionalities will be explained in the corresponding section.

The steps needed to create a query can be summarized as follows:

- Execute the QbE datamart document that you wish to query. If it requires parameters, assign them a value and execute.
- Once the QbE editor is open, drag attributes from the left panel onto the query editor area and build the query. Details on query creation are provided in the next section.
- Save the query clicking on the  icon, located in the toolbar at the right top corner. Choose a name, a description and the visibility criterion within the application.
- Execute the query clicking on the small  icon, located at the right top corner of the **Query Editor** area. The **Results** tab will open showing the retrieved data. Here you can hide or rename columns.
- If you want to export the query on file, click on the  icon, select the type and the name of the export file, and the destination directory.

This is the basic guideline for building QbE queries. In the next sections we explain in detail the different options for query creation and configuration by describing the query editor.

The query designer includes three areas (shown in FIGURE 6.78):

- Datamart schema (on the left)
- Query editor (center area)
- Query and parameters catalog (on the right).

We will start from the Schema section, then moving to Query and finally focusing on Catalogs. By describing all functionalities of each panel, we show how a QbE complex query can be created and managed.

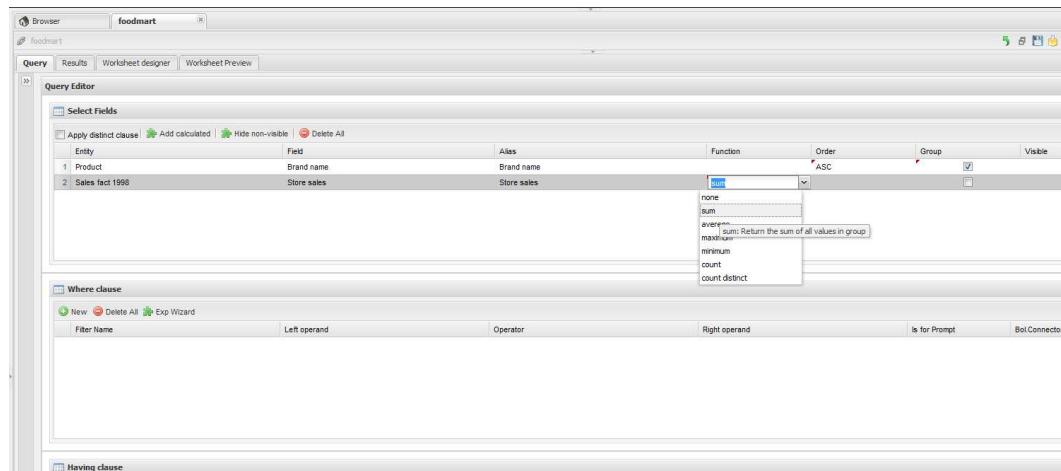


FIGURE 6.79 - Query ordering and grouping

Datamart Schema

The left panel shows the business model that serves as the datamart for the query. Entities and relationships are represented in a tree structure, with user-defined names. As said above, fields can be dragged from here and dropped onto the editor area to insert them into a query.

The left panel has a small toolbar allowing to configure the panel (e.g., expand, reduce) and to save changes made to the model.

Clicking on an item in the tree, the contextual menu will open to show additional functionalities (see FIGURE 6.80)

- **Add to SELECT/WHERE/HAVING clause.** To add the selected field to the corresponding clause in the query editor area.
- **Add calculated field.** To add an attribute that can be obtained via simple expressions combining existing attributes. Clicking on the contextual menu item, the wizard will open. Here you can combine fields with arithmetic and date functions. When you create a calculated

field, you can add it to the model by clicking on the Save  button. In addition, they can be used in queries.

Calculated fields may also be managed by expert users via advanced functionalities, which we will discuss at the end of the section.

- **Edit field.** To rename a field.
- **Add/Edit Range.** To add or manage a range for values of the selected attribute (details are provided below).
- **Remove calculated field.** To remove a calculated field that was added before.

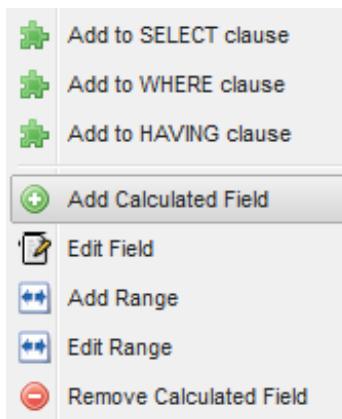


FIGURE 6.80 - Functionalities in the contextual menu of the datamart schema panel

Range management

It frequently happens that attributes of entities in a model have several different values. However, for the purpose of analyzing data, it is often more useful to group those values into categories.

For example, consider the age of customers: often analysis do not aim to know the precise age of customers, but they most probably aim to know if customers belong to a certain age range, e.g., young, adult and elderly. For this and similar cases, the QbE Engine is able to define and manage ranges in queries.

To create a new range for an attribute, click on **Add Range** in the contextual menu: the band creation wizard will open. As shown in FIGURE 6.81, you can

select fields, arithmetical and date functions to define the range. For example, click on the AA_up_today function, which returns the difference from a date and today; when prompted, set the customer's birth date as the function operand. The range is defined.

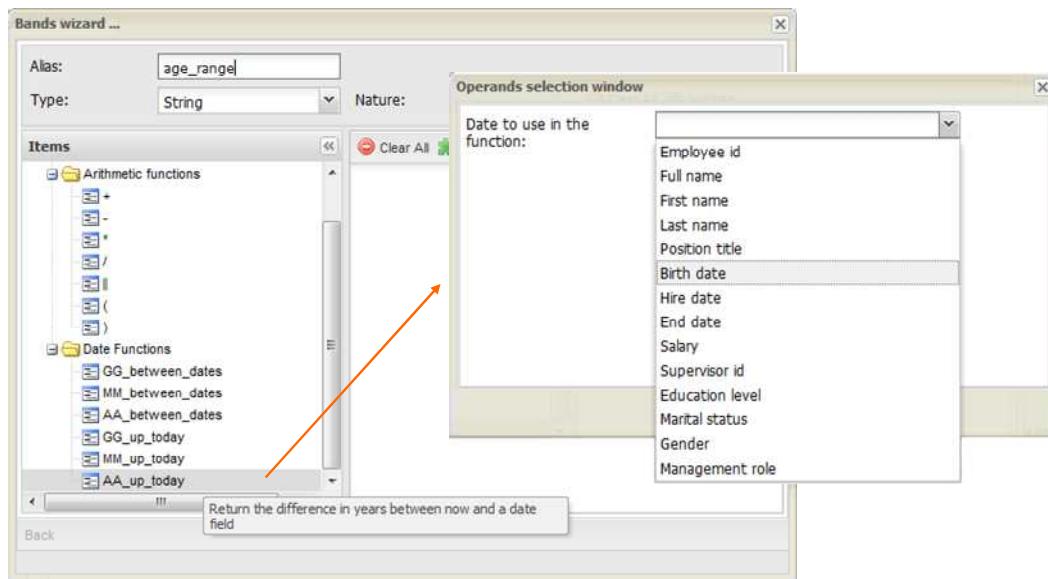


FIGURE 6.81 - Create a range for age with the band wizard

Clicking on the **Next** button at the bottom right corner of the wizard, you will be shown the window to create new range instances. Click on **Add Band** to add a new instance, set the corresponding values and labels. Then click on **Add Default** to insert a default range: this will create a new category called **Others**, which groups all values not falling into defined range intervals.

At this point, click on **Finish**. The range will appear as a node in the schema panel on the left. If you want to edit the range, click on **Edit Range**.

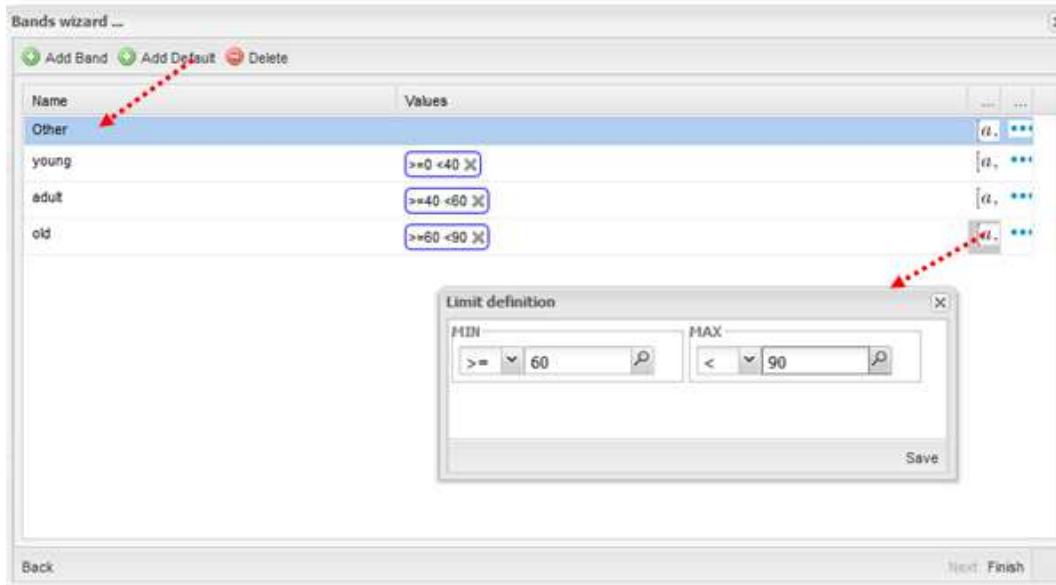


FIGURE 6.82 - Range instance creation

Query editor

The central panel is dedicated to query definition and includes three sub-panels:

- Select Fields
- Where clause (filter)
- Having clause (filter on groups).

Elements from the datamart schema can be dragged and dropped onto the query sub-sections. If a whole entity is selected, all its attributes will be dropped into the editor. Alternatively, elements can be added to the editor sub-sections using the contextual menu, as said before.

To remove an attribute from the query editor, select the left side of the row (multiple rows can be selected as well) and click on CANC.

Select Fields

This panel is structured as a table: rows contain the attributes selected from the datamart schema, while columns include applicable functions. If the selected attributes belong to the same entity or to entities that are directly linked, the QbE will automatically resolve relationships between attributes (implicit join). Otherwise, the developer shall set explicit join relationships in the filter section.

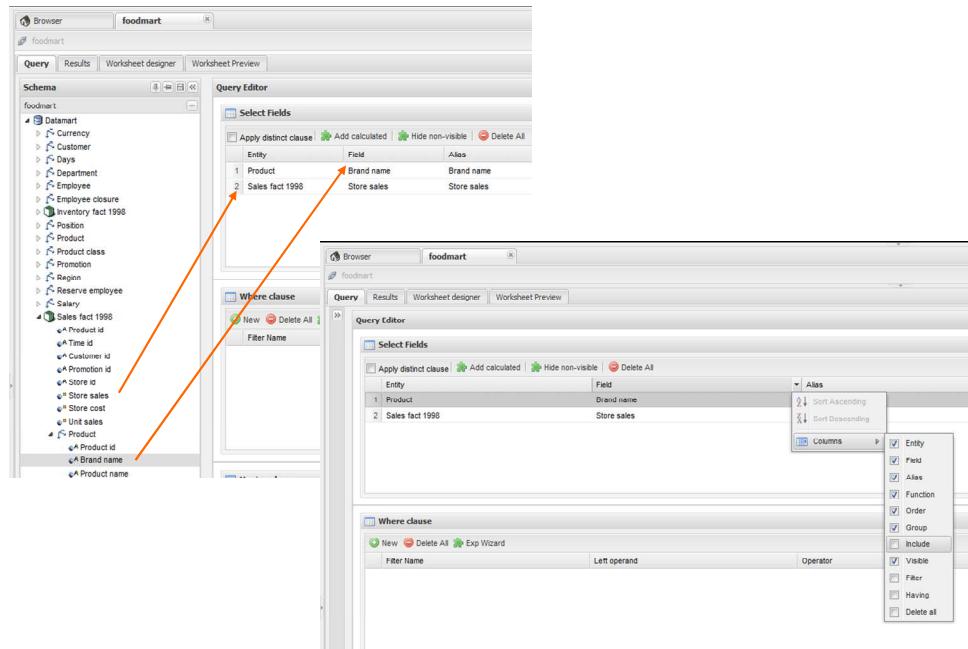


FIGURE 6.83 - Query creation : drag attributes and choose alias

Functions defined on columns allows the power user to:

- Define alias for fields: those aliases will be shown as column header in the result table.
- Group results based on aggregating functions. Double click on the **Functions** column of one attribute to select the aggregation type. Then tick on the other attribute on the **Group** column. This means that data on the first attribute will be aggregated and grouped according to the values of the second attribute (see FIGURE 6.84).
- Set ordering criteria. Double click on the **Order** column to set the ordering criteria.

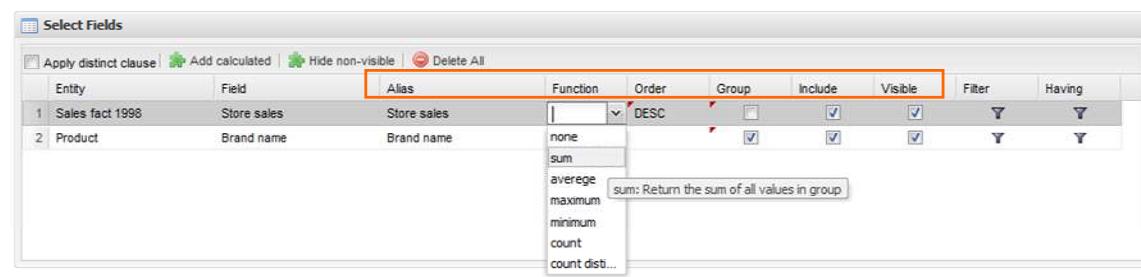
- Include or remove attributes from a query. Tick the column **Include** to include the attribute (this is the default behavior).
- Hide attributes in query results. Hidden attributes will still be used in the query but will not be shown in the result table. Tick the column **Visible** to make the attribute visible (this is the default behavior).
- Add a filter in the WHERE or HAVING clause section. Select the attribute on which you wish to create a filter and click on the filter symbol  of the corresponding column.

In addition, the select sub-section has a toolbar, whose functionalities are summarized in the following table:

Select fields toolbar options	
Apply distinct clause	Remove duplicated rows from results, if any
Hide non visible	Hide fields set as non visible in query results
Add calculated	Add calculated field to the query
Delete all	Remove all rows from the select area

TABLE 6.19 - Select fields toolbar

The order of rows and columns can be changed through drag and drop operations.



The screenshot shows the 'Select Fields' dialog box. At the top, there are four buttons: 'Apply distinct clause', 'Add calculated', 'Hide non-visible', and 'Delete All'. Below these are two rows of data:

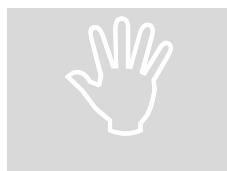
Entity	Field	Alias	Function	Order	Group	Include	Visible	Filter	Having
1 Sales fact 1998	Store sales	Store sales	none	DESC		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
2 Product	Brand name	Brand name	sum			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

A dropdown menu is open over the 'Function' column for the second row. The menu items are: none, sum, average, maximum, minimum, count, and count dist... A tooltip for 'sum' says: 'sum: Return the sum of all values in group'.

FIGURE 6.84 - Select fields: query configuration functions

Filters

The filtering panels allow power users to define filter criteria (WHERE clause), possibly applied to grouped data (HAVING clause). Similarly to the select area, filters are structured as a table: here rows contain filters and columns represent the elements of the filter.



Customization of the query editor

Not all columns of the editor are visible by default. To customize the editor appearance, double click on the arrow located on the column header and select **Columns**. Here you can choose which columns you want to see in the editor.

There are three ways to create a filter:

- Drag an attribute from the datamart schema to the chosen filter panel
- Click on the filter symbol  on the row of an attribute in the chosen panel
- Click on the  button of the panel

To remove a filter from the query editor, select the left side of the row (multiple rows can be selected as well) and click on **CANC**. To remove all filters from a Where/Having panel, click on **Delete All** on the toolbar of the corresponding panel.

Now let us consider the two different types of filters.

Where clause

Filters are expressions of type:

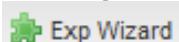
```
<Left operand, operator, right operand>
```

Once the power user has created the filter, he can configure it by using the proper setting values on columns. In particular:

- Column name contains the name of the filter (non editable) and column **Desc.** contains an editable description.
- Columns **Left operand**, **Operator**, **Right operand** allow the definition of the filter according to the syntax defined above.
- Columns **Left Operand Type** and **Right Operand Type** should contain the type of the operands.

Double clicking on the right operand, a lookup function will be activated to make the selection of a value easier.

- If the filter is dynamically assigned a value by the user executing the query, the column **Is for Prompt** should be checked. This way, the user will choose the value of the right operand at execution time.
- When multiple filters are defined, they can be combined with Boolean expressions, as defined in the **Bol. Connector** column.

Note that more complex combinations of filters can be defined using the expression wizard: to open it, create the base filters, then click on  **Exp Wizard** on the toolbar.

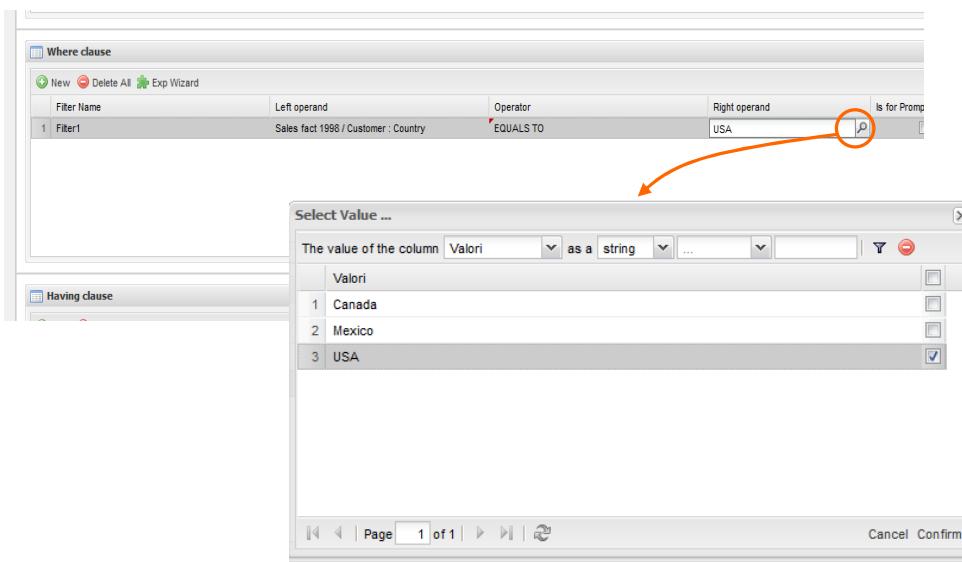


FIGURE 6.85 - Filter lookup for right operand selection

The possible types of filters in the QbE are summarized below, with the values that should be assigned to each column.

Filter type	Left operand	Operator	Right operand	Example
Basic	Entity.attribute	Any	value	Product.family = "Food"
Basic	Entity.attribute	Any	Entity.attribute	Sales 1998.Store sales > Sales 1998.Store cost
Parametric	Entity.attribute	Any	[parameter]	Product.family = [Product family]
Dynamic	Entity.attribute	Any	prompt	Product.family = ?
Explicit join	Entity1.attribute	Any	Entity2.attribute	Sales 1997.product name = Sales 1998. product name
Value list from subquery	Entity.attribute	In /not in	subquery	Sales 1998.customer in subquery
Single value from subquery	subquery	< = >	value	Subquery > 0

TABLE 6.20 - Possible combinations of filters in the QbE

The use of subqueries in filters is explained later in this section.

Having clause

The filter panel on the bottom allows the definition of filters according to the logic of the HAVING construct of the SQL language.

Columns are the same as those of the where clause. There are, however, additional columns related to grouping functions.

Once the power user has created the “having” filter, he can configure it using the proper setting values on columns. In particular:

- Column name contains the name of the filter (non editable) and column **Desc.** contains an editable description.
- Columns **Left operand**, **Operator**, **Right** operand allow the definition of the filter according to the syntax defined above.
- Two columns named **Function** are visible close to the right and the left operand, respectively: define here the aggregation function to use on the left, or right, operand.
- Columns **Left Operand Type** and **Right Operand Type** should contain the type of the operands.
- If the filter is dynamically assigned a value by the user executing the query, the column **Is for Prompt** should be checked. This way, the user will choose the value of the right operand at execution time.
- When multiple filters are defined, they can be combined with Boolean expressions, as defined in the **Bool. Connector** column.

Catalogs

This panel contains two elements:

- The catalog of queries (top)
- The list of analytical drivers linked to the QbE document (bottom).

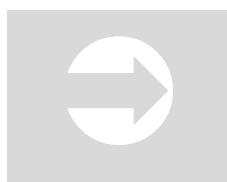
Query catalog

The catalog of queries is a list of all queries defined in the QbE document.

As we already discussed, several queries can be built over the same QbE datamart. The catalog will list all saved queries on the current datamart. The base query that we are creating in the query editor will appear with a default name (query-q1): to rename it, simply double click on the query item in the catalog tree.

To create a new query, click on the  icon. The query will appear in the catalog at the same level, as the base query. Using the query editor you can create the query and save it.

To create a new subquery, which will be used as a filter inside the main query, click on . The query will appear in the catalog as a child node of the base query.



Subqueries

The use of subqueries in a QbE document is described at section Advanced QbE Functionalities, later in this chapter.

Anaylitical drivers catalog

The lower panel lists all analytical drivers linked to the QbE document.

Although drivers are not editable, they can be used as a right/left condition of a filter by dragging and dropping them into the corresponding columns of the query editor. Here they will be represented with the following syntax:

[Product_Family]



Analytical Drivers

Read how to associate analytical drivers to documents at chapter 5 - SpagoBI Server, section Behavioral Model.

Advanced QbE functionalities

Until now we described the base functionalities of the QbE. This section is dedicated to advanced features, which can be comfortably managed by more expert users.

Multidatamart

The QbE engine supports the combination of multiple data sources into the same logical model. Multidatamart is a powerful feature because it allows users to create logical schemas that are completely independent of the physical structure and storage of data.

On the other hand, this features require some expertise. First, because the data sources must be set by hand in the template, as follows:

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

In addition, entities from different datamarts must be explicitly joined by the user, which is in charge of ensuring correctness and consistency of the links between entities.

The final schema will show entities as if they belonged to the same datamart.

Profiled Access

SpagoBI supports controlled access and visibility on any type of analytical document thanks to its Behavioural Model.



The behavioral model

The behavioral model is a very important concept in SpagoBI. For a full understanding of its meaning and functionalities, please refer to chapter 5 - SpagoBI Server.

As far as QbE documents are concerned, profiled access can be implemented at datamart level by editing the QbE datamart template. This critical operation can only be performed by an administrator. Once the template is defined, the user will not be able to modify the logics of profiled access.

Hereafter, a generic schema of QbE datamart template, which shows two different options for visibility restriction:

```
<QBE>
  <DATAMART name="Metamodel name"/>
    ... more datamart definitions
  <MODALITY name='name of a modality'>
    <TABLE name='JPA object class name' accessible='true|false'>
      <FIELDS>
        <FIELD name='JPA attribute name' accessible=' true|false
      />
          ...many other fields rules
      </FIELDS>
      <FILTERS>
        <FILTER>
          F{JPA attribute name} = 'P{parameter name}'
        </FILTER>
          ...many other filter rules
      </FILTERS>
    </TABLE>
  </MODALITY>
</QBE>
```

Where:

- The `name` attribute of the `datamart` tag is the name of the datamart. In case of deploy from SpagoBI Meta this value is automatically assigned. There may be one or more datamarts, as explained later in section Multidatamart.

- The `modality` section contains all profiling logics for both model and data. In particular:
- The `table`, `field` and `filters` tags rule visibility of entities and attributes, statically and dynamically.
- The `name` attribute for table, field and filters must contain the name of the class generated by the business model compilation via the JPA middleware.

In the template above, we can see two types of visibility restrictions.

In the `table` element, some fields are statically set as visible or not visible. For example, if we wish to hide the entity `salary` and the attribute `phone1` of the `customer` entity, we should write:

```
<QBE>
    <DATAMART name="foodmart"/>
    <MODALITY name='salary_and_customer' >
        <TABLE name='it.eng.spagobi.meta.Salary'
accessible='false' />
            <TABLE name='it.eng.spagobi.meta.Customer'
accessible='true'>
                <FIELDS>
                    <FIELD name='phone1' accessible='false' />
                </FIELDS>
            </TABLE>
        </MODALITY>
    </QBE>
```

In the `Fields` section of the template, we are defining a rule for profiled access. The rule is bound to the value of an attribute in the user profile. For example, if we wish to set a profiled access based on product family, we may define the `family` attribute for SpagoBI users and assign it to the field `productFamily` in the template.

At query execution time, all products whose family is different from the value defined by the user profile attribute `family` will be filtered out from results. Thus, the user will not see the data that are out of his visibility scope.

```

<QBE>
    <DATAMART name="foodmart"/>
    <MODALITY name='family profiled'>
        <TABLE name=' it.eng.spagobi.meta.productClass '
accessible='true'>
            <FILTERS>
                <FILTER>
                    F{productFamily} = 'P{family}'
                </FILTER>
                ...many other filter rules
            </FILTERS>
        </TABLE>
    </MODALITY>
</QBE>

```

Note that the filtering logic will be applied even if the query does not affect directly the field in the template. For example, if the user asks to retrieve sales facts in January, he will only be shown sales in January for products whose family is equal to his profile attribute `Family`.

Cross Navigation & Advanced Calculated Fields

We have already seen how to add calculated fields to a QbE query. In addition to the basic types of calculated fields, more advanced options can be configured. In particular, calculated fields can provide a link to the following elements:

- Images
- HTML pages
- Other SpagoBI documents (cross navigation).

To enable advanced functionalities, open the Calculated Field Wizard by selecting **Add Calculated** in the toolbar. Check the box **Expert User** and you will see three types of Groovy script fields on the left.

If you select image, the editor will show you the HTML template code to be used to reference an image on the server:

```
'</img>'
```

Replace the URL with the path of your image.

For example, the following piece of code will produce a graphical result like the one shown in FIGURE 6.86 (column Bullet chart).

```
baseUrl = "http://localhost:8080/SpagoBIQbeEngine/img/inline/";
if(dmFields['it.eng.spagobi.meta.Sales_fact_1998:store_cost']<2){
return '<center></img></center>';
} else
if(dmFields['it.eng.spagobi.meta.Sales_fact_1998:store_cost']>3){
return '<center></img></center>';
} else {
return '<center></img></center>';
}
```

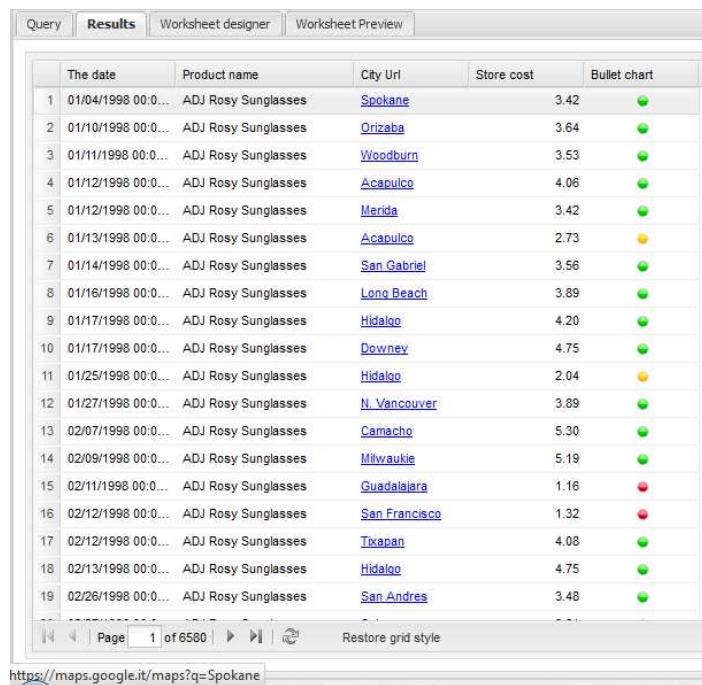


FIGURE 6.86 - QbE results tab with custom image and hyperlink

Similarly, if you select a link, you will need to customize the template as follows:

```
'<a href="${URL}"> ${LABEL} </a>'
```

For example, if you add the following code, you will obtain the result shown in FIGURE 6.86 (column City URL).

```
'<a href="https://maps.google.it/maps?q='
+
dmFields['it.eng.spagobi.meta.Sales_fact_1998::rel_customer_id_in
_customer(rel_customer_id_in_customer):city']
+ '>'
+
dmFields['it.eng.spagobi.meta.Sales_fact_1998::rel_customer_id_in
_customer(rel_customer_id_in_customer):city']
+ '</a>'
```

If you select cross navigation, you will have to customize the code so as to call the cross navigation service. In the wizard you can find an example:

```
String label = 'bestByRegion';
String text= fields['salesRegion'];
String params= 'region=5';
String subobject;

String result = '';

result += '<a href="#" onclick="javascript:sendMessage({';
result += '\'label\': \'' + label + '\'';
result += ', parameters:\'\'' + params + '\'\'';
result += ', windowName: this.name';
if(subobject != null) result += ', subobject:\'\'' + subobject
+'\'\'';
result += '},\'crossnavigation\')"';
result += '>' + text + '</a>';

return result;
```

Where:

- **Label** is the name of the target document
- **Text** is the clickable field in the QbE query results table.
- **Params** is the list of parameters passed to the target document

- **Subobject** is the subobject name of the target document on which you want to cross navigate. This will permit to go directly to the subobject. If your don't want to see a specific subobject just leave it blank.



Cross Navigation

For more details on cross navigation and how it works, please refer to the corresponding section in chapter 5 - SpagoBI Server.

Subqueries

As discussed above, the QbE Engine supports the definition and usage of subqueries similarly to the SQL language. So, it is possible to define a subquery and use it within a filter in association to the in/not in operator, as shown in **Errore. L'origine riferimento non è stata trovata..**

Once defined the main query and the filter that will contain the subquery, go to the Query Catalog panel and click on . The query will appear in the catalog as a child node of the base query.

To use the sub-query inside the main query, simply drag and drop it into the columns corresponding to the left or right operand of the filter. Set the type of operand to “sub-query”. Now the sub-query will be used to provide values within the filter, in a similar way to SQL sub-query.

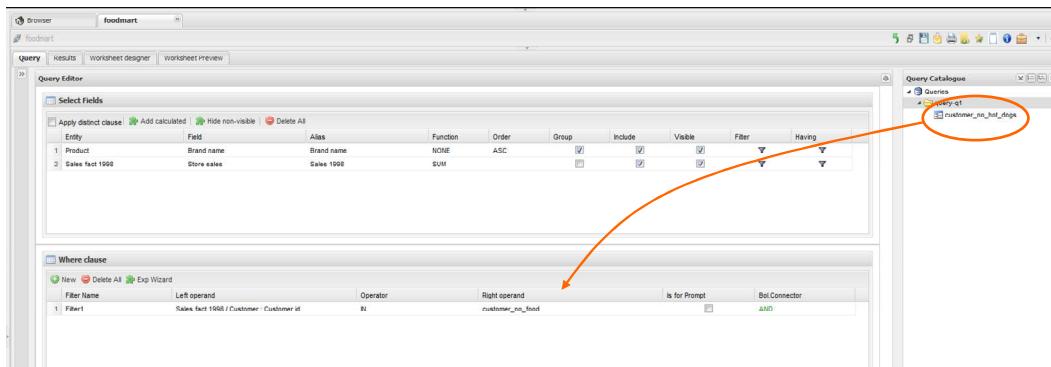


FIGURE 6.87 - QbE query: use of a subquery in a filter

SpagoBISmartFilterEngine

In the previous section we learned what the QbE engine is and how you can use it. Using the QbE Engine, power users can define complex queries, without dealing with SQL code but still with a similar degree of complexity. In fact, end users may not feel comfortable with complex queries and often prefer to simply execute queries that have already been defined by expert users.

SpagoBISmartFilterEngine provides end users with an intuitive tool to customize queries that have already been defined by power users. The logics behind the smart filter is more similar to a filter (hence, the name) than to a query builder: this has the advantage of being simple and intuitive for end users. At the same time, users are enabled with different customization options to meet their various needs in accessing and analyzing data.

Before going into details on the creation of a smartfilter document, let us have a look at the features provided by this engine.

Smart filter overview

A smart filter document is based on an underlying query defined via the QbE engine. When the document is executed, a form is generated starting from the query and containing all filtering criteria defined over the base query. The end user can set filters to extract the data of interest.

Once he has extracted the data, the user can further manipulate them by grouping them, viewing them in a master/detail structure, or use them to build an ad-hoc analysis using the Worksheet Engine.



SpagoBIWorksheetEngine

The Worksheet Engine allows end users to build ad-hoc analytical documents using an intuitive graphical interface. Read section SpagoBIWorksheetEngine, later in this chapter.

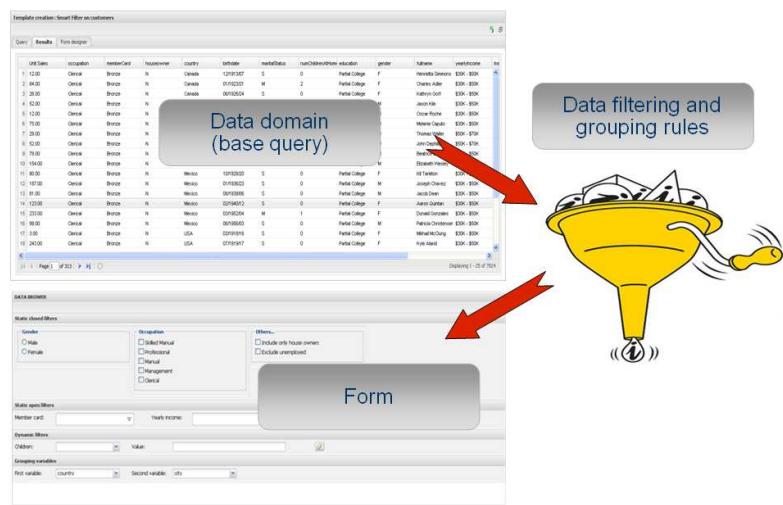


FIGURE 6.88 - Smart filter logical architecture

The screenshot shows the results of a grouping filter execution. On the left, a list of countries and cities with their record counts (e.g., Canada: Burnaby 1, Canada: Offside 1, Canada: Metchosin 1, Canada: Port Hamond 2, Canada: Royal Oak 1, Mexico: La Cruz 1, Mexico: San Andres 2, USA: Albany 1, USA: Anacortes 1, USA: Arcadia 1, USA: Beaverton 1, USA: Bremerton 1, USA: Buren 1, USA: Corvallis 1, USA: B Cajon 1, USA: Lake Oswego 1, USA: Lemon Grove 1). An orange arrow points from this list to a detailed table on the right. The detailed table displays specific data for Mexico and USA, including Unit Sales, occupation, memberCard, homeowner, country, birthdate, maritalStatus, Num. children at home, and education. The table shows two entries for Mexico (Unit Sales 122.00, Professional, Bronze, Y, Mexico, 08/1922/10, M, 2, Bachelors Deg) and two entries for USA (Unit Sales 95.00, Professional, Bronze, Y, Mexico, 09/1977/08, M, 2, Bachelors Deg).

FIGURE 6.89 - Grouping filter results after execution

The end user will see a form like the one shown in FIGURE 6.90. He will select filters according to his preferences and click on **Execute** to view results. From the results page, the user can go back to the form by clicking on **Back to form**. This allows him to change the filtering criteria and view data modified accordingly.

The screenshot shows a 'DATA BROWSER' interface titled 'Smart filter selection form'. It includes sections for 'Static closed filters' (Gender: Male, Female; Occupation: Skilled Manual, Professional, Manual, Management, Clerical; Others...: Include only house owners, Exclude unemployed), 'Static open filters' (Member card, Yearly income dropdowns), 'Dynamic filters' (Children dropdown, Value input field, search icon), and 'Grouping variables' (First variable: country, Second variable: city dropdowns). A 'Execute' button is located at the top right.

FIGURE 6.90 - Smart filter selection form

Smart filter creation

The steps to create a smart filter document are:

- Create the QbE base query
- Add filters and grouping variables
- Register the document on the server.

The first two steps correspond to the creation of a query template, while the last one is common to all SpagoBI documents. You can create a query template in two different ways:

- Manual editing. This option is recommended to very expert users only.
- Use the template build wizard provided within the Smart Filter document development interface.

In the following we will adopt the second approach.

Select Analytical Model > Document Development and click on Add document to create a new document. Select a Smart Filter document. Then open the wizard clicking on the small icon at the bottom .



Engine configuration

The engine configuration parameters to be used during document creation are summarized at section Engines Management, in chapter 5 - SpagoBI Server. Please refer to that section for configuration details.

The template build wizard will guide the user along these tasks.

Query creation

The first step is the definition of a QbE query. Therefore, the first information required by the template wizard is the metamodel over which the query will be built. Once you have selected it, click on **Start the designer** to start creating the template.

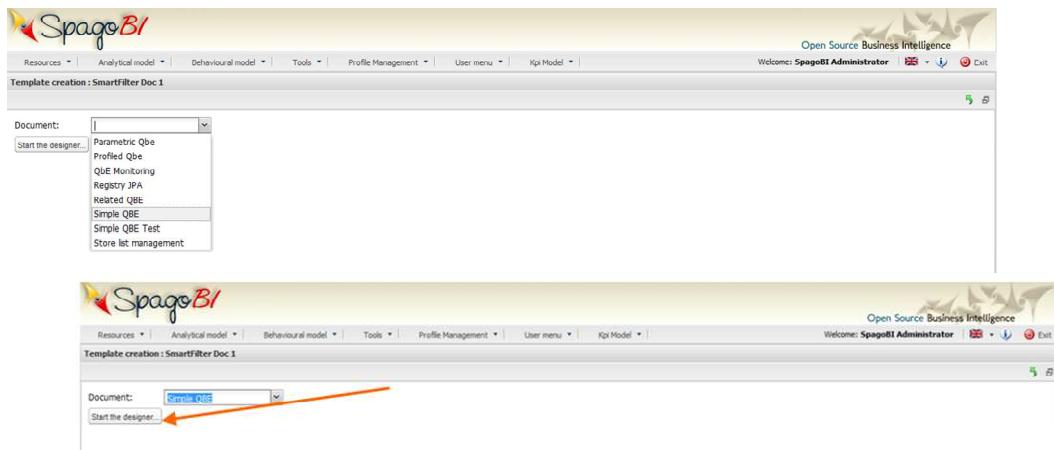


FIGURE 6.91 - Choose the metamodel for the QbE query

The query editor is the same as the one available in the QbE engine, where the power user can easily create queries without dealing with SQL code.



Building QbE queries

The QbE engine allows the creation of complex queries via an intuitive interface for end users. The same interface is used to create a Smart Filter document. Please refer to section SpagoBIQbEEngine, in this chapter, for more details on query building.

Once you have defined the query, it is a good practice to execute it, to check that the results are correctly displayed. If everything is correct, move to the **Form designer** tab.

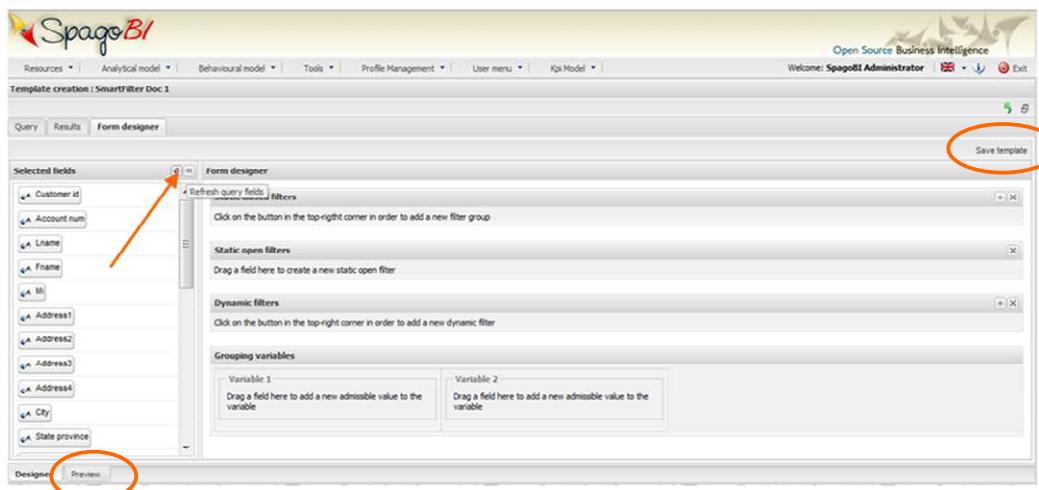
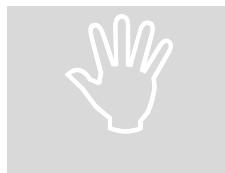


FIGURE 6.92 - Smart filter form editor

As shown in FIGURE 6.92, the form editor consists of two parts. On the right, the base structure of a form is shown. As the user creates new filters (open, closed, dynamic filters or grouping variables), they will be added to the corresponding section of the form. The left panel is used to visualize query fields that will be used in filters.

This section is initially empty: click on **Refresh query fields** to view all the fields of the query that you have just defined. If you modify the query, remember to refresh the fields in this panel.



Refresh query fields

Any time the query is modified, fields in the form editor must be refreshed clicking on **Refresh query fields**.

To have a preview of the form, click on the **Preview** tab located at the bottom of the editor. When you are happy with the result, click on **Save template** to save the form as a new template for the Smart Filter document.

Filters

Designing the form basically involves the definition of filters. There are four types of filters in a smart filter document. They are grouped together in the following categories (from top to bottom in the graphical interface):

- **Static closed filters.** Filtering criteria based on complex conditions. They can be single or multiple.
- **Static open filters.** Filtering criteria based on the selection of one or more values within a pre-defined list. They can be single or multiple.
- **Dynamic filters.** Filtering criteria allowing the user to build his own filtering conditions (as: operator, expression, value).
- **Grouping variables.** Grouping criteria of data obtained from the initial result set.

The grouping section is always displayed, while the others are shown only if one filter at least has been defined.

In the following we discuss each type of filter in detail and show how to define them in a smart filter template.

▪ **Static closed filters**

Static closed filters are defined as a group. Therefore, the first step to create a static closed filter is the configuration of the properties related to the management of the filter group.

To add a static closed filter, click on the button **Add static closed filters group** located at the right top corner of the corresponding section. A configuration window will open, where the following properties can be set:

Static closed filters configuration options	
Name	Name of the filter group
Single selection	If set to YES, the user will be enabled to select only one filter within the filter group. Otherwise, multiple selections are allowed.
Allow No selection	If set to YES, the user will be allowed to leave all filters in the undefined group.
No selection option label	Label of the option corresponding to no selection. Can be assigned a value only if the Allow No selection is set to YES.
Boolean connector	Define the logical connector between filters in the group. Can be set only if Single selection is set to NO.

TABLE 6.21 - Static closed filters configuration options



FIGURE 6.93 - Static closed filters configuration editor

Once you have set all properties, you can actually create the filter group clicking on **Apply**. To edit the group again, click on **Edit**: this will reopen the editor window.

Now you can add filters to the group that you have just created. Clicking on **Add**, the filter editor will open (see FIGURE 6.94). The **name** attribute defines the name that will be shown to the user beside the filter. The attributes **Field**, **Operator** and **Value** define the filter as a condition of the form:

```
[field] [operator] [value]
```

Where **Field** is a column of the underlying query and can be selected using the drop down menu. Multiple conditions can be combined using the **Boolean Connector** field (in the table).

You can save the filter clicking on **Apply**. To edit the filter again, click on **Edit**: this will reopen the filter editor window.

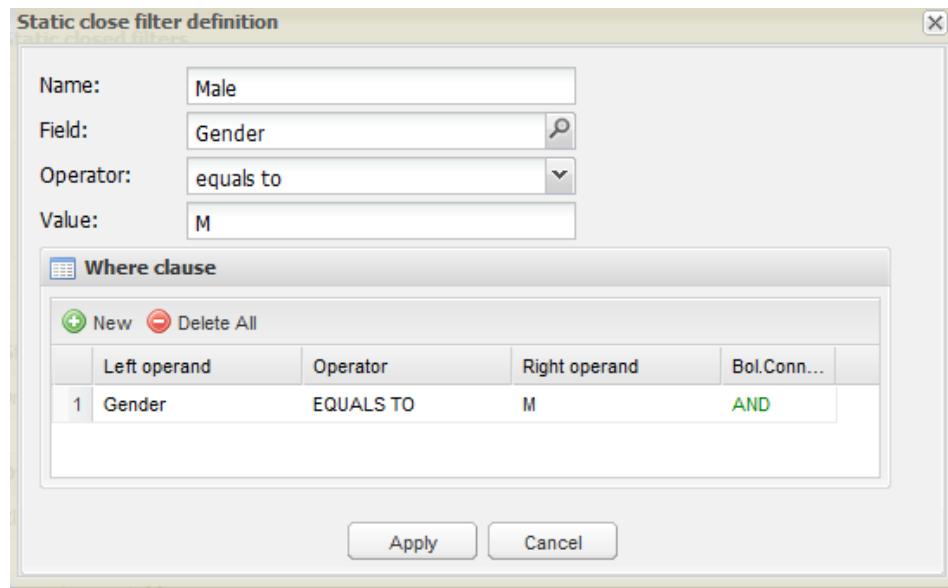


FIGURE 6.94 - Static closed filter editor

- **Static open filters**

Static open filters allow the user to select values that are not pre-determined but retrieved at runtime via a query on the database. For this reason they are called open. They are still static because the column that will be filtered is determined at filter definition time. Differently from closed filters, they are not grouped.

To create a new static open filter, drag the field that you wish to filter from the left panel onto the editor area that corresponds to static open filters. An editor window will open as soon as the field is dropped onto the editor area.

As shown in FIGURE 6.95, the open filter is similar to the closed one, except that the Field cannot be modified (because it was dragged before). Furthermore, the Value is not editable since values will be retrieved from a query on the database at runtime. There are two options for the query:

- **Standard query.** In this case all distinct values for the given field will be retrieved. It is possible to choose how to order results, and if values should be retrieved from the entity to which the field belongs, or its associated first level entity.
- **Custom query.** With this option it is possible to choose a query defined via QbE to retrieve admissible values. This means that the level of control on admissible values is enforced, including the use of the behavioral model to show different values to different users.

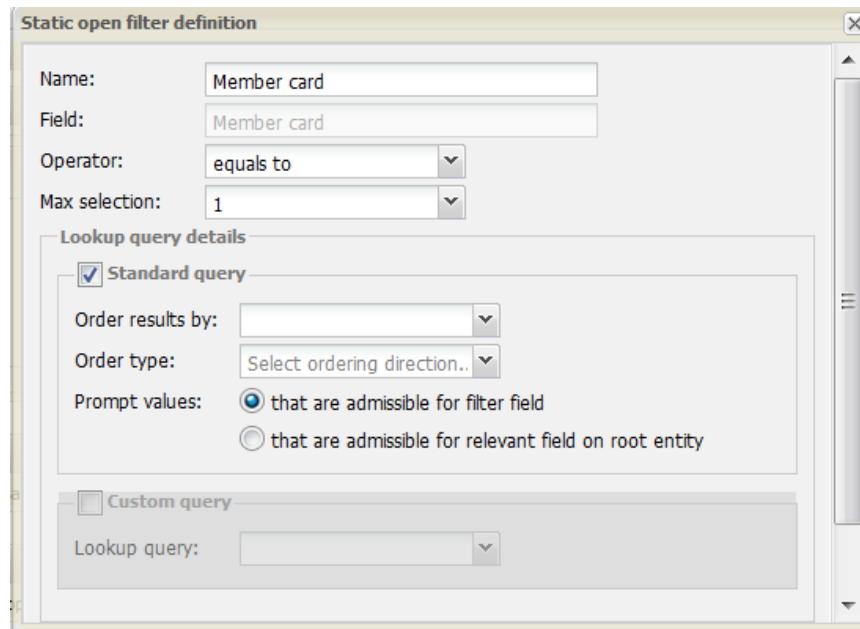


FIGURE 6.95 - Static open filter editor

▪ Dynamic filters

Dynamic filters, like closed ones, require the statical definition of the fields that shall be filtered. Nevertheless the user can freely choose the value attribute to which the field will be compared. They generally replace static closed filters whenever the field domain is a continuum, which makes the definition of discrete values scarcely meaningful.

The comparison operator is statically defined for a group of filters. Different groups will have in general different operators. Hence, dynamic filters, like static closed ones, are defined in groups.

Therefore, before creating a filter, you should create a group clicking on the + button at the top right corner of the dynamic filter area, **Add dynamic filters group**.

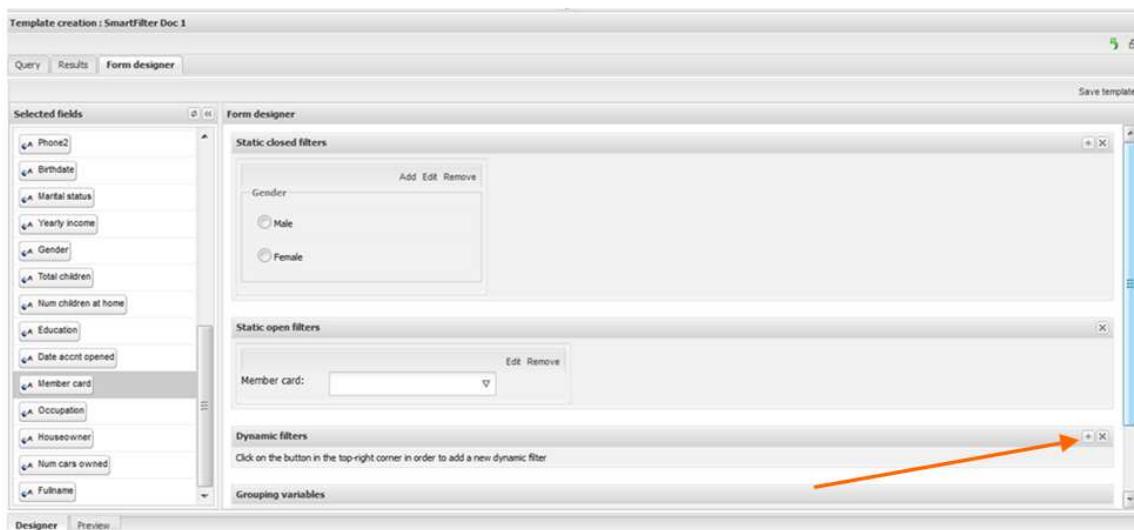


FIGURE 6.96 - Add dynamic filter group

When creating the group, select both the name and the characterizing operator. Then you can simply drag fields from the left panel onto the filter group area: this will create a new dynamic filter in that group.

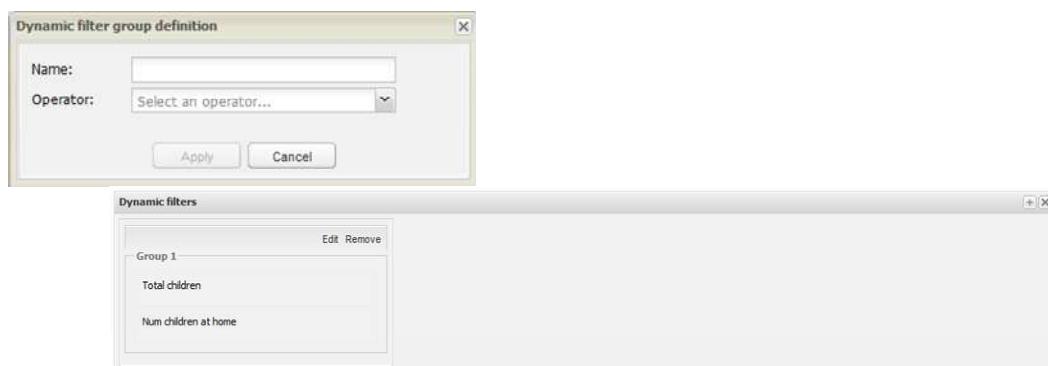


FIGURE 6.97 - Create a group of dynamic filters

■ Grouping variables

Beside filtering the results of the base query, as seen before, the Smart Filter allows its grouping on two levels. For each level, you can add fields from the base query. You will be shown the different options for first and second level grouping, when executing the document. To add a field on a level, drag it from the left panel onto the corresponding editor area.

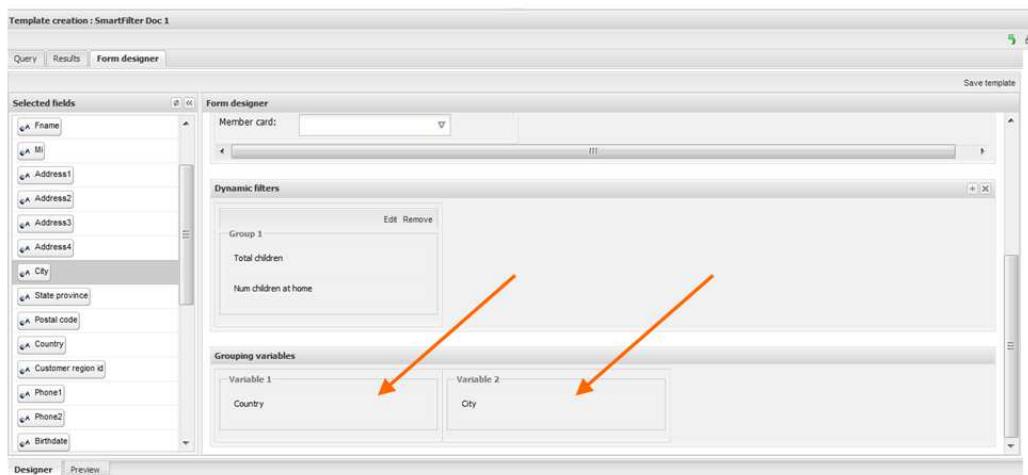


FIGURE 6.98 - Grouping variables definition

Now save it by clicking on **Save Template**. For a preview, clicking on the **Preview** tab.

Ad hoc reporting

Ad-hoc reporting allows end-users to freely build their own structured views on their own data. The main characteristics of a tool in this area are:

- simple and free data selection (free inquiry)
- no need of technical skills
- able to format results list
- free and graphical creation of tabular reports and charts with customizable layout
- saving reports in the user's personal working area
- private use
- easily accessible and modifiable.

Some ad-hoc reporting features are always required, even if not for all types of users. It is usually for analysts and operative people wishing to build their own synthetic or detailed report dealing with a medium level of difficulty.

SpagoBI supports ad-hoc reporting features with the Worksheet engine (SpagoBIWorksheetEngine), which allows end-users to freely create their own multi-sheet reports, by defining simple tables, cross-tables and different chart types in their document layout.

SpagoBIWorksheetEngine

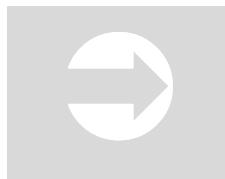
The Worksheet Engine represents the logical evolution of the QbE Engine: while the latter is aimed at enabling free query on data, the former allows the end user to easily create and customize reports on free queries built with the QbE. The combination of these two engines realizes a complete support for ad-hoc data inquiry and reporting, targeted at non technical users wishing to build their own analyzes.

The Worksheet Engine allows the creation of a separate sheet for each piece of analysis, which makes the document modular and easy to read. Worksheets are designed using an intuitive web interface: no designer is needed since it is all web-based. Documents are mainly thought for web execution, although it is possible to export them in formats other than HTML.

The creation of a worksheet document follows a slightly different procedure from other SpagoBI documents. In fact, a worksheet can only be created starting from two other types of documents:

- QbE documents
- Smart filter documents.

This means that you first need to create either a QbE or a smart filter document, and produce a result set from these documents; then, you can create a worksheet. The worksheet will show those results as a report (composed of multiple sheets).



Free inquiry

The QbE engine and the Smart Filter engine support free inquiry on business data. Learn how to create QbE and smart filter documents at section Free Inquiry, in this chapter.

Now let us see in detail how to build a worksheet. In our example, we will use a QbE document: as said above, a smart filter can be also used in order to access the Worksheet Engine.

First, create a query and look at its preview by clicking on the **Results** tab. Once you are happy with the results, you can start designing the worksheet by clicking on the **Worksheet designer** tab (shown in FIGURE 6.99). The worksheet designer is composed of different sections:

- **Selected Fields** (top left). The list of fields resulting from the query (QbE or smart filter).
- **Palette** (left side). The list of available graphical widgets that can be added to a sheet. The palette includes charts (bar, pie or line) and tables (flat or pivot).
- **Layout** (bottom left). Here you select the layout of the sheet.
- **Filters** (right side). The main editor area.

By default, the editor area is divided into three parts:

- A header with a text editor and options to insert images in a given position
- A central body, where you can insert only one widget per sheet
- A footer mirroring the header.

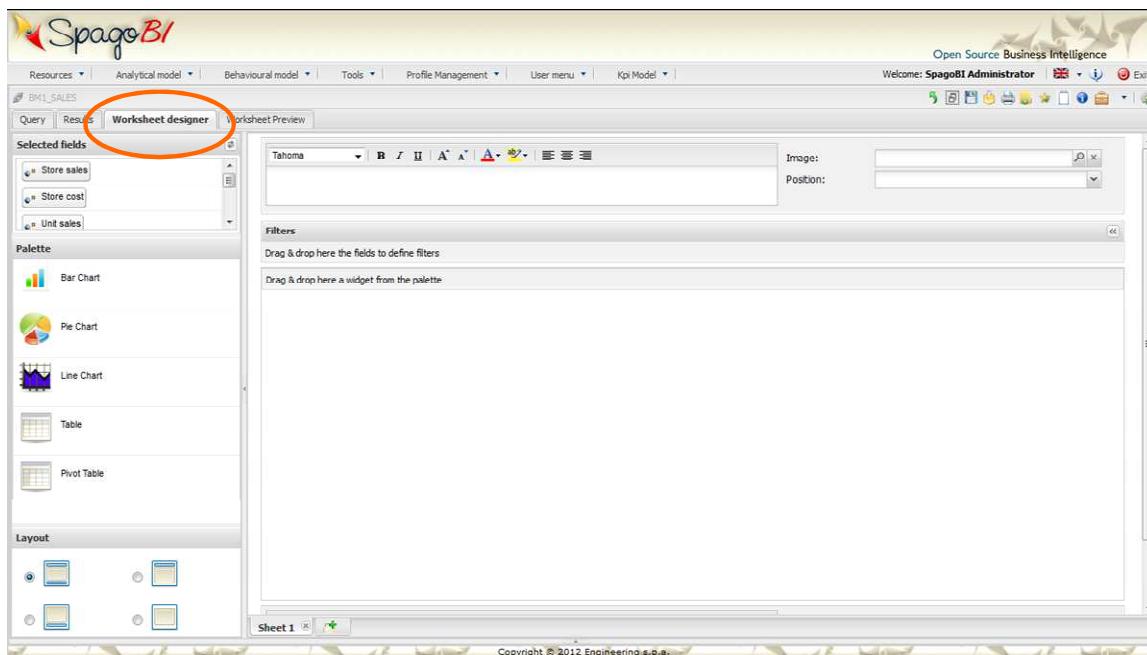


FIGURE 6.99 - Worksheet designer

Before proceeding to design the worksheet, you need to refresh the available data in the **Selected Fields** section: click on the icon to load fields generated by the last execution of the query. Remember to refresh data also when you modify the query: first generate the results preview, then force the refresh of data.

Now you can start designing the first sheet. Drag and drop a graphical widget onto the central body. Drag fields from the left list and drop them into the appropriate area of the widget (the widget itself will show explanatory tooltips). Set the title and insert images if you wish.

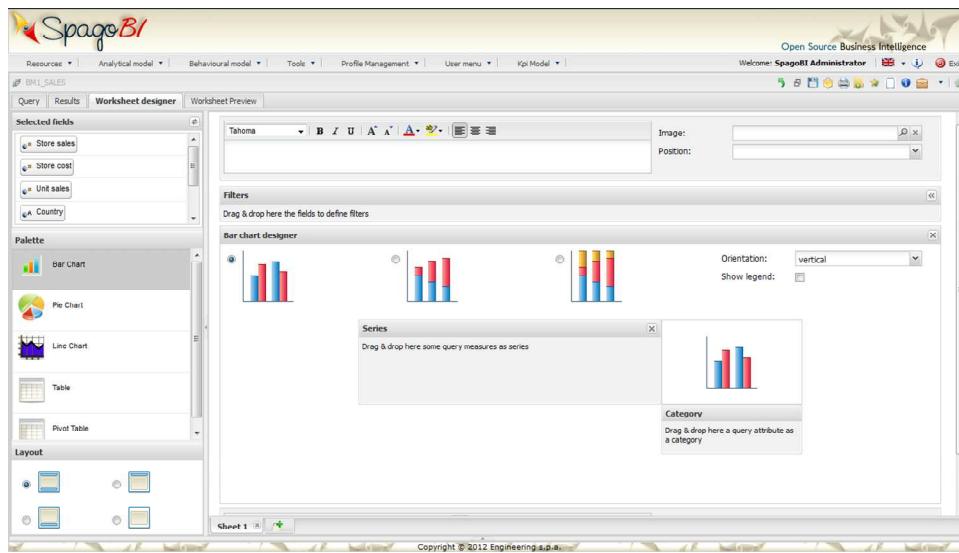


FIGURE 6.100 - Designing the first sheet

At any time you can click on the **Preview** tab to see the result. Once you are happy with the result, click on the icon, located in the top right toolbar. Choose where you want to save the document (multiple locations are possible) and save. If you are not satisfied yet, just go back to the designer tab, or to the **Query** tab to modify the query.

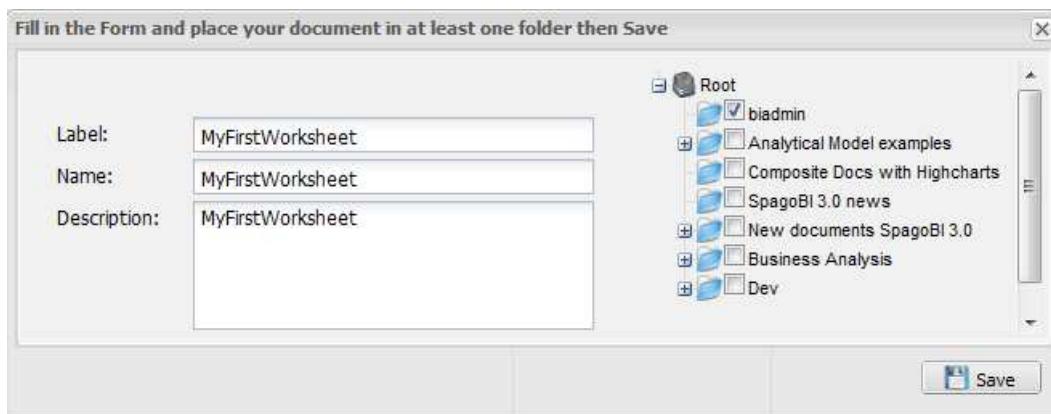


FIGURE 6.101 - Save the worksheet document

You can add further sheets to the document clicking on the icon at the bottom of the main editor area. To delete a sheet, just click on the small cross

beside its name. A tab can be closed or renamed by right-clicking on its name and choosing the appropriate item in the contextual menu.

Right-clicking on a field of the list on your left, you can configure the visibility of this field, depending on whether it is a measure or an attribute. Click on Options in the contextual menu and select your preferred choice, as shown in FIGURE 6.102.



FIGURE 6.102 - Field configuration options

Let us explain more in detail the elements defining a sheet: graphical widgets and filters.

Graphical widgets

As said above, there are two types of graphical widgets available in the worksheet:

- Charts: bar, pie and line.
- Tables: flat and pivot.

Charts

Charts require the definition of one or more categories, and one or more series. In the case of a pie chart, there is only one series allowed. Series and categories are dragged and dropped from the left list of fields. The designer enforces controls, so that the definition of series/categories is consistent. This makes the tool usable to end users without technical skills.

All charts allow the designer to:

- Hide/show the legend via the checkbox.
- Choose chart orientation and data structure (except the pie chart, which has only one graphical layout)
- Choose the grouping function measure. This must be done the first time a field is dropped onto the editor area, and can be later modified via the configuration table shown in FIGURE 6.103.
- Rename series, change their color, set precision, suffix and separator symbols via the configuration table of FIGURE 6.103.

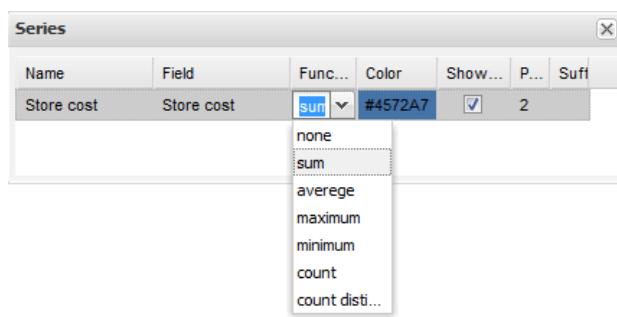


FIGURE 6.103 - Choosing the grouping function for charts

To see a preview of your worksheet, click on the **Preview** tab. Here you can also choose the sheets to be exported. Clicking at the top of the chart area, you can select the series that will be visible in the exported document (see FIGURE 6.104). This may be particularly useful in case there are several series and only some of them are relevant to the exported document.

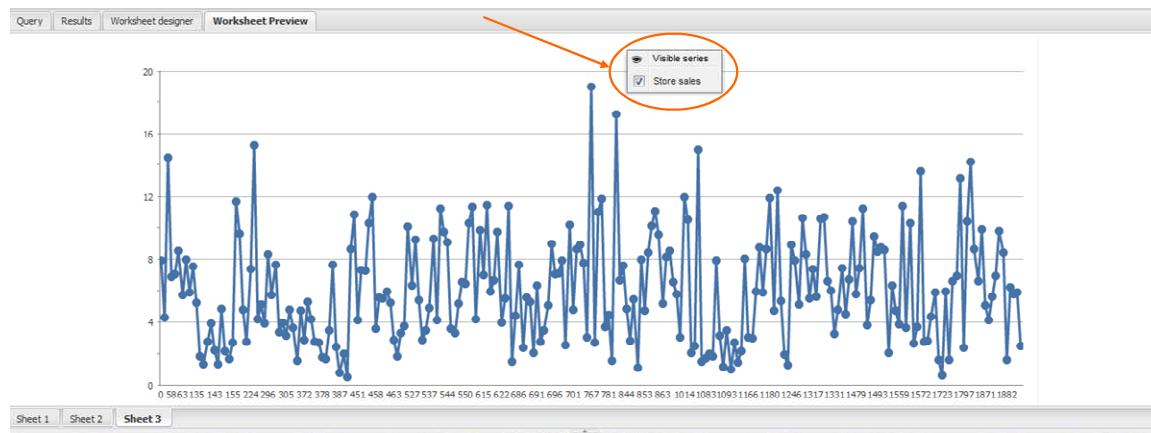


FIGURE 6.104 - Setting export options

Tables

To fill tables, just drag and drop fields from the left panel. Flat tables show flat list of values, while pivot tables allow users to select the fields to be plotted on rows and on columns, as shown in FIGURE 6.105. In the central area, called **Measures**, you can drop your measures (e.g., sales or costs). On columns and rows, you can drop at least one dimension each. This will produce a cross table like the one shown in FIGURE 6.105.

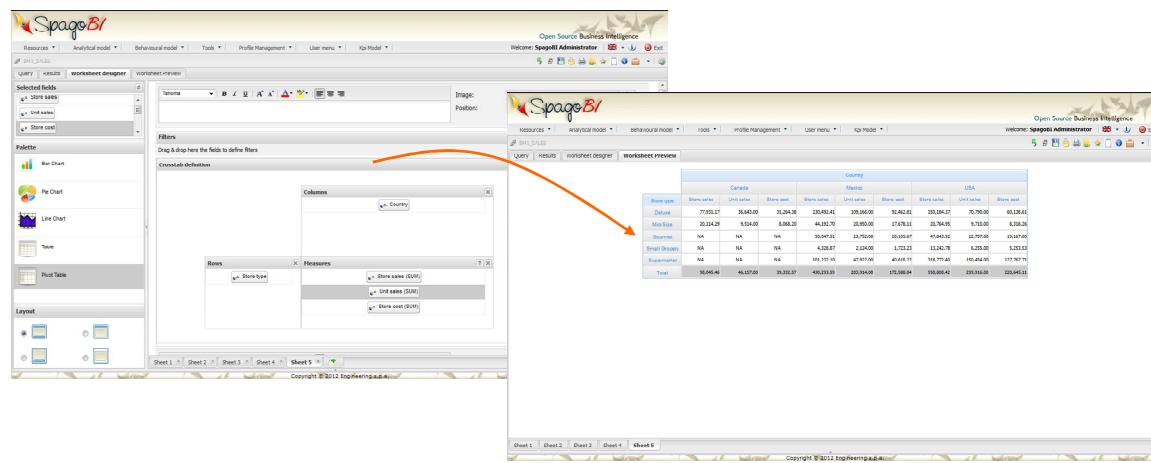


FIGURE 6.105 - Pivot table in a worksheet document

When you drag and drop a measure onto the pivot table, you will be asked which aggregation function you wish to use. If you wish to change it afterwards,

double click on the measure and select a new aggregation function. You can configure the pivot table by clicking on the  icon in the Measure area, as shown in FIGURE 6.106

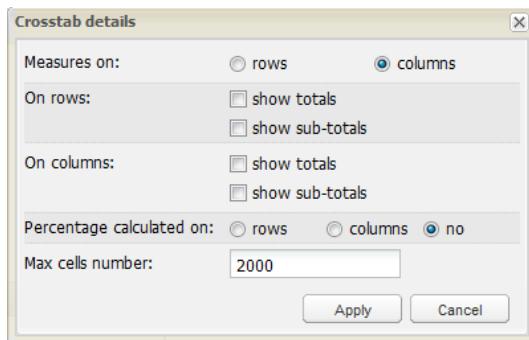


FIGURE 6.106 - Pivot table details configuration

When you have edited it, go to the preview. Here you can see your pivot table and still make changes. Clicking on the header of a column in a pivot table, you can hide or show it. Using the same menu you can also add calculated fields to the table. Furthermore, you can swap columns by dragging them.

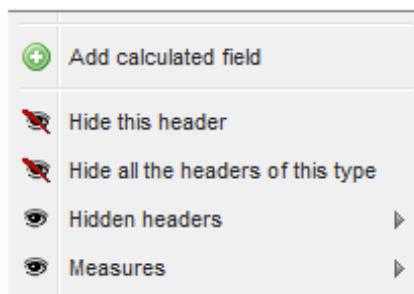


FIGURE 6.107 - Editing options on pivot tables

Filters

The worksheet allows users to add filters on data in different places and at different levels:

- **Global filters.** These filters apply to all sheets in the documents. To create a global filter on a field, double click on it from the list **Select Fields**. A popup window will open and allow you to pick the values that you wish to select for that field.
- **Local filters.** These filters apply locally, to the sheet where they are created. To create a local filter on a field, double click on it from within the editor area (rows or columns). A popup window will open to allow you to pick the values that you wish to select for that field.
- **Conditional dynamic filters.** These filters apply locally, to the sheet where they are created. To create a conditional filter, drag a field onto the **Filters** section. Then click on the + symbol to edit the filter. In this case you will not statically select a value, but the user executing the document will choose the value for the filter.

A particular type of conditional filter is the *splitting filter* (see FIGURE 6.108). A It allows users to choose a value for the given field, while assigning all possible values to that field when the document is exported. In other words, the exported document will contain one sheet for each admissible value of the field. In addition, the history of value selection for the filter will be registered and shown in the exported document.

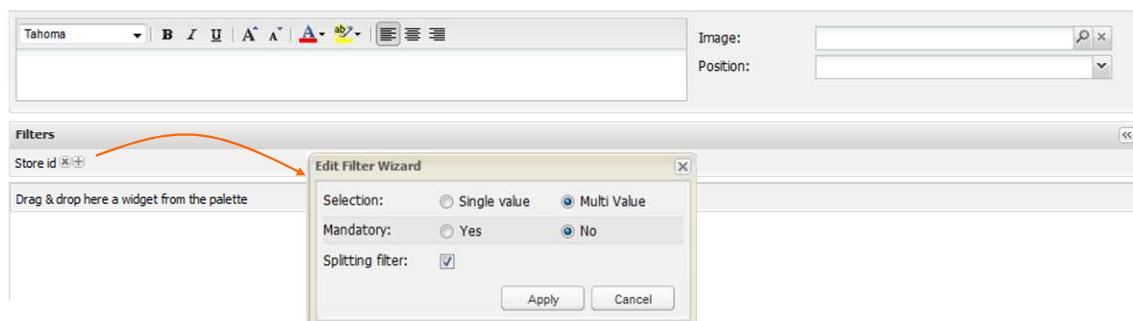


FIGURE 6.108 - Splitting filter in the worksheet designer

Below we summarize the list of possible actions for worksheet configuration.

Editing actions on pivot tables		
Element	Action	Result
Measure in pivot table editor	Double click	Change the aggregation function, according to the menu selection
Row or Column in pivot table editor	Double click	Create local filter
Attributes in field list	Double click	Create global filter
Attributes in field list	Right click	Options > then select display configuration
 icon	Click	Set configuration details for the pivot table

TABLE 6.22 -- Editing actions on pivot tables

Location Intelligence

Location Intelligence is based on the idea that geographical spaces are a particular analytical dimension in the BI domain. It is based on:

- the geographical representation of data
- interaction with GIS systems
- spatial data
- spatial operators

Location Intelligence usually guarantees:

- an immediate perception of a phenomena distribution over a geographical area
- interactivity
- multivariate analysis
- temporal snapshots

Location Intelligence is becoming widely used, mostly thanks to the emergence of location services such as Google Maps. This domain is very easy to use for all kinds of users, usually analysts and operational profiles. By contrast, its management is not as easy, especially if it implies an internal management of the geographical data base.

SpagoBI offers two geographical engines allowing to set a real-time connection between geographical data and business data of the data warehouse. In particular:

- The GEO engine (SpagoBIGeoEngine) uses a static catalogue in order to display data, allowing users to dynamically re-aggregate information, according to geographical hierarchies (e.g., nation, country, city). This engine can also be used in a broader context, beyond the strictly geographical context: indicators distribution can be displayed on any graphical structure (such as process flow charts or topological schemes of hardware infrastructures).
- the GIS Engine (SpagoBIGisEngine) interacts with real spatial systems, according to the WFS/WMS scheme. This engine integrates the open source GEO Report solution.

Basic concepts

The term Location Intelligence refers to all those processes, technologies, applications and practices capable to join spatial data with business data, in order to gain critical insights, to better support decisional processes and to optimize business activities.

At the technological level, this correlation is the result of the integration between the software systems that manage these two heterogeneous types of data: geographic information systems (GIS), which manage spatial data, and Business Intelligence systems (BI), which manage business data. This integration gives rise to new technological tools supporting decision-making processes, and the analysis on those business data that are directly or indirectly related to a geographic dimension.

Location Intelligence applications significantly improve the quality of users' analysis based on a geographic dimension. Indeed, a Data Warehouse (DWH)

almost always included such information. By representing the geographic distribution of one or more business measures on interactive thematic maps²⁴, users can quickly identify patterns, trends or critical areas, with an effectiveness that would be unfeasible using traditional analytical tools.

Spatial data

The term *spatial data* refers to any kind of information that can be placed in a real or virtual geometric space. In particular, if the spatial data is located in a real geometric space – which is a geometric space that models the real space –, it can be defined as *geo-referenced* data.

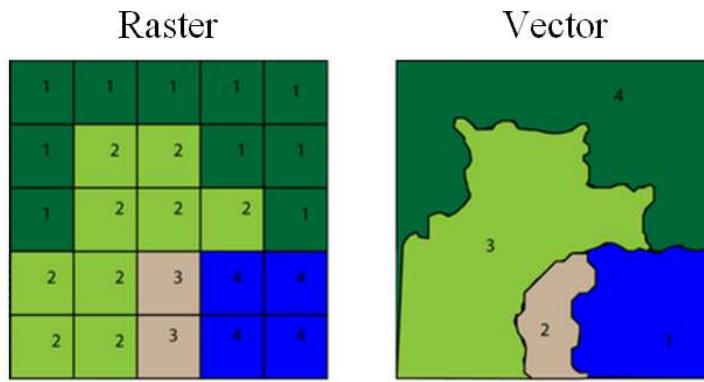


FIGURE 6.109 – A base layer in raster and vector format

Spatial data are represented through graphical objects called *maps*. Maps are a portrayal of geographic information as a digital image file suitable for display on a computer screen.

According to the *Open Geospatial Consortium* (OGC) definition, a map is made of overlapping *layers*: a *base layer* in raster format (e.g. satellite photo) is integrated with other layers (*overlays*) in vector format. Each overlay is made of homogeneous spatial information, which models a same category of objects, called *features*.

²⁴ http://en.wikipedia.org/wiki/Thematic_map

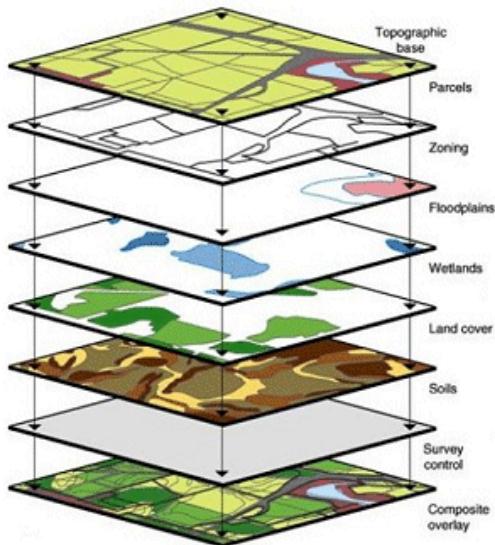


FIGURE 6.110 – Overlapping layer

A feature is called *geographic feature* when the constituting objects are abstractions of real-world physical objects and can be located univocally within a reference coordinate system, according to their relative position.

A feature includes:

- a set of attributes that describes its geometry (vector encoding). Geometric attributes must describe its relative shape and position in an unambiguous way, so that the feature can be properly drawn and located on the map, according to the other features of the layers.
- a set of generic attributes related to the particular type of physical object to be modeled. Generic attributes are not defined: they vary according to the type of abstraction that users want to give to each real-world physical object.

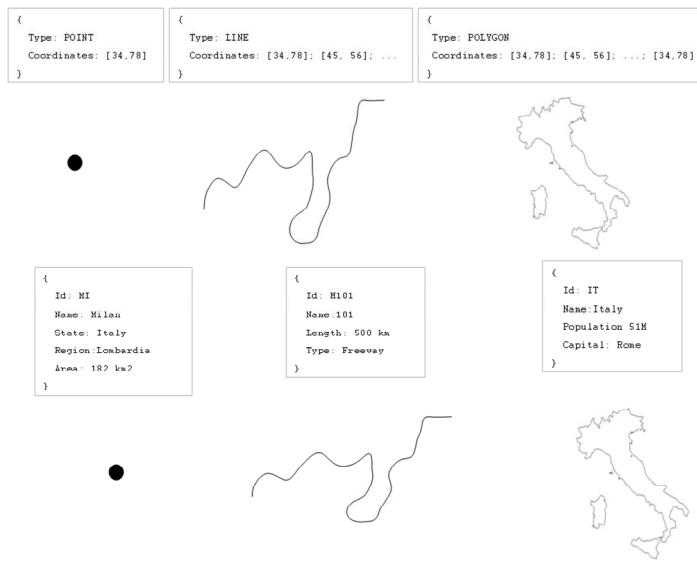


FIGURE 6.111 – Examples of feature

There is a wide range of standards that can be used for the vector encoding of spatial data (e.g. GeoJSON, GML, Shape File, etc.). Most geographic information systems can perform the needed conversions among various encodings.

GIS

Geographic information systems (GIS) provide a set of software tools designed to capture, store, extract, transform and display spatial data²⁵. Therefore, the term GIS refers a set of sole technological components that manage the spatial data during its whole life cycle, starting from the capture of the data up to its representation and re-distribution.

From a logical point of view, the key functionalities of a GIS do not differ from those of a BI system. Both systems are characterized by some specific components supporting the effective storage of data, some others supporting

²⁵ “Principles of Geographical Information Systems” - Peter A. Burrough, Rachael A. McDonnell (1986, Oxford University Press)

their manipulation, their re-distribution or their visualization. On the other hand, the implementation of these functionalities deeply differs between GIS and BI systems, since they deal with two different types of data (alphanumeric and spatial data).

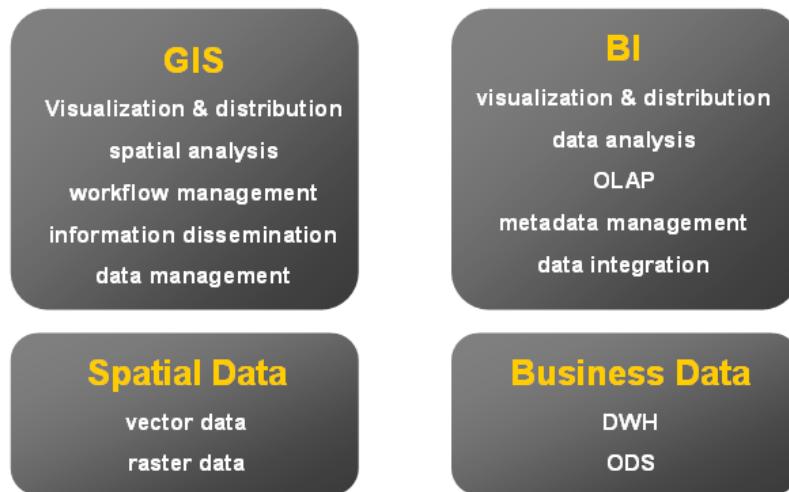
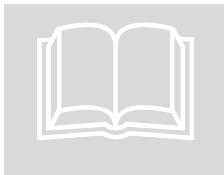


FIGURE 6.112 – Definition of GIS, BI, spatial data and business data

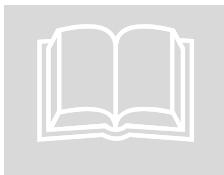
Unlike the market of BI suites, the market of GIS is characterized by a wide spread of open standards, adopted by all main vendors, which regulate the interaction among the various components of the system at all architectural levels. The most important International organization for standardization in the GIS domain is the *Open Geospatial Consortium* (OGC), involving 370 commercial, governmental, non-profit and research organizations. As for the integration between GIS and BI systems, the OGC has defined two main standards supporting the re-distribution of the spatial data:

- the *Web Map Service* (WMS). It describes the interface of services that allow to generate maps in a dynamic way, using the spatial data contained in a GIS.
- the *Web Feature Service* (WFS). It describes the interface of services that allow to query a GIS, in order to get the geographic features in a format that allows their transformation and/or spatial analysis (e.g. GML, GeoJson, etc.).



Open Gesospatial Consortium (OGC)

The most important International organization for standardization in the GIS domain is the Open Geospatial Consortium (OGC), involving 370 commercial, governmental, non-profit and research organizations. Read more at www.opengeospatial.org.



WMS and WFS standards for spatial data distribution

Full documentation about the WMS and WFS standards can be found at www.opengeospatial.org/standards/wms and www.opengeospatial.org/standards/wfs.

SpagoBI suite offers two engines supporting the Location Intelligence analytical area: *Geo* and *Geo Report*. Both engines generate thematic maps. The functionalities allowing the dynamic interaction with the generated maps varies in these two engines, since they implement two different architectures.

SpagoBI**Geo**Engine

The Geo engine integrates the spatial data through a *catalog-based* architectural solution.

Generally speaking, a catalog-based integration focuses on either the BI or the GIS by creating a catalogue of selected information that also stands in for the other system. The catalog is loaded in a static way and provides a set of minimal data manipulation capabilities respect to the system that replaces. Specifically, as far as a BI suite is concerned, the catalog contains a finite set of maps (e.g. SVG, Flash).

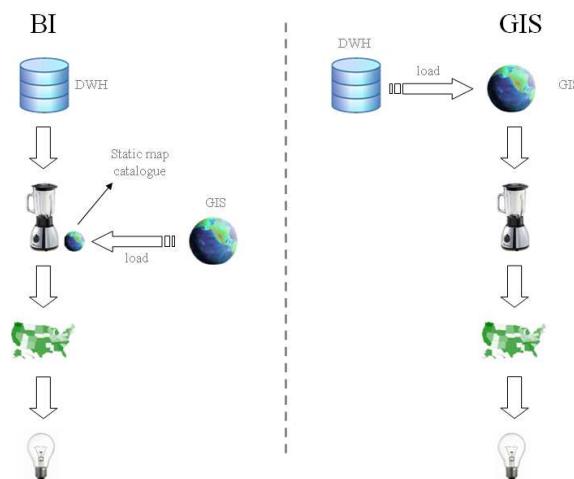


FIGURE 6.113 – Catalog-based integration of spatial and BI data

In particular, the Geo engine implements an architecture based on a catalog of maps, coded in SVG format. In other words, no GIS is required to perform analysis through the Geo engine. The maps can represent any graphical object, even not directly related to a real-world object, such as a process structure diagram.

To create a GIS document, the developer needs to perform the following steps:

- Find (or possibly create) a map in SVG format (spatial data) and register it in SpagoBI Map catalog
- Define a SpagoBI dataset (business data)
- Build a template that describes how the integration between the two types of data included in the map and in the dataset shall be performed.

As for interaction features, the GEO engine allows the visualization and thematization of the map according to the selected indicators as well as *drill down* and *roll up* operations. The user can freely navigate through the geographical dimension, re-aggregating the interested measures at different levels. For instance, the user can pass from the visualization of the measure distribution by region to the visualization of the measure distribution by province (drill down) and vice versa (roll up).

SVG maps

The GEO engine thematizes the maps included in a catalog, where they are stored in SVG format. The first step is therefore to find, create or modify an SVG map for the chosen geographical area.

Any file saved in SVG format is a text file in XML format. As a consequence, they can easily be queried, indexed, enriched with scripts and, if necessary, zipped.



SVG format

The SVG, acronym for Scalable Vector Graphics, refers to an XML-based encoding format, used to describe two-dimensional vector graphical objects. SVG is an open standard, defined by the World Wide Web Consortium (W3C), which released its first version in 1999. Read more at <http://www.w3.org/Graphics/SVG/>.

All recent browsers (Mozilla Firefox, Internet Explorer 9, Google Chrome, Opera and Safari) support the SVG format at different levels. Some older versions of Internet Explorer do not natively support the SVG standard. As a consequence, in order to display contents in this format, they require the installation of a specific plug-in, such as the one provided by Adobe²⁶.

²⁶ Adobe SVG Viewer Plugin (www.adobe.com/svg/viewer/install/main.html)



SVG visualization

The thematic maps in SVG format produced by the GEO engine have been tested using Adobe SVG plug-in, on Explorer and Firefox.

The GEO engine can also be used on native rendering engines of the various browsers. However, in these cases, the proper functioning of all objects included in the thematic map is not guaranteed. For further details, please visit the following page of SpagoBI wiki: http://wiki.spagobi.org/xwiki/bin/view/spagobi_server/ge_o_requirements.

Not all graphical objects of an SVG map can be thematized. Using the SVG grouping operator `<g>`, the developer can create one or more subsets of graphical objects and specify which groups should be subject to thematization. Each group has an unique name, corresponding to the value of the `id` attribute of the `<g>` tag (e.g. `<g id="regions">`).

Considering that graphical objects grouped in a SVG file are usually homogeneous elements (in other words, they model a same category of objects: regions, towns, streets, etc.), we can consider these groups as layers and the objects can be considered as features.

For example, the map shown in FIGURE 6.114 includes two objects groups (layers), both containing two features each:

- `Product_family_labels` and
- `Product_family`

```

1 <svg width="300" height="300"
2 viewBox="0 0 300 300"
3 version="1.1"
4 xmlns="http://www.w3.org/2000/svg">
5
6 <g id="product_family_labels">
7 <text id="food_label" fill="black" x="85" y="50">Food</text>
8 <text id="drink_label" fill="black" x="185" y="50">Drink</text>
9 </g>
10
11 <g id="product_family">
12 <circle id="Food" cx="100" cy="100" r="40" stroke="black"
13 stroke-width="2"/>
14 <circle id="Drink" cx="200" cy="100" r="40" stroke="black"
15 stroke-width="2"/>
16 </g>
17
18 </svg>
19

```

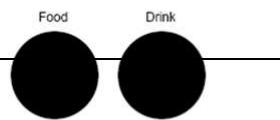
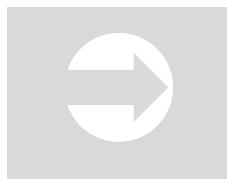


FIGURE 6.114 – Example of two layers included in the map

Once obtained the map, you should register it into SpagoBI map catalogue.



SpagoBI Map catalogue

The Map catalogue contains all maps that can be used in location intelligence documents. It is a cross functionality of SpagoBI. Read how to register a map in the catalogue at section Map Catalogue, chapter 5 – SpagoBI Server.

Template building

The template allows the GEO engine to properly join business data (SpagoBI dataset) and spatial data (map included in the catalog), in order to produce the analytical documents.

In order to describe the structure of the template, we will refer to the map described in the SVG file shown in FIGURE 6.114 and to the query-type dataset shown in FIGURE 6.115.

The screenshot shows the SpagoBI interface for defining a dataset. At the top, there's a 'Detail Data Set' configuration panel with fields for LABEL (TEST_DATASET), NAME (TEST_DATASET), DESC (TEST_DATASET), Transformer (dropdown), TYPE (query), and a large Query text area containing a SQL SELECT statement. Below this is a 'Data Source' dropdown set to 'FoodMart'. At the bottom, there's a 'Predefined List of values details - Test results' table with columns for PRODUCT_FAMILY, STORE_COST, and STORE_SALES, showing data for Food, Non-Consumable, and Drink categories.

PRODUCT_FAMILY	STORE_COST	STORE_SALES
Food	311993.6419	778135.8000
Non-Consumable	83073.4180	207269.5100
Drink	37498.6690	93742.1600

FIGURE 6.115 – Template structure definition

The groups included in the map will be called *layers*; the objects included in the layers will be called *features*.

The map includes two layers:

- `product_family_labels`. The `product_family_labels` layer includes the `food_label` and `drink_label` features.
- `product_family`. The `product_family` layer includes the `Food` and `Drink` features.

The dataset is composed of three columns, two of which are measures (`store_cost` and `store_sales`) while the third one (`product_family`) indicates the grouping class. The dataset includes three datasets, respectively one for each possible grouping class (Food, Drink and Non-Consumable).

We are going to build a template so that the engine is able to produce a thematic map in FIGURE 6.116. In this thematic map, the color intensity of each feature included in the `product_family` layer proportionally increases according to the value of the selected measure (one of the two measures of the dataset) in the corresponding record.

We will also set the template to associate each feature to all the dataset records in which the value of the `product_family` column corresponds to the id of the feature itself.

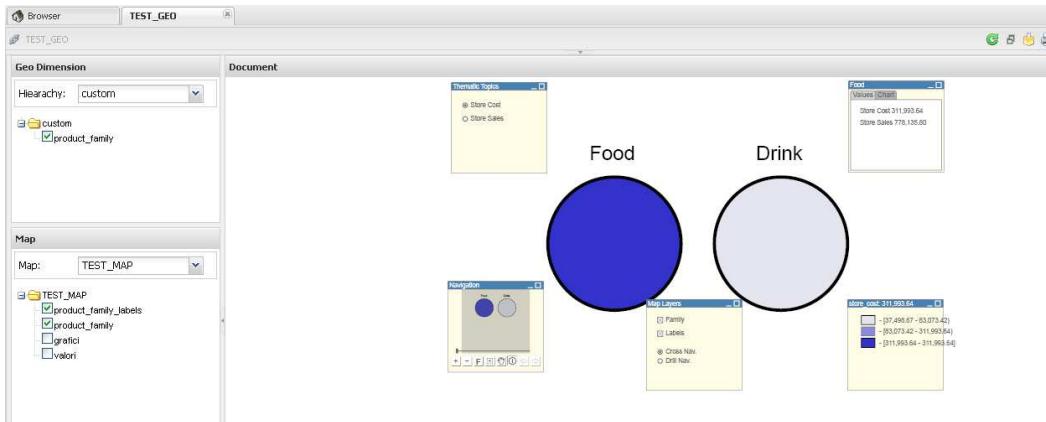


FIGURE 6.116 – Template definition

The template of the GEO engine is an XML file with the following structure.

```
<MAP>
  </MAP_PROVIDER>
  <DATAMART_PROVIDER>
    </METADATA>
    </HIERARCHIES>
  </DATAMART_PROVIDER>
  <MAP_RENDERER>
    </MEASURES>
    </LAYERS>
  </MAP_RENDERER>
</MAP>
```

The three main configuration blocks (`MAP_PROVIDER`, `DATAMART_PROVIDER` and `MAP_RENDERER`) as well as the attributes and configuration sub-blocks will be described in the following paragraphs.

Map provider

The MAP_PROVIDER block allows to specify the name of the map that needs to be thematized, which is located in the catalog. To this end, use the MAP_NAME attribute, as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<MAP>
<MAP_PROVIDER
class_name="it.eng.spagobi.engines.geo.map.provider.SOMapProvider"
map_name="SBI_MAP_GEO_ES01" />
```

Datamart provider

The DATAMART_PROVIDER block and its sub-blocks allow to define the structure of the dataset, including the business data to thematize the map. Specifically, it describes how to associate the dataset records to the features included in the map (METADATA block) and how to re-aggregate the basic database in order to re-calculate the measures on the different levels of the reference geographic dimension (HIERARCHIES block).

The excerpt defining the datamart provider is shown below:

```
<DATAMART_PROVIDER
class_name="it.eng.spagobi.engines.geo.datamart.provider.DataMart
Provider"
hierarchy="custom"
level="product_family">

<METADATA>
    <COLUMN type="geoid" column_id="product_family"
hierarchy="custom" level="product_family"/>
    <COLUMN type="measure" column_id="store_sales"
agg_func="sum"/>
    <COLUMN type="measure" column_id="store_cost"
agg_func="sum"/>
</METADATA>

<HIERARCHIES>
    <HIERARCHY name="custom" type="custom">
```

```

        <LEVEL name="product_family"
column_id="product_family" column_desc="product_family"
feature_name="product_family"/>
    </HIERARCHY>
</HIERARCHIES>

</DATAMART_PROVIDER>

```



Geographic dimension

This term refers to any dimension that is navigable through the map with a spatial representation. This does not necessarily refer to the geographic dimension included in the DWH. For instance, in the document described in this chapter, we will refer to the product dimension.

The **METADATA** block includes a meta description of the columns included in the input dataset. To this end, the user should define the **COLUMN-type** block for each column, using the **column_id** to indicate the name of the column. The other attributes of the **COLUMN** block form the real meta information associated to the column.

The **ttype** attribute can assume the following values:

- **geoid** and
- **measure**.

Each input dataset can include only one **geoid**-type column. The engine uses this column to join the dataset records and the corresponding features in the map.

On the other hand, each input dataset can contain several **measure-type** columns. These columns include the values of the measures which can be used to produce the thematic map. The dataset may also include some columns that are not **geoid**-type, nor **measure**-type. For these columns, you do not need to specify any **COLUMN** sub-block in the **METADATA** block.

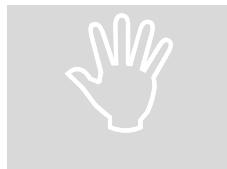
In the geographic dimension, you can define one or more hierarchies. To this end, it is necessary to specify the **geoid**-type column, as well as the **level**

(`level` attribute) and hierarchy (`hierarchy` attribute) to which it has to be associated.

The `geoid`-type column does not necessarily refer to the lowest granularity level in the referenced hierarchy. However, during the execution phase, users are allowed to drill down up to the specific level referenced by this column. Furthermore, in the measure-type columns, users must specify which type of aggregation function shall be used during roll-up operations (`agg_func` attribute).

The `HIERARCHIES` block describes the hierarchies defined along the geographic dimension. Each hierarchy is characterized by a univocal name, one hierarchy type and one or more levels. There are two types of hierarchies:

- `custom`
- `default`.



Hierarchy type

The `default`-type hierarchies supports a hierarchy whose structure was not define directly in the document template but in an external file. Starting from 3.x version, this type is not supported anymore and the only available type is `custom`.

The level of each hierarchy is defined through one or more `LEVEL` tags included in the `HIERARCHY` tag, set out in ascending order according to their granularity level.

Each `LEVEL` tag includes:

- a name (`level_name`)
- a description (`column_desc`)
- a dataset column (`column_id`). This is the column that will be linked to the feature defined below in the map.
- the layer of the referring map (`feature_name`).

For instance, in this case, the engine will set an association between the values of the `store_sales` and `store_cost` measures with the `product_family` column set as "Food" and the features of the `product_family` layer whose id is "Food", as shown in FIGURE 6.117.

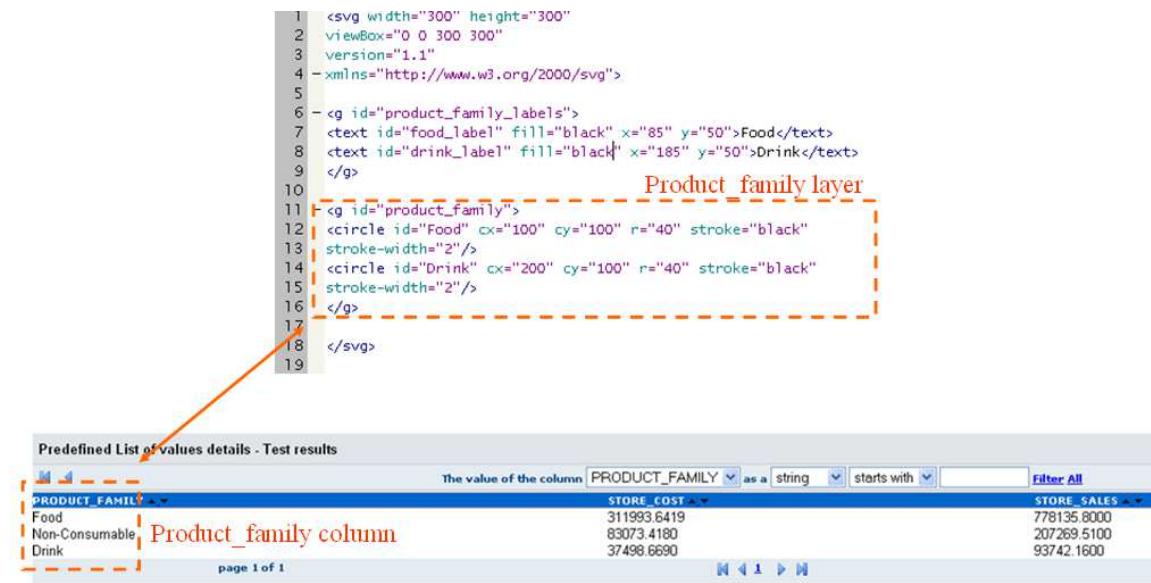


FIGURE 6.117 – Correspondence between dataset measures and SVG layers

The interface of the Geo Engine allows users to dynamically change the selected hierarchy and the aggregation level. However users should specify the data hierarchy and aggregation level in the document template for the first execution, by the `hierarchy` and `level` attributes of the `MAP_PROVIDER` tag.

Map renderer

The `MAP_RENDERER` block and its sub-blocks allow to describe the thematization methods. Specifically, it is possible to define a different thematization method for each measure (KPI tag).

The following two elements can be configured:

- the method allowing to partition the value interval into a finite number of sub-intervals (**THRESHOLDS** tag)
- the method allowing to define the color to be associated to each sub-intervals (**COLOURS** tag).

The engine exploits these two pieces of information to color the features of the layer to be thematized, according to the following procedure:

- the engine associates the features and the raw of the input dataset, according to the mapping rule specified in the **LEVEL** tag of the selected level, within the selected hierarchy;
- the engine sets the value sub-interval to be associated to the dataset raw of each feature;
- the engine uses the color associated to the sub-interval and colors the feature accordingly.

Below you can see an example of MAP_RENDERER definition in the document template.

```
<MAP_RENDERER
class_name="it.eng.spagoobi.engines.geo.map.renderer.InteractiveMa
pRenderer">

    <MEASURES default_kpi="store_cost">
        <KPI column_id="store_sales" description="Store
Sales" agg_func="sum" colour="#E19696" >
            <THRESHOLDS type="quantile" lb_value="0"
ub_value="none" >
                <PARAM name="GROUPS_NUMBER" value="5" />
            </THRESHOLDS>
            <COLOURS type="grad"
outbound_colour="#FFFFFF" null_values_color="#CCCCCC" >
                <PARAM name="BASE_COLOR"
value="#009900" />
            </COLOURS>
        </KPI>
        <KPI column_id="store_cost" description="Store Cost"
agg_func="avg" colour="#9696B9" >
            <THRESHOLDS type="quantile" lb_value="0"
ub_value="none" >

```

```

        <PARAM name="GROUPS_NUMBER" value="5"
/>
    </TRESHOLDS>
    <COLOURS type="grad"
outbound_colour="#FFFFFF" null_values_color="#CCCCCC" >
        <PARAM name="BASE_COLOR"
value="#3333CC" />
    </COLOURS>
</KPI>
</MEASURES>

<LAYERS>
    <LAYER name="product_family" description="Family"
selected="true" default_fill_color="#4682B4" />
        <LAYER name="product_family_labels"
description="Labels" selected="true" default_fill_color="#4682B4"
/>
    </LAYERS>

</MAP_RENDERER>

```

- The **KPI** block identifies the measure to which the thematization method refers, which is defined by the **column_id** attribute.
- The value of the **column_id** attribute should correspond to the name of the column of the input dataset that includes the measure. Moreover, this column must have previously been defined as a **measure**-type column, using the **METADATA** tag. In the **KPI** tag, the user must always define the **TRESHOLDS** and **COLOURS** meta-tag.

The following table includes the heuristics supporting value interval partition into a finite number of sub-intervals (**type** attribute of the **TRESHOLDS** tag).

Type	Description
static	Static : it partitions the interval into smaller fixed-size subintervals, statically defined by the RANGE parameter <TRESHOLDS type="static" lb_value="0" ub_value="none" > <PARAM name="range" value="0,256,512,1024" /> </TRESHOLDS>
quantile	Quantiles : it partitions the interval into N quintiles. The exact amount of quintiles to be created is defined by the GROUPS_NUMBER parameter

	<pre><TRESHOLDS type="quantile" lb_value="0" ub_value="none" > <PARAM name="GROUPS_NUMBER" value="5" /> </TRESHOLDS></pre>
perc	<p>Percentage : it partitions the interval into subintervals whose extent represents a specific fraction of the overall interval extent. The extent of each single subinterval is defined by the RANGE parameter.</p> <pre><TRESHOLDS type="perc" lb_value="0" ub_value="none" > <PARAM name="range" value="30,20,30,20" /> </TRESHOLDS></pre>
uniform	<p>Uniform : it partitions the interval into N subintervals of a same extent. The exact number of sub-intervals is defined by the GROUPS_NUMBER parameter.</p> <pre><TRESHOLDS type="uniform" lb_value="0" ub_value="none" > <PARAM name="GROUPS_NUMBER" value="4" /> </TRESHOLDS></pre>

TABLE 6.23 - Heuristic for value interval partition

For each heuristics, users can specify two threshold values (`lb_value` and `ub_value`) outside of which no value is considered.

The following table defines the heuristics supporting color definition for each value sub-interval (`type` attribute of the `COLOURS` tag).

Type	Description
static	<p>Static: it assigns each sub-interval a specific color that is statically defined, through the RANGE parameter</p> <pre><COLOURS type="static" null_values_color="#FFFFFF" > <PARAM name="range" value="#CCD6E3,#6699FF,#4a7aaaf,#283B64" /> </COLOURS></pre>
grad	<p>Gradient : it assigns each sub-interval a specific color that is dynamically calculated through a gradient function, which progressively scales the base color intensity. This is defined through</p>

	<p>the BASE_COLOR parameter</p> <pre><COLOURS type="grad" outbound_colour="#CCCCCC" null_values_color="#FFFFFF" > <PARAM name="BASE_COLOR" value="#3333CC" /> </COLOURS></pre>
--	--

TABLE 6.24 - Heuristics supporting color interval definition

For each heuristics, users can also specify the color to be assigned to the values that fall outside of the acceptability thresholds, through the `lb_value` and `ub_value` attributes of the `THRESHOLDS` block (`outbound_color` attribute), as well as the color to be assigned to the features included in the map that have no valid value in the dataset (`null_values_color`).

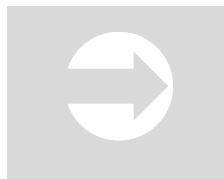
Sometimes users need to color the map and, at the same time, to continue to see the underlying objects, through a transparency effect (e.g. a raster image). In this case, specify the `opacity` parameter in order to properly regulate the transparency level of colors (1 = no transparency; 0 = invisible).

In addition to the `MEASURES` configuration sub-block, which allows to define the thematization methods to be used for each measure, the `MAP_RENDERERS` block also includes the `LAYERS` sub-block, which allows to specify which layers shall be included into the map at first execution of the document. These layers can be hidden or made visible at any time, without necessarily re-start the analysis.

Creation of a GEO document

Now we have all the necessary elements to develop a new GEO analytical document: map, dataset, template. Check that the engine is properly installed and configured at **Resources > Engine management**.

If this does not work, go to <http://myhost:myport/SpagoBIGeoEngine> and check that the `webapp` (/SpagoBIGeoEngine) which implements the GEO engine functionalities is working. Once checked, go back to the engine list and add the GEO engine manually.



Engine configuration

Engines configuration parameters are summarized at section Engines Management, in chapter 5 – SpagoBI Server .

At this point, the SVG map, included in the map catalog can be deployed.



Map catalog

Please refer to section Map catalog, at chapter 5 – SpagoBI Server for details about registering a map on SpagoBI Server.

The dataset should also be created before creating the analytical document.



Dataset Definition

To learn how to build a dataset of type query, please refer to section Data Sets in chapter 5 – SpagoBI Server.

To create the analytical document, select **Analytical Model > Document Development** and click the **Add document** button.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Document, chapter 5 – SpagoBI Server.

The results of the document execution is shown in FIGURE 6.118.

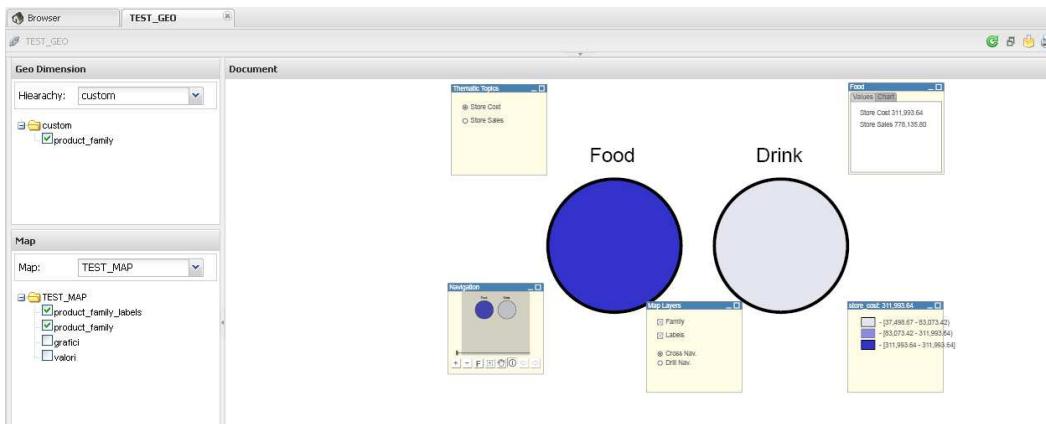


FIGURE 6.118 – Document execution results

Geo Engine advanced functionalities

The next paragraphs provide an overview on the advanced functionalities of the Geo engine, including:

- the definition of multilevel hierarchies
- the contextual cross navigation towards detailed documents
- the customization of elements in the graphical user interface not directly related to thematization.

Multilevel hierarchies

As mentioned above, the GEO engine allows to define multilevel hierarchies, allowing users to navigate the geographic dimension dynamically, through *drill down* and *roll up* operations.

In the previous example, we have used the product dimension as the geographic dimension in order to spatially represent some measures on the product sales. Now, in addition to the already existing `product_family` level, we will add a new one (`product_department`) to the hierarchy. Users can freely navigate through these levels, using the developed thematic map.

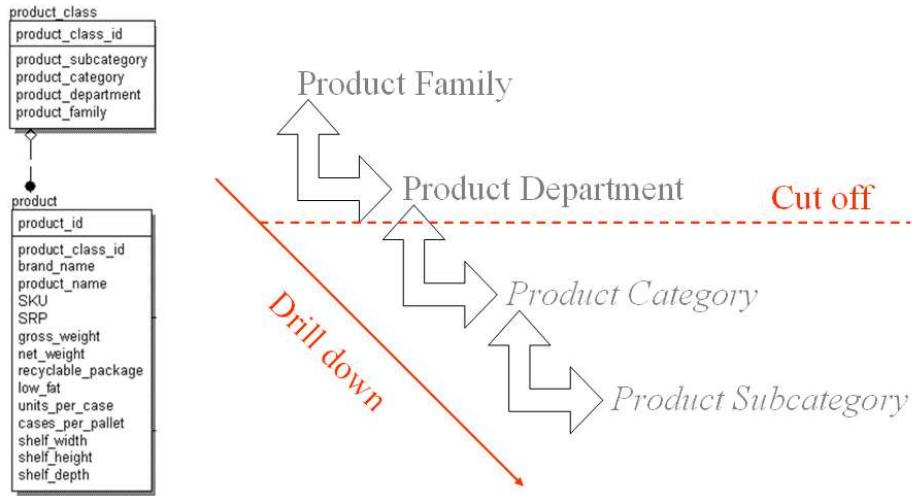


FIGURE 6.119 – Dynamic navigation through the geographic dimension

In order to build the original document, we have followed a bottom up approach: we started from the definition of the input elements (map and dataset), in order to come to the definition of the template. This time we will adopt a top down approach: starting from the modification of the template to come to the modification of the input elements (map and dataset).

In the template, only the `HIERARCHIES` block shall be modified, by adding a new level, as shown below:

```
<HIERARCHIES>
    <HIERARCHY name="custom" type="custom">
        <LEVEL name="product_department"
column_id="product_department" column_desc="product_department"
feature_name="product_department"/>
        <LEVEL name="product_family" column_id="product_family"
column_desc="product_family" feature_name="product_family"/>
    </HIERARCHY>
</HIERARCHIES>
```

The new `product_department` level refers to the column of the `product_department` input dataset, which does not exist yet. Moreover, it refers to the `product_department` layer, which is not available in the input map yet.

Let us start by encoding the dataset as shown in FIGURE 6.120, by adding another grouping level and consequently reducing data granularity.

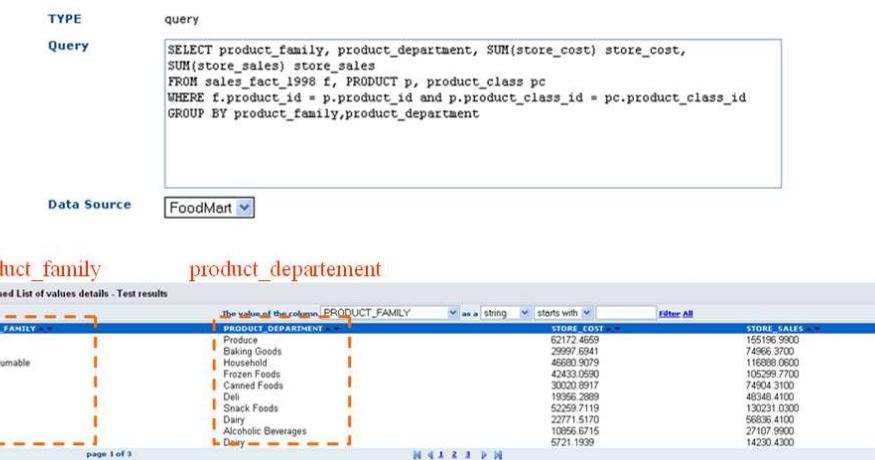


FIGURE 6.120 – Encoding the dataset

We create a new SVG map that include the `product_department` layer. To this end, we cannot simply add a new layer to the existing map, since it does not respect the partitioning of features of the upper-level layer. Therefore, displaying both layers on a same map would not be effective.

Once the new map is registered in the catalog, the analytical document can be re-executed, in order to navigate through the product hierarchy.

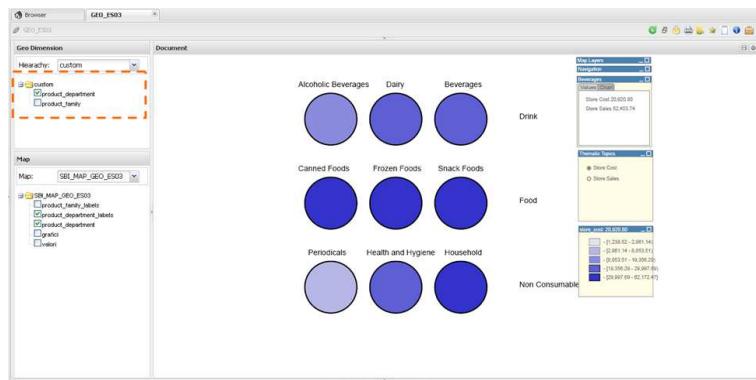
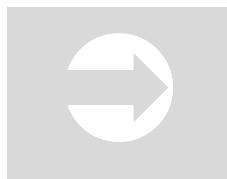


FIGURE 6.121 – Re-executing the analytical document and navigating through the product hierarchy

Cross navigation

The GEO engine allows to define cross-navigation paths. Clicking on the different parts of the map, users can navigate through the detailed documents and filter them according to the selected region.



Cross Navigation

To understand what cross navigation is and how it works, please refer to the corresponding section in chapter 5 – SpagoBI Server.

To define the navigation paths, add a template to the document and include the **CROSS_NAVIGATION** block into the **DATAMART_PROVIDER** block, as shown below:

```
<CROSS_NAVIGATION>

    <!--Product department level-->
    <LINK hierarchy="custom" level="product_department">
        <PARAM type="absolute" name="DOCUMENT_LABEL"
value="RPT_CUSTOMER_DETAIL" />
        <PARAM type="absolute" name="ParAgeGroup" value="F30-40"
/>
        <PARAM type="relative" scope ="dataset"
name="ParDepartment" value="product_department" />
    </LINK>

    <!-- Product family level -->
    <LINK hierarchy="custom" level="product_family">
        <PARAM type="absolute" name="DOCUMENT_LABEL"
value="TEST_CROSS OLAP" />
        <PARAM type="relative" scope ="dataset" name="family"
value="product_family" />
    </LINK>
</CROSS_NAVIGATION>
```

To define new cross-navigation paths, add a **LINK** tag into the **CROSS_NAVIGATION** tag, for each single path. Each level (**level** attribute) of each hierarchy (**hierarchy** attribute) may have different cross-navigation paths.

For each **LINK** tag, you can specify the order of parameters when they will be passed to the service that manages the cross-navigation operations (**PARAM** tag).

Parameters, defined as attribute/value pairs, can be defined

- statistically (**type** attribute set as **absolute**) or
- according to the specific execution environment (**type** attribute set as **relative**).

In the latter case, users can specify whether the parameter value should be read

- by the dataset during the document execution (**scope** attribute set as a “**dataset**”) or
- by the input parameters (**scope** attribute set as “**environment**”).

When the attribute is set as relative, the value attribute is not a real parameter value, but it corresponds to the name of the column that includes the values to be read (**scope** equal to **dataset**) or the name of the input parameter that includes the values to be read (**scope** equal to **environment**). In any case, each link shall include the **DOCUMENT_LABEL** parameter set as **absolute**, whose value indicates the label of the document registered in SpagoBI which you want to navigate.

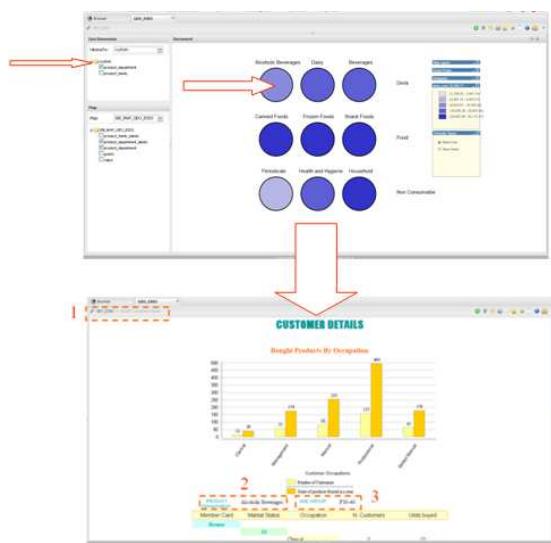


FIGURE 6.122 – Cross navigation starting from a geo document

GUI Settings

The `GUI_SETTINGS` tag included in the `MAP_RENDERER` tag allows to define some properties of the user interface and the types of visualization of the analysis. In particular, users can partition this configuration into different parts: properties of the control windows (`WINDOWS` block), label properties (`LABELS` block) and generic properties (`PARAM` block).

In the `WINDOWS` configuration block, the developer can specify the style properties of the control window that appears in the map.

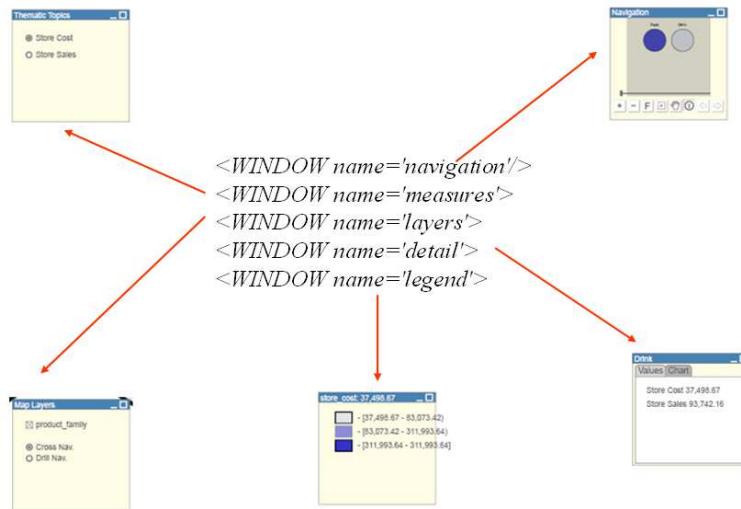


FIGURE 6.123 – Windows configuration block

Moreover, it is possible to use the `DEFAULTS` configuration block to modify the settings common to all the windows, as shown below:

```

<GUI_SETTINGS>
  <WINDOWS>
    <DEFAULTS>
      <PARAM name='visible'>true</PARAM>
      <PARAM name='y'>3</PARAM>
      <PARAM name='transform'>scale(1.0)</PARAM>
      <PARAM name='minimized'>true</PARAM>
      <PARAM name='styles'>
        <![CDATA[ 
          
```

```

        {
            winPlaceholderStyles:
            {"fill": "none", "stroke": "dimgray", "stroke-width": 1.5}
                , windowStyles:
            {"fill": "#fffce6", "stroke": "dimgray", "stroke-width": 1}
                , titlebarStyles:
            {"fill": "steelblue", "stroke": "dimgray", "stroke-width": 1}
                , titlebarHeight: 17
                , statusbarStyles:
            {"fill": "aliceblue", "stroke": "dimgray", "stroke-width": 1}
                , statusbarHeight: 13
                , titletextStyles: {"font-
family": "Arial,Helvetica", "font-size": 14, "fill": "white"}
                , statustextStyles: {"font-
family": "Arial,Helvetica", "font-size": 10, "fill": "dimgray"}
                , buttonStyles:
            {"fill": "steelblue", "stroke": "white", "stroke-width": 2}
                }
            ]]>
        </PARAM>
    </DEFAULTS>

<WINDOW name='navigation'>
    <PARAM name='visible'>true</PARAM>
    <PARAM name='title'>Navigation</PARAM>
</WINDOW>

<WINDOW name='measures'>
    <PARAM name='visible'>true</PARAM>
    <PARAM name='title'>Measures</PARAM>
</WINDOW>

```

The following table provides a complete list of configuration parameters to be used in the control windows.

Configuration parameters for the GUI window		
Parameter	Type	Description
visible	boolean (true false):	It indicates whether the Window should be created.
width	number	Width of the Window in viewBox coordinates.
height	number	Height of the Window (incl. title and status bar) in viewBox coordinates.

x	number	The position of the left edge of the window in viewBox coordinates.
y	number	The position of the upper edge of the window in viewBox coordinates.
movable	boolean (true false):	It indicates whether the Window may be moved or not.
xMin	number	The left constraint, the constraints define the area where the Window can be moved within.
yMin	number	The upper constraint.
xMax	number	The right constraint.
yMax	number	The lower constraint.
showContent	boolean (true false):	Value may hold true or false, indicates whether the Window content should be visible or hidden during window movements.
margin	number	A number in viewBox coordinates describing a margin. Used e.g. for placing text and buttons in statusBar or titleBar.
titleBarVisible	boolean (true false):	It indicates whether the Window should have a title bar.
statusBarVisible	boolean (true false):	It indicates whether the Window should have a status bar.
title	String or undefined	A string specifying the Window title text.
statusBarContent	String or undefined	A string or undefined value specifying the Window status text.
closeButtonVisible	boolean (true false):	It indicates whether the Window should have a closeButton.
minimizeButtonVisible	boolean (true false)	It indicates whether the Window should have a minimizeButton.
maximizeButtonVisible	boolean (true false)	It indicates whether the Window should have a maximizeButton.
minimized	boolean (true false)	It indicates whether the Window should be

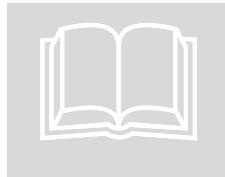
		minimized.
transform	String	It lists the transformations to apply to the windows. Please refer to SVG specifications for more information about valid transformations.
styles	JSON	<p>It indicates the styles to apply to the different components of the window</p> <pre>{ winPlaceholderStyles: {"fill":"none","stroke":"dimgray","stroke-width":1.5} , windowStyles: {"fill":"#fffce6","stroke":"dimgray","stroke-width":1} , titlebarStyles: {"fill":"steelblue","stroke":"dimgray","stroke-width":1} , titlebarHeight: 17 , statusbarStyles: {"fill":"aliceblue","stroke":"dimgray","stroke-width":1} , statusbarHeight: 13 , titletextStyles: {"font-family":"Arial,Helvetica","font-size":14,"fill":"white"} , statustextStyles: {"font-family":"Arial,Helvetica" ,"font-size":10 ,"fill":"dimgray"} , buttonStyles: {"fill":"steelblue","stroke":"white","stroke-width":2} }</pre>

TABLE 6.25 - Configuration parameters for GEO Engine GUI window

In the **LABELS** configuration block, users can specify the labels to be added to the header (left, center, right) and/or in the footer (left, center, right) of the map. Each label, according to its position, has a dedicated configuration block.

Each label is generated at run-time by an object that implements the **ILabelProducer** interface. This implementation allows users to generate text strings (**TEXT** tag) that may include the date (**FORMAT** tag).

```
<LABELS>
    <LABEL position="footer-right"
class_name="it.eng.spagobi.engines.geo.map.renderer.DateLabelProducer">
        <FORMAT day="dd/MM/yyyy" hour="HH:mm"/>
        <TEXT>Last update ${day} at ${hour}</TEXT>
        <PARAM name='font-size'>32px</PARAM>
        <PARAM name='fill'>red</PARAM>
    </LABEL>
</LABELS>
```



SVG allowed parameters

You can retrieve the complete list of possible parameters in the SVG specification: <http://www.w3.org/Graphics/SVG/>

Finally, in the **GUI_SETTINGS** configuration block, users can define a set of generic parameters, which are used by the engine during the execution phase:

```
<PARAM name='defaultDrillNav'>false</PARAM>

<PARAM name='highlightOnMouseOver'>true</PARAM>
<PARAM name='normalizeChartValues'>true</PARAM>
<PARAM name='chartScale'>1.0</PARAM>
<PARAM name='chartWidth'>80</PARAM>
<PARAM name='chartHeight'>160</PARAM>
<PARAM name='valueFont'>12px</PARAM>
<PARAM name='valueScale'>1.0</PARAM>
```

The complete list of these parameters is listed in the following table.

Parameter name	Description
defaultDrillNav	It indicates whether the initial navigation modality shall be cross-type or drill-type.
highlightOnMouseOver	It indicates whether the areas of the map shall color when the mouse passes over them
normalizeChartValues	It indicated whether the values of the graphs shall be normalized on a 0-100 scale (this is useful when the various measures use different units of measurement and/or highly variable ones)
chartScale	Scale factor to be applied to the graphs plotted on maps (1.0 = original dimensions)
chartWidth	Width of the graphs plotted on the map
chartHeight	Height of the graphs plotted on the map
valueFont	Font to be applied to the values of the measures plotted on the map
valueScale	Scale factor to be applied to the values plotted on the map (1.0 = original dimensions)

TABLE 6.26 - List of parameters to be used during the GUI setting process

SpagoBIGEOResportEngine

The GEOResport engine implements a *bridge integration* architecture.

Generally speaking, a bridge integration involves both the BI and the GIS systems, still keeping them completely separated. The integration between spatial data and business data is performed by a dedicated application that acts as a *bridge* between the GIS and the BI suite. This application extracts the spatial data from the GIS system and the business data from the BI suite, to answer the users' requests. Afterwards, it joins them and provides the desired results.

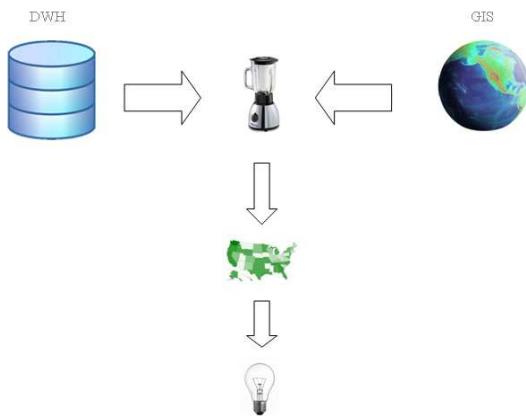


FIGURE 6.124 – Bridge integration architecture of the GEOREport engine

In particular, the GEOREport Engine extracts spatial data from an external GIS system and join them dynamically with the business data extracted from the Data Ware House, in order to produce a thematic map, according to the user's request. In other words, it acts as a *bridge* between the two systems, which can consequently be kept totally decoupled.

The thematic map is composed of different overlapping layers that can be uploaded from various GIS engines at the same time. Among them just one layer is used to produce the effective thematization of the map: this is called *target layer*.

If specified in the template, the GEOREport engine can also read the target layer from a GeoJSON file, stored in the /georeport sub-folder of the Resources folder of SpagoBI. In addition, it can also work as an engine based on a catalog architecture, without using any GIS.

There are no constraints on the encoding of the various layers or on the standard used to upload them from the external GIS system (WFS, WMS, etc.), except the target layer, which shall be encoded in GeoJSON format and read through a WFS call.

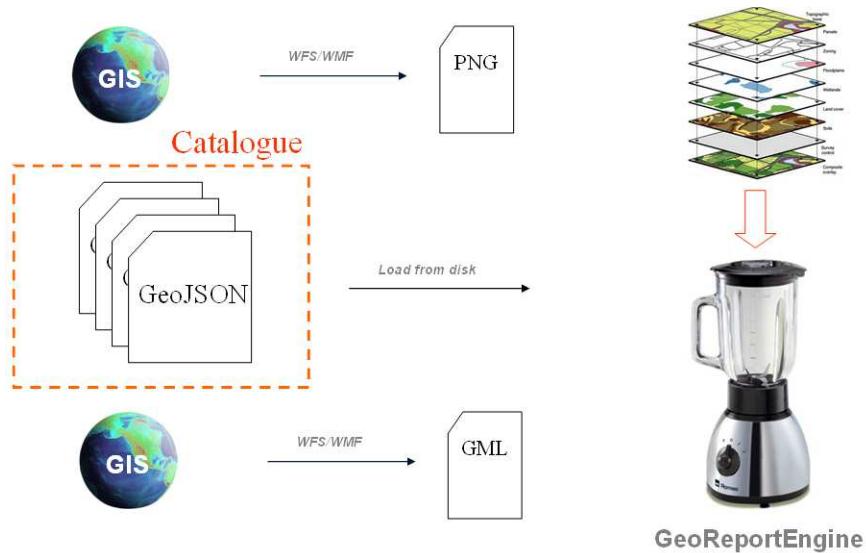


FIGURE 6.125 – Input elements to produce a thematic map

From a functional point of view, the engine uses three input elements to produce thematic maps:

- a map, i.e., a set of layers, one of which – the target layer – is encoded in GeoJSON format (spatial data)
- SpagoBI dataset (business data)
- a template that mainly describes how these two types of data (map and dataset) can be matched.

As for interaction features, the GEOREport engine can exploit the potentials of spatial data to perform the typical operations supported by modern web GIS, such as the dynamic inclusion of information layers, zoom in/out, pans, area and distance calculation.

Unlike the GEO engine, the GEOREport engine does not allow to perform drill operations on the geographic dimension. This engine aims to extend the functionalities provided by a common reporting engine, in order to allow the spatial visualization of spatial data, stored in any flat dataset – a dataset that potentially has no hierarchical structure.

Creating the map in GeoJSON format

GeoJSON is an open standard of spatial data. Any geometric object that is encoded in GeoJSON is first of all a JSON object. This is particularly useful in order to easily manipulate spatial data in this format through scripting client-side languages (such as Javascript). Furthermore, this format allows users to make data encoding much more compact than XML-based formats.



GeoJSON

GeoJSON is an open standard of spatial data . Its name comes from the fact that it is based on the JSON (JavaScript Object Notation) standard. Read more at www.geojson.org .

Unlike the GEO engine, the GeoReport engine does not read the whole map on the source systems. Actually, it reads a set of elementary layers that are composed during the drawing phase, to produce and display the final map.

As mentioned in the previous paragraphs, the various layers that compose the final map can be read even by various sources. The list of layers and their configurations that are used to produce the map, except the target layer, is available in this file:

- SPAGOBI\Webapps\SpagoBIGeoReportEngine\js\src\ext\sbi\commons\Settings.js

On the other hand, the configuration of the target layer is directly specified in the document template, as set forth in the following paragraph. The target layer can be read by an external GIS, through the WFS standard, or by a file encoded in GeoJSON format and stored in the following folder:

- RESOURCES/georeport

An excerpt of a possible GeoJSON file is shown below:

```
{"type": "FeatureCollection",
  "features": [
    {"type": "Feature", "id": "Campania",
```

```
"properties": {"id": "CAMPANIA"},  
"geometry": {  
    "type": "MultiPolygon",  
  
    "coordinates": [[[ [15.335488, 40.833768], [15.340536, 40.827696]  
    , ...  
    , "geometry_name": "store_state_geom" } ], "bbox": [6.623231887817  
38, 35.4903984069824, 18.520544052124, 47.091739654541] }
```

In the first case, the engine works as a bridge architecture, while in the second case it works as a catalog architecture. When working as a bridge architecture, it can interact with any GIS that implements the WFS standard and supports the GeoJSON format to export results.



GIS

The reference GIS server that tests the proper working of the GeoReport engine is called GeoServer²⁷. GeoServer is an open source GIS server written in Java, which supports most OGC standards. For some OGC standards, it is the reference implementation. Another well-known open source GIS server is MapServer²⁸, developed by the University of Minnesota since 1990. Among proprietary solutions, the most popular is ArcGIS Server²⁹ by ESRI.

Template building

The template of the analytical documents executed by the GeoReport engine allows this engine to properly join business data (SpagoBI dataset) and spatial data (target layer) in order to produce and visualize the output map.

²⁷ <http://geoserver.org>

²⁸ <http://mapserver.org>

²⁹ <http://www.esri.com/software/arcgis/arcgisserver/index.html>

In order to describe the structure of the template, we will use the target layer of the `usa_states.json` file, stored in the `RESOURCES/georeport` folder. As shown below, this layer includes the separation of the various States of the USA (52 features, one for each State).

```
1 usa_states.json *
 1
 2   "type": "FeatureCollection"
 3   "features": [
 4     {
 5       "type": "Feature"
 6       "id": "states.1"
 7       "geometry": {
 8         "type": "MultiPolygon",
 9         "coordinates": [[[[-88.071564, 37.51099000000001], [-88.087883, 37.47627300000006], [-88.311707, 37.442852], [-88.35917,
10
11
12       "geometry_name": "the_geom"
13       "properties": {
14         "STATE_NAME": "Illinois"
15         "STATE_FIPS": "17"
16         "SUB_REGION": "E N Cen"
17         "STATE_ABBR": "IL"
18         "LAND_KM": 143986.61
19         "WATER_KM": 1993.335
20         "PERSONS": 1.1430602E7
21
22     }, {
23       "type": "Feature"
24       "id": "states.2"
25       "geometry": {
26         "type": "MultiPolygon"
27         "coordinates": [[[[-77.008232, 38.96655699999995], [-76.911209, 38.88998799999999], [-77.045448, 38.78811999999999], [
28
29       "geometry_name": "the_geom"
30       "properties": {
31         "STATE_NAME": "District of Columbia"
32         "STATE_FIPS": "11"
33       }
34     }
35   }
36 }
```

Then we will use a SpagoBI query dataset, connected to the food mart data source, whose SQL query is shown below:

```
SELECT
    r.sales_state
    , SUM(f.store_sales) store_sales
    , AVG (f.unit_sales) unit_sales
SELECT
    r.sales_state
    , sum(f.store_sales) store_sales
    , avg(f.unit_sales) unit_sales
    ,sum(f.store_cost) store_cost
FROM
    sales_fact_1998 f, store s, sales_region r
WHERE
    s.store_id=f.store_id AND
    s.region_id = r.region_id
GROUP BY
    r.sales_state
```

In this template, we will include information that allows the engine to produce a thematic map identical to the one shown in Figure 59. The color intensity of each feature included in the `usa_states.json` file proportionally increases according to the value of the selected measure (one of the three measures of the dataset) in the corresponding record.

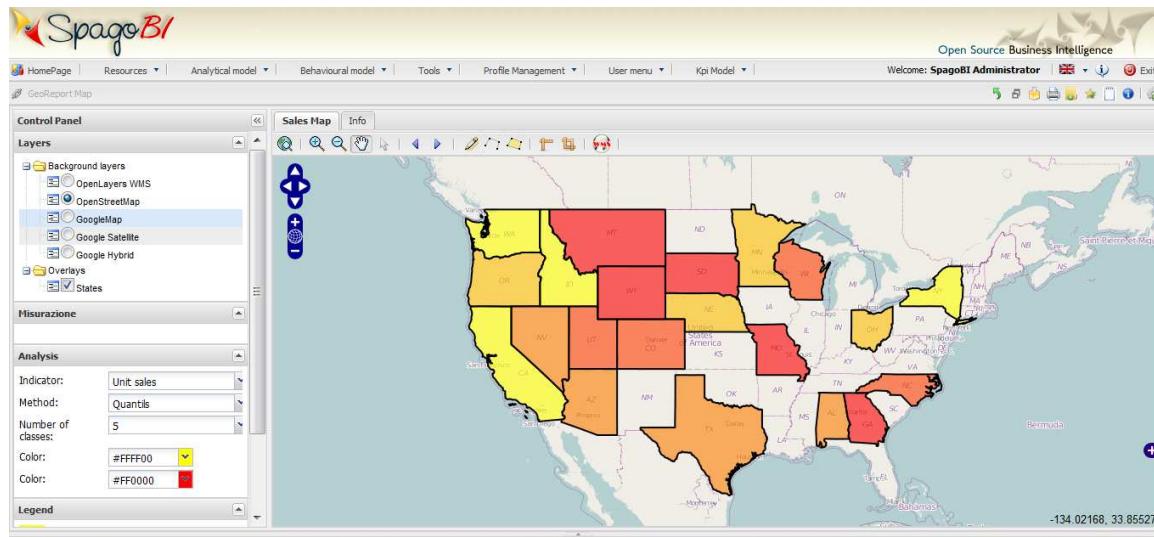


FIGURE 6.126 – The map uses different color intensity according to the value of the selected measures

The template of the GeoReport engine is a JSON file. The key information included in this file are:

- name of the map
- type of analysis
- method for joining spatial data and business data
- measure definition
- definition of the target layer and
- visualization coordinates.

Below you can see an example of a complete template that we'll use to generate the analysis shown in FIGURE 6.126. This template tells the engine to associate each feature to all the dataset records in which the value of the `product_family` column corresponds to the `STATE_ABBR` property of the feature itself.

In the next paragraphs, we describe it in detail.

```
{
    mapName: "Sales Map",
    analysisType: "choropleth",
    indicators: [["unit_sales", "Unit sales"], ["store_sales", "Sales"], ["store_cost", "Cost"]],
    businessId: "sales_state",
    geoId: "STATE_ABBR",
    targetLayerConf: {
        text: 'States',
        name: 'usa_states'
        //, url: 'http://localhost:8080/geoserver/wfs'
        , data: 'usa_states.json'
    },
    lon: -96.800,
    lat: 40.800,
    zoomLevel: 4
}
```

Name of the map

The `mapName` attribute allows to specify the name of the map. It is displayed in the related tab, as shown in FIGURE 6.127.

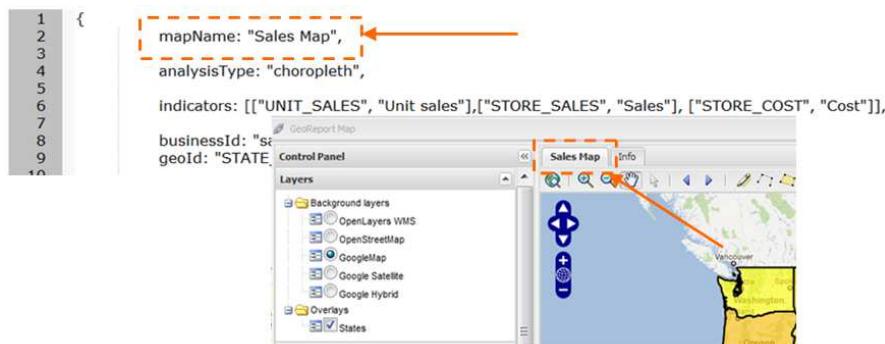


FIGURE 6.127 – Defining the name of the map

Types of analysis

The `analysisType` attribute allows to specify the type of thematization that the user wants to produce³⁰.

The engine supports two types of thematization:

- **Choropleth.** It changes the intensity of fill colors of the features included in the target layer, according to users' needs.
- **proportionalSymbols.** It changes the dimension of graphical objects.

Using the GeoReport engine, the `proportionalSymbols` analysis can be applied to target layers that are composed of features whose geometry is represented by a dot > point. The symbol used to perform the thematization of features is a circle whose center is located in the feature itself and whose radius is proportional to the value of the measure of that feature.

On the other hand, the choropleth analysis can only be applied to target layers that are composed of features whose geometry is represented by a plane figure.



FIGURE 6.128 – Comparison between the choropleth and propoportionalSymbols analysis

³⁰ http://en.wikipedia.org/wiki/Thematic_map#Methods_of_thematic_mapping

Measures

The indicators attribute allows to specify the measures that can be used to perform the thematization of the map.

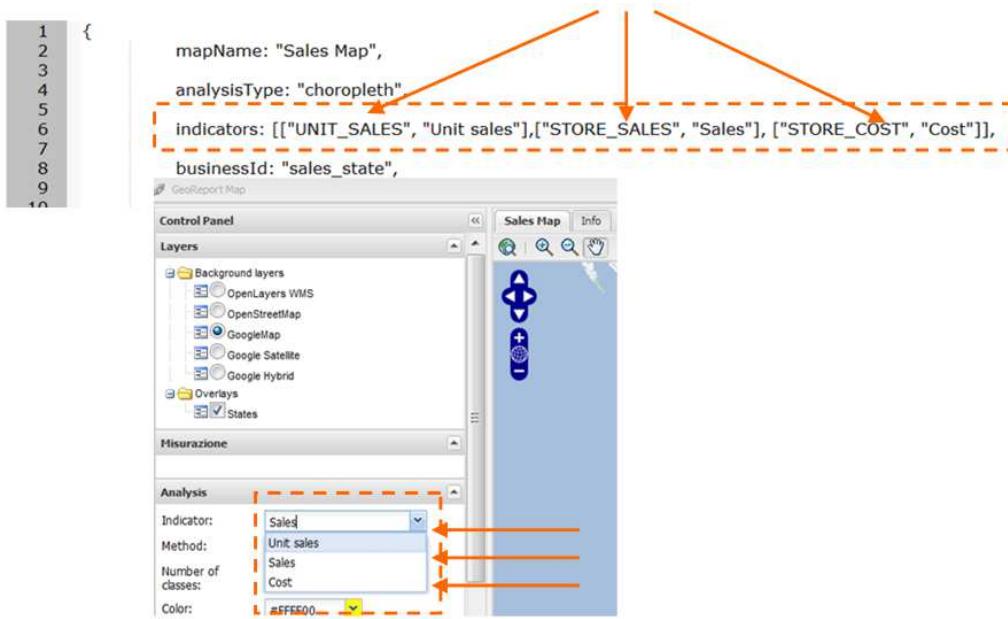


FIGURE 6.129 – Indicator attribute definition

Each measure is defined by an array (e.g. ["UNIT_SALES", "Unit sales"]) in which the first value ("UNIT_SALES") represents the name of the column of the input dataset that includes the measure. The second value ("Unit sales") includes the description of the measures that will be listed in the selection combo, through the engine interface.

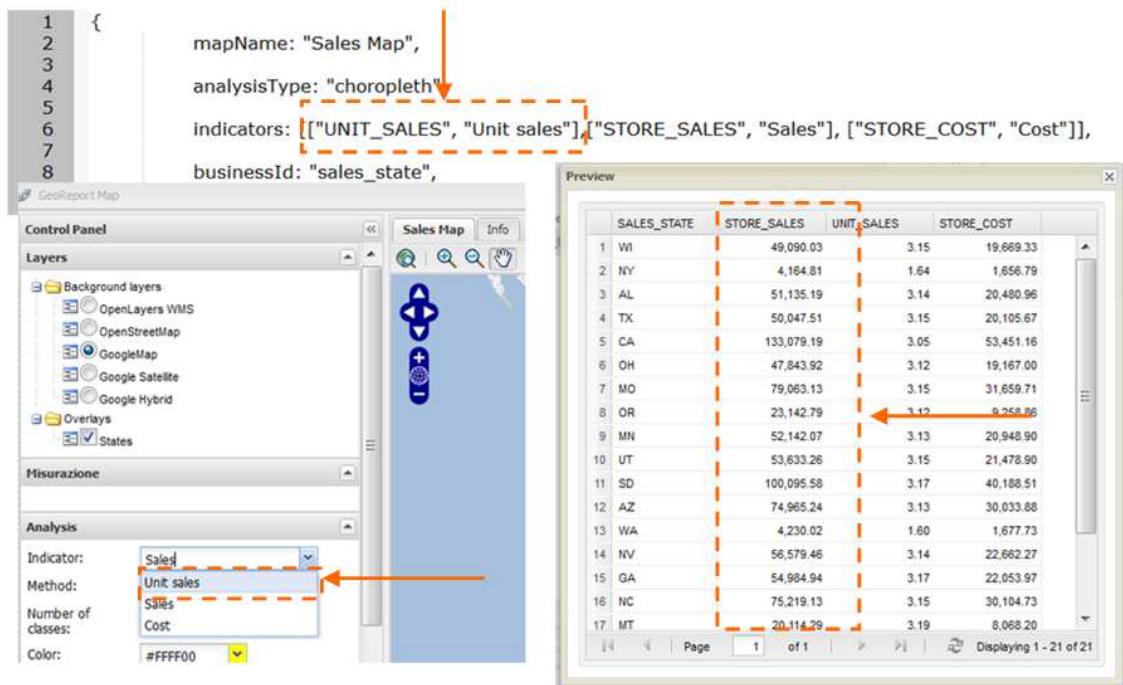


FIGURE 6.130 – Measure definition

How to perform join operations

The **businessId** and **geoId** attributes allow to define how the features included in the target layer and the data of the input dataset shall be joined. To this end, the value of the column included in the input dataset defined as **businessId** is compared with the value of the property of the feature defined as **geoId**.

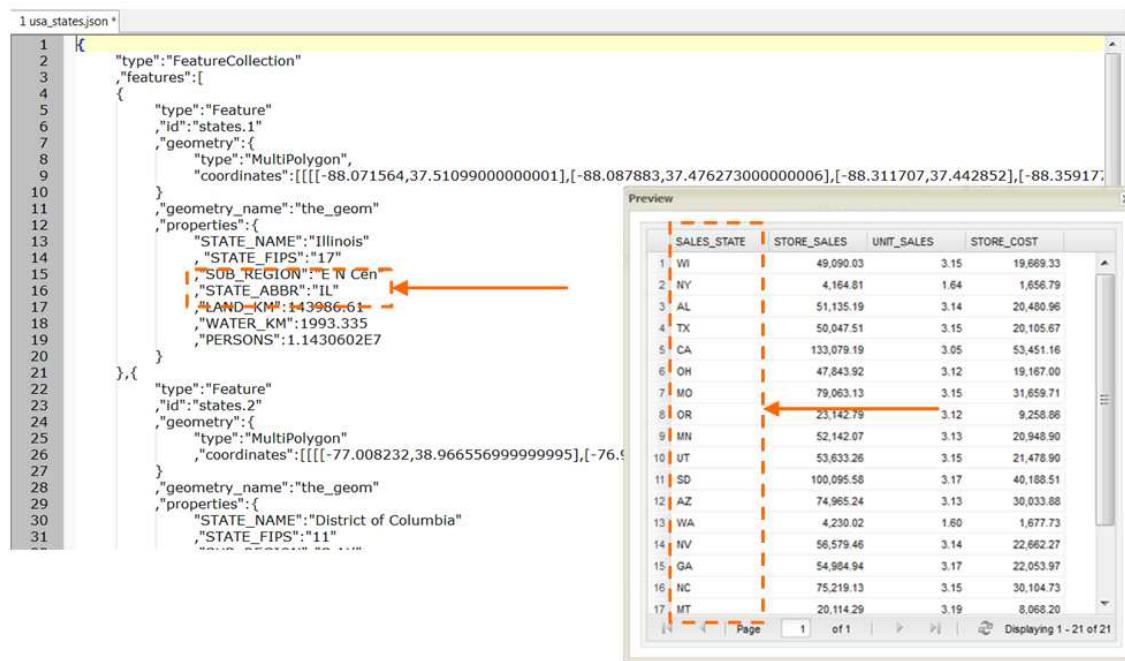


FIGURE 6.131 – Defining how target layer and input dataset shall be joined

Target Layer

The `targetLayerConf` attribute allows to specify the configuration of the del target layer. Specifically, users can specify the name (`name`), description (`text`) and the target layer uploading modality (`url` or `data`).

```
targetLayerConf: {
    text: 'States'
    , name: 'usa_states'
    //, url: 'http://localhost:8080/geoserver/wfs'
    , data: 'usa_states.json'
}
```

As previously mentioned, the target layer can be uploaded from an external GIS using a WFS call or from the RESOURCE/georeport folder. In the first case, users shall specify the endpoint of the WFS services, called through the `url` attribute. In the second case, users shall use the `data` attribute in the RESOURCE/georeport folder, in order to specify the name of the file to be read by the engine.

Visualization coordinates

The `lon`, `lat` and `zoomLevel` attributes allow to respectively specify the longitude, latitude and zoom level of the center of the map, displayed by the engine.

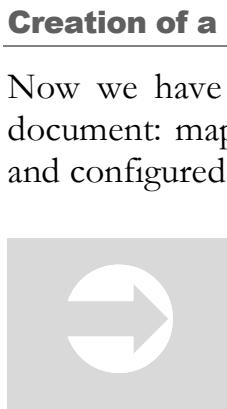
```
lon: -96.800,  
lat: 40.800,  
zoomLevel: 4
```

Latitude and longitude shall be expressed through a EPSG:4326 reference system.



Setting latitude and longitude

If you do not have a clear idea of the values to be used, we suggest keeping the ones proposed in this example, which focus on the USA. Afterwards, re-execute the document, move over the area of your interest and copy the coordinates of the new centre, which are shown at the bottom right of the window.



Creation of a GEOResport document

Now we have all the necessary elements to develop a new GEO analytical document: map, dataset, template. Check that the engine is properly installed and configured at [Resources > Engine management](#).

Engine configuration

Engines configuration parameters are summarized at section Engines Management, in chapter 5 – SpagoBI Server.

Now the usa_states.json file used in our example, including the target layer encoded in GeoJSON format, can be copied into the RESOURCES/georeport folder.



Map catalog

Please refer to section Map catalog, at chapter 5 – SpagoBI Server for details about registering a map on SpagoBI Server.

Add the dataset to those already registered in SpagoBI suite. The dataset should also be created before creating the analytical document.



Dataset Definition

To learn how to build a dataset of type query, please refer to section Data Sets in chapter 5 – SpagoBI Server.

To create the analytical document, select **Analytical Model > Document Development** and click the **Add document** button.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Model, chapter 5 – SpagoBI Server

The results of the execution of the new document are shown in FIGURE 6.132.

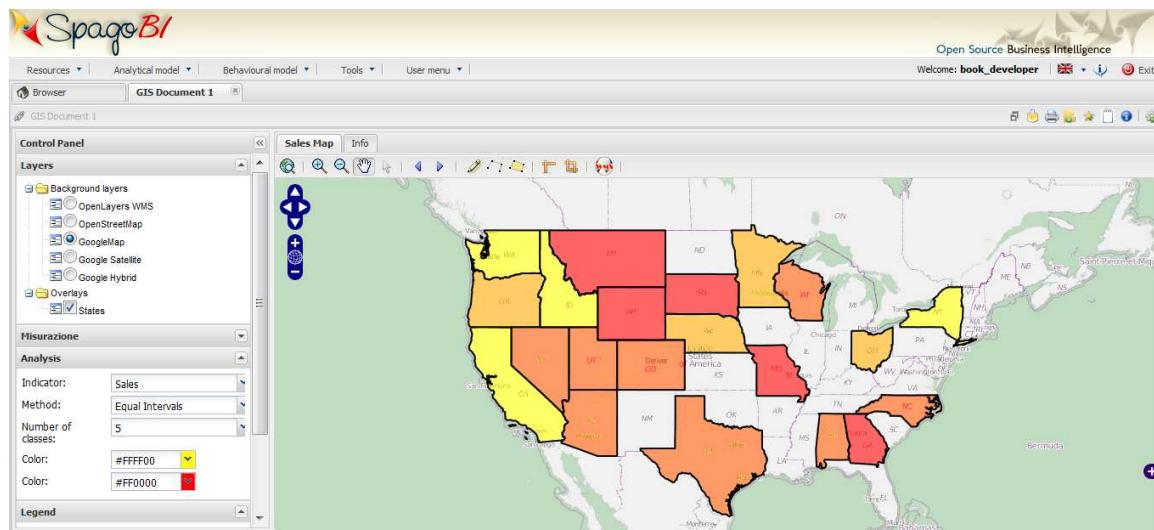


FIGURE 6.132 – Creating an analytical document

GEOReport Engine advanced functionalities

The next paragraphs provide an overview on the advanced functionalities of the GeoReport engine, including the contextual cross navigation towards detailed documents and the customization of the elements of the graphical user interface which are not directly related to thematization.

Inline document

Unlike the GEO engine, the GeoReport engine does not have the standard cross navigation features provided by SpagoBI suite.



Cross Navigation

To understand what cross navigation is and how it works, please refer to the corresponding section in chapter 5 – SpagoBI Server.

Nevertheless, the GeoReport Engine supports the definition of a document that will be displayed in a contextual pop-up window (*inline document*) once the user clicks on a feature of the target layer.

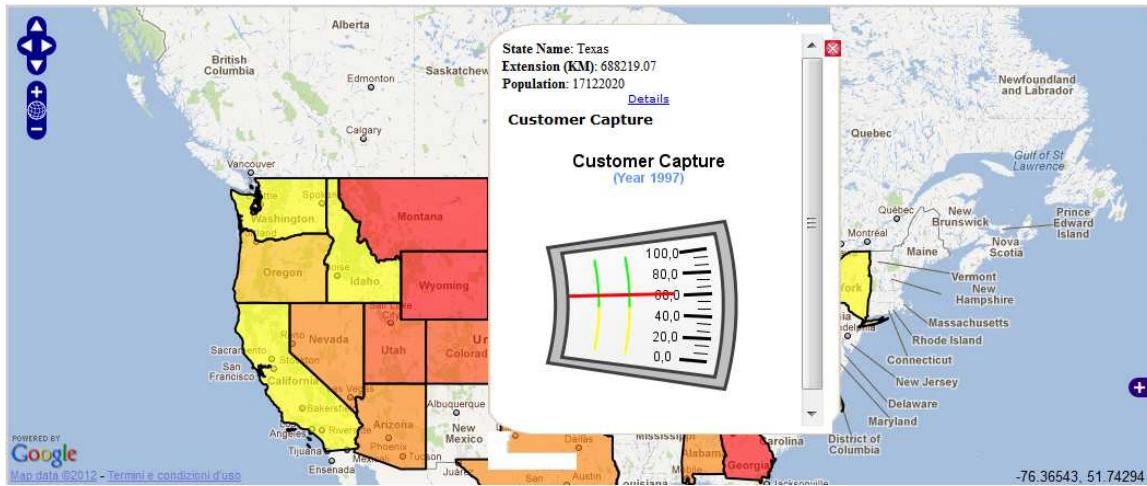


FIGURE 6.133 – Contextual pop-up window

This feature is specified in the template using the `inlineDocumentConf` attribute, as shown below:

```
inlineDocumentConf: {
    label: 'DIALCHART_simpledial'
    , staticParams: {
        param1: 'andrea'
    }
    , dynamicParams: {
        state: 'STATE_NAME'
    }
    , displayToolbar: 'false'
    , displaySliders: 'false'
}
```

Where:

- the `label` attribute refers to the label of the document registered in SpagoBI that shall be displayed in the pop-up window.

- The `staticParams` and `dynamicParams` attributes allow to specify the static parameters (whose values are defined in the template) and dynamic parameters (whose values refer to the properties of the selected features, providing the actual runtime value) to be passed to the document.

Properties shown in the upper part of the pop-up window are directly read by the property of the selected feature. The developer can specify the properties to be displayed in the template, by using the `featureInfo` attribute.

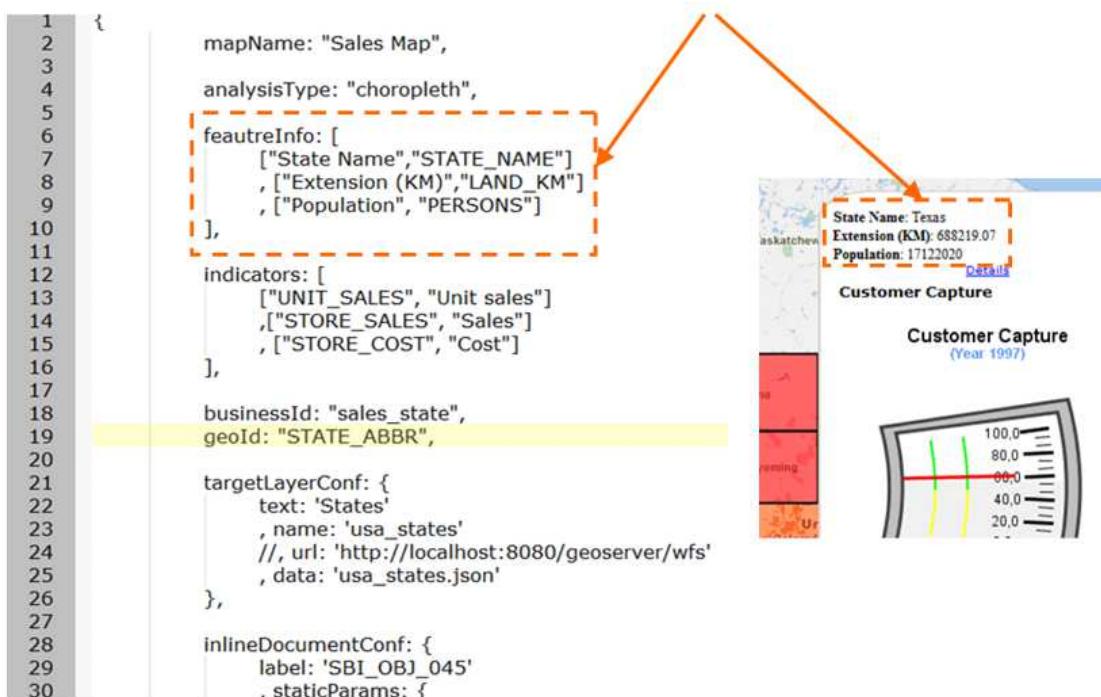


FIGURE 6.134 – Specifying the properties of the pop-up window

The `details` link included in the pop-up window opens another detailed document in the `info` tab. This document is specified in the template through the `detailDocumentConf` attribute, whose configuration parameters are the ones used to configure the `inlineDocumentConf` attribute.

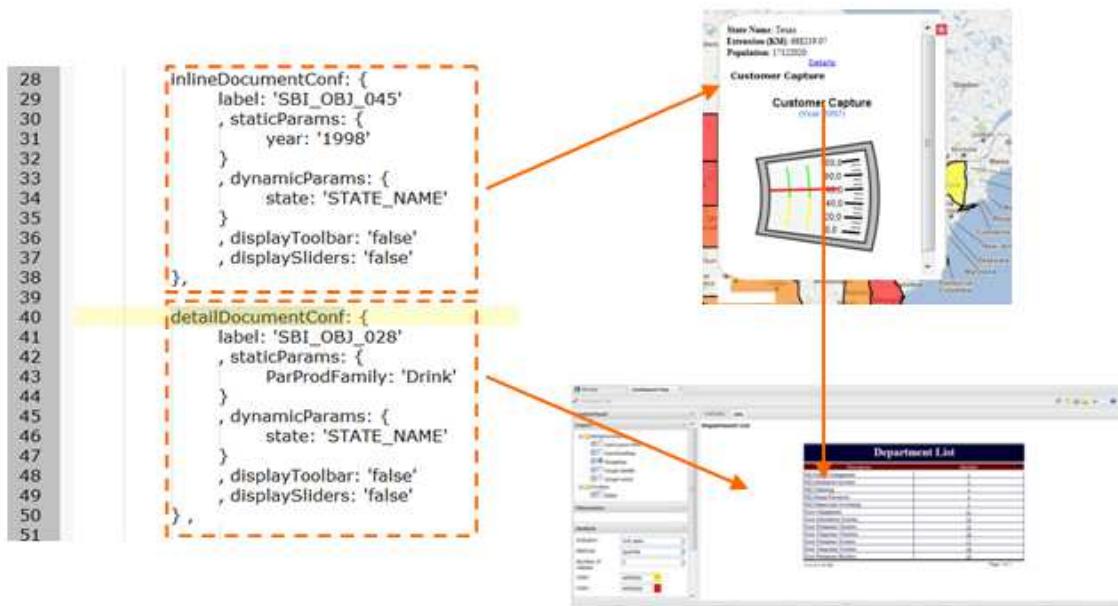


FIGURE 6.135 – Defining the detail document included in the pop-up window

GUI Customization

All buttons in the toolbar, as well as all the panels of the control panel, can be enabled or disabled by modifying the `Setting.js` file, at `\webapps\SpagoBIGeoReportEngine\js\src\ext\sbi\commons`

```

Sbi.settings.georeport = {

  georeportPanel: {

    controlPanelConf: {
      layerPanelEnabled: true
      , analysisPanelEnabled: true
      , measurePanelEnabled: true
      , legendPanelEnabled: true
      , logoPanelEnabled: false
      , earthPanelEnabled: false
    }

    , toolbarConf: {
      enabled: true
      , zoomToMaxButtonEnabled: true
    }
  }
}

```

```
, mouseButtonGroupEnabled: true  
, measureButtonGroupEnabled: true  
, wmsGroupEnabled: true  
, drawButtonGroupEnabled: true  
, historyButtonGroupEnabled: true  
}
```

Real time

Real time business intelligence (RT BI) allows the analysis of business data as soon as they occur. Differently from traditional BI systems, where data are retrieved and analyzed offline, RT BI requires to gather relevant information by processing data on-the-fly as they flow between enterprise systems. This often takes the form of *events*, which should be recognized and brought to the user's attention in real time.

Another characterizing aspect of RT BI is that business users should be able to react and take actions in response to detected events, in order to partially or totally solve a problem before the process is closed. This means that the interaction between information and front-end users becomes bi-directional: instead of simply being presented with operational data, users can now alter them by taking appropriate actions. This should support decision makers to act on time and manage the complexity of today's business.

To sum up, the main characteristics of real time BI tools are:

- monitoring of data as soon as they occur
- continuous update
- event-driven
- bi-directional interaction with data (actions)
- support for detailed and/or aggregated analysis
- applicable to business and operational scenarios
- easy to understand, especially aggregate analysis (e.g. KPI)

SpagoBI supports the real time monitoring, analysis and presentation of business data and processes, handling real-team data as far as historical ones.

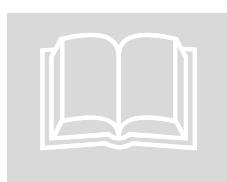
In particular, SpagoBI offers two engines allowing to manage and show real-time data:

- A graphical engine (SpagoBIDashboardEngine) that provides speedometers and other SWF widgets, allowing to display measures or KPIs (Key Performance Indicators) for real-time graphical performance views.
- An advanced engine for full real-time monitoring consoles (SpagoBIConsoleEngine), to be used in business, applicative or Business Activity Monitoring (BAM) processes.

SpagoBIDashboardEngine

The Dashboard Engine supports the execution of four types of real time dashboards:

- Rotation
- Multi-rotation
- Live lines
- Live Table



OpenLazslo

SpagoBI Dashboard Engine is developed using the open source framework OpenLazslo. Read more about the project at <http://www.openlaszlo.org>.

As usual with SpagoBI documents, creating a dashboard requires to perform the following steps:

- Create a dataset, in the appropriate format for the chosen dashboard

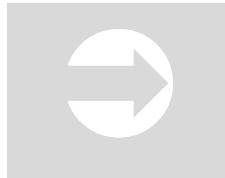
- Define the template
- Create the analytical document on SpagoBI Server.

Datasets can either be queries or Business Inquiries, which are queries over a metamodel (in other words, a business model built with SpagoBI Meta).



Business Model Query

Please refer to chapter 3, SpagoBI Meta, to learn how to define and query a business model.



Dataset Creation

To learn how to build a dataset of type query, please refer to section Data Sets in chapter 4, SpagoBI Server.

To create a dashboard document on the Server, the procedure is the same as for any other SpagoBI analytical document.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Model, chapter 4, SpagoBI Server.

In case you wish to make the document parametric, you will need to insert parameters in the dataset, following the well known syntax, and associate the corresponding analytical drivers to the document when you create it on the Server.



Analytical Drivers

To learn how analytical drivers are linked to document parameters, please refer to section The Behavioral Model in chapter 5 – SpagoBI Server.

In the following we provide details about the first two steps, which are specialized for each type of dashboard: dataset definition and template specification.

Rotation

The rotation dashboard, commonly defined a speedometer, shows a value within a pre-defined range, possibly divided into sub-intervals (see FIGURE 6.136). The value is dynamically refreshed to reflect real time variations.

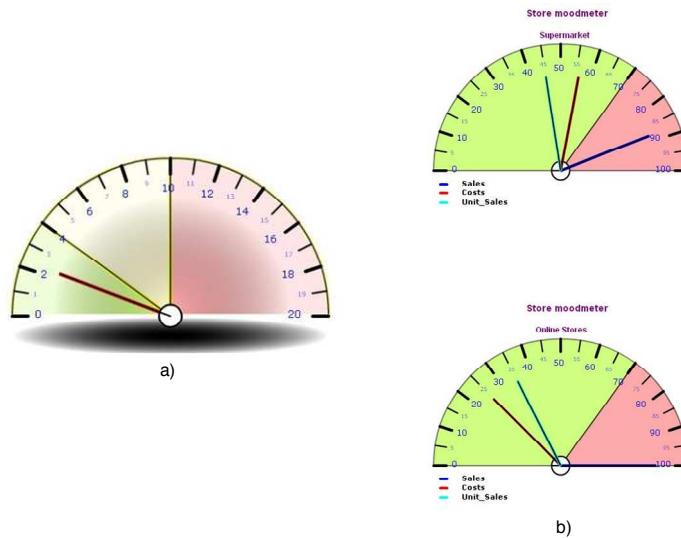
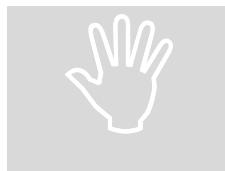


FIGURE 6.136 - Rotation (a) and multi rotation (b) dashboardDataset structure



Speedometers

The speedometer is also available as a dial chart. However, its value is set at first execution, while in the real time speedometer it is dynamically updated.

Dataset structure

The dataset must return one column whose name is `value`, as follows:

```
SELECT 2.3 AS value FROM dual
```

The `value` column corresponds to the needle in the speedometer.

Template definition

The template of a rotation dashboard is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DASHBOARD movie='/dashboards/rot.lzx.swf'
displayTitleBar='true'>
    <DIMENSION width='400' height='400' />
<CONF>
    <PARAMETER name='value' value='Sales' />
    <PARAMETER name='refreshRate' value='15000' />
    <PARAMETER name='minValue' value='0' />
    <PARAMETER name='lowValue' value='4' />
    <PARAMETER name='highValue' value='10' />
    <PARAMETER name='maxValue' value='20' />
    <PARAMETER name='colorArc1' value='0xff8080' />
    <PARAMETER name='colorArc2' value='0xffff80' />
    <PARAMETER name='colorArc3' value='0x80ff80' />
</CONF>
<DATA
url="/servlet/AdapterHTTP?ACTION_NAME=GET_DATASET_RESULT">
</DATA>
</DASHBOARD>
```

Where:

- The attribute `movie` inside the `DASHBOARD` tag defines the type of chart

- The attribute `displayTitleBar` defines whether the title should be visible
- The tag `DIMENSION` defines the size of the dashboard in pixels.
- The tag `CONF` includes configuration options, listed below:

Configuration attributes in the rotation dashboard template	
value	Label for the needle
refreshRate	Rate of refresh, in milliseconds
minValue	Minimum value of the interval scale
maxValue	Maximum value of the interval scale
lowValue	Threshold between the lower section and the middle section
highValue	Threshold between the middle section and the upper section
colorArc1	Color of the lower section
colorArc2	Color of the middle section
colorArc3	Color of the upper section
Number of main divisions	Number of sections in the dashboard

TABLE 6.27 - Configuration attributes for rotation dashboard template

Multi Rotation

Multiple rotation dashboards can be shown together in a multi rotation dashboard, as shown in FIGURE 6.136 (b). In addition, each of them can have more than one needle.

Dataset structure

The dataset is the union of multiple datasets, one for each rotation dashboard. Each dataset must return a column with a fixed name (called “series” in the example below) that identifies the single dashboard. The label of this column may change but it must be the same across single datasets. It is defined in the template in the attribute `xSerieAttributeName` (see below).

Other columns give name to needles. Their name must be specified in the template with the attributes value1, value2, value3 and so on.

```
select 'Supermarket' as series, 88 as Sales, 56 as Costs, 45 as
Unit_Sales from dual
UNION
select 'Online Stores' as series, 56 as Sales, 28 as Costs, 43 as
Unit_Sales from dual
```

Template definition

The template of a multi rotation dashboard is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DASHBOARD movie='/dashboards/multirotation.lzx.swf' >
  <DIMENSION width='400' height='400' />
  <CONF>
    <PARAMETER name='minValue' value='-100' />
    <PARAMETER name='maxValue' value='100' />
    <PARAMETER name='lowValue' value='-50' />
    <PARAMETER name='highValue' value='50' />
    <PARAMETER name='numNeedles' value='3' />
    <PARAMETER name='refreshRate' value='15000' />
    <PARAMETER name='xSerieAttributeName' value='serie' />
    <PARAMETER name='value1' value='Sales' />
    <PARAMETER name='value2' value='Costs' />
    <PARAMETER name='value3' value='Unit_sales' />
    <PARAMETER name='colorNeedle1' value='red' />
    <PARAMETER name='colorNeedle2' value='blue' />
    <PARAMETER name='colorNeedle3' value='aqua' />
    <PARAMETER name='multichart' value='false' />
    <PARAMETER name='numCharts' value='1' />
    <PARAMETER name='orientationmultichart' value='vertical' />
    <PARAMETER name='colorTitleSerie' value='purple' />
    <PARAMETER name='sizeTitleSerie' value='12' />
    <PARAMETER name='fontTitleSerie' value='Arial' />
    <PARAMETER name='displayTitleBar' value='true' />
    <PARAMETER name='title' value='4 Models Moodmeter' />
    <PARAMETER name='colorTitle' value='purple' />
    <PARAMETER name='sizeTitle' value='14' />
    <PARAMETER name='fontTitle' value='Arial' />
    <PARAMETER name='legend' value='true' />
  </CONF>

  <DRILL document="chtHistorySalesDirec">
    <PARAM name="param1" type="RELATIVE" value="$F{serie}" />
```

```

<PARAM name="paramStat" type="ABSOLUTE" value="static
parameter"/>
</DRILL>

<DATA
url='/servlet/AdapterHTTP?ACTION&#95;NAME=GET&#95;DATASET&#95;RES
ULT'></DATA>

</DASHBOARD></span>
```

Where:

- The attribute `movie` inside the `DASHBOARD` tag defines the type of chart
- The tag `DIMENSION` defines the size of the dashboard in pixels.
- The tag `CONF` includes configuration options, listed below:

Configuration attributes in the multi rotation dashboard template	
value	Label for the needle
refreshRate	Rate of refresh, in milliseconds
minValue	Minimum value of the interval scale
maxValue	Maximum value of the interval scale
lowValue	Threshold between the lower section and the middle section
highValue	Threshold between the middle section and the upper section
numNeedles	Number of needles for each dashboard
xSerieAttributeName	Name of the column defining each dashboard
value[i]	Name of the column providing the value for the i-th needle (e.g., value1, value2)
colorNeedle[i]	Color of the lower section
multichart	If true, multiple dashboard are represented
numCharts	Number of dashboards
orientationmultichart	Layout in case of multiple dashboards. Possible options: horizontal or vertical.
colorTitleSerie	Color of the series title, in alphabetical encoding (e.g., yellow, red)

sizeTitleSerie	Size of the series title (single dashboard)
fontTitleSerie	Font of the series title
displayTitleBar	If true, the title will be visible
title	Text of the general title
colorTitle	Color of the general title, in alphabetical encoding (e.g., yellow, red)
sizeTitle	Size of the general title
fontTitle	Font of the general title
legend	If true, the legend will be visible

TABLE 6.28 - Configuration attributes for multi rotation dashboard template

Live Line

The live line represents a chart with multiple lines that move over time, along a time axis.

Dataset structure

The dataset must return the value that each line has at a given moment, such as:

```
Select 90 as Effort_Index, 75 as Competitiveness, 55 as
Cost_Optimization, 70 as Health from dual
```

Template definition

The template of a live line dashboard is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DASHBOARD movie='/dashboards/livelines.lzx.swf'
displayTitleBar='true'>
    <DIMENSION width='560' height='200' />
    <CONF>
        <PARAMETER name='minYValue' value='0' />
        <PARAMETER name='maxYValue' value='200' />
        <PARAMETER name='stepNumValue' value='10' />
        <PARAMETER name='refreshRate' value='15000' />
        <PARAMETER name='paramWidth' value='500' />
        <PARAMETER name='paramHeight' value='300' />
        <PARAMETER name='stepYValue' value='25' />
        <PARAMETER name='timeStartValue' value='22:15' />
```

```

<PARAMETER name='timeStopValue' value='23:15' />
<PARAMETER name='timeGridStepValue' value=':15' />
<PARAMETER name='line1Color' value='0xf92525' />
<PARAMETER name='line2Color' value='0x17e512' />
<PARAMETER name='line3Color' value='0xbe4ce3' />

</CONF>
<DATA
url="/servlet/AdapterHTTP?ACTION_NAME=GET_DATASET_RESULT">
</DATA>
</DASHBOARD>

```

Where:

- The attribute `movie` inside the `DASHBOARD` tag defines the type of chart
- The tag `DIMENSION` defines the size of the dashboard in pixels.
- The tag `CONF` includes configuration options, listed below:

Configuration attributes in the live line template	
minYValue	Minimum value on the Y axis
maxYValue	Maximum value on the Y axis
stepNumValue	Number of ticks on the X axis
refreshRate	Rate of refresh, in milliseconds
paramWidth	Width of the internal dashboard area
paramHeight	Height of the internal dashboard area
stepYValue	Distance between ticks on the Y axis
timeStartValue	Start time for the x axis (hh:mm) Default is the current time
timeStopValue	Stop time for the x axis (hh:mm) Default is the current time
timeGridStepValue	Distance between ticks on the X axis, in hh:mm
line[i]Color	Color of the i-th line, in hexadecimal coding

TABLE 6.29 - Configuration attributes in the live line template

Live Table

Dataset structure

The dataset must return the name and the value for each row in the table:

```
select 'Supermarket' as BestDistributionLine, 614.245 as Score
from customer where customer_id = 1
union
select 'Small-Size Grocery' as BestDistributionLine, 814.853 as
Score from customer where customer_id = 1
```

Note that the name of columns must match those defined in the template (see below).

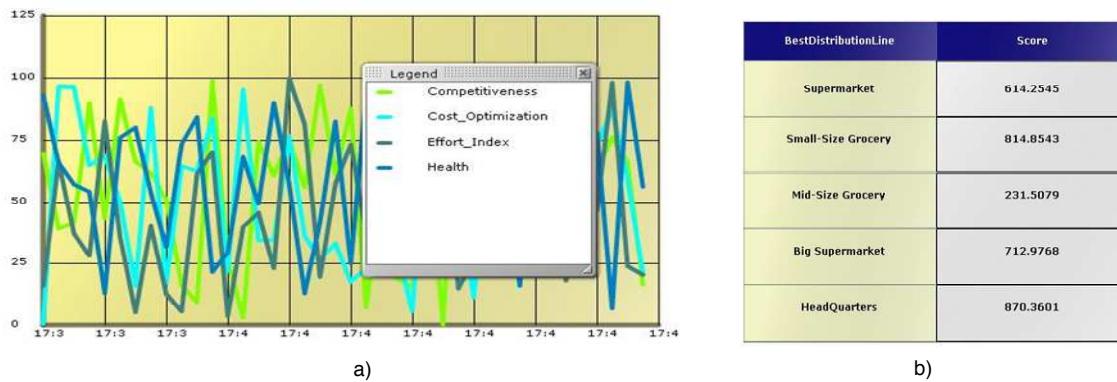


FIGURE 6.137 - Live line (a) and live table (b) dashboards

Template definition

The template of a live line dashboard is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DASHBOARD movie='/dashboards/livetabler.lzx.swf'
displayTitleBar='true'>
<DIMENSION width='280' height='200' />
<CONF>
<PARAMETER name='xColName' value='BestDistributionLine' />
<PARAMETER name='headerHeight' value='12' />
<PARAMETER name='rowsNumber' value='5' />
<PARAMETER name='rowsSpacing' value='0' />
<PARAMETER name='refreshRate' value='15000' />
```

```

<PARAMETER name='paramWidth' value='300' />
<PARAMETER name='paramHeight' value='180' />
<PARAMETER name='xColBGColor' value='0xf2f4c5' />
<PARAMETER name='colBGColor' value='0xffffffff' />
<PARAMETER name='gradOpacityBck' value='0.5' />
<PARAMETER name='gradOpacityCell' value='0.3' />
<PARAMETER name='vertScrollBar' value='true' />
<PARAMETER name='horizScrollBar' value='true' />
<PARAMETER name='fontSizeHeader' value='12' />
<PARAMETER name='fontSizeCell' value='10' />
</CONF>
<DATA url='/servlet/AdapterHTTP?ACTION_NAME=GET_DATASET_RESULT'>
</DATA>
</DASHBOARD>

```

Where:

- The attribute `movie` inside the `DASHBOARD` tag defines the type of chart
- The tag `DIMENSION` defines the size of the dashboard in pixels.
- The tag `CONF` includes configuration options, listed below:

Configuration attributes in the live table template	
xColName	Name for the column on the left
headerHeight	Height of the table header in percentage wrt. table
rowsNumber	Number of rows in the table
rowsSpacing	Spacing between rows
refreshRate	Rate of refresh, in milliseconds
paramWidth	Width of the internal dashboard area
paramHeight	Height of the internal dashboard area
xColBGColor	Background color for the left column
colBgColor	Background color for other columns
gradOpacityBck	Opacity gradient of the background
gradOpacityCell	Opacity gradient of cells
vertScrollBar	If true, the vertical scrollbar is enabled
horizScrollBar	If true, the horizontal scrollbar is enabled
fontSizeHeader	Font size for the header
fontSizeCell	Font size for cells

TABLE 6.30 - Configuration attributes in the live table template

SpagoBIConsoleEngine

SpagoBIConsoleEngine supports the design and execution of monitoring consoles, showing real time data and allowing the user to interact with them.

Data can be shown both at aggregate and detail level. The Engine offers a set of graphical elements that can be combined depending on the requirements of the analysis. Data are refreshed on the fly with a customizable frequency.

As shown in FIGURE 6.138, the real time console consists of two main subsections:

- A summary section (upper page). This section consists of a set of dashboards showing aggregate indicators, e.g., KPIs, to provide a quick overview of the situation.
- A detail section (lower page). Here information is shown with a fine level of detail, with one row for each relevant event/data.
- The detail section also includes a set of possible *actions* that can be performed on single items (e.g., open a document, view details, start/stop a process). Actions are represented by buttons on the right side of each row.

There are two types of actions in a RT console:

- generic actions, which apply to all items in the detail section of the console (i.e., all rows)
- item-specific actions, which apply to the selected row only.

Because the different parts of a RT console are completely customizable, using the Console Engine you can realize several types of monitoring console. Actions can be customized as well, by making visible only those that are meaningful for the current analysis (possibly none).

In addition, each console may include more than one page, visualized as tabs. So it is possible to group together different types of analysis in the same console and view them as tabs.

In the remainder of the section we will explain each part in detail and show how to configure them when building the document.

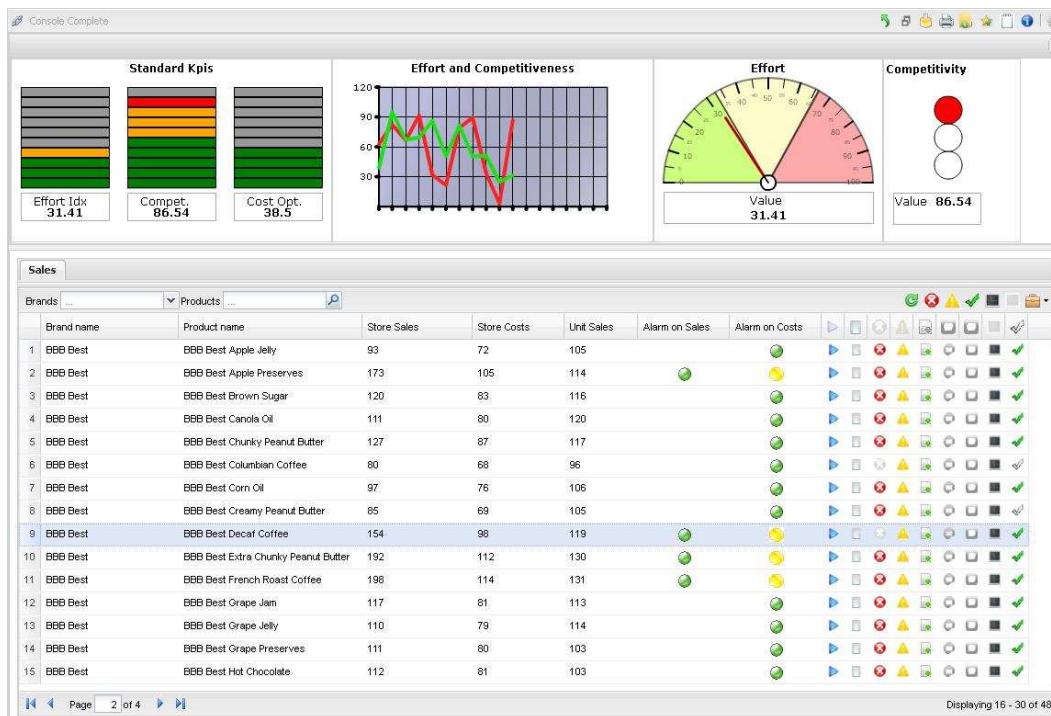


FIGURE 6.138 - SpagoBI real time console

The steps to create a console document are the following:

- Define the dataset, which contains both actual data to show and configuration options
- Create the document template
- Configure the management of actions (if any)
- Create the analytical document on SpagoBI Server

Dataset definition

The RT console shows data that are refreshed accordingly to the settings defined by the developer. Each component of the console may have its own

dataset, but it is also possible to share the same dataset across multiple components.

Datasets should be defined on SpagoBI Server following the usual procedure. It is important to remember their names as they will be used later in the template.



Dataset Creation

To learn how to create a dataset in SpagoBI, please refer to section Data Sets in chapter 5 – SpagoBI Server.

Since the engine allows the modification of data, the dataset must not only retrieve values from the data warehouse, but also provide information to enable data modification.

A complete example of dataset is shown below. This example shows all possible columns that a dataset may return. Most of them are optional and the choice to include them will depend on the intended design of the console.

```
select ID, brand_name,
      'Brand Name' as label_brand_name, product_Name, 'Product Name'
      as label_product_name,
      store_sales, store_cost, unit_sales,
      Anomalies as Anomalies, Anomalies as Anomalies2, Anomalies as
      Anomalies3,
      minValue_unit_sales, maxValue_unit_sales,
      maxValue_unit_sales as threshold,
      concat(maxValue_unit_sales,'%', ' ') as threshold_tooltip,
      minValue_unit_sales as thresholdFirst, maxValue_unit_sales as
      thresholdSecond,
      monitor_check, errors_flag, errors_check, alarms_flag,
      alarms_check,
      views_check, views_check views_check2, round(1*rand(100),0)
      views_flag, pid ,
      1 as select_flag
from(
select ID, brand_name, product_Name, store_sales, store_cost,
unit_sales, store_sales as Anomalie,
      round(90 + (2*rand(100))) as minValue_unit_sales, round(120 +
      (2*rand(110))) as maxValue_unit_sales,
      monitor_check, errors_flag, errors_check, alarms_flag,
      alarms_check, views_check, pid
```

```
from sales_fact_1997 t, product_class t2, product t3
where t.product_id = t3.product_id
and t2.product_class_id = t3.product_class_id
group by product_name
) data limit 50) a
```

The meaning of columns and the correspondence with each action is explained in the table below:

Dataset column driving console actions		
Dataset column	Action	Meaning
pid		If null the icon will be active, otherwise inactive (i.e., the process is already running).
pid		If null the icon will be inactive, otherwise active (i.e., the process is running and can be stopped).
errors_check		If null or 0 the icon will be active, otherwise inactive (i.e., error was already acknowledged)
errors_flag		If not null, the errors_check icon will be visible (because an error occurred)
alarms_check		If null the icon will be active, otherwise inactive (i.e., alarm was acknowledged)
alarms_flag		If not null, the alarms_check icon will be visible (because an alarm occurred)
		This can be linked to a column created on purpose or to an existing one, for example the errors_check. In this case, logs can be downloaded only for processes with errors. Otherwise it can be visible and active for all processes
monitor_check		If null or 0, the item can be added to the list of items that have been discarded. The action can be activated again by deselecting the item from the list.
views_check		If null or 0, the icon will be active. Otherwise inactive (i.e., already acknowledged)
select_row		No need to set values to activate/deactivate because this action is managed in memory.
select_flag		If not null, the select_row checkbox will be visible

TABLE 6.31 - Dataset column driving console actions

To make console documents more manageable, data pagination is normally managed at client side. In case of huge volumes of data, it is possible to define a server-side management.

Template creation

The RT template is a JSON document following a specific structure.



Designer for the real time console

At present SpagoBI Studio does not include a designer for RT console yet, but it will soon be available. You can check the syntax of your JSON document using a tool like <http://jsoneditor.appspot.com/>.

The template has a base structure that includes three parts:

- **Dataset.** In this section datasets used by the elements of the console are declared.
- **Summary panel.** In this section graphical widgets composing the aggregated view can be defined.
- **Detail panel.** In this section the detail table and its actions are configured.

```
{  
    // -----  
    // DATASETS  
    // -----  
    // -----  
    datasets: [  
        dataset definition  
    ]  
    // -----  
    // -----  
    // SUMMARY PANEL
```

```

// -----
-----
, summaryPanel: {
    widgets definition
}

// -----
-----
// DETAIL PANEL
// -----
-----
, detailPanel: {
    detail definition
}
}

```

Let us now describe each section in detail.

Datasets

This section is used to define all datasets used by any component of the console. This also includes dataset used to manage internazionalization (as explained in section Advanced Functionalities).

Below you can see an example of dataset template section:

```

datasets: [
    {
        id: 'testConsLiveLine'
        , label: 'BOOK_DS_CONSOLE_003'
        , refreshTime: 10
    }, {
        id: 'testConsole'
        , label: 'BOOK_DS_CONSOLE_002'
        , refreshTime: 50
        , memoryPagination: true
        , rowsLimit: 60
    }, {
        id: 'testConsoleLabels'
        , label: 'BOOK_DS_CONSOLE_004'
    }
]

```

Where:

- The `dataset` element is an array of datasets
- Each dataset is defined within {} brackets and may have a number of properties:

Dataset configuration properties	
<code>id</code>	Identifier of the dataset within the template
<code>label</code>	Dataset label as defined in SpagoBI
<code>refreshTime</code>	Interval between data refresh (sec)
<code>memoryPagination</code>	Optional. If true, pagination will be performed client side, otherwise server side
<code>rowsLimit</code>	Optional. If pagination is client side, sets the maximum number of rows to be read

TABLE 6.32 – Dataset configuration properties

Summary Panel

This section defines KPI graphical widgets in the summary section, if any. It is an optional section: if not specified, no summary panel will be shown in the console.

```
, summaryPanel: {
    collapsable: true
    , collapsed: false
    , hidden: false
    , height : 230
    , layoutManagerConfig: {
        layout:'column'
        , columnNumber: 4
        , columnWidths: [.25, .25, .25, .25]
    }
    , charts: [...]
}
```

Where:

- The `summaryPanel` object contains a set of properties, followed by the list of widgets as an array called `charts`.
- The properties `collapsible`, `collapsed`, `hidden` set visibility properties for the panel, while `height` sets its vertical dimension
- The `layoutManagerConfig` object sets the layout of the panel.
- The `layout` attribute may be: '`column`' or '`table`'. The latter allows the creation of more complex layouts like tables (as we will see later in the chapter).

The panel contains one or more graphical widgets. Each widget must be defined in the template, within the JSON `charts` object.

They are:

- Live line
- Multi led
- Speedometer
- Semaphore

Widgets are SWF files, implemented via the OpenLazslo library. Each widget and its configuration template will be described in the following.



OpenLazslo

SpagoBI Dashboard Engine is developed using the open source framework OpenLazslo. Read more about the project at <http://www.openlaszlo.org> .

Live line

This type of widget represents the evolution of an indicator along time. It is composed of a line chart with time on the x axis, as shown in FIGURE 6.139 a).

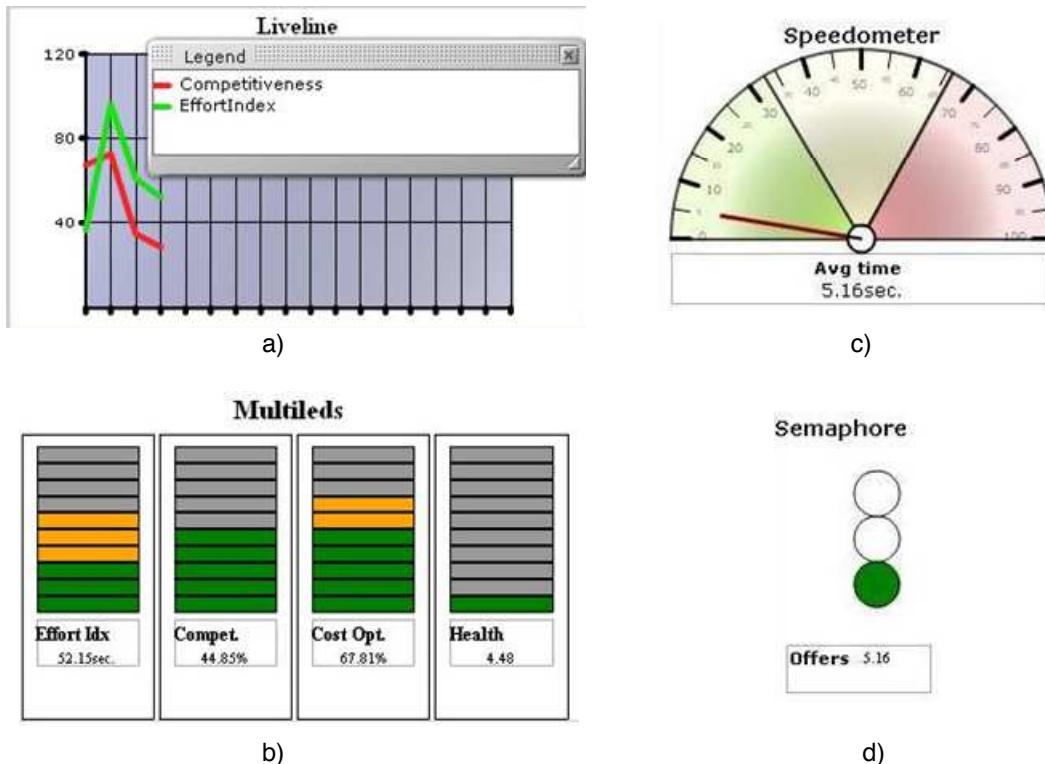


FIGURE 6.139 - RT console graphical widgets: a) live line b) multi led c) speedometer d) semaphore

Below you can see an excerpt of a console template that configures a live line widget.

```
{
    dataset: 'testConsLiveLine'
    , width:350
    , height: 200
    , widgetConfig: {
        type: 'chart.sbi.livelines'
        , styles: {
            title: {text: 'Liveline', visible:true, align:
"center", valign: "top", font: {size: 12, style: "bold"}},
```

```
        plot: {widthScale: 1, heightScale: .9, valign: 'top', halign: 'center'},
                border: {color: "white", size: 0}
            }
        , rangeMinValue: 0
        , rangeMaxValue: 120
        , stepY: 30
        , domainValueNumber: 18
    ,
domainValues:['', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'']
        , fields: ['EffortIndex', 'Competitiveness']
    }
}
```

Where:

- dataset contains the logical identifier of the dataset (the id in the dataset section). Values will be read from this dataset.
 - width and height set the size of the widget (in pixels)
 - `widgetConfig` is an object defining properties for the widget, as listed below:

Live line widget configuration properties	
type	Constant value defining the type of chart: 'chart.sbi.livelines'
title	Optional. Title of the widget
rangeMinValue	Minimum value on the Y axis
rangeMaxValue	Maximum value on the Y axis
stepY	Interval between ticks on the Y axis
domainValueNumber	Number of ticks on the X axis (time)
domainValues	Values on the X axis (time)
fields	Columns (as defined in the dataset) to be visualized

TABLE 6.33 - Live line widget configuration properties in the RT console

Multi leds

This widget shows KPI values as colored leds. Led colors change according to the KPI value, giving a dynamic effect upon data refresh, as shown in FIGURE 6.139 b)

Below you can see an excerpt of a console template that configures a multi led widget.

```
{
dataset: 'testConsLiveLine'
    , width: 350
    , height: 200
    , widgetConfig: {
        type: 'chart.sbi.multileds'
        , title: 'Multileds'
        , styles: {
            border: {color: 'white', size:0},
            title: {visible: false},
            padding: {top: 30, right: 0, bottom: -10,
left: 0},
            gauge: {
                padding: {right: 0, left: 0},
                colors: {firstInterval: "green",
secondInterval: "orange", thirdInterval: "red"},
                led: {height: 5, gap: 1, margin:
0},
                title: {align: "center", valign:
"top", font:{size: 12, style: "plain"}},
                value: {align: "center", valign:
"bottom", font:{size: 12, style: "bold"}}
            }
        }
        , fields: [
            {header: 'Effort Idx',name:'EffortIndex',
rangeMaxValue: 100, secondIntervalUb: 66, firstIntervalUb: 10,
rangeMinValue: 0},
            {header: 'Compet.',name:'Competitiveness',
rangeMaxValue: 100, secondIntervalUb: 66, firstIntervalUb: 33,
rangeMinValue: 0},
            {header: 'Cost Opt.',name:'CostOptimization',
rangeMaxValue: 100, secondIntervalUb: 66, firstIntervalUb: 33,
rangeMinValue: 0}
        ]
    }
}
```

Where:

- `dataset` contains the logical identifier of the dataset (the id in the dataset section). Values will be read from this dataset.
- `width` and `height` set the size of the widget (in pixels)
- `widgetConfig` is an object defining properties for the widget, as listed below:

Multi led widget configuration properties		
type		Constant value defining the type of chart: 'chart.sbi.multileds'
title		Optional. Title of the widget
styles		Optional. Object that sets the style for the border
	border	Color and dimension of the border
	title	Style of the title
	padding	Space around the widget
	gauge	Defines style for the single led
	padding	Space around the single led
	colors	Color for each area
	led	Size of the single led
	title	Title of the single led
	value	Style for values of the single led
fields		Columns (as defined in the dataset) to be visualized
	header	Description of the single value
	name	Name of the field (i.e., dataset column returning the value)
	descValue	Column of the dataset that return the description of the shown value. If not defined, the value will be displayed.
	rangeMinValue	Minimum value of the range
	firstIntervalUp	Threshold of the green area
	secondIntervalUp	Threshold of the yellow area

	range.MaxValue	Maximum value of the range

TABLE 6.34 - Multi led widget configuration properties in the RT console

Speedometer

This widget shows KPI values as a speedometer (see FIGURE 6.139 c).

Below you can see an excerpt of a console template that configures a speedometer widget.

```
{
    dataset: 'testConsLiveLine'
    , width: 250
    , height: 200
    , widgetConfig: {
        type: 'chart.sbi.speedometer'
        , paramWidth: 100
        , paramHeight: 150
        , minValue: 0
        , maxValue: 100
        , lowValue: 33
        , highValue: 66
        , field: 'EffortIndex'
        , styles:{
            title: {text:'Speedometer',
visible:true, align: "center", font:{size: 12, style: "bold"}}
            , padding: {top: 0, right: 0, bottom:0,
left: 0}
            , header: { text: 'Value ', visible:
true, align: "center", font:{size: 12, style:"plain"}}
            , headerValue: {align: "center",
valign:"bottom", font:{size: 12, style: "bold"}}
        }
    }
}
```

Where:

- dataset contains the logical identifier of the dataset (the id in the dataset section). Values will be read from this dataset.
- width and height set the size of the widget (in pixel)

- `widgetConfig` is an object defining properties for the widget, as listed below:

Speedometer widget configuration properties		
type		Constant value defining the type of chart: 'chart.sbi.speedometer'
title		Optional. Title of the widget
paramWidth		Width of the component
paramHeight		Height of the component
minValue		Minimum value of the range
maxValue		Maximum value of the range
lowValue		Threshold value of the first area
highValue		Threshold value of the second area
field		Column of the dataset returning the value
descValue		The field returned by the dataset with the description value to visualize
styles		Optional. Object that sets the style for the border
	title	Style of the title
	padding	Space around the value box
	header	Style for the title of value
	headerValue	Style for value

TABLE 6.35 - Speedometer widget configuration options in the RT console

Semaphore

This widget shows KPI values as a semaphore (see FIGURE 6.139 d).

Below you can see an excerpt of a console template that configures a semaphore widget.

```
{
    dataset: 'testConsLiveLine'
    , width: 150
    , height: 200
    , widgetConfig: {
        type: 'chart.sbi.semaphore'
        , paramWidth: 50
    }
}
```

```

        , paramHeight: 50
        , rangeMinValue: 0
        , rangeMaxValue: 100
        , rangeFirstInterval: 33
        , rangeSecondInterval: 66
        , field: 'EffortIndex'
        , styles: {
            padding: {top: 5, right: 0, bottom: 0,
left: 45}
            , title: { text: 'Semaphore', visible: true,
align: "center", valign: "top", font: {size: 12, style: "bold"}}
            , header: { text: 'Offers', visible: true,
valign:'top', align: "right", font:{size: 12, style: "plain"}}
            , headerValue: { align: "center",
valign:"bottom", font:{size: 12, style: "bold"}}
        }
    }
}

```

Where:

- dataset contains the logical identifier of the dataset (the id in the dataset section). Values will be read from this dataset.
- width and height set the size of the widget (in pixel)
- widgetConfig is an object defining properties for the widget, as listed below:

Semaphore widget configuration properties		
type		Constant value defining the type of chart: "chart.sbi.semaphore"
paramWidth		Width of the component
paramHeight		Height of the component
rangeMinValue		Minimum value of the range
rangeMaxValue		Maximum value of the range
rangeFirstInterval		Threshold value for the green circle
rangeSecondInterval		Threshold value for the yellow circle
field		Column of the dataset returning the value
styles		Optional. Object that sets the style for the border
	title	Style of the title
	padding	Space around the value box

	header	Style for the title of value
	headerValue	Style for value

TABLE 6.36 - Semaphore widget configuration options in the RT console

Composite Widget

Using the table layout, you can define more complex layout (see for example the summary panel in FIGURE 6.140), where the panel may contain multiple widgets.

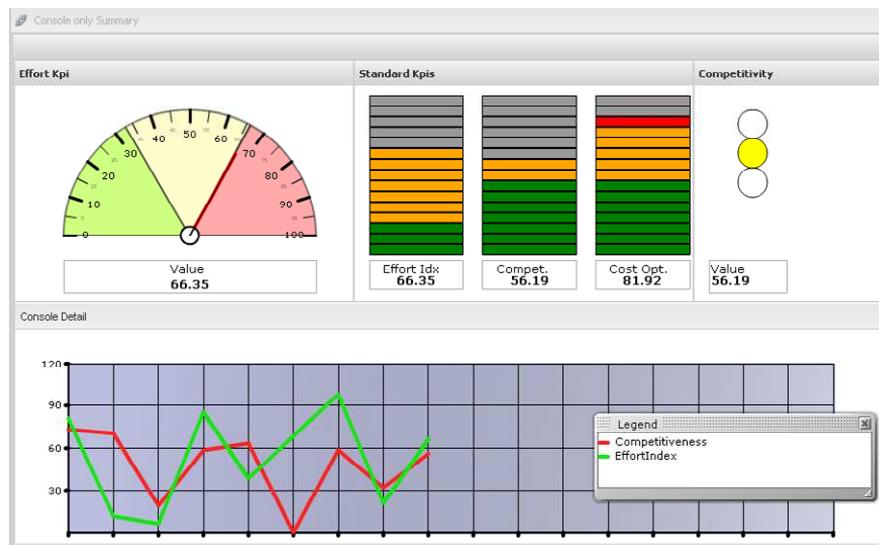


FIGURE 6.140 - Example of table layout in the console summary panel

An example of template excerpt is shown below:

```
, summaryPanel: {
    collapsable: true
    , collapsed: false
    , hidden: false
    , height : 200
    , layoutManagerConfig: {
        layout:'table'
        , layoutConfig: {
            columns: 3          // The total column
count must be specified here
            , tableAttrs: {style:
{float:'left'}}}}
```

```
        }
        , defaults: {bodyStyle:'padding:5px',
columnHeightDefault: 250, dataset:'testConsLiveLine',
hideBorders: true } // applied to each contained panel
    }
, charts: [
    {   widgetConfig: {
        type: 'chart.composite'
        , title: 'Effort Kpi'
        , dataset: 'testConsLiveLine'
        , width: 350
        , subcharts: [
            {
                widgetConfig: {
                    type: 'chart.sbi.speedometer'
                    , paramWidth: 100
                    , paramHeight: 150
                    , minValue: 0
                    , maxValue: 100
                    , lowValue: 33
                    , highValue: 66
                    , field: 'EffortIndex'
                    , styles:{title: {text:'Effort',
visible:false, align: "center", font:{size: 12, style: "bold"}},
                    , padding: {top: 15, right: 0, bottom:0, left: 0}
                    , plot: {widthScale: .8,
heightScale: .8, valign: 'top', halign: 'center'}
                    , header: { text: 'Value '},
visible: true, align: "center", font:{size: 12, style:"plain"}},
                    , headerValue: {align: "center", valign:"bottom", font:{size: 12, style: "bold"}}
                }
            }
        ]
    }
}
]
```

We only focus on specific properties of the table layout, while we refer the reader to previous sections for other properties.

- `layoutConfig` is an object defining properties for the layout, as listed below:

Table layout configuration properties		
columns		The number of columns
tableAttrs		Generic HTML attributes
defaults		Default element for all panels
	bodyStyle	HTML properties for the body element
	columnHeightDefault	Default height for each panel (in pixel)
	dataset	Default dataset for each widget, if not specified otherwise.
	hideBorders	If true, panel borders will be hidden

TABLE 6.37 - Table layout configuration properties for the RT console summary panel

- `widgetConfig` is an object defining properties for the widget, as listed below:

Table layout widget configuration properties	
type	Constant value defining the type of chart: 'chart.composite'
title	Title of the panel
dataset	Id of the dataset retrieving data. If not defined, the id is retrieved from the defaults section
width	Width of the single panel.
colspan	Number of columns fitting in the panel (see in figure the second row)
subcharts	A list of <code>widgetConfig</code> objects that define single widgets. Configuration options defined for each type of widget apply.

TABLE 6.38 - Table layout widget configuration properties for the RT console summary panel

Detail Panel

This section is optional and may define information about different parts of the console:

- a single page (console tab)

- the navigation bar
- the filters and actions bar

In the following we provide details for each case.

Page definition

Below an excerpt of console template focusing on the detail panel.

```
//-----  
-----  
// DETAIL PANEL  
// -----  
-----  
, detailPanel: {  
    pages: [  
  
    //-----  
-----  
    // Page 1  
    // -----  
-----  
    {  
        //title: 'Detail console for $P{TYPE} and $P{ALARM}'  
        title: LN(titlePage)  
        , msg: ''  
        ...  
    }  
]  
}
```

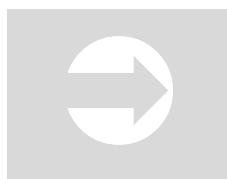
Recall that a console may be composed of multiple tabs. Each page is defined within the JSON element pages[].

```
, pages: [{page1},{page2}..],{pagen}
```

The page is the main element of the detail panel. For each page, you must define a title, as shown above. The title can be parameterized or internationalized (both topics will be later discussed).

Navigation bar

Each page may include a navigation bar. This bar consists of clickable names pointing to the documents that can be reached in cross navigation (see FIGURE 6.141). Clicking on the name of a document, a new window will be opened with the target document.



Cross Navigation

Please refer to section Cross Navigation in chapter 4, SpagoBI Server, for a full explanation of this SpagoBI feature.

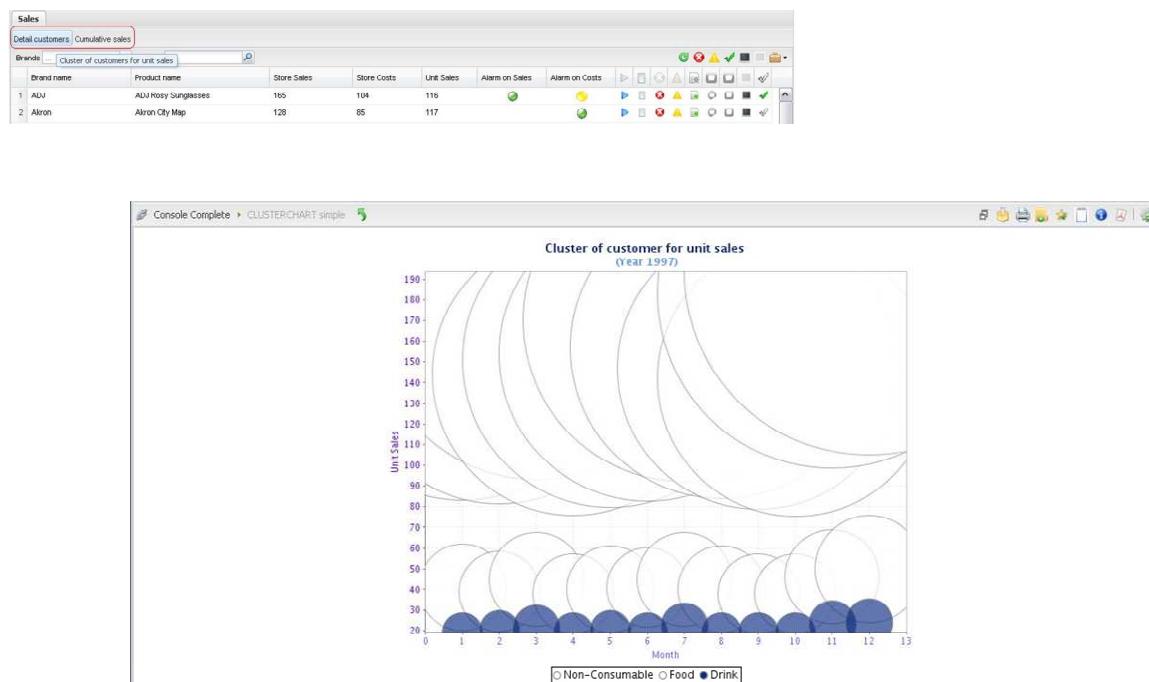


FIGURE 6.141 - Cross navigation bar in the RT console

This feature can be enabled defining a set of information in the console template.

```
{
  name: 'crossnav'
, hidden: false
```

```

        , tooltip: 'Detail sales and costs'
        //, imgSrc:'ico_info.gif'
        , config: {
            target: 'new'
            , document: {label: 'chtHistorySalesDirec',
staticParams: {typeCross: 'INTERNAL'}, dynamicParams: []}
        }
    , {
        name: 'crossnav'
        , hidden: false
        , tooltip: 'Detail sales on store type'
        , config: {
            target: 'self'
            , document: {label: 'SCATTER_simple', staticParams:
{}}, dynamicParams: []
        }
    }
}

```

Where:

- The value of the `text` attribute will be shown as the text of the button
- Similarly for the `tooltip`
- The `label` corresponds to the label of the SpagoBI document that should be invoked in cross navigation
- The `target` attribute defines whether the target document will be opened in the same window of the current one ('`self`') or in a new window ('`new`').
- The `parameters` attribute contains parameters that will be passed to the linked document. If the document requires parameters but this element is not defined, they will be asked to the user before execution.

Definition of parameters is explained in the dedicated section, later in this chapter.

Table

This section is the core of the detail panel as it practically defines all detail elements of a console. Specifically, you can define and configure:

- Table columns
 - Filter bar
-

- Action bar
- Inline charts
- Inline actions

Column layout

Columns that compose the main detail table can be customized according to the developer's needs. In particular, the following elements can be configured for each column:

- Header
- Size (width)
- Data type: string (default), number, data or timestamp (to correctly handle column ordering)

Below an example of configuration:

```
, table: {
    dataset: 'testConsole'
    , datasetLabels: 'testConsoleLabels'
    , columnId: 'ID'
    , columnConfig: {
        'brand_name': {
            width: 150
            , header: 'label_brand_name'
            , headerType: 'dataset'
        }, 'product_name': {
            width: 200
            , header: 'label_product_name'
            , headerType: 'dataset'
        }, 'store_sales': {
            width: 120
            , header: 'cod_StoreSales'
            , headerType: 'i18N'
            , type: 'int'
        }, 'store_cost': {
            width: 100
            , header: 'cod_StoreCosts'
            , headerType: 'datasetI18N'
            , type: 'int'
        }, 'unit_sales': {
```

```

        width: 80
        , header: 'cod_UnitSales'
        , headerType: 'datasetI18N'
        , type: 'int'
    }, 'Alarms': {
        width: 100
        , header: 'Alarm on Costs'
        , headerType: 'static'
    }, 'Alarms2': {
        width: 100
        , header: 'Alarm on Sales'
        , headerType: 'static'
    }
}
...
}
```

Where:

- **dataset** is the dataset that will determine table columns. Columns will appear in the same order as they are retrieved. Note that you should put here the logical identifier defined at the beginning of the template (not SpagoBI label).
- **datasetLabels** is the dataset that will be used for internationalization (if any). It is not mandatory, but if you wish to enable internationalization it must be specified.
- **columnId**: la colonna del dataset contenente l'identificativo dell'oggetto analizzato. Il campo columnId non viene visualizzato nella griglia.
- **columnConfig** is an optional object that allows the configuration of a single column, as detailed below.

Column configuration properties	
width	Width of the cell (in pixel)
header	Header of the cell
type	Data type of the column: string (default), int, date, timestamp
headerType	Defines how the header should be retrieved. Available options:
static	Default option. The value of the "header" property will

		be displayed.
	dataset	Value is retrieved from the dataset. A column with the name “header” must be defined in the dataset.
	i18N	Value is retrieved from a localization file (.js)
	datasetI18N	Value is retrieved from the internationalization dataset defined above.

TABLE 6.39 -- Column configuration options in the detail panel of the console

Filter bar

It is possible to add a bar with filters: once selected, they will be applied to values shown in the detail table.

You can see an example of where two filters, Brands and Products, are defined. All filters are included in the `filters[]` element, as shown below:

```
// -- Filter bar -----
-----
, filterBar: {
    type: 'custom'
    , defaults: {
        //operator: 'EQUALS_TO'
        operator: 'IN'
        , operand: 'DISTINCT'
    }
    , filters: [{
text: 'Brands'
        , column: 'brand_name'
        , operator: 'EQUALS_TO'
        , operand: 'DISTINCT'
    }, {
text: 'Products'
        , column: 'product_name'
        , operator: 'IN'
        , operand: 'DISTINCT'
    }]
    ...
}
```

Where:

- The property `type` has two possible values: ‘`custom`’ or ‘`automatic`’. By specifying `automatic`, a filter for each column will be created.

Otherwise, you have to explicitly specify which columns will have a filter in the bar. This is the case of our example.

- Both in the defaults (if type is automatic) and in each single filter, you need to specify the operator to define the type of filter: single value corresponds to EQUALS_TO, while multiple possible values correspond to IN.
- If the type is ‘custom’, the filters[] element contains all filters you wish to define. Each filter element has the following properties:

Filter configuration properties	
text	Text of the filter
column	Column that will be filtered
operator	Defines the type of filters. Options: EQUALS_TO and IN.
operand	Set to DISTINCT. Defines that filtering will be applied on distinct values.

TABLE 6.40 - Single column filter configuration properties

Actions

As said at the beginning of the section, actions allow users to perform operations on data. There are two types of actions: generic (massive) and item-specific (inline).

Generic actions

It is possible to add generic (massive) actions on the same bar that hosts filters. Each action is represented by an icon and has in general two states: enabled and disabled. These states are managed persistently on the database, as we will explain later.

Generic actions include:

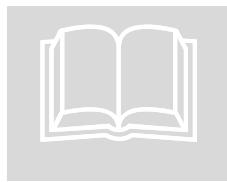
Generic actions

	Refresh	Force manual refresh of data
	Errors acknowledge	Acknowledge that errors have been viewed (or cancel acknowledgement)
	Warning acknowledge	Acknowledge that alarms have been viewed (or cancel acknowledgement)
	Show discarded/monitored objects	Show objects that are monitored or have been discarded, i.e., removed from the list
	Acknowledge	Acknowledge that all objects in the list have been viewed (or cancel acknowledgement)
	Select all	Select all rows in the current page (to perform further actions on them).
	Deselect all	Deselect all rows in the current page
	Swap selection	Invert selected items with unselected

TABLE 6.41 - Generic actions in the RT console

Note that selection actions (select, deselect, swap) are useful to define a set of items that will be subject to further actions, chosen among those listed in TABLE 6.41. Grey icons represent an action whose state is disabled.

Actions on alarms, errors, logs or process start/stop are particularly useful when the console is used in a Business Activity Monitoring (BAM) system.



Business Activity Monitoring (BAM)

BAM systems are used to monitor and detect events relevant for a business and make them meaningful to end-users. Their implementation in business contexts allows Real Time BI features. Read more about real time BI with SpagoBI at <http://www.spagoworld.org/xwiki/bin/view/SpagoBI/RealTimeBI>.

Below you can see an excerpt of template where generic actions are configured in the filter bar. Each action must be included in an `actions[]` object, except the “export” that is automatically included. Each action includes a number of objects that are used for configuration.

```

// -- Filter bar -----
-----
, filterBar: {
    ...
    , refreshDataAfterAction: true
    , actions: [{
        name: 'refresh'
        , tooltip: 'Refresh console data'
        , hidden: false
        , config: {}
    }, {
        name: 'errors'
        , tooltipActive: 'Mark all errors as not viewed'
        , tooltipInactive: 'Mark all errors as viewed'
        , checkColumn: 'errors_check'
        , flagColumn: 'errors_flag'
        , hidden: false
        , config: {
            staticParams: {stmt: 'UPDATE_ALL_ERRORS_STATE_TEST',
numPars: '4'}
            , dynamicParams: [{errors_check: 'errors_check',
scope: 'env'},
                                {'userId': 'userId',
scope: 'env'},
                                {'brand_name': 'brand_name',
scope: 'env'},
                                {'product_name':
product_name, scope: 'env'}]
            , metaParams: {queryParams:
[{"name":"errors_check","type":"num"}, {"name":"userId","type":"string"},
{"name":"brand_name","type":"string", "default": "%"}, {"name":"product_name",
"type":"string", "default": "%"}]}
        }
    }, {
        name: 'alarms'
        , tooltipActive:'Mark all alarms as not viewed'
        , tooltipInactive:'Mark all alarms as viewed'
        , checkColumn: 'alarms_check'
        , flagColumn: 'alarms_flag'
        , imgSrcActive:'ico_point.gif'
        , imgSrcInactive: 'ico_point_grey.gif'
        , msgConfirm: 'This operation will update ALL
alarms...Continue?'
        , hidden: false
        , config: {...}
    }, {
        name: 'viewsName'
    }
}

```

```

        , type: 'views'
        , tooltipActive: 'Mark all rows as not viewed'
        , tooltipInactive: 'Mark all rows as viewed'
        , checkColumn: 'views_check'
        , hidden: false
        , config: {...}
    }, {
        name: 'monitor'
        , hidden: false
        , tooltip: 'View all active'
        , checkColumn: 'monitor_check'
    }, {
        name: 'monitor_inactive'
        , hidden: false
        , tooltip: 'View all inactive'
        , checkColumn: 'monitor_check'
    }, {
        name: 'selectAllRow'
        , type: 'selectRow'
        , tooltip: 'Check all rows in the page'
        , columnID: 'ID' //the column on db that we want to add
at the memory list
        , hidden: false
        , flagColumn: 'select_flag'
        , msgConfirm: 'This operation will check update ALL records
on the page...Continue?'
    }, {
        name: 'unselectAllRow'
        , type: 'unselectRow'
        , tooltip: 'Uncheck all rows in the page'
        , columnID: 'ID' //the column on db that we want to to
remove from the memory list
        , hidden: false
        , flagColumn: 'select_flag'
        , msgConfirm: 'This operation will uncheck update ALL records
in the page...Continue?'
    }, {
        name: 'invertSelectionRow'
        , type: 'invertSelectionRow'
        , tooltip: 'Uncheck all rows in the page'
        , columnID: 'ID' //the column on db that we want to use
for invert check values
        , hidden: false
        , flagColumn: 'select_flag'
        , msgConfirm: 'This operation will invert the selection of ALL
records in the page...Continue?'
    }
]
} //filterBar

```

Where:

- `tooltip`, `tooltipActive` and/or `tooltipInactive` allow the definition of tooltip messages. In case the action has two possible states, you can specify the tooltip in both cases.
- The `flagColumn` and `checkColumn` properties are used to set the visibility and the state of an action. In particular, the former defines whether the action should be visible, while the latter defines whether it was selected by the user. Admissible values are 0 (false) and 1 (true).

Note that the values for those attributes are defined by two columns that must be returned in the dataset. In our example, those dataset columns are called “select_flag” and “monitor_check”. Names are not fixed but they correspond to the columns returned by the dataset.

- If the `hidden` property is true, the action will not be visible. This overrides any other setting of the `flagColumn` property.
- The `config` element must be present to enable any action on the database (even just to save the new state when clicking on the button). Its content will be discussed in detail in a later paragraph, Action state management.
- If `refreshDataAfterAction` is true, the content of the table will be refreshed after execution of a massive action (generic). This is useful when the refresh time is not very short.
- If `singleExecution` is true, the action can only be executed once.
- The `imgSrcActive` and `imgSrcInactive` are strings defining the name of an image that will be shown instead of the default one. Images must be available under the `\webapps\SpagoBIConsoleEngine\img` folder.
- Similarly, the `imgSrc` string defines the name of an image for cross navigation icons (window and popup). Images must be available under the `\webapps\SpagoBIConsoleEngine\img` folder.
- The `msgConfirm` string allows the definition of a message that will be shown to confirm the general action.
- The `type` string assigns the action to a group (among views, alarms, errors,...) This allows the creation of alternative groups of actions (e.g., one group is disabled when the other is enabled).

- The `name` element contains the logical name of the action button. To ensure backward compatibility, the name is used by the engine to recognize the type of button in case the type attribute is not defined.
- The `column` string is defined only for actions that manage in memory the list of selecting actions ('`selectAllRow`', '`unselectAllRow`' and '`invertSelectionRow`'). It defines the name of the column that should be kept in memory. This name can then be used by action statements associated to generic actions, as a multivalue parameters (see paragraph Parameter Management later in this section).

Item-specific actions

Item-specific (inline) actions can be added on each row of the table. Although some inline actions share some functionalities with generic ones, they differ regarding interaction with the user.



FIGURE 6.142 - Inline actions in the RT console

Item-specific include:

Item-specific actions		
	Start process	Start external process
	Stop process	Stop external process
	Errors acknowledge	Acknowledge that errors have been viewed (or cancel acknowledgement)
	Warning	Acknowledge that alarms have been viewed (or cancel

	acknowledge	acknowledgement)
	Log download	Download log files related to the specific process
	Detail window	Open a SpagoBI detail document in new window (cross navigation)
	Detail popup	Open a SpagoBI detail document in new popup (cross navigation)
	Enable/disable monitoring	Enable/disable monitoring for the selected item
	Acknowledge	Acknowledge that the object has been viewed (or cancel acknowledgement)
	Select row	Select the item for further actions (e.g., generic ones on selected items). Does not force any operation on the DB.

TABLE 6.42 - Item-specific actions in the RT console

Grey icons represent the action when its state is disabled.

Let us now see each action in detail.

▪ Start process

Start and stop process actions allow end users to execute and terminate an external process, such as CommonJ processes. They are particularly meaningful when the console is part of a BAM infrastructure, which allows users to monitor and act on event data.

Upon start action, it is possible to allow the specification of one or more parameters, which will be passed to the target process.

Inline actions are included in the `inlineActions[]` object of the template. Below the configuration template for a Start process action:

```
, inlineActions: [
{
  name: 'start'
, hidden: false
, checkColumn: 'pid'
```

```

        , tooltip: 'Start the process'
        , config : {
            document: {
                label: 'processTest'
                , staticParams: {idService: '2', resourceName: 'azTest', formatDate: 'dd-MM-yyyy HH:mm:ss'}
                , dynamicParams: [{`'combo_field_1': 'Measure`}], scope: 'promptable'
                , values: {type: 'combo', defaultValue: '', data: [['', 'Total'], ['costs', 'Costs'], ['sales', 'Sales']] }
                , {'text_field': 'Note'}
            ,
                , scope: 'promptable'
                , values: {type: 'text' , defaultValue: 'aaa'}
                , {'date_field': 'Date'}
            ,
                , scope: 'promptable'
                , values: {type: 'data', format:'d/Y/m', defaultValue: ''}
                , {'radio_field': 'Group values'}
                , scope: 'promptable'
                , defaultValue: ''
                , values: {type: 'group', options:[{values: {type: 'combo' , title: 'Measure' , data: [['', 'Total'], ['costs', 'Costs'], ['sales', 'Sales']] }}, {values: {type: 'data' , title: 'Date' , format:'d/m/Y'}}]}
            ,
        }
    }
}

```

```

        }
    }]}
}
...
]

```

Where:

- The `checkColumn` element specifies which column of the dataset determines icon visualization. If it is assigned 0 or null, the process is not currently executing, therefore it can be started. If, otherwise, the column value is different from 0 or null, the process is already executing, so the start action icon is not active.
- The `config` element includes configuration options:
 - `label` is the name of the CommonJ process, registered in SpagoBI, that will be started.
 - `staticParams` and `dynamicParams` define parameters (if any) and how to ask for values when the process is started. For parameters of type ‘promptable’ a popup window will be created to ask for values.

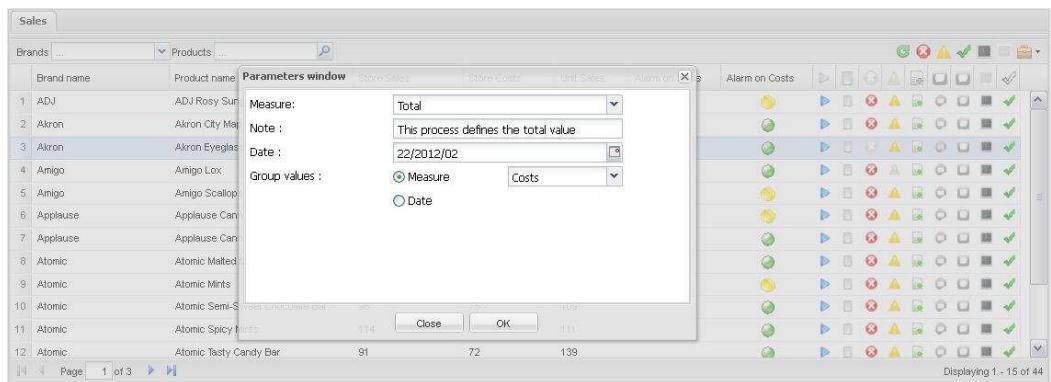


FIGURE 6.143 - Parameters to start process execution

▪ Stop

Below the configuration template for a Stop process action:

```
, inlineActions: [
{
    name: 'stop'
    , checkColumn: 'pid'
    , hidden: false
    , tooltip: 'Stop the process'
    , config: {
        document:
        {
            label: 'processTest'
            , dynamicParams: [ {'id': 'ID', scope: 'dataset'},
            {'PROCESS_ID': 'pid', scope: 'dataset'}]
        }
    }
}
...
]
```

Where:

- The `checkColumn` element specifies which column of the dataset determines icon visualization. If it is assigned 0 or null, the process is not executing, therefore it cannot be stopped. If, otherwise, the column value is different from 0 or null, the process is currently executing, so the stop action icon is active.
- The `config` element includes configuration options:
 - `label` is the name of the CommonJ process, registered in SpagoBI, that will be started.
 - `staticParams` and `dynamicParams` define parameters (if any) and how to ask for values when the process is started. The PID of the process is a mandatory parameter to terminate it. Only processes that were manually started via the start action can be terminated: third party processes cannot be stopped from the RT console.
- **Error acknowledgement**

Clicking on this action item, a popup window will open, showing information about the error. Error detail information is retrieved from a dataset with label <table_dataset_label>Errors.

This dataset should define a query that retrieves error information. Regardless of the total number of columns, it must return one column whose name is 'detailColumn'. This is mandatory to let the engine retrieve error information from that column.

For example, the following dataset is called BOOK_DS_CONSOLE_002Error :

```
SELECT column1 as BrandName, column2 as ProductName, 'Error
during the execution of the batch ' as detailColumn FROM
TEST_CONSOLE p where id = $P{id}
```

As shown in FIGURE 6.144, the popup window shows in the upper part the list of errors that occurred, and in the lower part the details of the row selected from the upper list.

Once seen the details, the user can acknowledge this by clicking on **Mark as checked**. Upon re-opening of the window, he will see the same information and can uncheck the acknowledgement.

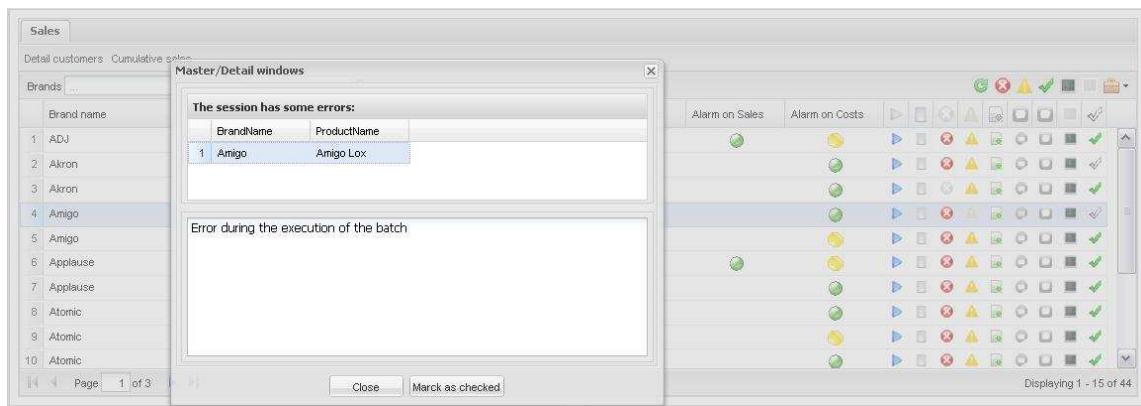


FIGURE 6.144 - Error acknowledge popup window

Below the corresponding configuration template:

```
, inlineActions: [
  {
    name: 'errors'
    , hidden: false
    , titleWin: 'The session has some errors: '
    , checkColumn: 'errors_check'
    , flagColumn: 'errors_flag'
    , tooltipActive: 'View unchecked error messages'
    , tooltipInactive: 'View checked error messages'
    , config: {staticParams: {stmt: 'UPDATE_ERROR_STATE_TEST',
numPars: '2'}
               , dynamicParams: [{errors_check': 'errors_check', scope: 'dataset'}, {ID: 'ID', scope: 'dataset'}]
               , metaParams: {queryParams:
[{"name": "errors_check", "type": "num"}, {"name": "ID", "type": "num"}]
}
    }
  ...
]
```

Where:

- The string `titleWin` defines the title of the error window
- `tooltipActive` and/or `tooltipInactive` allow the definition of tooltip messages.
- The `flagColumn` and `checkColumn` properties are used to set the visibility and the state of the error action. Admissible values are 0 (false) and 1 (true).
- The `config` element must be present to enable the action. Its content will be discussed in detail in a later paragraph, Action state management.

▪ Alarms acknowledgement

Similarly to errors, clicking on this icon a popup window will open, showing information about alarms. Alarm detail information is retrieved from a dataset with label <table_dataset_label>Alarms.

This dataset should define a query that retrieves alarms information. Regardless of the total number of columns, it must return one column whose name is 'detailColumn'. This is mandatory to let the engine retrieve alarm information from that column.

For example, the following dataset is called BOOK_DS_CONSOLE_002Alarms :

```
SELECT column1 as BrandName, column2 as ProductName, 'Alarms
during the execution of the batch ' as detailColumn FROM
TEST_CONSOLE p where id = $P{id}
```

As shown in, the popup window shows in the upper part the list of alarms that occurred, and in the lower part the details of the row selected from the upper list.

Once seen the details, the user can acknowledge this by clicking on **Mark as checked**. Upon re-opening of the window, he will see the same information and can uncheck the acknowledgement.

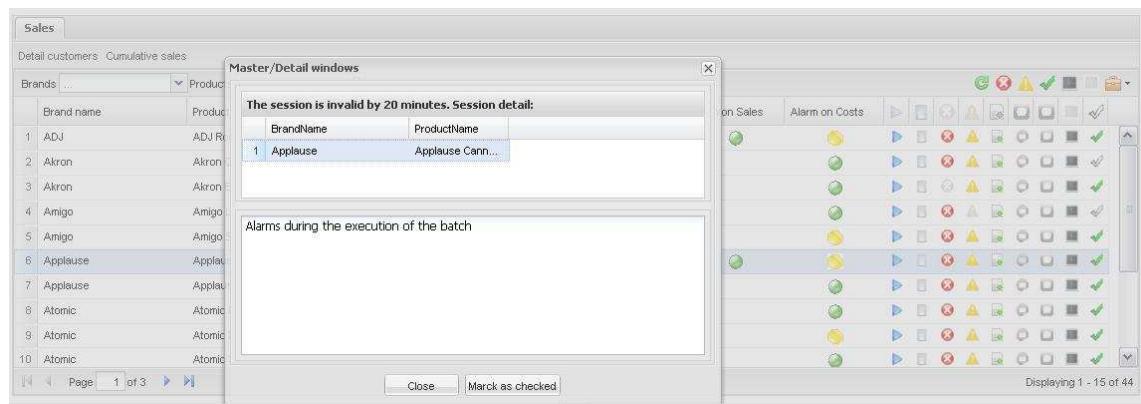


FIGURE 6.145 – Alarm acknowledge popup window

Configuration template:

```

, inlineActions: [
{
    name: 'alarms'
    , hidden: false
        , titleWin: 'The session is invalid by 20 minutes.
Session detail:'
        , checkColumn: 'alarms_check'
        , flagColumn: 'alarms_flag'
        , tooltipActive: 'View unchecked alarm messages'
        , tooltipInactive: 'View checked alarm messages'
            , singleExecution: false
        , config: {staticParams: {stmt: 'UPDATE_ALARM_STATE_TEST',
numPars: '2'}
            , dynamicParams: [{'alarms_check': 'alarms_check',
scope: 'dataset'},
                {ID: 'ID', scope:
'dataset'}]
            , metaParams: {queryParams:
[{"name":"alarms_check","type":"num"}, {"name":"ID","type":"num"}]
}
        }
    ...
]

```

Where:

- The string `titlewin` defines the title of the alarm window
- `tooltipActive` and/or `tooltipInactive` allow the definition of tooltip messages.
- The `flagColumn` and `checkColumn` properties are used to set the visibility and the state of the alarm action. Admissible values are 0 (false) and 1 (true).
- The `config` element must be present to enable the action. Its content will be discussed in detail in a later paragraph, Action state management.

- **Log files download**

This action allows the download of log files for the given process. Log files are retrieved by the engine at a given location and compressed in a zip file. The name of the zip archive can be configured using static and dynamic parameters. For example, P_Atomic_0113091521.log.

Because the timestamp is actually part of the name, it is possible to insert via the web interface the start and the stop date/time to set the range for logging.

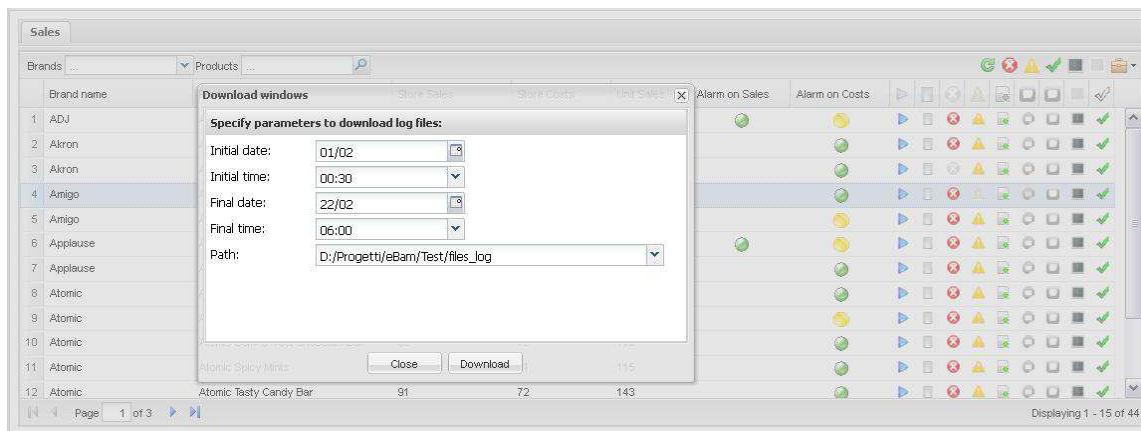


FIGURE 6.146 - Setting the start and stop date for downloading log files.

Below an example of configuration template:

```
, inlineActions: [
  {
    name: 'informationlog'
  },
  hidden: false
  ,
  tooltip: 'Download log files'
  ,
  config: {
    staticParams: {
      DIRECTORY: ['/home/spagobi/runtime/apache-tomcat-6.0.18/resources'],
      PREFIX2: 'Akron'
    }
    ,
    dynamicParams: []
  }
}
...
]
```

Where:

- `tooltip` allows the definition of a tooltip message.
- The `flagColumn` element is used to set visibility of the action. Admissible values are 0 (false) and 1 (true).
- The `config` element must be present to enable the action. Its content will be discussed in detail in a later paragraph, Action state management.

- **Cross navigation**

Similarly to the navigation bar, the detail section can be configured to host cross navigation links. Target documents usually provide detail information that may be useful to understand visualized data.

The difference between cross navigation from the navigation bar and the detail panel is that in the second case documents are executed from a single row. Therefore, they may be parameterized with data that are specific of the selected item. In addition, target documents are always opened as a popup window.



Cross Navigation

Please refer to section Cross Navigation in chapter 4, SpagoBI Server, for a full explanation of this SpagoBI feature.

Below an example of configuration template:

```
, inlineActions: [
{
    name: 'crossnav'
    , hidden: false
    , tooltip: 'Detail sales and costs'
    //, imgSrc:'ico_info.gif'
    , config: {
        target: 'new'
        , document: {label: 'chtHistorySalesDirec', staticParams:
{typeCross: 'INTERNAL'}, dynamicParams: []}
    }
}
```

```

        }
    }
    ...
]
```

Where:

- `tooltip` allows the definition of a tooltip message.
- The `imgsrc` string contains the name of an alternative image for the cross navigation icon.
- The `config` element must be present to enable the action. Its content will be discussed in detail in a later paragraph, Action state management.
- The `typecross` parameter should be defined in case you wish the target document to update the current one, instead of opening a new window. If this is the case, define the parameter as '`INTERNAL`'. If `typeCross` is not defined, the default behaviour (i.e., open an external window) will be adopted. This is particularly useful when the console is part of a composed document and you need to implement internal navigation within the document.

▪ **Enable/Disable monitoring**

This action applies at single item level and allows the user to add or remove the selected row to the list of monitored items.

Configuration template:

```

, inlineActions: [
  {
    name: 'monitor'
  , hidden: false
  , tooltipActive: 'Enable monitoring'
  , tooltipInactive: 'Disable monitoring'

  , checkColumn: 'monitor_check'

  , config: {staticParams: {stmt: 'UPDATE_MONITOR_STATE_TEST',
  numPars: '2'}}
```

```

        , dynamicParams: [{ 'monitor_check':
'monitor_check', scope: 'dataset' },
{ ID: 'ID', scope:
'dataset' }]
        , metaParams: {queryParams:
{ "name": "monitor_check", "type": "num" }, { "name": "ID", "type": "num" } }
}
}
...
]

```

Where:

- `tooltipActive` and/or `tooltipInactive` allow the definition of tooltip messages.
- The `checkColumn` element sets the state of the action. Admissible values are 0 (false) and 1 (true). If the state is set to 1, the item is added to the list of monitored objects, so it can only be removed (and vice versa).
- The `config` element must be present to enable the action. Its content will be discussed in detail in a later paragraph, Action state management.

▪ Row acknowledgement

This action allows the user to acknowledge that the item has been viewed. In other words, it changes the state of the action from enabled to disabled. Setting the row as viewed may possibly allow to remove it from the detail panel. This can be done through a statement, as explained in later paragraph “Action state management”.

Configuration template:

```

, inlineActions: [
{
    name: 'views'
    , hidden: false
    , checkColumn: 'views_check'
    , tooltipInactive: 'Checked. Click to uncheck'
    , tooltipActive: 'Unchecked. Click to check'

```

```

        , config: {staticParams: {stmt: 'UPDATE_CHECK_STATE_TEST',
numPars: '2'}
            , dynamicParams: [{views_check: 'views_check',
scope: 'dataset'},
{ID: 'ID', scope:
'dataset'}]
            , metaParams: {queryParams:
[{"name": "views_check", "type": "num"}, {"name": "ID", "type": "num"}]}
        }
    ...
]
```

Where:

- **tooltipActive** and/or **tooltipInactive** allow the definition of tooltip messages.
- The **checkColumn** element sets the state of the action. Admissible values are 0 (false) and 1 (true).
- The **config** element must be present to enable the action. Its content will be discussed in detail in a later paragraph, Action state management.

- **Select row**

This action allows the user to select a row for further operations. In other words, it does not force any action on the database by itself. Selection is valid across pages, so the user can scan multiple pages selecting items of interest and his selection will be kept. After selection, the user will choose the generic action to be executed on selected items.

It is also possible to revert the operation by clicking again on the icon. This will remove the row from the selection.

Note that the list of selected items is managed by the engine as a multivalue parameter. Therefore, it can be referenced from within statements defining generic actions (see “Action State Management”).

Inline widgets

Another option for the detail section is to show numerical values within the table as graphical widgets. Available widgets are:

- Point. Shows a rounded point only in case the value exceeds a pre-defined threshold.
- Semaphore. Shows a rounded point with one of the three semaphore colors, based on the range to which the value belongs.
- Bar. Shows a grey bar whose length is proportional to the value.

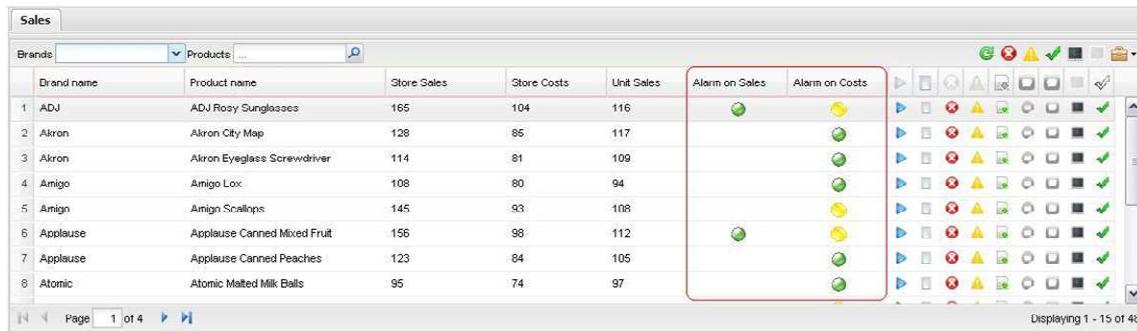


FIGURE 6.147 - Inline widgets in the RT console

Below an example of template showing inline widgets configuration:

```
, inlineCharts: [
  {
    type: 'point'
    , column: 'Alarms2'
    , color: 'green'
    , threshold: 'maxValue_costs'
      , thresholdType: 'dataset'
    , tooltip: 'Value greater than $F{maxValue_costs} '
  },{
    type: 'semaphore'
    , column: 'Alarms'
    , thresholdFirstInt: 'minValue_unit_sales'
      , thresholdSecondInt: 'maxValue_unit_sales'
      , thresholdType: 'dataset'
    , tooltipGreen: 'Value lower than $F{minValue_unit_sales} '
```

```

        , tooltipYellow: 'Value between
$F{minValue_unit_sales} and $F{maxValue_unit_sales} '
        , tooltipRed: 'Value greater than
$F{maxValue_unit_sales} '
}, {
    type: 'bar'
    , column: 'Anomalies3'
    , color: 'gray'
    , totValue: 500 //initial value
}
]

```

Where:

- `column` defines the column whose value will be represented as a graphical widget.
- The elements `tooltip`, `tooltipGreen`, `tooltipYellow`, `tooltipRed`: allow the definition of tooltips on images. Tooltips can be parameterized and contain values from the dataset (denoted with `$F{<column_name>}`) or console parameters (denoted with `$P{<parameter_name>}`).
- The elements `threshold`, `thresholdFirstInt`, `thresholdSecondInt` are used to define thresholds when relevant (point or semaphore), while
- `thresholdType` specifies threshold calculation modality. Possible values are: `static`, `dataset` or `env`. According to the chosen value, the threshold elements will contain a value, a column from the dataset or then name of a console parameter.

Parameter management

Some actions, such as cross navigation, may require the definition of parameters upon execution. In this section we explain how to define parameters.

There are two possible types of parameters:

- static parameters
- dynamic parameters

Static parameters are assigned a fixed value in the configuration template, while dynamic parameters are defined at runtime. The value for dynamic parameters can be retrieved from three sources:

- dataset
- execution context (environment)
- user input

In the example below you can an excerpt of template that configures parameters.

```
{
    name: 'start'
    , hidden: false
    , checkColumn: 'pid'
    , tooltip: 'Start the process'
    , config : {
        document: {
            label: 'processTest'
            , staticParams: {idService: '2', resourceName: 'azTest', formatDate: 'dd-MM-yyyy HH:mm:ss'}
            , dynamicParams: [
                {'multi_value_field': 'Multiple values '
                 , scope: 'promptable'

                 , values:{}
                   type: 'checkList' //combo, text,
checkList, lookup
                   , datasetLabel: 'DS_TEST_CONSOLE_PROMPT'
// only if it uses a dataset
                   , valueField: 'value' //the column with
the value information
                   , descField: 'descriz' //the column with
the description information
                   , defaultValue: ''
                 }
            ],{'combo_field_1': 'Combo value 1'
             , scope: 'promptable'

             , values: {type:'combo'
               , defaultValue: ''
             }
           }
         }
       }
     }
   }
 }
```

```

        , data: [['', 'Parziale'], ['full',
'Parziale Full'], ['reset', 'Reset']]
    }
}, {'combo_field_2': 'Combo value 2 '
, scope: 'promptable'

, values: {type:'combo'
, defaultValue: ''
, datasetLabel:
'DS_TEST_CONSOLE_PROMPT'
}
}, {'single_value_field': 'Single value '
, scope: 'promptable'

, values: {type:'lookup'
, datasetLabel:
'DS_TEST_CONSOLE_PROMPT'
, valueField: 'value' //the column
with the value information
, descField: 'descriz' //the column
with the description information
, defaultValue: ''
}
}, {'text_field': 'Textarea value '
, scope: 'promptable'

, values: {type:'text'
, defaultValue: 'aaa'
}
}, {'date_field': 'Data value '
, scope: 'promptable'

, values: {type:'data'
, format:'d/Y/m'
, defaultValue: ''
}
}, {'radio_field': 'Group value '
, scope: 'promptable'
, defaultValue: ''
, values: {type: 'group'
//, defaultValue: 'aaa'
, options:[{values:
{ type:'combo'
, title: 'Combo
value'
, data: [['', 'Parziale'], ['full', 'Parziale Full'], ['reset', 'Reset']]
}
}, {values:
{ type:'data'

```



```

        , valueField:
        'value' //the column with the value information
        , descField:
        'descriz' //the column with the description information
    }
}
]
}
} ]}} //no notifyStartAction
}

```

Where:

- **staticParams** defines static parameters
- **dynamicParams** defines dynamic parameters. The type of source is defined by the **scope** attribute. In particular, the scope attribute has three possible values:
 - **dataset** means that the value will be searched in the dataset columns.
 - **env** refers to the execution environment of the console. The parameter will be given the value of the context parameter specified in the console.
 - **promptable** means that the user will be asked to assign a value to the parameter (see FIGURE 6.148). Possible options for user input are: free input (type is '**text**') '**combo**', '**lookup**' (for single value), '**checkbox**' (for multiple values) or '**date**'.

An example of dynamic parameter with env or dataset attributes is shown below:

```

dynamicParams: [ {
    brandname: 'P1'
    , scope: 'dataset'
}, {
    month: 'P4'
    , scope: 'env'
} ]

```

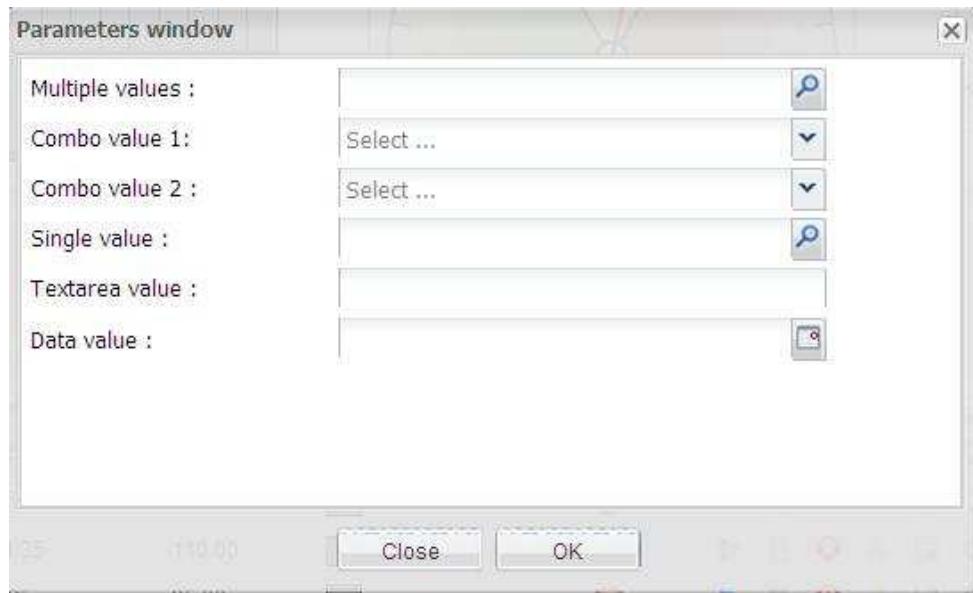


FIGURE 6.148 - Parameters window for user input

Objects like checklist, lookup or combo need to retrieve values that will be shown to the user. There are two options:

- Manually set possible values using the object `data[]`. This is only possible for combo objects (see in the example above the definition of `combo_field_1`)
- Use a dataset

In the second case the following information should be configured:

- `datasetLabel`. The label of the dataset as registered in SpagoBI. In this case the dataset does not have to be included in the initial list of datasets (at the beginning of the template).
- `valueField`. The dataset column providing the actual value.
- `descField`. The dataset column providing the description. If this option is not defined, the value of `valueField` will be taken here.

For the ‘date’ type, you can set the format using the JSON property `format`. Note that the value must be a valid format for ExtJS.

Action state management

So far we have explained all possible actions, both massive and inline. Some of those actions, typically acknowledgement actions (view, errors and alarms) require that their effect is saved and kept for further updates. For example, if a user has already checked an error as acknowledged, this information should remain valid and visible whenever the console is executed again.

Persistence of action effects is managed by the Console Engine by means of two elements:

- A specific configuration file, where statements defining the effects of actions are specified. Statements consist of SQL code, possibly containing placeholder for parameters. This file must be located at \webapps\SpagoBIConsoleEngine\WEB-INF\conf\commons with the name *statements.xml*
- A dedicated section of the console template, where parameters (if any) to be passed to statements are defined.

Let us first consider the following example of statements.xml file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<STATEMENTS>
    <!-- start statements for test console -->
    <STATEMENT name="UPDATE_MONITOR_STATE_TEST"
               query="UPDATE TEST_CONSOLE SET FLG1 = ?
WHERE ID = ? "/>

    <STATEMENT name="UPDATE_CHECK_STATE_TEST"
               query="UPDATE TEST_CONSOLE SET FLG6 = ?
WHERE ID = ? "/>

    <STATEMENT name="UPDATE_ERROR_STATE_TEST"
               query="UPDATE TEST_CONSOLE SET FLG3 = ?
WHERE ID = ? "/>

    <STATEMENT name="UPDATE_ALARM_STATE_TEST"
               query="UPDATE TEST_CONSOLE SET FLG5 = ?
WHERE ID = ? "/>

    <STATEMENT name="UPDATE_ALL_ERRORS_STATE_TEST"
               query="UPDATE TEST_CONSOLE SET FLG3 = ?, COLUMN3 = ?
WHERE FLG2 = 1
```

```

        AND COLUMN1 like ? and COLUMN2
like ? "/>

<STATEMENT name="UPDATE_ALL_ALARMS_STATE_TEST"
            query="UPDATE TEST_CONSOLE SET FLG5 = ?,"
COLUMN3 = ?
                WHERE FLG4 = 1
                AND COLUMN1 like ? and COLUMN2
like ? "/>

<STATEMENT name="UPDATE_ALL_VIEWING_STATE_TEST"
            query="UPDATE TEST_CONSOLE SET FLG6 = ?,"
COLUMN3 = ?
                WHERE COLUMN1 like ? and COLUMN2 like
? "/>

<STATEMENT name="UPDATE_ALL_VIEWING_STATE_TEST_BY_CHECK"
            query="UPDATE TEST_CONSOLE SET FLG6 = 1,
COLUMN3 = ?
                WHERE ID IN ($) "/>

<!-- end statements for test console -->

</STATEMENTS>

```

As you may notice, statements are SQL statements with placeholders. In particular:

- The ? symbol stands for single-value parameters
- The \$ symbol stands for multi-value parameters

At action execution time, placeholders will be substituted by corresponding parameters' values.

Now let us see how to define those parameters in the dedicated section of the template. The section is denoted by the config{} element and should be added to any acknowledge action to enable statements. More precisely, each statement in the statements.xml file should have a corresponding config object in the correct action.

In the following we consider the generic action of acknowledgement. The same logics applies to any other action.

```
{
    name: 'viewsName'
    , type: 'views'
    , tooltipActive: 'Mark all rows as not viewed'
    , tooltipInactive: 'Mark all rows as viewed'
    , checkColumn: 'views_check'
    , hideSelectedRow: true //true: hides the checkbox of
selectedRows, false: reset checkbox value
    , hidden: false
    , config: {
        staticParams: {stmt:
'UPDATE_ALL_VIEWING_STATE_TEST_BY_CHECK'
, numPars: '1'
//the number of params ? in the query
        }
        , dynamicParams: [{ 'userId': 'userId' ,
scope: 'env' },
{'ID': 'ID'
, scope: 'env' , useSelectedRow: true}]
        , metaParams: {queryParams:
[ { "name": "userId", "type": "string" },
{ "name": "ID", "type": "list_string", "default": "" } ] }
    }
}
```

Where:

- The `config` object contains parameters configuration. In particular it contains three objects:

Config objects		
staticParams		
	stmt	Label of the corresponding statement
	numPars	Number of single value parameters (only ? placeholder)
dynamicParams		List of all parameters that will be passed to the statement, in the format <parameter_label>:<object_name>.
	scope	Source of values for parameters. For generic actions, only 'env' is allowed. For inline actions, 'env' or 'dataset'. If dataset, the dataset column value for the current row will be passed.

	useSelectedRow	If true, the list of selected row (via check action) will be passed. Because this is a list, the corresponding statement contains a \$ placeholder.
metaParams		
	queryParams	<p>Includes 'name', 'type' and 'default' value.</p> <p>For single value (placeholder ?) type can be 'string' or 'num'.</p> <p>For multivalue (placeholder \$) type can be 'list_string' or 'num_string'</p>

TABLE 6.43 - Configuration options for parameters management in the RT console template

Advanced Functionalities

In addition to all functionalities describe above, the Console Engine offers two extra features that may be useful in specific situations.

The first is the ability to contextualize the items of the detail panel, according to the execution environment language and/or other dynamic conditions (localization). The second is the export functionality, available in most SpagoBI engines.

Localization

The Console Engine is able to manage the contextualization of all headers in the detail panel, as well as tooltips and messages. This feature can be useful not only in international contexts, but also when table headers may depend on retrieved data and cannot be set a priori.

Column headers in the detail table can be localized in two different ways:

- With the dataset modality, default column headers will be replaced by specific column names returned by the same dataset containing actual data. The column to be used is set in the template.
- With the datasetI18N modality, default column headers will be replaced by values retrieved from a decode table. This dataset must exactly

contain three columns named: ‘code’, ‘label’ and ‘locale’. Label values will be selected based on code and locale, and set as column header.

In both cases, the default header of the table column will be automatically hidden and replaced by the dynamic one.

Note that all datasets used by the console automatically receive the LOCALE parameter, so there is no need to define it in the list of parameters when defining the dataset.

Below an excerpt of configuration template.

```
, table: {
    columnConfig: {
        'brand_name': {
            width: 150
            , header: 'label_brand_name'
            , headerType: 'dataset'
        }, 'product_name': {
            width: 200
            , header: 'label_product_name'
            , headerType: 'dataset'
        }, 'store_sales': {
            width: 120
            , header: 'cod_StoreSales'
            , headerType: 'i18N'
            , type: 'int'
        }, 'store_cost': {
            width: 100
            , header: 'cod_StoreCosts'
            , headerType: 'datasetI18N'
            , type: 'int'
        }, 'unit_sales': {
            width: 80
            , header: 'cod_UnitSales'
            , headerType: 'datasetI18N'
            , type: 'int'
        }, 'Alarms': {
            width: 100
            , header: 'Alarm on Costs'
            , headerType: 'static'
        }, 'Alarms2': {
            width: 100
            , header: 'Alarm on Sales'
            , headerType: 'static'
        }
    }
}
```

Where:

- Localization is managed within the `columnConfig{}` object, which contains the configuration for each column.
- `i18N` specifies the internationalization option
- For each column, there are three possible options for `headerType`:

Options for column headerType	
static	The header is set to the value of the property 'header'
dataset	The header will be retrieved from the column whose name is the value of the property 'header'
datasetI18N	The header will be retrieved from the decode table, setting code = the value of the property 'header' and locale = current LOCALE value

TABLE 6.44 - Options for column headers with internationalization

In addition to table headers, it is possible to localize name of detail pages, tooltips as well as messages that are shown to confirm an action.

Differently from the previous case, where only template modifications are required, here you also need to update the locale files located at `webapps\SpagoBIConsoleEngine\user_messages`. Their names are: `en.js`, `it.js`, `fr.js` and `es.js`.

```
Ext.ns("Sbi.locale");
Sbi.locale.ln = Sbi.locale.ln || new Array();
//=====
=====
// USER MESSAGE EXAMPLES
//=====
=====
Sbi.locale.ln['cod_StoreSales'] = 'Store Sales';
Sbi.locale.ln['cod_StoreCost'] = 'Store Cost';
Sbi.locale.ln['titlePage'] = 'Console detail of allarms for
pratic $P{PAR} and $P{PAR2}';
```

Then you should modify the console template as follows:

```
// Page 1
{
    title: LN(titlePage)
    , msg: ''
...
}
```

The LN(<code>) function translates the argument code by reading the correct locale file (recall that the console has the locale parameter within its execution context).

Export

The Console Engine supports the export of data from the detail panel into XLS or CSV formats. This functionality is automatically added to the generic action (and filter) bar.

Exported tables may be different from the original detail table. Export columns are set using a specific dataset for export. This can be useful, for example, when only a subset of data is shown in the detail panel, but all of them should be exported.

Below an example of template configuration:

```
, table: {
    dataset: 'testConsole'
    , datasetLabels: 'testConsoleLabels'
    , columnId: 'ID'
    , columnConfig: {...}
    , datasetExport : {
        datasetExp : 'BOOK_DS_CONSOLE_002_EXP'

        , columnConfig : {
            'brandname': {
                width: 120
                , header: 'label_brand_name'
                , headerType: 'dataset'
            }, 'productName': {
                width: 120
                , header: 'label_product_name'
```

```
        , headerType: 'dataset'
    }, 'Alarms': {
        width: 120
        , header: 'Alarm on costs'
        , headerType: 'static'
    }
}
}
```

Where:

- datasetExp is a string specifying the label of the dataset to be used in the exported table. If this dataset is not defined, the same dataset of the original detail table will be used by default.
- columnConfig specifies column headers for the exported table. Similarly to the detail table, options are: 'static', 'dataset' and 'datasetI18N'.

Mobile BI

Architecture

The general architecture of the Mobile engine reflects the common strategy of all engines, well known by now. First, a template is created for the specific type of document. Then, a specific application for mobile devices, accesses the same Server and directly interacts with it to produce interactive documents available on mobile devices.

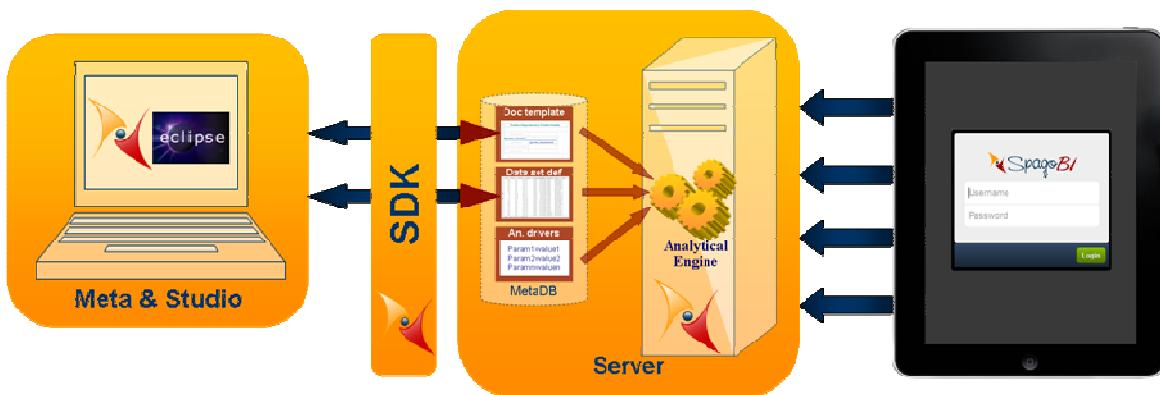


FIGURE 6.149 - Final architecture of the SpagoBI Mobile module

As for many other document types, the templates for mobile documents are very easy to understand and manually write. So, they can be directly uploaded on SpagoBI Server using the developer/administrator web interface.

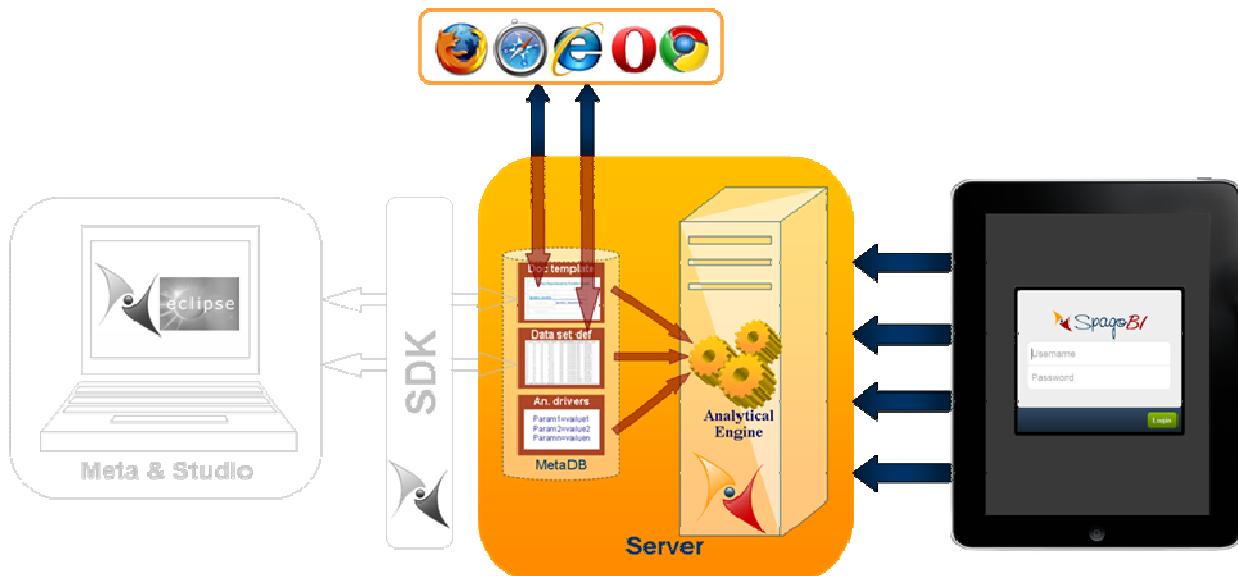


FIGURE 6.150 - Current architecture of the SpagoBI Mobile module



Graphical designers for mobile

At present, the graphical designer for mobile documents are not yet available. So, the developer must write by hand the templates and upload them into SpagoBI Server.

Installation

SpagoBIMobileEngine is a web application that needs to be deployed under an application server. This is the actual endpoint where users can access SpagoBIMobileEngine to execute documents.

Documents are executed by users via their mobile browsers, such as Safari or Chrome. Users connect to the web application and authenticate themselves on SpagoBI Server. This web-based technology allows users to take advantage of SpagoBI Mobile engine functionalities from several mobile devices, such as Apple iPad and Android-based tablets, without the need to download it from a dedicated online shop (e.g., the AppStore).

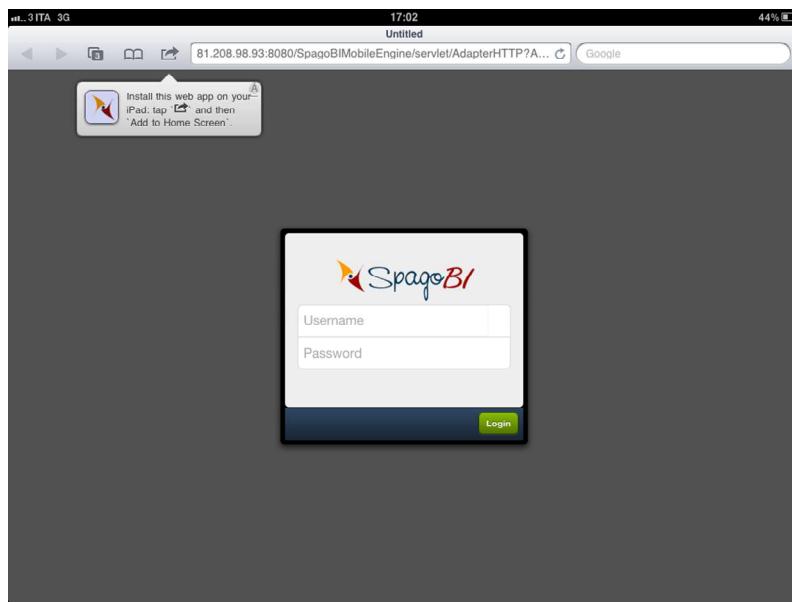


FIGURE 6.151 - Installation and login of SpagoBI Mobile client on iPad

A customized application for Apple iPad is available, which installs a link to SpagoBI Mobile engine. After the first installation, users will see SpagoBI icon on their home screen: clicking on it, they will directly access documents without any further need to open the browser and connect to the web application (see FIGURE 6.151). This ensures transparent and native-like access to SpagoBI documents.

SpagoBIMobileEngine is designed to access and execute documents: as such, it provides neither document development features nor administration tools. To create or modify documents, as well as to manage authentication credentials, you need to install an instance of SpagoBI Server. This Server may run on the same application server of SpagoBI Mobile, or even on another one: the only requirement is that the two servers work with the same SpagoBI metadata database schema.



On-line and off-line

SpagoBIMobile Engine is implemented using HTML5 and CSS3, which actually support offline working. In the current release, it is able to work online only and the user is authenticated at any new session. In the near future, the offline working modality will be implemented.

Navigator

The login page is the entry-point to SpagoBI Mobile engine. The user authenticates by providing username and password. Authentication credentials are stored in SpagoBI Server profile management system.

Once logged in, the user will see the mobile version of the document browser (see FIGURE 6.151).



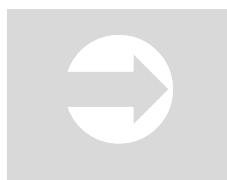
Document browser

The document browser is a cross functionality of SpagoBI Server. It used to explore document hierarchies and execute them via an intuitive graphical interface. Read more at section Cross Services, chapter 5 – SpagoBI Server.

On the left, the list of available mobile documents is shown in a tree format. The tree only contains documents of type *mobile*, which are accessible to the user according to his rights (role and user profile). Therefore, access is profile-based as in all the other engines, as described by the Behavioural Model.

Documents can be organized in functionalities (folders), in the same way as with documents on the traditional SpagoBI Server. Functionalities must be

managed using SpagoBI traditional web interface, via **Analytical Model > Functionalities Management**. Then, they can be seen on a mobile device. Roles and user profiles must also be managed on the Server via tools available for the management of the Behavioural Model.



The behavioral model

The behavioral model rules visibility and rights on documents, based on roles and user profiles. For a full understanding of its meaning and functionalities, please refer to section The Behavioral Model, chapter 5 – SpagoBI Server.

You can navigate through the hierarchical documents tree. Each level can be composed of documents and folders. Folders are identified by an arrow icon, while each document type has its own icon.

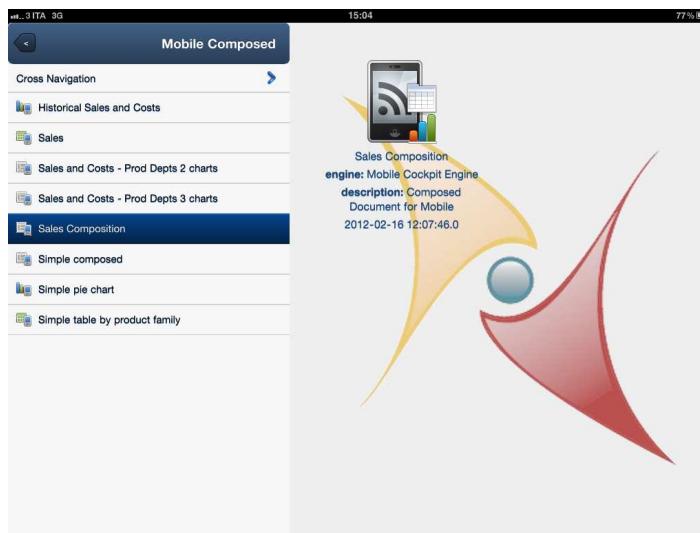


FIGURE 6.152 - Mobile navigator

Tapping on a menu item will cause two different actions depending on the type of node: if the node is a folder, you will see all documents contained in that folder, which appear in the tree menu. If it is a mobile document, you will start the execution of the document. Note that documents other than mobile are not shown in the mobile navigator.

There are currently three types of mobile documents:

- Mobile Report. To display data in a table layout
- Mobile Chart. To represent data using graphical charts
- Mobile Cockpit. To aggregate documents of type Report and Chart, enabling their dynamic refresh via internal document navigation.

In the following sections we discuss each type of report in detail.

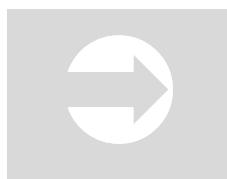
SpagoBI Table Mobile Engine

The Mobile Table document, displays data in a table layout. The user can interact with this widget sorting the columns, tapping on the column header, or highlight the rows, tapping over them.

To create an interactive table layout report that can be accessed from a mobile device, follow these steps:

- Create a dataset that will provide data to the report
- Create the template
- Register the document on SpagoBI Server.

The procedure to register a document on the Server is the same as with all other analytical documents.



Create analytical documents on SpagoBI Server

The process to create and register a document on the Server is described in detail at section Analytical Model, chapter 5 – SpagoBI Server

Specific configuration settings for the document creation interface are:

- **Mobile Report** as Document type

- **Mobile Report Engine** as Engine

As said above, templates must be currently edited by hand. Below an example of template for a mobile report.

```
<?xml version="1.0" encoding="windows-1250"?>
<TABLE_WIDGET>
    <TITLE value="Foodmart Products" style="font-weight:bold; font-family: Verdana; font-size: 16px;"/>
    <COLUMNS>
        <COLUMN header="Brand" value="BRAND_NAME" style="color: red;">
            </COLUMN>
        <COLUMN header="Product" value="PRODUCT_NAME" style="background-color: silver; border: 1px solid grey;">
            <COLUMN header="SRP" value="SRP" alarm="ALARM1">
                <CONDITIONS>
                    <CONDITION style="color: blue;">
                        <![CDATA[=0]]>
                    </CONDITION>
                    <CONDITION style="color: pink;">
                        <![CDATA[=1]]>
                    </CONDITION>
                </CONDITIONS>
            </COLUMN>
            <COLUMN header="Low fat" value="LOW_FAT" alarm="ALARM2">
                <CONDITIONS>
                    <CONDITION style="color: green;">
                        <![CDATA[<2]]>
                    </CONDITION>
                    <CONDITION style="color: orange;">
                        <![CDATA[=2]]>
                    </CONDITION>
                    <CONDITION style="color: red;">
                        <![CDATA[>2]]>
                    </CONDITION>
                </CONDITIONS>
            </COLUMN>
        </COLUMNS>
    </TABLE_WIDGET>
```

Where:

- **TITLE:** tag containing the document title in the value attribute with a specific CSS inline style through the style attribute.
- **COLUMN:** each of these tags represents the dataset column to be displayed as a table column. You can customize it, using the following attributes:

Columns attributes	
header	User friendly column header name
value	Name of the dataset column providing data
style	Standard CSS3 inline style for the cell
alarm	Name of the dataset column whose value will be tested against the conditions specified in the CONDITIONS tag. If conditions are satisfied, the cells of the corresponding row in the COLUMN will have their style changed according to the STYLE tag.

TABLE 6.45 - Column attributes for mobile tables

- **condition:** each of these tags determines a condition to be applied to the alarm column of the current **column** tag. Mandatory attributes are:

Condition attributes	
STYLE	CSS style to apply to the current COLUMN cell if the condition of the alarm column value on the corresponding row is verified
CDATA	Simple Javascript condition

TABLE 6.46 - Condition attributes for mobile reports

Cross navigation

Mobile reports support cross navigation.



Cross Navigation

Cross navigation is the key feature of SpagoBI that allows navigation from one document to another. To understand how it works, please refer to the corresponding section in chapter 5 - SpagoBI Server.

To define a cross navigation starting from a Mobile Report document, you should configure a `drill` tag on its template. An example follows:

```
<DRILL document="MobileDoc">
  <PARAM name="family" type="CATEGORY"/>
  <PARAM name="value" type="SERIE"/>
  <PARAM name="state" type="RELATIVE"/>
  <PARAM name="text" type="ABSOLUTE" value="Analysis"/>
</DRILL>
```

The cross navigation target document is specified by the `document` attribute in the `DRILL` tag.

The inner `PARAM` tags allows the definition of parameter values for analytical drivers of the target document. The `name` attribute must match the URL of the parameter in the target document. The `type` attribute accepts the following values:

Values for parameter type attribute	
CATEGORY	The name of the column that you tapped will be considered as the value for the analytical driver of the target document
SERIE	The value of the cell that you tapped will be considered as the value for the analytical driver of the target document
RELATIVE	The analytical driver of the source document and its value will be propagated to the target document, with the same name (URL).
ABSOLUTE	Name (URL) of the parameter to be propagated in cross navigation. The value that will be passed to the target document is fixed, as given by the "value"

	attribute.
--	------------

TABLE 6.47 - Values for parameter type attribute in mobile reports

SpagoBIChartMobileEngine

The Mobile Chart document displays data in various types of charts. Depending on the chart type, the user can zoom in and out, pinch the chart, rotate or enable/disable categories tapping over the chart legend.

To create an interactive chart that can be accessed from a mobile device, you should follow these steps:

- Create a dataset that will provide data to the report
- Create the template
- Register the document on SpagoBI Server.

The procedure to register a document on the Server is the same as with all analytical document.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in at section Analytical Model, chapter 5 – SpagoBI Server.

Specific configuration settings for the document creation interface are:

- **Mobile Chart** as Document type
- **Mobile Chart Engine** as Engine

As said above, templates must be currently edited by hand. Below an example of template for a mobile chart:

```
<?xml version="1.0" encoding="UTF-8"?>
<CHART_WIDGET width='80%' height='80%'>

<LEGEND>
```

```

<POSITION portrait='bottom' landscape='bottom' />
</LEGEND>

<INTERACTIONS_LIST>
    <INTERACTIONS type='reset' confirm='true' />
    <INTERACTIONS type='rotate' />
</INTERACTIONS_LIST>

<ANIMATE>true</ANIMATE>

<SERIES_LIST>
    <SERIE field='value' type='pie' showInLegend="true"
highlight="true">
        <LABEL field='family' contrast='true' font='18px
Arial' display='rotate' />
    </SERIE>
</SERIES_LIST>
</CHART_WIDGET>
```

Where:

- The tag `CHART_WIDGET` includes the template definition
- The tag `LEGEND`, if present, specifies position and orientation of the legend. If the tag is not specified in the template, the legend will not be displayed.
- The tag `INTERACTIONS_LIST` includes a sequence of possible interactions with the chart.
- The tag `ANIMATE` enables animation effects when the chart is first drawn
- The tag `SERIES_LIST` includes the definition of series. Each series is included in the `SERIE` tag and has the following attributes:

Series attributes	
field	Contains values for the series
type	Available options for charts: Area/Bar/Column/Gauge/Line/Pie/Radar/Scatter
showInLegend	If true, the series will be shown in the legend.

TABLE 6.48 - Series attributes for mobile charts

- The **LABEL** tag, inside a series, specifies how values in the series should be visualized. Its attributes are: **field**, which specifies the name of the label; **contrast**, **font**, **display**, which configure the style.

Another relevant tag, which is not shown in the example above, is the **AXES** tag. It describes the structure of the axes in the chart. For example:

```
<AXES_LIST>
    <AXES type='Numeric' position='left' title='Kpi Value'>
        <FIELDS_LIST>
            <FIELDS>KPI1</FIELDS>
            <FIELDS>KP2</FIELDS>

        </FIELDS_LIST>
    </AXES>
    <AXES type='Category' position='bottom' title='Time'>
        <FIELDS_LIST>
            <FIELDS>date</FIELDS>
        </FIELDS_LIST>
    </AXES>
</AXES_LIST>
```

Where:

- AXES_LIST** is the tag that includes axes definition.
- The tag **AXES** includes one axis. Its properties are summarized below

Axis attributes		
type	If value = Category	The corresponding axis is the axis of categories
	If value = Gauge	Typically series axis. Series of type gauge.
	If value = Radial	Typically series axis. Series of type radial.
	If value = Numeric	Typically series axis. Series of type numeric.
	If value = Time	Typically series axis. Series of type datetime.
position	Position of the axis. Possible options: top, left, bottom, right.	
title	Title of the axis	

TABLE 6.49 - Axis attributes for mobile charts

- The <FIELDS_LIST> tag inside <AXES> contains values for the axis. If this is an axis category, there is only one <FIELDS> element, whose value is the name of the dataset column returning categories. If this is a series list, there is one tag <FIELDS> for each series and its value is the name of the series.
- The tag <OPTIONS> contains a set of options. Currently the only supported option is showValueTip. If true, by tapping on a point, its details will be made visible.

```
<OPTIONS showValueTip='true' />
```

New template creation

The template is an XML representation of the JSON chart configuration of Javascript Sencha Touch framework.



Sencha Touch Javascript Framework

Read more about the Sencha Touch framework at <http://www.sencha.com/products/touch/>.

For a list of chart examples with configuration properties, you can browse the following links:
<http://dev.sencha.com/deploy/touch-charts-1.0.0/examples/>
<http://docs.sencha.com/touch-charts/1-0/>

Expert developers may add more chart types to available ones by translating the JSON configuration template into a SpagoBI Mobile template. The translation from JSON to XML should follow these instructions:

- Each JSON object is an XML tag
- Each JSON object attribute is an XML attribute of the corresponding tag.

For example, the following JSON object :

```
legend: {  
    position: {  
        portrait: 'bottom',  
        landscape: 'bottom'  
    }  
}
```

becomes:

```
<LEGEND>  
    <POSITION portrait='bottom' landscape='bottom' />  
</LEGEND>
```

- An array of n JSON objects is a list of n XML tags, nested inside a tag having the same name followed by _LIST suffix.*

For example, the JSON array:

```
interactions: [  
{  
type: 'reset',  
confirm: true  
,  
{  
type: 'rotate'  
}  
]
```

becomes:

```
<INTERACTIONS_LIST>  
    <INTERACTIONS type='reset' confirm='true' />  
    <INTERACTIONS type='rotate' />  
</INTERACTIONS_LIST>
```

- To translate JSON object names into XML tags, simply transform each lowercase letter to uppercase and put “_” before the uppercase letter of the original name.

For example:

```
markerConfig: {
    radius: 5,
    size: 5
}
becomes:
<MARKER_CONFIG radius="5" size="5">
</MARKER_CONFIG>
```

- JSON object attributes are XML tag attributes and maintain the same case sensibility.



Writing the template

Note that the template is case sensitive: use uppercase tags and lowercase properties.

Cross navigation

Cross navigation on mobile charts is very similar to cross navigation on reports. The only difference lies in the cross navigation configuration and concerns the type attribute of PARAM tag:

Values for parameter type attribute	
CATEGORY	The category associated to the point where you tapped will be considered as the value for the analytical driver of the target document
SERIE	The value of the series associated to the point where you tapped will be considered as the value for the analytical driver of the target document
RELATIVE	The analytical driver of the source document and its value will be propagated to the target document, with

	the same name (URL).
ABSOLUTE	Name (URL) of the parameter to be propagated in cross navigation. The value that will be passed to the target document is fixed, as given by the “value” attribute.
SERIE_NAME	The name of the series associated to the point where you tapped will be considered as the value for the analytical driver of the target document

TABLE 6.50 - Values for parameter type attribute for mobile charts

SpagoBICockpitMobileEngine

The mobile cockpit has the same interaction patterns of its composing documents (table and chart), in addition to a slider bar and internal navigation features, as explained below.

To create cockpits that can be accessed from a mobile device, you should follow these steps:

- Create the mobile documents composing the cockpit (report or chart)
- Create the template
- Register the document on SpagoBI Server

The procedure to register a document on the Server is the same as with all analytical document.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Document, chapter 4.

Specific configuration settings for the document creation interface are:

- **Mobile Cockpit** as Document type
- **Mobile Cockpit Engine** as Engine

As said above, templates must be currently edited by hand. Below an example of template for a mobile cockpit.

This is an example of Mobile Cockpit template:

```
<?xml version="1.0" encoding="windows-1250"?>
<COMPOSED_WIDGET>
    <TITLE value="Sales and Costs" style="font-weight:bold;
font-family: Verdana; font-size: 16px;"/>
    <DOCUMENTS>
        <DOCUMENT label="SBI_OBJ_301" width="50%" height="100%">
            </DOCUMENT>
        <DOCUMENT label="SBI_OBJ_302" width="50%" height="100%">
            <IN_PARAMETERS>
                <PARAMETER urlName="ProdFamily" defaultValue="Food"/>
            </IN_PARAMETERS>
        </DOCUMENT>
    </DOCUMENTS>
</COMPOSED_WIDGET>
```

Where:

- Each **DOCUMENT** tag defines a document part of the composition. The document specified by the **label** attribute must be an already existing and valid SpagoBI Server document for mobile devices; **width** and **height** attributes specify the dimension of the document inside the composition.
- The **IN_PARAMETERS** tag is a reference to the parameters' name of the composing (inner) document. If the composing document and the cockpit share the same name for a parameter, the association between the two is done automatically. If not, the name (URL) of the parameter of the composing document must be specified in the **urlName** attribute of the **parameter** tag. The default value should be specified in the **value** attribute.

Cross navigation

In a Mobile Cockpit document, the cross navigation mechanism defines the interactions among the documents of the composition. When tapping on a document, the other documents can be refreshed with new values for their analytical drivers, depending on the point where you tapped.

For example, in FIGURE 6.153 we can tap on a state on the left report. As a result, the chart on the right is refreshed with sales and costs for that state.

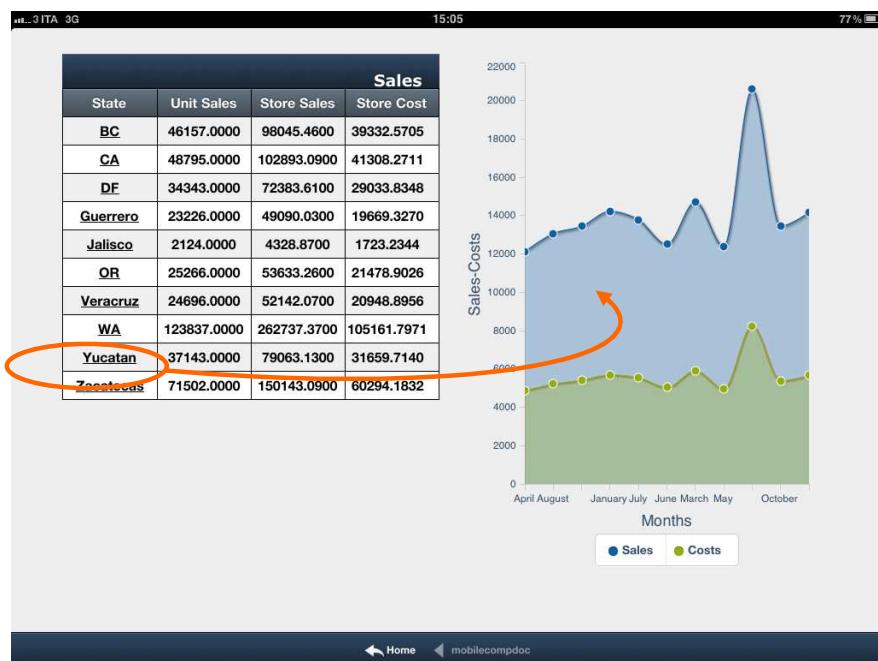
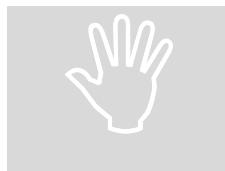


FIGURE 6.153 – Mobile cockpit composed of a table and a chart

To define a cross navigation in a Mobile Cockpit, you only have to define cross navigation inside each document of the composition. No further configuration is required at the cockpit level. This means that, in order to be refreshed, target documents must have a parameter with the same name of the category/series/parameter passed by the source document. Documents will be re-executed only if at least one parameter had its value changed from the current execution settings.



Attention

Note that the `attribute` document of the `DRILL` tag specified in the document template is not considered when cross navigating inside the cockpit document.

Note that each document of the composition will be re-executed only if the value of at least one analytical driver is changed from the current value: considering the example above, if you tap on the “Food” category on the chart, the other documents will be re-executed only if their “family”analytical driver has not the “Food” value.

Adding the tag `SLIDER` as a child of `COMPOSED_WIDGET` tag, it is possible to display a Slider field at the bottom of the cockpit. The slider allows users to pass numeric values to one or both documents that compose the cockpit. By dragging the pointer of the slider, the user can pass the selected value to all documents accepting a parameter called with the exact `name` attribute of the slider. Each document accepting this parameter will be refreshed.

Add a tag as the following one:

```
<SLIDER name="year" value="2008" maxValue="2011" minValue="2008"
label="Year"/>
```

Where:

- `slider`: tag containing the slider configuration.

The attributes that you can use to customize it are:

Slider attributes	
<code>name</code>	The field's HTML name attribute, which must correspond to the input parameter's name of the single documents that compose the cockpit.
<code>value</code>	The starting value set when the document starts
<code>maxValue</code>	The max value of the slider

minValue	The minimum value of the slider
label	The value of the label that could be set over the HTML slider field

TABLE 6.51 - Slider attributes for mobile documents

The Slider item will appear as follows:

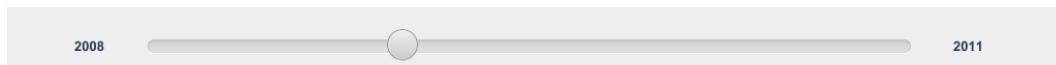


FIGURE 6.154 - Slider in a mobile cockpit

Office integration

Office integration is always required in a BI product. SpagoBI guarantees this integration with the multiformat exporters that usually support .xls, .doc and .rtf formats.



Multiformat exporters

For details about the exporting capability, please refer to the Exporters and Massive export paragraphs at section Cross-services, chapter 5 - SpagoBI Server.

Moreover, SpagoBI offers a specific engine for the publication of personal documents in BI environments created via commonly used Office tools (Open Office or MS Office).

SpagoBIOfficeEngine

SpagoBIOfficeEngine supports the provisioning of additional information to BI analysis as static documents of any format: text (e.g., PDF, XLS, DOC,

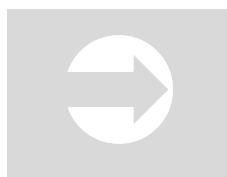
ODT, TXT, RTF) or images (e.g., GIF, JPG, PNG) or presentations (e.g., PPT, ODP).

These documents are typically created via commonly used Office tools (Open Office or MS Office), or they may be themselves the result of a BI analysis. For example:

- Scheduled reports. If saved in the document tree (typically as PDF files), they are automatically associated to SpagoBIOfficeEngine.
- Output of a workflow process. This is a PPT file that is automatically associated to SpagoBIOfficeEngine.
- Documents exported from analytical reports. These documents can be registered themselves as analytical documents, associated to SpagoBIOfficeEngine.

To create an Office document, follow the steps described below. Note that these operations are allowed for users with the *developer* role.

- Generate the static document in the desired format, either with SpagoBI or with Office tools (as discussed above)
- Create a new analytical document in the **Analytical Model > Documents Development** section. Select type **Office** and upload the static document as a template.



Create analytical documents on SpagoBI Server

The process to create and register a document on the Server is described in detail at section Analytical Model of chapter 5 - SpagoBI Server.

At document execution time, the engine simply shows the static document loaded as the template. If the current format is supported, the document will be opened within the browser. Otherwise, the user will be prompted with a download window.

Because this is a static type of document, no further processing takes place on the document. Therefore, unlike with other SpagoBI analytical documents, here it is meaningless to associate analytical drivers or to schedule document execution.

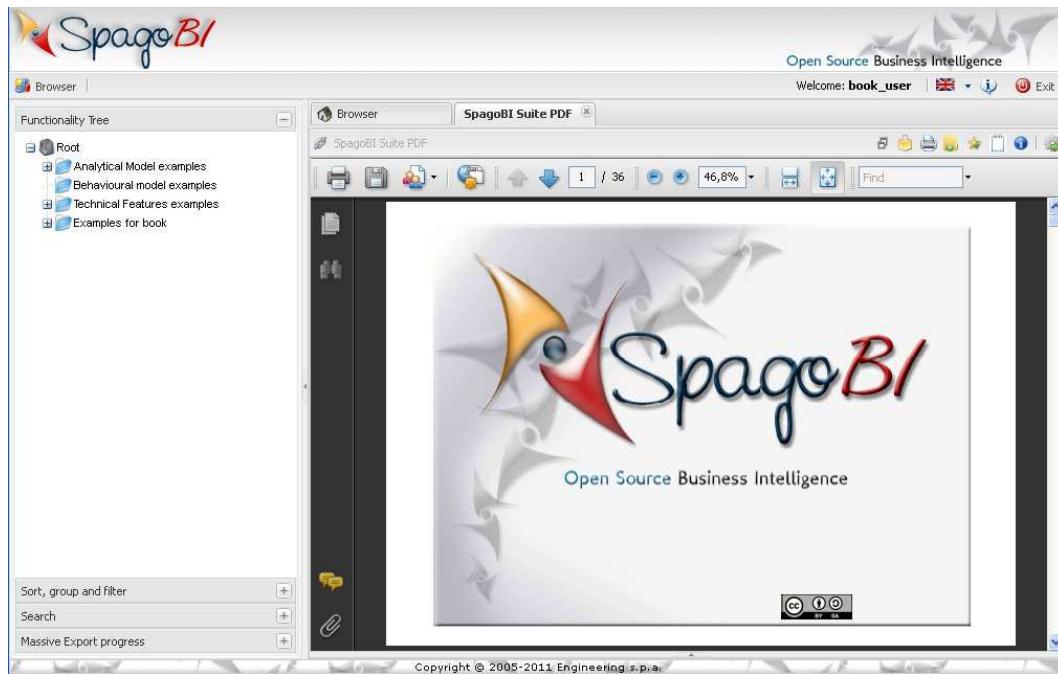


FIGURE 6.155 - SpagoBI Office document executed within the browser

Collaboration

Collaboration is an important topic for companies, which goes far beyond standard BI capabilities. SpagoBI supports collaboration between users by allowing them to share information, private knowledge and notices, thanks to annotate features available on all document types.



Notes

For any detail on the annotate capability, please refer to the 'Notes' paragraph, section Cross Services, in chapter 5 - SpagoBI Server.

Furthermore, SpagoBI offers a specific engine (SpagoBIDossierEngine) that allows to automatically create organized report dossiers, enriched with notes and comments posted by users to comment data. A collaborative workflow manages dossier components that are sent to users to later receive their notes.

SpagoBIDossierEngine

SpagoBIDossierEngine supports information sharing and collaboration by providing documents that contain both analytical data and information provided by users. Dossier documents can be used to:

- collect data at the end of a business period
- enrich business analysis with knowledge brought by single users, which may not be visible at first sight from data
- set up a meeting by involving all participants before the meeting itself, sharing a common base of information to help later communication
- automatically produce presentations, which may be repeated over time with updated data.

A dossier is a document containing a collection of report snapshots, enriched with a customizable page layout and, most importantly, with notes inserted by a collaboration process involving SpagoBI users. More specifically, the execution of a dossier document produces a presentation document (i.e., a PowerPoint file).

A dossier is an actual SpagoBI document, but it is slightly different from others for two reasons:

- the template of the document is not a unique file, but it contains some files and the configuration of the reports being part of the dossier;
- the execution of the dossier is a process, led by a workflow engine, which involves some SpagoBI users and automatic mechanisms, in some different activities.

Requirements

A dossier execution requires an OpenOffice (or LibreOffice) 2.x (or later) installation on the same server hosting SpagoBI Server. You have to start OpenOffice as a service, launching the following command from OPENOFFICE_HOME/program:

```
soffice -quickstart -  
accept="socket,host=localhost,port=9000;urp;StarOffice.ServiceManager"
```

This command may open a new OpenOffice window. In order to avoid this behaviour, you can add the following text as a command line argument:

```
- invisible
```

Note that every time you launch the soffice command a new soffice process is created and so, if you use this option, the only way to stop open office is to kill the process.

On SpagoBI side, check the configuration on the file SpagoBI/WEB-INF/conf/engines/dossier/dossier.xml:

```
<OFFICECONNECTION host="localhost" port="9000" />
```

The port number must match the one declared on the command above.

Here the creation of a new dossier document is the same as for all other SpagoBI documents. The only relevant point is that you shall edit the template via the wizard provided by the document creation interface. Let us focus on the specific template for this type of document.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Model of chapter 5 – SpagoBI Server. Correspondence between engines and document type is listed in table TABLE 5.3.

Configuring the dossier template

Clicking on the "Template build", you can configure a dossier template using a graphical interface (GUI):

The template GUI allows you to:

- add a SpagoBI document to the dossier (**Configured Document List** section). Up to 3.5 release, users are allowed to add JasperReports documents only
- load the PowerPoint template. This is a PowerPoint file which contains some placeholders that will be replaced by SpagoBI document snapshots. Once you have loaded the template file, its name will appear near the **Loaded Template** label together with a button that will allow you to download it.
- add the workflow process definition file. Once loaded, the name of the workflow process definition 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 dossier choose it from the **Add Document** tree on the right side of the GUI: select it and press **Add Document**.

This operation will get you into the configuration page (see FIGURE 6.157), which contains:

- summary information about the selected document (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 dossier. It will be used to identify the place holder, inside the PowerPoint template, which has to be replaced by the snapshot of the document;
- a parameters section where all the parameters of the document are listed together with an input text, useful to specify an execution value: you can set a value here (configuration of static drivers) or you can leave it empty and then create a new analytical driver in the dossier

document (with the same URL name of the report driver) in order to let the user choose when he will execute the dossier document.

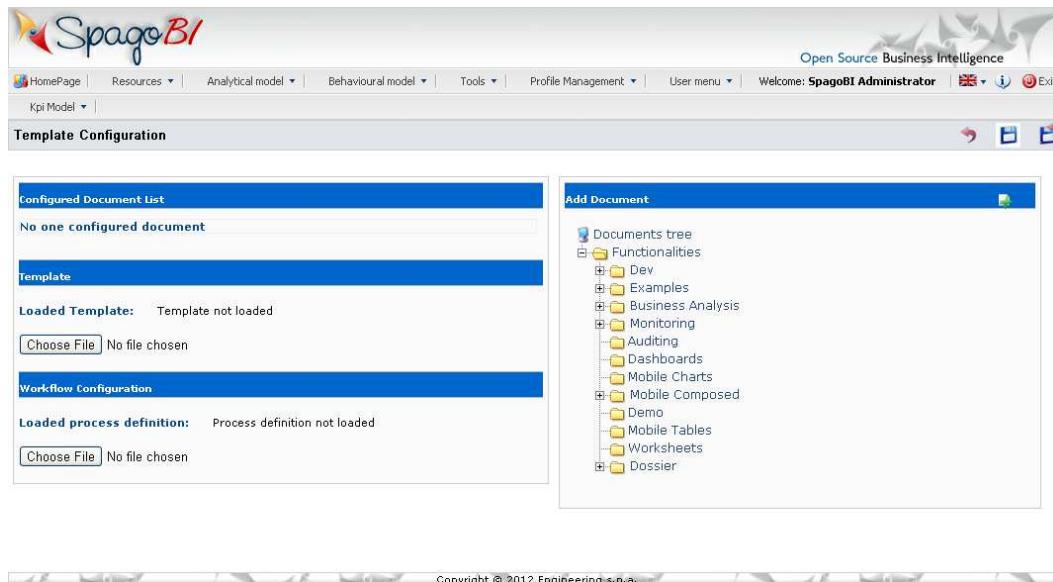


FIGURE 6.156 - Dossier template configuration GUI

Remember that values entered will be used to produce the snapshot of the documents composing the dossier. You cannot change the output format parameter (it is filled with a default value).

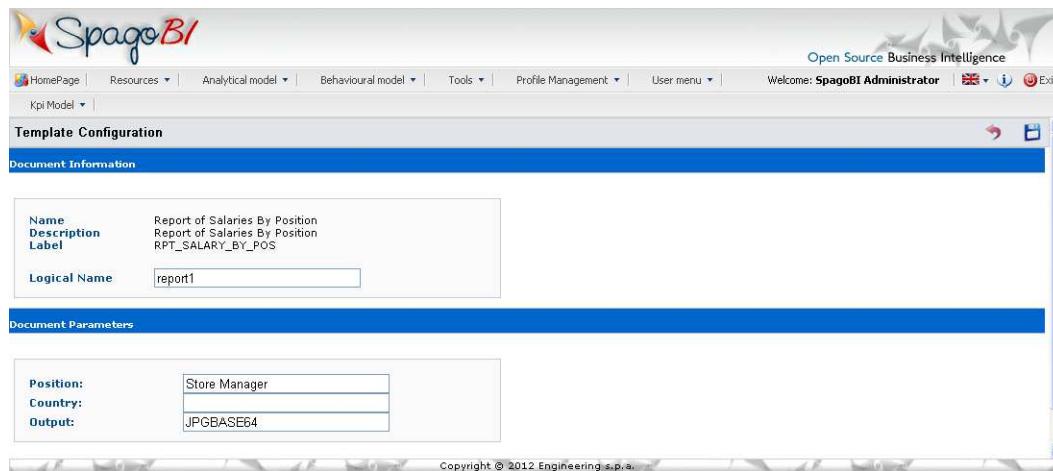


FIGURE 6.157 - Configuration page of the dossier document

Once done with configuration, save and return to the dossier template page. Here you should first load composing documents, then the presentation template and finally the booklet workflow (see the right panel of FIGURE 6.156).

Composing documents should have already been created and registered on SpagoBI Server. Click on the lookup icon to load them.

Now we will discuss the template and afterwards the workflow.

Creating a presentation template

A presentation template is a PowerPoint document, which will be the layout template for the final dossier presentation.

When you build the template, you must set the position of each snapshot. When you decide the position for one snapshot, draw a rectangular box containing a text string like this one:

```
spagobi_placeholder_logicalNameOfTheDocument
```

Replace the final part of the string with the logical name of the configured document (see above). The snapshot of the document will exactly fill the area of the rectangle, so remember to choose the right dimensions.

A second critical aspect concerns the choice of the documents to be put in the same slide. As we will shortly see, each slide is assigned to a group of users that will be enabled to make comments on it. So it is important to check whether the document you insert in a slide is actually accessible to each user belonging to that group, according to his role.



Role management

SpagoBI behavioural model includes roles, users and profiles. To understand how they relate to each other and how to assign permissions to a role, please refer to section Behavioural Model, in chapter 5 - SpagoBI Server.

Once you have defined the presentation template, load it using the interface.

Configuring booklet workflow

When you execute a dossier document, a new workflow process is started. The process consists of automatic tasks and others performed by the users. The workflow document defines these tasks.

For the current release the workflow process definition must be written using the JBPM syntax.



Building the booklet workflow

Since the dossier engine requires a fixed structure for process definition, we suggest to retrieve and modify the example contained in SpagoBI Demo binary distribution (the All-in-one package), and then change the allowed parts using a text editor.

Automatic tasks, which analyze the configuration and produce metadata, cannot be changed. Users' task definition can be changed, added or removed, within the following limits:

- the swimlane definition. A swimlane is a role of the workflow. Each task in the process definition is assigned to a swimlane. Swimlanes are the bridge between workflow roles and SpagoBI roles. You can add or remove swimlane definitions, but each of them must have the xml structure reported below:

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

You can only change the name attribute that gives it the name of a SpagoBI role as a value.

- the “Add notes” tasks between the fork node and the join node. The xml definition of the task is shown below.

```

<task-node name=" * ">
  <task name="add_note_task1" 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>
* give it a name
** name of a previous defined swimlane
*** index of the ppt document slide where the notes must be
pasted

```

The number of this task kind must be equal to the number of presentation template slides containing placeholders for SpagoBI documents. If you need to add an 'add_note_xxx' task definition, copy an existing one and paste it below the last one. Then add the reference to the task in the subsequent fork node.

For each 'add_note' you can change name, swimlane assignment and value of the spagobi_booklet_pageindex variable, which it is the index of the presentation document slide where the notes will be added. Remember that changing the swimlane name will allow SpagoBI users belonging to the same SpagoBI group to see documents on that slide and insert new notes.

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

```

<task-node name="ValidateFinalDocument">
  <task name="validate_document_task" swimlane=" * ">
    <event type="task-create">
      <script>
        <expression>

```

```

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

```

Load the modified workflow using the interface and save the whole template clicking on  at the right top corner of the page.

If composing documents are associated to any analytical driver, you must associate the same driver, with the same URI, to the dossier containing it. The document is ready for execution.

Dossier execution

A dossier is executed like any other SpagoBI document, from either the document browser or the **Analytical Document** menu. Insert parameter values, if any, and execute.

The first result of the execution is the list of generated presentations. You can download or delete the previous version from the list or generate a new version by clicking on .

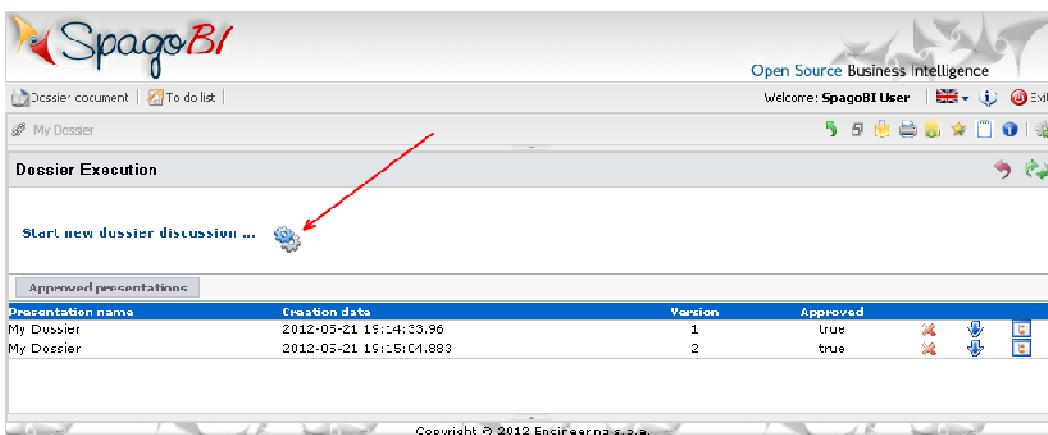


FIGURE 6.158 – Starting a new execution of a dossier document

The workflow process executes all automatic tasks and assigns to SpagoBI users the add_note activities, based on the assignments made during the workflow process definition.

The **Worklist** functionality allows users to check if the execution of a workflow task is needed. Note that the administrator should configure this functionality on the user's menu. You can accept the activity clicking on the button located at the right corner of each row (see FIGURE 6.159).

The screenshot shows the SpagoBI Worklist interface. At the top, there is a header with the SpagoBI logo, the text "Open Source Business Intelligence", and a welcome message "Welcome: SpagoBI User". Below the header, there are links for "Dossier document" and "To do list". The main area is titled "Workflow To Do List". It features a search bar with the placeholder "The value of the column Activity name as a string starts with" and a "Filter All" button. Below the search bar is a table with four columns: "Activity name", "Activity description", "Activity priority", and "Process name". The table contains two rows of data:

Activity name	Activity description	Activity priority	Process name
add_note_task2		3	DossierWorkflowProcess
add_note_task1		3	DossierWorkflowProcess

At the bottom of the table, it says "page 1 of 1". There are navigation buttons for the table and a magnifying glass icon for filtering. The footer of the page includes a copyright notice "Copyright © 2012 Engineering s.p.a.".

FIGURE 6.159 - List of add note activities for a user

Once you have accepted the add_note activity, open the notes editor. This will make the activity invisible to all other users. The editor shows snapshots of the slides associated to the activity, and a text area for the notes insertion. You can insert different notes and save them. Finally, click '**End Discussion**' at the right upper corner.

Document execution will remain on hold until all “add notes” activities have been completed: at this point the final presentations are produced and submitted for validation. Users allowed to execute the document will see in their task list a new activity (i.e., validation).

To validate the document, accept the task and open the validation form. Download the presentation by clicking on and check it. If you are happy with it, select **True** and save the document by clicking on . If not, select **False** and save.

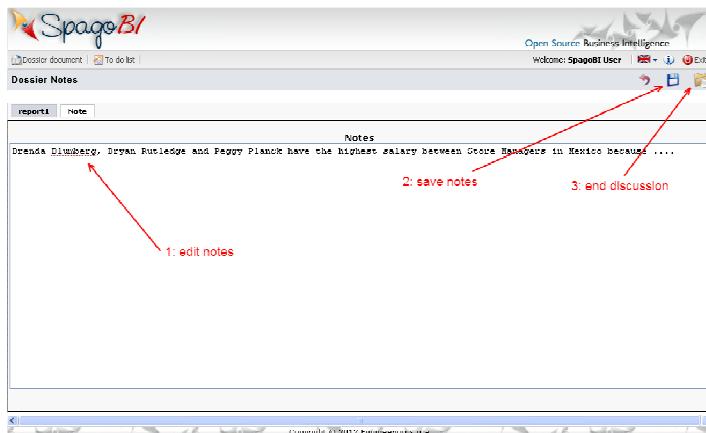


FIGURE 6.160 - Add notes editor for the dossier document

All saved presentations will be shown in the list of generated versions (see FIGURE 6.158). However, only approved presentations can be made visible to other users. To do so, click on in the row of an approved document: the fucntionalities tree will open and allow you to select the location where you wish to save the document.



FIGURE 6.161 - Approve the execution of a dossier document

7. Operational engines

ETL

SpagoBI allows the upload of data from source systems according to a common ETL logic, as well as the monitoring of data flows continuously feeding the data warehouse.

To this end, SpagoBI provides an ETL engine: SpagoBITalendEngine.

SpagoBITalendEngine

SpagoBITalendEngine integrates the open source tool Talent Open Studio (TOS). Talend Open Studio (TOS) is a graphical designer for defining ETL flows. Each designed flow produces a self-alone Java or Perl package. TOS is based on Eclipse and offers a complete environment including test, debug and support for documentation.

The integration between Talend and SpagoBI is twofold. TOS includes SpagoBI as a possible execution target for its job, while SpagoBI implements the standard context interface for communicating with Talend. Jobs can be directly executed from SpagoBI web interface or possibly scheduled.

Furthermore, the analytical model for monitoring ETL flows can be successfully applied to the analysis of audit and monitoring data generated by a

running job. Note that this is not a standard functionality of SpagoBI suite, but it can be easily realized within a project with SpagoBI.

To create an ETL document, you should perform the following steps:

- Job design (on Talend)
- Job deploy
- Template building
- Analytical document building
- Job execution.

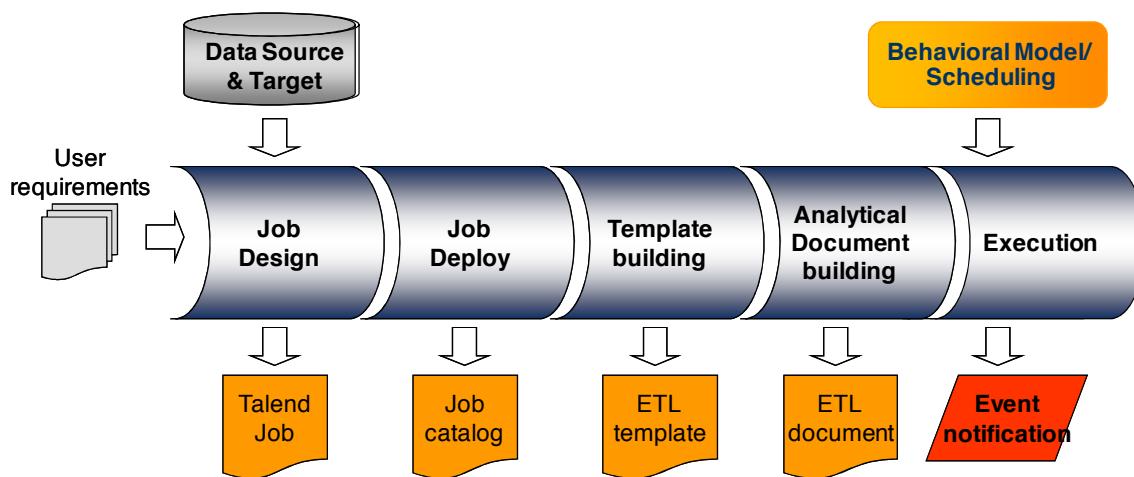


FIGURE 7.1 - Creation process of an ETL analytical document

In the remainder of the section, we discuss in detail all steps by providing examples.

Job design

The job is designed directly using Talend.

Designing an ETL job requires to select the proper components from Talend tool palette and connect them to obtain the logic of the ETL flow. Talend will map to appropriate metadata both the structure of any RDBMS and the structure of any possible flow (e.g., txt, xls, xml) acting as input or output in the designed ETL.

To design the ETL, several tools are available: from interaction with most RDBMS engines (both proprietary and open source) to support for different file formats; from logging and auditing features to support for several communication and transport protocols (ftp, POP, code, mail); from ETL process control management to data transformation functionalities.

Talend also supports data quality management. Furthermore, it enables the execution of external processes and can interact with other applications, e.g., open source CRM applications, OLAP and reporting tools.



Talend Open Studio

Talend Open Studio provides a wide set of functionalities and tools for ETL design and management. For full documentation and white papers please refer to <http://www.talend.com>.

FIGURE 7.2 shows the design of an ETL process using the tMap tool. The tMap tool allows the association of sources to targets according to defined rules. The main input is the source table in the data warehouse, while secondary (or lookup) inputs are dimensions to be linked to data. The output (target) is the data structure used for aggregation.

It is also possible to design parametric ETL jobs. We will see how to manage them along the next steps.

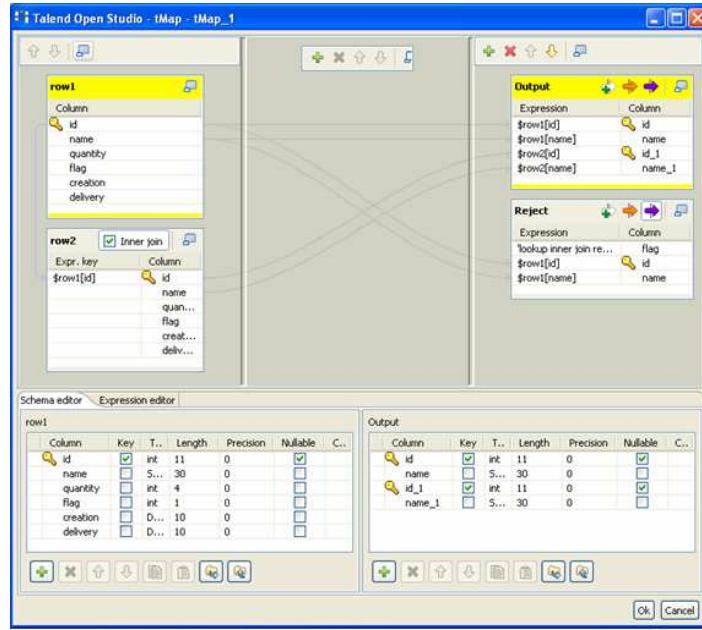


FIGURE 7.2 - Talend tMap operator

The result of the design phase is the job shown in FIGURE 7.3.

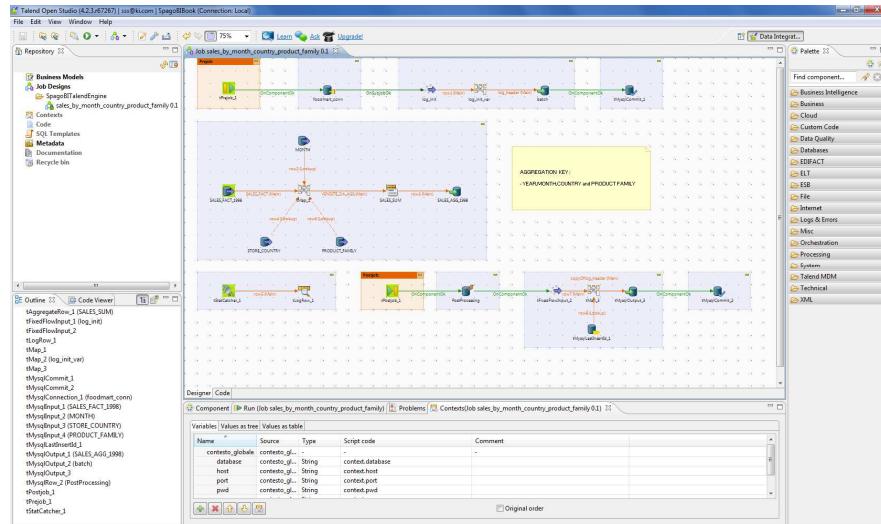


FIGURE 7.3 - Talend ETL job sample

Job deploy

Once you have designed the ETL job, you should deploy it on SpagoBI Server.

First of all, configure connections properties to SpagoBI Server. Select **Preferences > Talend > Import/export** from within Talend. Then set connection options as follows:

Parameter	Value
Engine name	SpagoBITalendEngine
Short description	Logical name that will be used by Talend
Host	Host name or IP address of the connection URL to SpagoBI
Port	Port of the connection URL to SpagoBI
Login	SpagoBI user that will perform job deploy
Password	Password of the user that will perform the deploy

FIGURE 7.4 – Setting connections through parameters and values

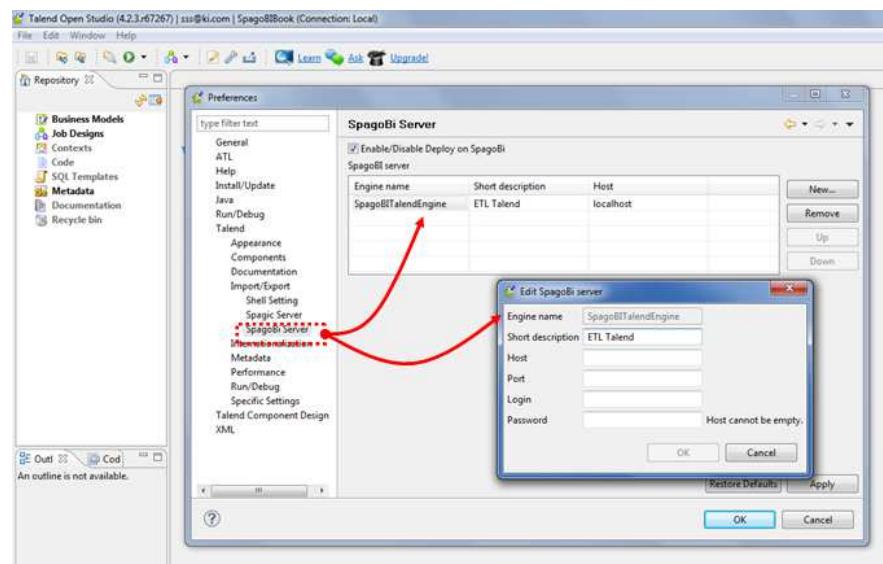


FIGURE 7.5 - Configuring connection to SpagoBI in Talend Open Studio

Once you have set the connection, you can right click on a job and select **Deploy on SpagoBI**, as shown in FIGURE 7.6. This will produce the Java code implementing the ETL and make a copy of the corresponding jar file at

\resources\talend\RuntimeRepository\java\Talend project name of SpagoBI Server.

It is possible to deploy multiple jobs at the same time. Exported code is consistent and self-standing. It may include libraries referenced by ETL operators and default values of job parameters, for each context defined in the job.

On its side, SpagoBI manages Talend jobs from an internal repository at /resources/talend/RuntimeRepository, under the installation root folder.

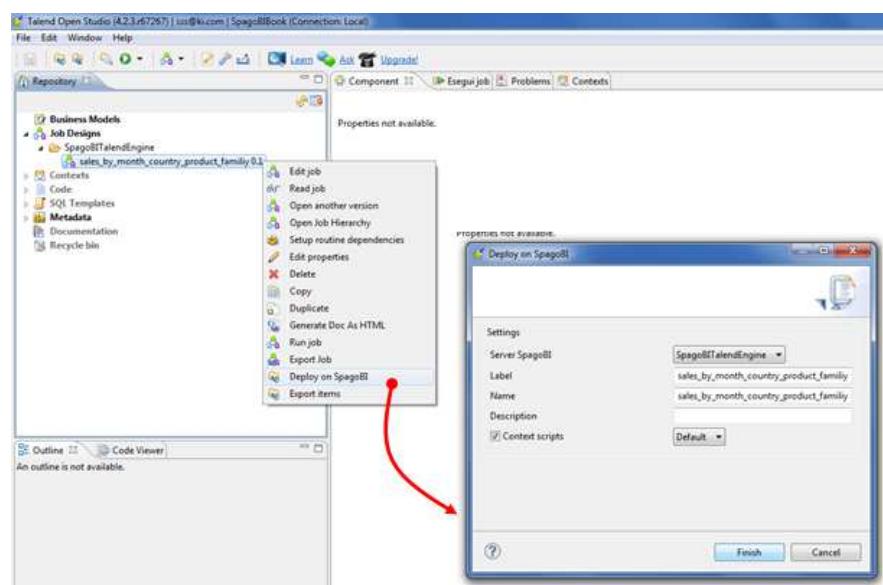


FIGURE 7.6 - Deploying a Talend ETL job on SpagoBI

Template building

As with any other SpagoBI document, you need to define a template for the ETL document that you wish to create.

The ETL template has a very simple structure, as shown in the example below:

```
<etl>
<job project="Foodmart"
      jobName="sales_by_month_country_product_familiy"
      context="Default"
      version="0.1"
```

```
language="java"/>  
</etl>
```

Where:

- The tag `job` includes all the following configuration attributes
- `project` is the name of the Talend project
- `jobName` is the label assigned to the job in Talend's repository.
- `context` is the name of the context grouping all job parameters. Typically it is the standard context, denoted with the name `Default`.
- `version` is the job version
- `language` is the chosen language for code generation. The two possible options are: Java and Perl.

Values in the template must be consistent with those defined in Talend, in order to ensure the proper execution of the ETL document on SpagoBI Server.

Creating the analytical document

Once we have created the template, we can create a new analytical document.

Before starting to create the document, it is recommended to check whether the engine is properly installed and configured. In case the engine is not visible in the Engine Configuration list (**Resources > Engine Management**), you should check that the web application is active by invoking the following URL:

<http://myhost:myport/SpagoBITalendEngine>



Engine management

Please refer to section Cross Services, Engine management, at chapter 5 – SpagoBI Server, to check configuration options for SpagoBI Talend Engine

Now you can create the analytical document on the Server, following the standard procedure. The template for this document is the one we have just created.



Create analytical documents on SpagoBI Server

The process to create and register a document on the Server is described in detail at section Analytical Model, chapter 5 - SpagoBI Server.

If the job has parameters, they should be associated to the corresponding analytical drivers, as usually. In other words, you have to create an analytical driver for each context variable defined in the Talend job.

Job execution

A Talend job can be executed directly from the web interface of SpagoBI Server, and of course from a Talend client.

To execute the job on SpagoBI, click on the document icon in the document browser, like with any other analytical document. The execution page will show a message to inform that the process was started.

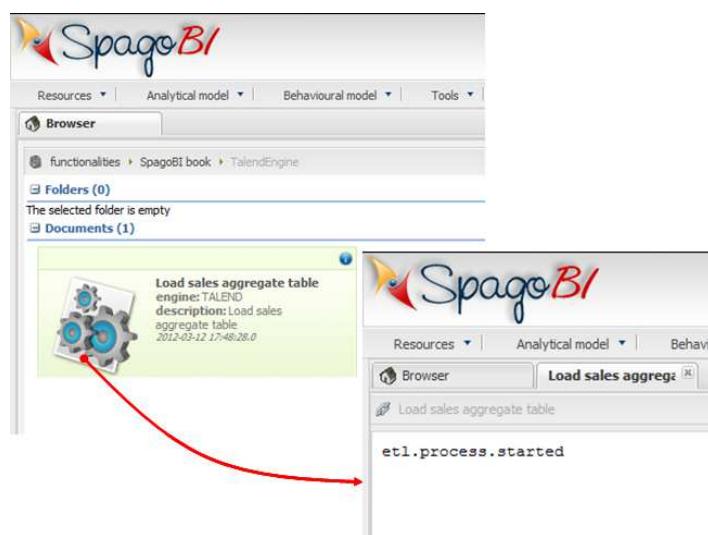


FIGURE 7.7 - Starting an ETL process from SpagoBI document browser

The engine generates an event when the process starts and when it stops. This allows the user to track job execution. To see event details, select **User Menu > Events** in the main window.

The screenshot shows two pages from the SpagoBI interface. The top page is titled 'Events list' and displays a table with two rows of event data. The columns are 'Event id', 'Event date', 'User', and 'Description'. The first row shows '1' as the event ID, '12/03/2012' as the date, 'bsadmin' as the user, and 'Started execution of ETL process: Execution Analy...' as the description. The second row shows '2' as the event ID, '12/03/2012' as the date, 'bsadmin' as the user, and 'Execution of ETL process successfully terminated...' as the description. A red arrow points from the bottom of the first page to the top of the second page. The bottom page is titled 'Event details' and shows a detailed view of the first event. It includes sections for 'Identifier', 'Date', 'User', 'Document', 'Description', and 'Related documents'. The 'Description' section contains a large amount of technical log output. Another red arrow points from the bottom of the second page back up to the top of the first page.

FIGURE 7.8 - Talend ETL event management

Job scheduling

Most often it is useful to schedule the execution of ETL jobs instead of directly running them. You can rely on SpagoBI scheduling functionality to plan the execution of Talend jobs.



SpagoBI Scheduler

SpagoBI Server provides a set of administrative tools, including a Scheduler. To learn how to make the best use of this tool, please refer to section Administrative Tools, at chapter 5 - SpagoBI Server.

Select **Tools > Scheduler**. Then perform the following steps:

- Create a new activity by clicking on . Choose a name and a description. Select the jobs you want to schedule.

- Assign values to job parameters. Leave default settings that do not correspond to parameters used in the job. Save.
- Once back to the main window, click on the symbol of the row containing the job you just created. Create a new scheduled execution and save (see FIGURE 7.9).

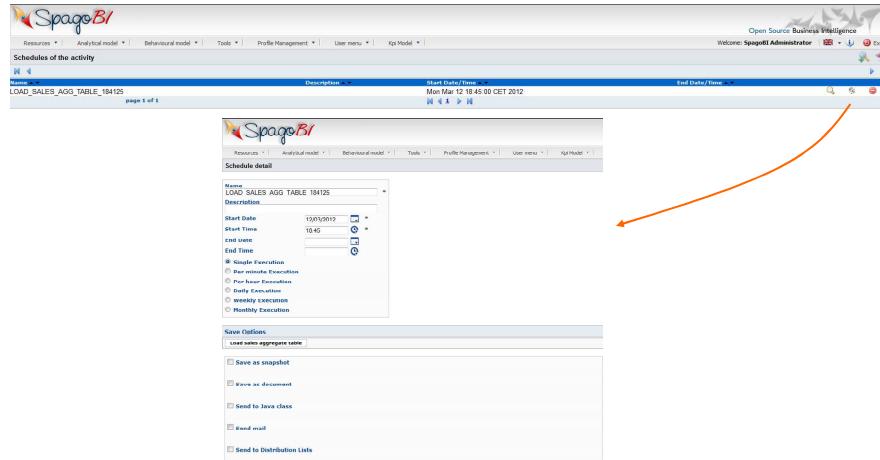


FIGURE 7.9 - Scheduling job execution

While defining a scheduled execution, you can set a notification option which will send an email to a set of recipients or a mailing list once the job has completed its execution.

To enable this option, check the flag **Send Mail** (as shown in FIGURE 7.10). Configure a set of parameters in the configuration panel reachable at Tools > Manage Configuration. Select the category Mail and set parameters as follows:

Parameter	Value
MAIL.PROFILES.scheduler.smtphost	Host name or address of the SMTP server
MAIL.PROFILES.scheduler.smtpport	Port number of the SMTP server
MAIL.PROFILES.scheduler.from	Username of the email recipient
MAIL.PROFILES.scheduler.password	Password of the email recipient

TABLE 7.1 - Parameters to be used in the configuration panel

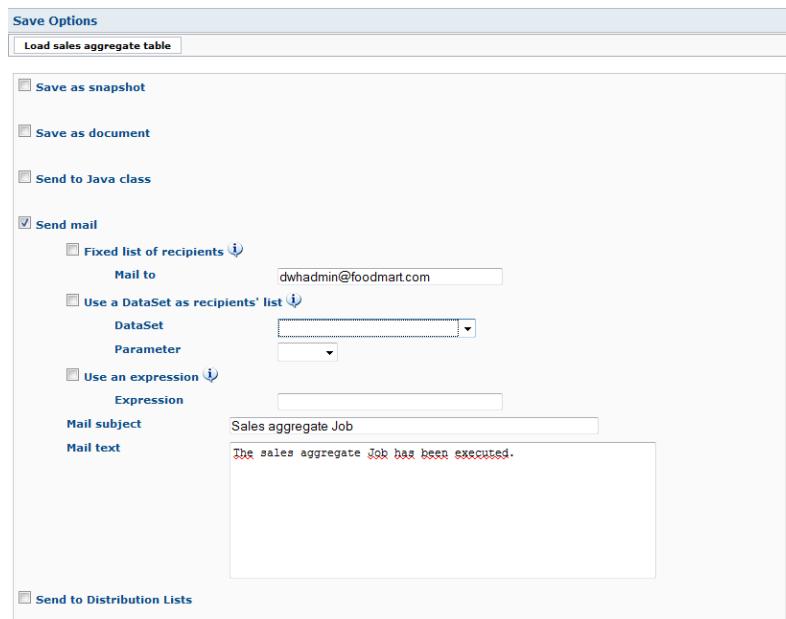


FIGURE 7.10 - Save options in job scheduling

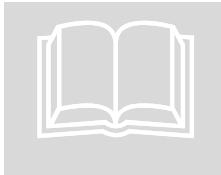
External processes

SpagoBI supports the execution of processes that are external to its own activity. When analyzing data, for example through the real time console, it may be useful to perform activities such as sending notification emails or taking actions on the components of the monitored system (e.g., business processes, network nodes).

SpagoBI provides the SpagoBIProcessEngine, which supports the management of external processes.

SpagoBIProcessEngine

The Process engine allows the execution and management of external processes. With the term “process” we refer to a Java instruction, however complex it may be. Processes can be executed in background or via the interface of the Console Engine. It is also possible to schedule their start and stop.



CommonJ

SpagoBI Process Engine exploits the functionalities of the CommonJ library. Read more at <http://commonj.myfoo.de/>.

To enable the management of an external process, the following steps are required:

- Create a Java class defining the execution logic
- If needed, create a Java class defining the logic of the process, i.e., which tasks the process is supposed to perform (optional).
- Create a template that will be associated to the SpagoBI document
- Create the SpagoBI CommonJ analytical document.

In the following sections, we provide details about both class and template creation, and document creation.

Class definition

First of all, the developer should write a Java class that defines the desired logics for processing start and stop. In particular, this class must extend one of these two classes of the engine:

- **SpagoBIWork.** In this case the class to be defined only needs to reimplement the `run()` method. This class is the base case: the logic of the external process will be contained in the `run()` method.
- **CmdExecWork.** In this case, the class to be defined must implement the method `execCommand()`. The logic of the external process can be delegated to an external class, which will be invoked by the `execCommand()` method, as specified in the document template (see below).

To stop the process, the developer is in charge of checking programmatically whether the process is still running, using the method `isRunning()`, or not.

Note that the class CmdExecWork extends SpagoBIWork by providing additional methods.

To better understand the difference between the two options, let us have a look at some code snippets.

Here you can see a class implemented as an extension of SpagoBIWork:

```
package it.eng.spagobi.job;

import java.util.Iterator;
import it.eng.spagobi.engines.commonj.process.SpagoBIWork;

public class CommandJob extends SpagoBIWork{
    @Override
    public boolean isDaemon() {
        return true;
    }
    @Override
    public void release() {
        System.out.println("Release!!!");
        super.release();
    }
    @Override
    public void run() {
        super.run();
        System.out.println("Job started! ");

        java.util.Map parameters=getSbiParameters();
        for (Iterator iterator =
parameters.keySet().iterator(); iterator.hasNext();) {
            String type = (String) iterator.next();
            Object o=parameters.get(type);
            System.out.println("Parameter "+type+ " value
"+o.toString());
        }

        for(int i=0;i<50 && isRunning();i++){
            System.out.println("job is running!");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Job finished!");
    }
}
```

Note that we only implement the `run()` method, embedding the logic of the process in it.

Below you can see an example extension of `CmqExecWork`, called `CommandJob`:

```
package it.eng.spagobi.job;

import it.eng.spagobi.engines.commonj.process.CmdExecWork;
import java.io.IOException;

public class CommandJob extends CmdExecWork{
    public boolean isDaemon() {
        return true;
    }
    public void release() {
        super.release();
    }
    public void run() {
        super.run();
        if(isRunning()){
            try {
                execCommand();
            } catch (InterruptedException e) {
            } catch (IOException e) {
            }
        }
    }
}
```

Note that this class implements the `execCommand()` method and uses the `isRunning()` method. No logic is directly embedded in this class.

Therefore, we also define an external class, called `ProcessTest`, which contains the actual logic (in our example printing the content of a file):

```
package it.eng.test;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;

public class ProcessTest {

    public static void main(String[] args) {
```

```
FileOutputStream file=null;
try {
    file = new FileOutputStream("C:/file.txt");
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
PrintStream output = new PrintStream(file);

while (true){
    output.println("New row");
    output.flush();
    try {
        Thread.currentThread().sleep(5000l);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        output.close();
    }
}
```

Now that classes are ready, we pack them in a .jar file containing all classes and their paths. Then we copy the jar file under the resource folder of SpagoBI at: [RESOURCE_PATH]/commonj/CommonjRepository/[JAR_NAME].

In the next section we will explain how to define the template, based on the class definition chosen above.

Template definition

As with any other SpagoBI document, we need to define a template for an external process document.

The example below shows a template that corresponds to the classes `CommandJob` and `ProcessTest` defined in the examples above. Let us note that this template corresponds to the option of implementing an extension of `CmdExecWork`.

```
<COMMONJ>
    <WORK workName='JobTest'
className='it.eng.spago.job.CommandJob'>
        <PARAMETERS>
```

```

<PARAMETER name='cmd'
value='C:/Programmi/Java/jdk1.5.0_16/bin/java' />
    <PARAMETER name='classpath'
value='C:/resources/commonj/CommonjRepository/JobTest/process.jar
' />
        <PARAMETER name='cmd_par'
value='it.eng.test.ProcessTest' />
            <PARAMETER name='sbi_analytical_driver' value='update' />
            <PARAMETER name='sbi_analytical_driver' value='level' />
        </PARAMETERS>
    </WORK>
</COMMONJ>
```

Where:

- <COMMONJ> is the main tag and includes all the document.
- The tag <WORK> specifies the process. In particular:
 - `workName` is the id of the process
 - `className` contains the name of the class implementing the process (as defined above).
- The tag <PARAMETERS> contains all parameters. Each <PARAMETER> tag includes a parameter. Some of them are mandatory.

CommonJ document template parameters	
cmd	Specifies the java command that will be launched, with its complete path
classpath	Specifies the classpath containing the jar file. This path will be added to the classpath for the process to run correctly.
cmd_par	Optional. In case it is defined, its value contains the Java class that will be launched instead of the job (i.e., the extension of CmdWorkExec or SpagoBIWork).
sbi_analytical_driver	Optional and repeatable. Each line with this attribute defines an analytical driver that should be associated with the process.

TABLE 7.3 – CommonJ document template parameters

The class CmdExecWork (and its extensions) allows the execution of the command specified in the template. In particular, the template above would produce the following command at runtime:

```
C:/Programmi/Java/jdk1.5.0_16/bin/java 'it.eng.test.ProcessTest'  
update=<val> level=<val>
```

Creation of SpagoBI document

The last step is the creation of the actual analytical document.

First, you should check that the engine is properly installed and configured.



Engine configuration

Engines configuration parameters are summarized at section Engines Management, in chapter 5 - SpagoBI Server.

Then, create a document of type External process, following the standard procedure.



Create analytical documents on SpagoBI Server

The process to create and register a document on the server is described in detail at section Analytical Model, chapter 5 - SpagoBI Server.

Manually starting jobs

SpagoBIProcessEngine provides a graphical interface to start and stop jobs, as shown in **Errore. L'origine riferimento non è stata trovata..**

Another option is to start and stop processes via the real time console, which provides support for these actions (see).



Real time console

SpagoBI supports real time monitoring with the Dashboard Engine and the Console Engine. Find details about the RT console at the corresponding section, in this chapter.

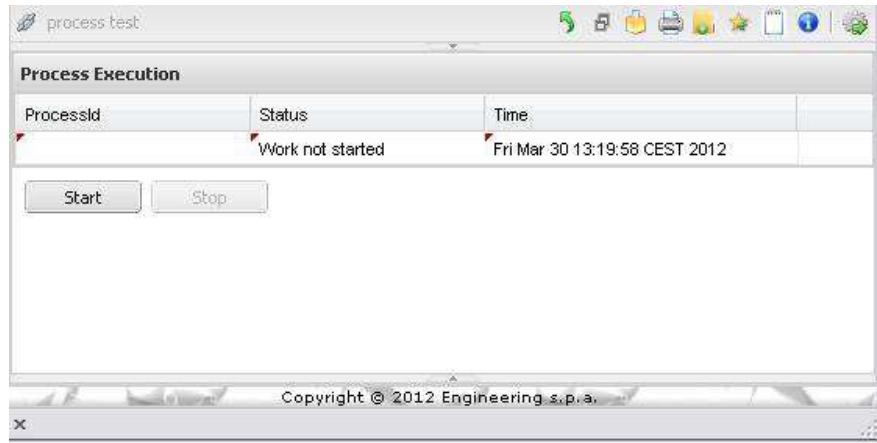


FIGURE 7.11 - ProcessEngine GUI for starting/stopping processes

A screenshot of a real-time monitoring interface. The top navigation bar includes "Detail customers", "Cumulative sales", "Brands...", "Products...", and a search icon. The main area is a grid table with columns: Brand name, Product name, Store Sales, Store Costs, Unit Sales, Alarm on Sales, and Alarm on Costs. The rows list products from brands like ADJ, Akron, Amigo, and Applause. To the right of the grid is a vertical column of process control icons, including arrows for start/stop, a magnifying glass, and other symbols. One row of these icons is highlighted with a red rectangle.

FIGURE 7.11 - Starting process via the real time console

In case the process requires parameters, they can be assigned values through a specific window, as shown in FIGURE 7.11.

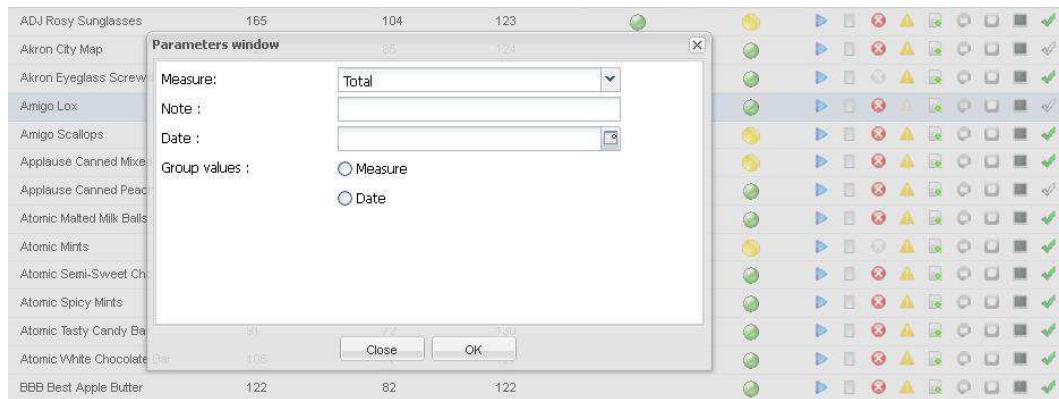


FIGURE 7.12 - Assigning values to process parameters

Master data management

SpagoBIRegistryEngine

SpagoBIRegistryEngine is a support tool for mastering data management. It allows the visualization and editing of data relevant for the business of an organization, which are typically not transactional.

Relevant use cases include:

- Management of simple dimensional registries
- Quality management and control of data imported via an ETL process
- Management of lookup and/or decoding tables.



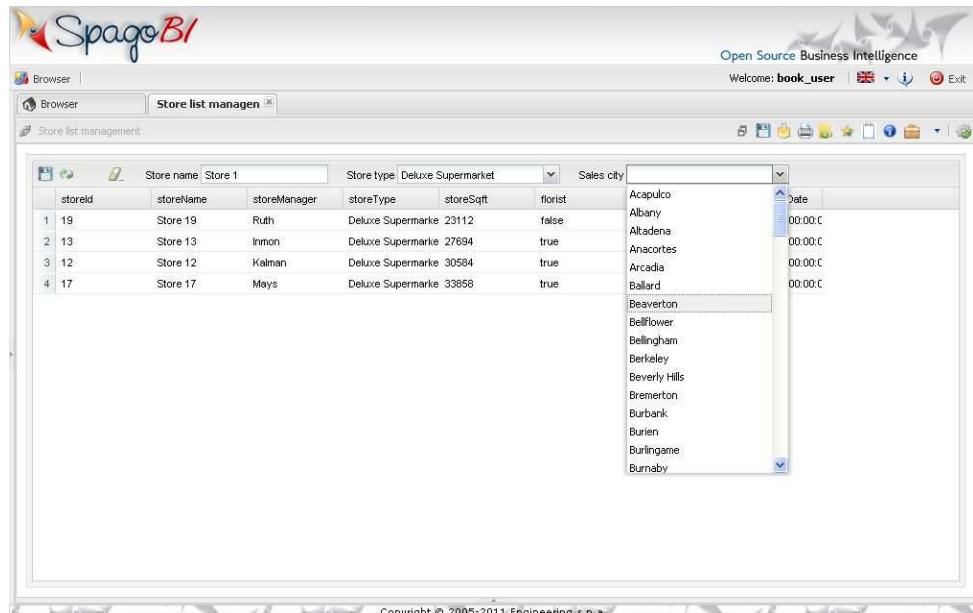
Data management

SpagoBI current version (3.5) allows users to modify data, but not to insert or delete them. These features will be added in the future.

FIGURE 7.12 shows a registry document. The document has a table structure, where columns represent the properties of the selected entity (or one of its subentities, as explained in the next section). In our example, we can see a list of stores and some of their properties.

On top, a set of filters is shown. Using them, the user can filter entities of interest. Filters can be either editable with free text or multiple selection, with a combo box showing all possible values (see FIGURE 7.12). To remove all filters, click on the  icon at the top left corner. To refresh data, click on .

Double clicking on a cell will make its content editable (provided that the field is editable). As with filters, some properties can be freely edited, while others can be set by choosing among predefined values in a combo box.



The screenshot shows the SpagoBI Registry document interface. At the top, there's a toolbar with various icons. Below the toolbar is a header bar with the SpagoBI logo, the text "Open Source Business Intelligence", and a "Welcome: book_user" message. The main area contains a table titled "Store list management". The table has columns: storeId, storeName, storeManager, storeType, storeSoft, florist, Sales city, and date. There are four rows of data. A dropdown menu is open over the "date" column of the fourth row, listing various city names: Acapulco, Albany, Altadena, Anacortes, Arcadia, Ballard, Beaverton, Bellflower, Bellingham, Berkeley, Beverly Hills, Bremerton, Burbank, Burien, Burlingame, and Burnaby. The bottom of the screen shows a copyright notice: "Copyright © 2005-2011 Engineering s.p.a."

FIGURE 7.12 - SpagoBI Registry document

Once you have edited some cells, you can save changes clicking on the save icon at the top left corner (in the toolbar).

storeId	storeName	storeManager	storeType	storeSqt	florist	salesCity	firstOpenedDate
1	HQ		HeadQuarter	22476	false	None	
2	Store 14	Strehlo	Deluxe Superma...	36509	true	San Francisco	1957-11-24 00:00:00
3	Store 9	Stuber	Gourmet Superma...	23598	false	Mexico City	1955-03-18 00:00:00
4	Store 7	White	HeadQuarter	24597	true	Los Angeles	1971-05-21 00:00:00
5	Store 5	Green	Mid-Size Grocery	23112	false	Guadalajara	1978-09-18 00:00:00
6	Store 19	Ruth	Supermarket	34452	true	Vancouver	1977-03-27 00:00:00
7	Store 20	Cobb	Mid-Size Grocery	21215	false	Victoria	1980-02-06 00:00:00
8	Store 24	Byrd	Supermarket	20319	true	San Diego	1979-05-22 00:00:00
9	Store 15	Ollom	Supermarket	34791	false	Seattle	1969-07-24 00:00:00
10	Store 11	Erickson	Supermarket	30584	false	Portland	1976-09-17 00:00:00
11	Store 13	Inmon	Deluxe Supermarke	38382	true	Salem	1957-04-13 00:00:00
12	Store 10	Merz	Supermarket	23759	false	Orizaba	1979-04-13 00:00:00
13	Store 12	Kelman	Deluxe Supermarke	23593	true	Hidalgo	1968-03-25 00:00:00
14	Store 18	Brown	Mid-Size Grocery	30797	false	Merida	1969-06-28 00:00:00
15	Store 8	Williams	Deluxe Supermarke	22688	true	Camacho	1994-09-27 00:00:00
16	Store 4	Johnson	Gourmet Superma...	23598	false	Acapulco	1982-01-09 00:00:00
17	Store 1	Jones	Supermarket	23598	false	Revere Hills	1981-01-03 00:00:00
18	Store 6	Maria	Gourmet Superma...	23598	true		

FIGURE 7.14 - Editing data

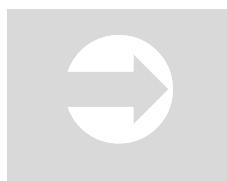
Document creation

SpagoBIRegistryEngine is implemented as an extension of SpagoBIQbEEngine. Thus, a registry document is a kind of QbE document and should be created accordingly.

The creation of a Registry document involves the following steps:

- Datamart generation
- Template creation
- Analytical document creation.

Note that, with respect to a “pure” QbE document, the final step, i.e., query creation via QbE interface, is missing.



Datamart generation

Datamart generation is similar to the case of a QbE document. Please refer to section SpagoBIQbEEngine, at chapter 6 - Analytical Engines.

As far as analytical document creation is concerned, the Registry is just like all other SpagoBI documents.



Create analytical documents on SpagoBI Server

The process to create and register a document on the Server is described in detail at section Analytical Model, chapter 5 – SpagoBI Server.

We focus hereinafter on step 2 and the specific format of the template.

Template creation

The template to be used to develop a Registry document is shown in the following example:

```
<?xml version="1.0" encoding="windows-1250"?>
<QBE>
    <DATAMART name="registry" />
    <REGISTRY>
        <ENTITY name="it.eng.spagobi.meta.Store">
            <FILTERS>
                <FILTER title="Store name" field="storeName"
presentation="MANUAL" />
                <FILTER title="Store type" field="storeType"
presentation="COMBO" />
                <FILTER title="Sales city" field="salesCity"
presentation="COMBO" />
            </FILTERS>
            <COLUMNS maxSize="200">
                <COLUMN field="storeId" editable="false" size="30"/>
                <COLUMN field="storeName" editor="MANUAL"
size="100" sorter="asc"/>
                <COLUMN field="storeManager" editor="MANUAL" />
                <COLUMN field="storeType" editor="COMBO" />
                <COLUMN field="storeSqft" editor="MANUAL" />
                <COLUMN field="florist" editor="COMBO" />
                <COLUMN field="salesCity" subEntity="regionId"
foreignKey="regionid" />
```

```

<COLUMN field="firstOpenedDate" editor="MANUAL" />
</COLUMNS>
</ENTITY>
</REGISTRY>
</QBE>

```

Where:

- The **datamart** tag references the datamart that provides data. The datamart should already have been created.
- The **entity** tag defines the entity that will be managed by the engine in this document. The name must correspond to the Java entity defined in the referenced datamart. This entity is required to contain one key only.
- The **filters** tag includes the list of filters on top of each list. For each filter the mandatory attributes are:

Filter attributes		
title		Title of the filter
field		The property the filter will be applied to
presentation	manual	Free text input
	combo	Combo box with possible values

TABLE 7.4 – Filter attributes

- The **columns** tag includes the attributes of the columns included in the registry table. The **maxsize** attribute defines the width in pixel of each column. For each column you should define:

Column attributes	
field	Name of the property that will be shown
editable	If set to true, the field can be edited by the user. Note that the key property for each entity must be declared as non editable.
editor	If editable, can be set to MANUAL (free text input) or COMBO (combo box with possible values).
size	Width (in pixel)
sorter	If assigned a value, values will be ordered. Set ASC

	for ascending order, DESC for descending order.
unsigned	Only for numeric values. If true, values will be shown with no sign.
subentity	If the property of a sub-entity is visible, specify the sub-entity name (together with the foreign key, see below)
foreign key	If the property of a sub-entity is visible, specify the foreign key here (together with the sub-entity, see above). Starting SpagoBI 3.3, this must be the natural key of the entity

TABLE 7.5 – Column attributes

8. SpagoBI SDK

SpagoBI offers a complete suite, covering the whole BI life cycle and providing tools for developers, administrators and end-users, with a customizable web application to build the customer delivery environment. That is the standard and complete usage of a Business Intelligence product, but many times the same solution needs to provide documents, services or information on other products the company is using.

Goals and targets

SpagoBI SDK is the module dedicated to the integration between SpagoBI and external applications, covering many integration scenarios.

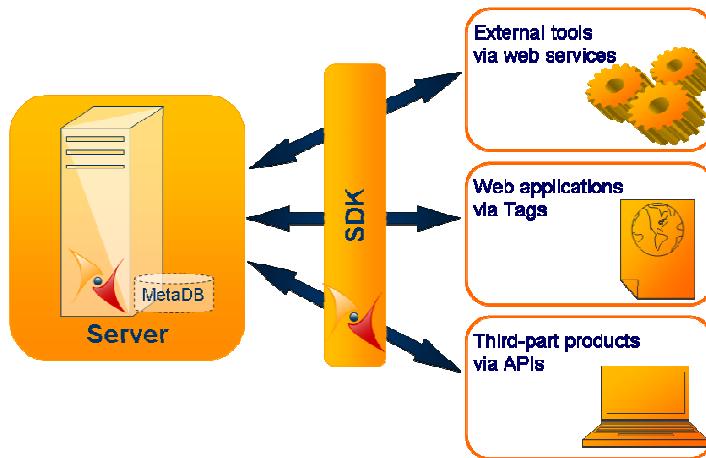


FIGURE 8.1 - SpagoBI SDK towards external applications

In fact, SpagoBI SDK provides:

- **Web services**, to interact with static documents, data sets, data sources, engines and maps without a direct action on the front-end
- **Tag libraries**, to directly include a SpagoBI document on a page (jsp) of an external application, having all the document features and interactivity available in the new context
- **Javascript APIs**, for a low-level integration with an external front-end, as instance to customize its menus, so as to add analytical features within the standard third-party product functionalities.

So, obviously SpagoBI SDK is not an end-user tool, but it is usable by:

- Java developers, who use the provided services to interact with a SpagoBI Server from their application
- Javascript developers, who use tag libraries and Javascript APIs to customize the front-end of their web application, so as to mix its standard functionalities with some analytical features provided by SpagoBI
- Administrators, who maintain this kind of integrations.

Installation and configuration

The first step to use the SpagoBI SDK module is to check that your environment satisfies the following requirements:

- a JDK 1.6.x already installed
- the same version of a SpagoBI Server reachable from your environment
- the application that will use the SDK must satisfy its technological requirements (java, javascript and jsp).

If the requirements are met, the second step is the download of the right package from the OW2 forge at:

- <http://forge.ow2.org/projects/spagobi> to look at SpagoBI latest releases

- http://forge.ow2.org/project/showfiles.php?group_id=204 to choose the latest release among all SpagoBI releases.

Now choose the right package according to the specific release and distribution matching your environment.

SpagoBI 2.5				2010-04-09
SpagoBIStudio_2.5_linux_20100412.zip	239,316.8	Any	.zip	
SpagoBIStudio_2.5_win_20100412.zip	210,511.2	Any	.zip	
SpagoBI 2.4				2009-12-18
SpagoBISTudio-bin-2.4.0_12182009.zip	300,365.4	Any	.zip	
SpagoBISTudio-src-2.4.0_12182009.zip	25,535.9	Any	Source .zip	

e. SpagoBI SDK				
SpagoBI 3.3				2011-12-22
SpagoBISDK-bin-3.3.0_12202211.zip	5,141.2	Any	.zip	
SpagoBI 3.2				2011-11-02
SpagoBISDK-bin-3.2.0_11042011.zip	5,140.1	Any	.zip	
SpagoBI 3.1				2011-07-21
SpagoBISDK-bin-3.1.0_07212011.zip	5,075.1	Any	.zip	
SpagoBI 3.0				2011-06-15
SpagoBISDK-bin-3.0.0_06152011.zip	5,068.4	Any	.zip	
SpagoBI 2.8				2011-02-24
SpagoBISDK-bin-2.8.0_02224011.zip	5,057.0	Any	.zip	
SpagoBI 2.7				2010-11-25
SpagoBISDK-bin-2.7.0_11262010.zip	5,059.7	Any	.zip	
SpagoBI 2.6				2010-07-07
SpagoBISDK-bin-2.6.0_07072010.zip	5,337.3	Any	.zip	
SpagoBI 2.5				2010-04-08
SpagoBISDK-bin-2.5.0_04092010.zip	5,315.0	Any	.zip	
SpagoBI 2.4				2009-12-17
SpagoBISDK-bin-2.4.0_12182009.zip	5,290.9	Any	.zip	
SpagoBISDK-src-2.4.0_12182009.zip	144.6	Any	Source .zip	

FIGURE 8.2 – Download SpagoBI SDK package from OW2 forge



SpagoBI SDK version

Using the same version of both SpagoBI Server and SpagoBI SDK is always recommended, even if not mandatory. This ensures that both modules maintain the same SDK release.

The downloaded SpagoBISDK-bin-<version>-<date>.zip contains a war named SpagoBISDK.war, structured as follows:

- / : the root contains some jsp as SDK usage examples
- /html : it contains an html test page for javascript APIs. The page must be modified with the environment settings related to the referenced

SpagoBI Server (host, port, user, password, documentLabel, executionRole).

- **/js** : it contains the SDK javascript APIs and their dependent files
- **/WEB-INF/lib** : the folder contains all the compiled libraries for the usage of SDK web services
- **/WEB-INF/tags/spagobi** : the folder contains the SDK tag library.

To use the SDK from an the external application, you further need to:

- use SDK web services: copy the /WEB-INF/lib contents under a folder visible from the classpath of the application
- use SDK tag library: copy the /WEB-INF/tags/spagobi/execution.tag file into the application context
- use SDK javascript APIs: copy the /js contents under the /js subfolder of the application context.

After copying the whole SpagoBISDK.war as a webapp on the referenced SpagoBI Sever and restarting it, you can test web services and APIs.

Type <http://<IP server address>:<port number>/SpagoBISDK> : a test page will appear, followed by a login page for the web services authentication.

Welcome to SpagoBI SDK demo

Examples:

- [web-services / tag library example](#)
- [javascript API example](#)

Welcome to SpagoBI SDK - Web Services / TAG library demo

Login with biuser/biuser

Name:

Password:

FIGURE 8.3 - Access the SDK test pages

Once logged in, you can choose which test you wish to perform. Clicking on web-services/tag library examples, you will be shown a test page like the one in FIGURE 8.4. Here you can pick a document: if it is executed correctly, the test will be successful.

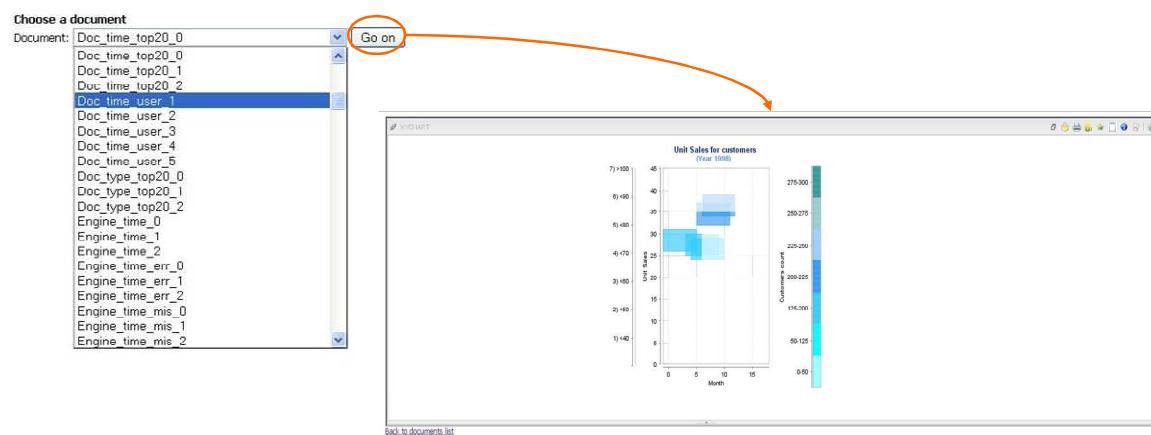


FIGURE 8.4 - Test page for web services

Alternatively, you can click on javascript API example. Here you will be shown some API use cases. For example, FIGURE 8.5 shows the function code of the `getDocumentHtml` API and the result of the execution of the document called `RPT_WAREHOUSE_PROF` (which must be registered on the Server).

The label of the document that will be executed in the lower part of the window is passed to the function as a parameter. Therefore, to change document, you should modify the JSP containing the code for invoking the API.

These JSP are stored in the main folder of the SpagoBISDK application, with names like `example1JSP`, `example2JSP`, and so on. By modifying these files, you can not only change the name of the document, but also the execution role.

Example 2 : getDocumentHtml

Description: Use `getDocumentHtml` function to get an html string that contains the definition of an iframe pointing to the execution service. The `src` property of the iframe is internally populated using `getDocumentUrl` function.

Code:

```
example2Function = function() {
    var html = Sbi.sdk.api.getDocumentHtml({
        documentLabel: 'RPT_WAREHOUSE_PROF'
        , executionRole: '/spagobi/user'
        , parameters: {warehouse_id: 19}
        , displayToolbar: false
        , displaySliders: false
        , iframe: {
            height: '500px'
            , width: '100%'
            , style: 'border: 0px;'
        }
    });
    document.getElementById('targetDiv').innerHTML = html;
};
```

XYCHART

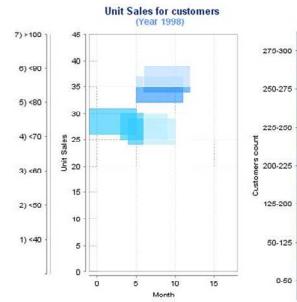


FIGURE 8.5 - Java API example use case

Below an example of the JSP file that can be modified:

```
execTest1 = function() {
    var url = Sbi.sdk.api.getDocumentUrl({
        documentLabel: 'RPT_WAREHOUSE_PROF'
// etichetta documento
        , executionRole: '/spagobi/user'
// ruolo
        , parameters: {warehouse_id: 19}
        , displayToolbar: false
        , displaySliders: false
        , iframe: {
            style: 'border: 0px;'
        }
    });
    document.getElementById('execframe').src = url;
```

SDK Web Services

SpagoBI SDK provides web services to interact with the most relevant objects:

- **DocumentService**: to interact with static analytical documents, the ones that do not require end-user interaction (i.e. reports, charts, office)
- **DataSetsSDKService**: to interact with SpagoBI data sets
- **DataSourcesSDKService**: to manage data sources
- **EngineService**: to manage analytical and operational engines
- **MapsSDKService**: to interact with the Map catalogue.

Every web service must be authenticated, so username and password are always sent to the security connector, as in the following proxy:

```
DocumentsServiceProxy proxy =
    new DocumentsServiceProxy("user", "password");
proxy.setEndpoint
    ("http://localhost:8080/SpagoBI/sdk/DocumentsService");
```

The following paragraph describes all the web services available in SpagoBI SDK.

Proxy

The proxies are classes that are invoked to establish the connection with the Server and recall web services. The constructor takes the username and password as input parameters, leaving the user free access to not specify any credentials.

Therefore, it is necessary to define the endpoints (*setEndpoint* method (String)) from which you can invoke methods provided by the proxy. Proxies are:

- *DataSetsSDKServiceProxy*: for the management of datasets

- *DataSourcesSDKServiceProxy*: for the management of data sources
- *DocumentsServiceProxy*: for the management of documents
- *EnginesServiceProxy*: for the management of engines
- *MapsSDKServiceProxy*: for the management of maps
- *TestConnectionServiceProxy*: for the control of the connection via the method boolean *connect()*.

Here is an example of using the proxy *DocumentsServiceProxy*.

```
// proxy initialization with user credentials
String user = ....;
String password = ....;
DocumentsServiceProxy proxy = new DocumentsServiceProxy(user,
password);
proxy.setEndpoint("http://localhost:8080/SpagoBI/sdk/DocumentsService");
// all visible documents list
SDKDocument[] documents = proxy.getDocumentsAsList(null, null,
null);
```

TestConnectionService: connection test

To use these functions we need to use the class *TestConnectionServiceProxy*.

- **boolean Connect()**: Method that tests the connection and validation in the credentials.

DocumentService

DocumentService is the web service interface that allows to interact with SpagoBI documents. Its interface provides the following methods:

- **SDKDocument[] getDocumentsAsList(String, String, String folderPath);**

It allows you to get a list of documents visible to the user as a list, which can be filtered by type, status, path of functionality.

- **SDKFunctionality getDocumentsAsTree(String);**

It allows you to get a list of documents visible to the user as a tree.

- **String[] getCorrectRolesForExecution(Integer) throws NonExecutableDocumentException;**

It allows you to obtain the list of valid roles for the execution of a document, given the id of the document.

- **SDKDocumentParameter[] getDocumentParameters(Integer, String) throws NonExecutableDocumentException;**

It allows you to get the parameter list of a document.

- **HashMap getAdmissibleValues(Integer, String) throws NonExecutableDocumentException;**

It allows you to obtain the list of permissible values of a parameter for a document execution.

- **Integer saveNewDocument(SDKDocument, SDKTemplate, Integer) throws NotAllowedOperationException;**
- **SDKDocument getDocumentById(Integer);**

It allows you to return the document with id "Id".

- **SDKDocument getDocumentByLabel(String);**

It allows you to return the document with the specified label.

- **SDKExecutedDocumentContent executeDocument(SDKDocument, SDKDocumentParameter[], String, String) throws NonExecutableDocumentException, NotAllowedOperationException, InvalidParameterValue, MissingParameterValue;**

It allows you to execute the document. The result are returned in the specified format by OutputType. It is important to value all mandatory analytical drivers.

- **SDKTemplate downloadTemplate(Integer) throws NotAllowedOperationException**

It allows you to download the template of the document with id "DocumentID".

- **Void uploadTemplate(Integer, SDKTemplate) throws NotAllowedOperationException;**

It allows you to upload the template of a document.

- `void uploadDatamartTemplate(SDKTemplate, SDKTemplate, String);`

It allows you to upload a file containing a QBE template, upload the xml file containing the definition of the calculated fields and create a QbE document.

- `void uploadDatamartModel(SDKTemplate);`

It allows you to upload a file containing a QBE template.

- `SDKTemplate downloadDatamartFile(String, String);`

It allows you to download a file located at path_resources\qbe\datamarts.

In the following, all the used data structures are described, before a detailed description of each method of the service.

Data structures

SDKDocument

It represents a SpagoBI document. Its structure follows:

Integer	<i>Id</i> Document id
String	<i>Label</i> Document Label
Integer	<i>Name</i> Document Name
String	<i>Description</i> Document description
Integer	<i>DataSetId</i>
Integer	<i>DataSourceId</i>
Integer	<i>EngineId</i>
String	<i>State</i> (DEV, TEST, REL, SUSP)
String	<i>Type</i> Document Type

TABLE 8.1 – SDKDocument structure

SDKFunctionality

It represents a feature on the Server. Its structure follows:

String	Code Functionality code
SDKDocument[<i>ContainedDocuments</i> Documents contained by the functionality
SDKFunctionality	<i>ContainedFunctionalities</i> Subsidiary functionalities
String	<i>Description</i>
Integer	<i>Id</i>
String	<i>Name</i> Functionality name
Integer	<i>ParentId</i>
String	<i>Path</i> Textual representation of the directory in the functionality tree
Integer	<i>Prog</i> Progressive number of a functionality sibling apparition.

TABLE 8.2 - SDKFunctionality structure

SDKConstraint

It represents a constraint of an analytical driver. Its structure follows:

Integer	Id
String	<i>Label</i>
String	<i>Type</i>
String	<i>Description</i>
String	<i>firstValue</i> First value of the constraint
Object[]	<i>secondValue</i> Cannot be enhanced (or it can be the maximum of a range in which the first value is the minimum)
it.eng.spagobi.sdk.documents.bo.SDKConstraint[]	<i>constraints</i> List of constraint

TABLE 8.3 - SDKConstraint structure

SDKDocumentParameter

It represents an analytical driver of a document on the Server. Its structure follows:

Integer	<i>id</i>
String	<i>label</i>
String	<i>type</i> Parameter type (STRING, NUM DATE, DATE_DEFAULT)
String	<i>urlName</i>
Object	<i>values</i>
it.eng.spagobi.sdk.documents.bo.SDKConstraint	<i>constraints</i>

TABLE 8.4 - SDKDocumentParameter structure

SDKExecutedDocumentContent

It represents the content of a document execution on the Server. Its structure follows:

String	<i>FileType</i>
String	<i>fileName</i>
javax.activation.DataHandler	<i>Content</i> javax.activation.DataHandler class allows to store data in various formats. In case of templates or models, the serialized file can be retrieved by the method of the class javax.activation.DataHandler <i>InputStream getInputStream()</i>

TABLE 8.5 - SDKExecutedDocumentContent structure

SDKTemplate

It represent a service that can be a template or a QbE model. Its structure follows:

String	FolderName
String	<i>Filename</i>
javax.activation.DataHandler	<i>Content</i> javax.activation.DataHandler class allows to store data in various formats. In case of templates or models, the serialized file can be retrieved by the method of the class javax.activation.DataHandler <code><u>InputStream getInputStream()</u></code>

TABLE 8.6 - SDKTemplate structure

getDocumentsAsList

This method gets the list of SpagoBI documents that satisfy some specified filtering criteria.

Specifically:

```
SDKDocument[ ] getDocumentAsList(
    String type, String state, String folderpath)
```

The method receives the parameters to specify the filtering criteria:

- **type**, if a single document type is required. Type can assume the following values, covering all SpagoBI analytical domains: REPORT, OLAP, DASH, DATAMART, DATA_MINING, MAP, DOSSIER, OFFICE_DOC, ETL, DOCUMENT_COMPOSITE, KPI, SMART_FILTER, CONSOLE, WORKSHEET, MOBILE_CHART, MOBILE_REPORT, MOBILE_COCKPIT
- **state**, if a single document state is required. State can assume the following values, according to the document life cycle: DEV, REL, TEST, SUSP

- **folderpath**, if only the documents in a single folder of the repository are required.

Parameters can be combined for a mixed filter, as follows:

- `getDocumentsAsList(null, null, null)`: to get all documents available in SpagoBI repository
- `getDocumentsAsList("REPORT", null, null)`: to get the reports only
- `getDocumentsAsList(null, "REL", null)`: to get all the released documents only
- `getDocumentsAsList(null, null, "/Functionalities/HR")`: to get all documents available in the HR folder of the repository
- `getDocumentsAsList("REPORT", "REL", "/Functionalities/HR")`: to get the released reports available in the HR folder of the repository.

The `getDocumentsAsList` method returns an array of `SDKDocument` that contains all details for the filtered documents.

getDocumentsAsTree

This method gets the tree of those SpagoBI documents that satisfy some specified filtering criteria.

The method returns a class object `it.eng.spagobi.sdk.documents.bo.SDKFunctionality`, which represents the functionality corresponding to the input path.

Each of them contains objects, represented by the class `it.eng.spagobi.sdk.documents.bo.SDKDocument` or other subsidiary features. If the parameter is passed as null, the returned functionality will be the root of the tree.

```
SDKDocument[] getDocumentAsTree(String initialpath)
```

Input parameters:

- **initialpath** is the functionality path where documents can be retrieved. For example, if you have a functionality called Examples, the string will be `/Functionalities/Examples`.

getCorrectRolesForExecution

This method gets the list of the allowed roles for a specific document execution.

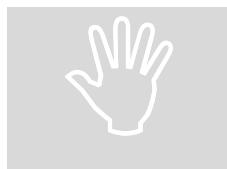
It takes the *id* of a document as the input parameter and returns all admissible roles for the execution of a document. Admissible means that the user calling the method can assume those roles: in other words, if the user can assume at least one role associated to a modality of one or more drivers associated to the document.



Users, roles and analytical drivers

For a complete understanding of users, roles and analytical drivers, and how they are used in documents, please refer to section The Behavioral Model in chapter 5 – SpagoBI Server.

If the user is a tester, all valid roles are returned, even if the user does not belong to this group. The number of returned roles is 0 if the user does not have the role of a modality for one or more drivers of the document. On the other hand, if only one role is returned, the user can accept and execute the document. If more than one item is returned, the user must choose the execution role.



Document not executable by the user

If the user is not allowed to see the desired document, an exception is thrown: *NonExecutableDocumentException*

```
List getCorrectRolesForExecution( Integer objectId,  
IEngUserProfile profile)
```

Input parameters:

- **objectId**, the id of the document
-

- **profile**, the user profile whose roles are used to extract the document.

getDocumentParameters

This method gets the list of parameters that a specific document can receive. Remember that a parameter is a solved analytical driver related to the document by means of a logical name (**Url name**).

The method returns an array containing the parameters associated with the document id "DocumentID". Each returned parameter is an instance of the class *it.eng.spagoobi.sdk.documents.bo.SDKDocumentParameter* that contains the following information:

- id association between analytical drivers and document (*DocumentParameter*)
- label
- name
- url
- type
- constraints, each constraint is an instance of the class *it.eng.spagoobi.sdk.documents.bo.SDKConstraint*.

This last class contains:

- constraint id (check)
- label
- name
- description
- type
- First constraint value (Optional)
- Second constraint value (Optional).

The constraint type can be:

DATE	Forcing a date, the first value is the data format
REGEXP	Regular expression (the first value is the string that defines the regular expression).
Maxlength	Length (this is the first value)
MinLength	Minimum length (this is the first value)
RANGE	The first value is the minimum, the second the maximum
DECIMALS	Number with decimal places (the first value is the number of decimal places)
INTERNET ADDRESS	Internet address
NUMERIC	Numeric value
ALFANUMERIC	Alphanumeric value
LETTERSTRING	Literal value
MANDATORY	Required
FISCALCODE	Social security number
EMAIL	Mail address

TABLE 8.7 - Constraint types



Attention

If the user is not able to see the desired document, an exception is launched: *NonExecutableDocumentException*

The signature is:

```
SDKDocumentParameter[] getDocumentParameters(Integer documentId,
String roleName)
```

Input parameters:

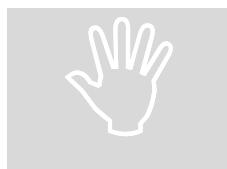
- **documentId**, id of the document whose drivers shall be retrieved
- **roleName**, role of the user executing the document.

getAdmissibleValues

This method gets the list of the allowed values for a specific document parameter. Remember that a parameter links a document to an analytical driver by means of a logical name (**Url name**).

The method returns all admissible values (in accordance with user roles) for the parameter that has the id "documentParameterId".

The output is a HashMap of strings, where the key is the value of the parameter and the value reported from the map is the description of the value. However if the parameter type is "Manual input", the returned map will be empty.



Attention

If the user is not able to see the desired document, an exception is launched: *NonExecutableDocumentException*

The signature is:

```
HashMap getAdmissibleValues(Integer documentParameterId, String  
roleName)  
documentParameterId
```

Input parameters:

- **documentParameterId**, id (URL) of the parameter
- **roleName**, role of user executing the document.

saveNewDocument

This method saves a new document into SpagoBI repository.

The signature is:

```
Integer saveNewDocument(SDKDocument document,  
SDKTemplate sdkTemplate, Integer functionalityId)
```

Input parameters:

- **document**, the document to be saved
- **sdkTemplate**, the template to be associated to the document
- **functionalityId**, the functionality where the document should be saved.

getDocumentById

This method retrieves a SpagoBI document using its numeric identifier.

The signature is:

```
SDKDocument getDocumentById(Integer id)
```

Input parameters:

- **id**, document id.

getDocumentByLabel

This method gets a SpagoBI document using its alphabetic identifier.

The signature is:

```
public SDKDocument getDocumentByLabel(String label)
```

Input parameters:

- **label**, the document label.

executeDocument

This method executes a SpagoBI document in the specified format and using the specified parameters values. All mandatory parameters must be set.

The execution result is returned in the format selected in input (PDF, XML, XLS ...).

For proper execution, you must assign values to all the required analytical drivers of the document, in particular with admissible values from the specified user. Otherwise, an exception will be thrown.

Note that web service execution is not possible for all SpagoBI documents. Documents requiring user interaction, such as OLAP or QBE, require the use of JavaScript API or tag.

The signature is:

```
public SDKExecutedDocumentContent executeDocument(SDKDocument  
document, SDKDocumentParameter[] parameters, String roleName,  
String outputType)
```

Input parameters:

- **document**, the document to be executed
- **parameters**, values for required parameters of the document
- **roleName**, role of the user executing the document
- **outputType**, type of output after execution.

downloadTemplate

This method downloads a SpagoBI document template.

The signature is:

```
SDKTemplate downloadTemplate(Integer documentId)
```

Input parameters:

- **id**, id of the document whose template will be downloaded.

uploadTemplate

This method uploads a new template for a SpagoBI document.

The signature is:

```
uploadTemplate(Integer documentId, SDKTemplate sdkTemplate)
```

Input parameters:

- **documentId**, id of the document whose template will be uploaded
- **sdkTemplate**, template that will be associated to the document.

uploadDatamartTemplate

This method uploads a document template allowing to use a business model on a SpagoBI Server. It coordinates with SpagoBIStudio to instantiate a QbE model on the resources available in the server. Optionally, it can be used to define calculated fields and create a document associated to the business model.

In practice, the method uploads the datamart.jar file under the \resources folder of SpagoBI; uploads the file defining calculated fields; and creates a QbE document in the user's personal folder, associated to the data source defined when calling the method. The last two steps are optional.

The signature is:

```
public void uploadDatamartTemplate(SDKTemplate sdkTemplate,  
SDKTemplate calculatedFields, String dataSourceLabel)
```

Input parameters:

- **sdkTemplate**, the file datamart.jar
- **calculatedFields**, the XML file containing the definition of calculated fields

- **dataSourceLabel**, the datasource label

During the creation of the document, the label and the name refer to the model, if this information is not available in the system yet.

uploadDatamartModel

This method uploads a business model onto a SpagoBI Server.

The service is used by SpagoBISTudio to upload a file containing a QbE template, according to the template parameter.

The signature is:

```
public void uploadDatamartTemplate(SDKTemplate sdkTemplate,  
SDKTemplate calculatedFields, String dataSourceLabel)
```

Input parameters:

- **sdkTemplate**, the template
- **calculatedFields**, XML file containing information about calculated fields
- **dataSourceLabel**, label of the data source that will be associated to the document.

downloadDatamartFile

This method downloads a business model on a SpagoBI Server.

This service is used by SpagoBISTudio to download the file called *fileName* located in directoy *FolderName*.

Foldername refers to the root folder [*path_resources*] / *QBE* / *datamarts*.

The signature is:

```
SDKTemplate downloadDatamartFile(String folderName, String  
fileName)
```

Input parameters:

- **folderName**, folder path where the template is stored
- **fileName**, name of the template file.

Example

Hereafter, an example for the execution of a SpagoBI document using the SDK services.

```
// proxy initialization with user credentials
String user = ....;
String password = ....;
DocumentsServiceProxy proxy = new DocumentsServiceProxy(user,
    password);
proxy.setEndpoint(
    "http://localhost:8080/SpagoBI/sdk/DocumentsService");
// all visible documents list
SDKDocument[] documents = proxy.getDocumentsAsList(null, null,
null);
for (int i = 0; i < documents.length; i++) {
    SDKDocument aDoc = documents[i];
    .....
}
// document identifier
Integer documentId = new Integer(45);
// request for valid roles for the execution of a document
String[] validRoles = null;
try {
    validRoles = proxy.getCorrectRolesForExecution(documentId);
    for (int i = 0; i < validRoles.length; i++) {
        .....
    }
} catch (NonExecutableDocumentException e) {
    System.err.println("User cannot see document " + documentId);
    return;
}
if (validRoles.length == 0) {
    System.err.println("User cannot execute document " +
documentId);
    return;
}
// request for document's parameters
SDKDocumentParameter[] parameters =
    proxy.getDocumentParameters(documentId, validRoles[0]);
for (int i = 0; i < parameters.length; i++) {
    SDKDocumentParameter aDocParameter = parameters[i];
```

```

// constraints on parameter
SDKConstraint[] constraints = aDocParameter.getConstraints();
for (int j = 0; j < constraints.length; j++) {
    SDKConstraint aConstraint = constraints[j];
    ....
}
// admissible values for parameter
HashMap values = proxy.getAdmissibleValues(
    aDocParameter.getId(), validRoles[0]);
Set entries = values.entrySet();
Iterator it = entries.iterator();
while (it.hasNext()) {
    Map.Entry entry = (Map.Entry) it.next();
    ....
}
// document execution
// setting parameters' values, if any
for (int i = 0; i < parameters.length; i++) {
    SDKDocumentParameter aParameter = parameters[i];
    String value = ....;
    if (value != null) {
        aParameter.setValues(new String[]{value});
    } else {
        aParameter.setValues(null);
    }
}
// execution
SDKExecutedDocumentContent export = proxy.executeDocument
    (sdkDocument, parameters, validRoles[0], "PDF");
InputStream is = export.getContent().getInputStream();
// now you can read the stream and write the PDF result into a
file

```

DataSetsSDKService

DataSetsSDKService is the web service interface that allows the interaction with SpagoBI data sets. Its interface provides the following methods:

- **SDKDataSet[] getDataSets()** throws NotAllowedOperationException;

It allows to return the list of datasets registered on the server.

- SDKDataSet **getDataSet**(Integer) throws
NotAllowedOperationException;

It allows to return the dataset with the specified id.

- SDKDataStoreMetadata **getDataStoreMetadata**(SDKDataSet) throws
NotAllowedOperationException, MissingParameterValue,
InvalidParameterValue;

It allows to return the dataset metadata.

- Integer **saveDataset**(SDKDataSet) throws
NotAllowedOperationException;

It allows to add a data set or to modify it, if it already exists.

- String **executeDataSet**(String, SDKDataSetParameter[]) throws
NotAllowedOperationException;

Execution of the dataset, returns the result as a JSON object.

Data structures

SDKDataSet

It represents an analytical driver of a document on the server. Its structure follows:

Integer	<i>Id</i>
String	<i>Label</i>
String	<i>Name</i>
Integer	<i>Description</i>
String	<i>fileName</i> For a dataset type file, recovers the file name
Object[]	<i>javaClassName</i> For a dataset type java, recovers the class name
Integer	<i>jdbcDataSourceId</i> data source id
String	<i>JdbcQuery</i> Query name for datasets jdbc type
Boolean	<i>NumberingRows</i> Get the columns numbered
SDKDataSetParam	<i>Parameters</i>

ter[]	
String	<i>PivotColumnName</i> Returns the pivot name column in the pivot transformation
String	<i>PivotColumnValue</i> Returns the pivot value column in the pivot transformation
String	<i>PivotRowName</i> Returns the pivot name row in the pivot transformation
String	<i>ScriptText</i> Returns Text script for Script datasets type
String	<i>ScriptLanguage</i> Back the script language for Script dataset type
String	<i>Type</i> Return dataset type
String	<i>Transformer</i> Return the transformation name
String	<i>JsonQuery</i> Return a Json query for Qbe dataset type
String	<i>Datamarts</i> Datamart Name
String	<i>WebServiceAddress</i> Return the Web Service address for a web service datasets type
String	<i>WebServiceOperation</i> Description of the operation to be carried on a Web Service
String	<i>CustomData</i> Return the content for Custom dataset type

TABLE 8.8 - SDKDataSet

SDKDataStoreMetadata

It represents the execution metadata of a dataset. Its structure follows:

SDKDataStoreFieldMetadata[]	<i>fieldsMetadata</i> Fields returned by the dataset
HashMap	<i>properties</i> A map containing potential properties of the dataset

TABLE 8.9 - SDKDataStoreMetadata

SDKDataStoreFieldMetadata

It represents the metadata for a field returned from the execution of the dataset. Its structure follows:

String	<i>className</i> Class name, identifies the type (eg. java.lang.String)
String	<i>Name</i>
HashMap	<i>properties</i> A map containing possible properties

TABLE 8.10 - SDKDataStoreFieldMetadata

SDKDataSetParameter

String	<i>name</i>
String	<i>type</i> (String, Number, Raw, Generic)
String[]	<i>values</i> parameter value

TABLE 8.11 - SDKDataSetParameter

getDatasets

It returns the list of datasets surveyed on the server.

The signature is:

```
SDKDataSet[ ] getDatasets()
```

getDataSet

It returns the dataset with the specified id.

The signature is:

```
public SDKDataSet getDataSet(Integer dataSetId)
```

Input parameters:

- **datasetId**, name of the dataset

getDataSetMetadata

It returns the metadata of the dataset as returned columns and its type. If the dataset has already been tested on the server, metadata is taken from the column of the table *ds_metadata sbi_dataset_history*, otherwise it is re-executed. If the dataset requires values for parameters to be executed, the method throws an exception *MissingParametervalue*.

The signature is:

```
public SDKDataStoreMetadata getDataStoreMetadata(SDKDataSet  
sdkDataSet)
```

Input parameters:

- **sdkDataSet**, dataset from which metadata are retrieved.

saveDataset

It saves the data set or change it, if it already exists.

The signature is:

```
public Integer saveDataset(SDKDataSet sdkDataSet)
```

Input parameters:

- **sdkDataSet**, which defines the dataset to be saved.

executeDataSet

It executes a dataset with the specified values for parameters (if any).

The signature is:

```
public String executeDataSet(String label, SDKDataSetParameter[] params)
```

Input parameters:

- **label**, label of the dataset
- **params**, required parameters for the execution.

DataSourcesSDKService

DataSourcesSDKService is the web service interface that allows the interaction with SpagoBI data sources. Its interface provides the following methods:

- **SDKDataSource[] getDataSources()** throws
NotAllowedOperationException;
- **SDKDataSource getDataSource(Integer)** throws
NotAllowedOperationException;

Data structures

SDKDataSource

It represent the data source. Its structure follows:

Integer	<i>id</i>
String	<i>label</i>
String	<i>name</i>
String	<i>descr</i> <i>Description</i>
Integer	<i>DialectId</i>
String	<i>Jndi</i> <i>String jndi</i>
Integer	<i>MultiSchema</i> <i>Indicates if it's multischema. (0 false or 1 true)</i>
String	<i>Pwd</i> <i>Password</i>
String	<i>URLConnection</i> <i>Jdbc url connection</i>

TABLE 8.12 - SDKDataSource

getDataSourcees

It returns the list of datasources.

The signature is:

```
SDKDataSource[ ] getDataSourcees()
```

getDataSource

It returns the datasource with the specified id. The signature is:

```
public SDKDataSource getDataSource(Integer dataSourceId)
```

Input parameters:

- **dataSourceId**, id of the data source to be retrieved.

EngineService

EngineService is the web service interface that allows the interaction with SpagoBI engines. Its interface provides the following methods:

- **SDKEngine[] getEngines()** throws NotAllowedOperationException;
- **SDKEngine getEngine(Integer)** throws
NotAllowedOperationException;

Data structures

SDKEngine

It represents the engine. The structure follows:

Integer	<i>id</i>
String	<i>label</i>
String	<i>name</i>
String	<i>description</i>
String	<i>ClassName</i>
String	<i>documentType</i>

Integer	<i>driverClassName</i> Driver class name if external engine
String	<i>DriverName</i>
String	<i>EngineType</i> Engine type EXT (external) or INT (internal)
String	<i>MainUrl</i>
String	<i>SecondUrl</i>
String	<i>Url</i> Engine Url
Boolean	<i>UseDataSet</i> Indicates if a dataset is used
Boolean	<i>UseDataSource</i> Indicates if a data source is used

TABLE 8.13 - SDKEngine

getEngines

It returns the list of engines.

The signature is:

```
public SDKEngine[] getEngines()
```

getEngine

It returns the engine with the specified id.

The signature is:

```
public SDKEngine getEngine(Integer engineId)
```

Input parameters:

- **engineId**, id of the engine to be retrieved.

MapsSDKService

MapsSDKService is the web service interface that allows the interaction with SpagoBI map catalogue. Its interface provides the following methods:

- **SDKMap[] getMaps()** throws NotAllowedOperationException;
- **SDKMap getMapById(Integer)** throws
NotAllowedOperationException;
- **SDKFeature[] getMapFeatures(Integer)** throws
NotAllowedOperationException;
- **SDKFeature[] getFeatures()** throws NotAllowedOperationException;
- **SDKFeature getFeatureById(Integer)** throws
NotAllowedOperationException;

To use these methods, the *MapsSDKServiceProxy* class is necessary.

Data structures

SDKMap

It represents the map feature. Its structure follows:

Integer	<i>mapId</i>
String	<i>name</i>
String	<i>descr</i> <i>description</i>
String	<i>format</i>
Integer	<i>binId</i> SVG id store in SBI_BIANRY_CONTENTS table
String	<i>url</i>
SDKFeature[]	<i>sdkFeatures</i>

FIGURE 8.6 - SDKMap

SDKFeature

Integer	<i>featureId</i>
String	<i>name</i>
String	<i>descr</i> <i>description</i>
String	<i>svgGroup</i>
String	<i>type</i> <i>feature type</i>
Booleano	<i>visibleFlag</i>

FIGURE 8.7 - SDKFeature

getMaps

It returns the maps list.

The signature is:

```
public SDKMap[ ] getMaps()
```

getMapById

It returns the map with the specified id.

The signature is:

```
public SDKMap getMapById(Integer mapId)
```

Input parameters:

- **mapId**, id of the map to be retrieved.

getMapFeatures

It returns the features related to the map with the specified id.

The signature is:

```
public SDKFeature[ ] getMapFeatures(Integer mapId)
```

Input parameters:

- **mapId**, id of the map whose features to be retrieved.

getFeatures

It returns all registered features.

The signature is:

```
public SDKFeature[] getFeatures()
```

getFeatureById

It returns the feature with the specified id.

The signature is:

```
public SDKFeature getFeatureById(Integer featureId)
```

Input parameters:

- **featureId**, id of the feature to be retrieved.

SDK Tag Library

SpagoBI SDK provides an execution tag that allows you to publish a SpagoBI Document on an external jsp page.

Execution tag

From a technical point of view, the execution tag defines an iFrame with the right request for a specific SpagoBI document. The document will be published into the produced iFrame, as an open window on SpagoBI Server.

For this reason, above all this tag is useful to publish dynamic documents (olap, cockpits, free inquiry, ad hoc reporting, etc.) that require end-user interaction. However, the tag may also be used to develop static documents, instead of the relative web service.

The execution tag accepts the following parameters:

- **spagoBIContext**: the URL of the SpagoBI context.
- **userId**: the user identifier. Documents are shown after the user's authentication only.
- **documentId**: the unique number (number identifier) of the document to be executed.
- **documentLabel**: the unique label (string identifier) of the document to be executed.
- **executionRole**: the end-user role that is used for the document execution. Note that if no value is passed for this parameter, the system will ask you for it.
- **parametersStr**: a string that collects all the parameter values for the document execution. The syntax is something like: ParUrlName1=ParValue1&ParUrlName2=ParValue2&...ParUrlNameN=ParValueN, where:
 - ParUrlNameN is the **Url Name** specified as a link to an analytical driver in the document details
 - ParValueN is the value to which the driver is sent during the document execution.



URL name of parameters

SpagoBI associates analytical drivers to documents by means of parameters. Parameters are identified by their URL, which links the corresponding driver to the document. For a detailed explanation, please refer to sections Analytical drivers and Register an analytical document in chapter 5 - SpagoBI Server.

- **parametersMap:** as an alternative to the parametersStr, the parametersMap organizes parameters on a map that uses the **Url Names** as entry keys and their values as entry values
- **displayToolbar:** TRUE or FALSE to show or hide the toolbar
- **displaySliders:** TRUE or FALSE to show or hide the sliders
- **iframeStyle:** to define the iFrame style settings
- **authenticationTicket:** authentication ticket if a SSO system is used.

Below an example of execution tag with the ParametersMap syntax:

```
<%@ taglib prefix="spagobi" tagdir="/WEB-INF/tags/spagobi" %>
....
<%
Map parameters = new HashMap();
parameters.put("ParMonth", "8");
parameters.put("ParTopNum", "5");
%>
<spagobi:execution
spagobiContext="http://localhost:8080/SpagoBI/"
    documentLabel="TOP_N_STORE"
    iframeStyle="height:500px; width:100%"
    executionRole="/role/guest01"
    parametersMap=<%= parameters %>
    displayToolbar=<%= Boolean.TRUE %>
    displaySliders=<%= Boolean.TRUE %> />
```

Below an example of execution tag with the ParametersStr syntax:

```
<%@ taglib prefix="spagobi" tagdir="/WEB-INF/tags/spagobi" %>
....
<%
String parametersStr = "ParMonth=4&ParTopNum=3";
%>
<spagobi:execution
spagobiContext="http://localhost:8080/SpagoBI/"
    documentId="1"
    iframeStyle="height:500px; width:100%"
    executionRole="/role/guest01"
```

```
parametersStr="<%= parametersStr %>"  
displayToolbar="<%= Boolean.FALSE %>"  
displaySliders="<%= Boolean.FALSE %>" />
```

SDK Javascript APIs

SpagoBI SDK provides javascript APIs in order to invoke services on the server within an HTML page. Similarly to the tag, APIs can be used to execute any type of document, but are particularly suited for those that require user interaction (OLAP, QBE ...).

From the context `http://<IP_server_address>:<port_number>/SpagoBISDK` (e.g. `http://localhost:8080/SpagoBISDK/`) it's possible to choose the option *javascript API example* to see some instances.

The functions are:

- **getDocumentUrl**: to get the execution URL of a SpagoBI document. Above all this function is useful to insert a menu item that run a SpagoBI document into an external application.
- **getDocumentHtml**: to get an iFrame that invokes a specific SpagoBI document. This function is particularly useful to include a SpagoBI document directly into a page of the external application
- **injectDocument**: to inject the above-mentioned iFrame into a <DIV> section of the HTML page.

getDocumentUrl

Function that creates the invocation URL for a specific execution, thus distinct from document, role of performance and parameters.

Parameters are:

- **documentLabel**: document label
- **executionRole**: execution role

- parameters: parameters for the execution, in the form of `{par_name: 'value', ...}`. If some document analytical drivers are not valued, SpagoBI calls for the enhancement.
- displayToolbar: it specifies whether the toolbar shall be drawn
- displaySliders: it specifies whether the sliders shall be drawn
- iframe: iframe style expressed in language 'css'.

Hereafter, an example of this function:

```
example1Function = function() {
    var url = Sbi.sdk.api.getDocumentUrl({
        documentLabel: 'DOCUMENT_LABEL'
        , executionRole: '/spagobi/user'
        , parameters: {warehouse_id: 19}
        , displayToolbar: false
        , displaySliders: false
        , iframe: {style: 'border: 0px;'}
    });

    document.getElementById('execiframe').src = url;
};
```

getDocumentHtml

Function that returns an HTML string containing the definition of an iframe that points to the service execution. Therefore the property of the iframe src is populated with the value returned from the function described above `getDocumentUrl()`.

The parameters are the same as those of the previous function. An example of this function follows:

```
example2Function = function() {
    var html = Sbi.sdk.api.getDocumentHtml({
        documentLabel: 'DOCUMENT_LABEL'
        , executionRole: '/spagobi/user'
        , parameters: {warehouse_id: 19}
        , displayToolbar: false
        , displaySliders: false
        , iframe: {height: '500px'
                    , width: '100%'
                    , style: 'border: 0px;'}
    });
};
```

```
        });
        document.getElementById('targetDiv').innerHTML = html;
    };
}
```

injectDocument

This function injects an HTML string into an existing <div>. This string contains the definition of an iframe that points to the execution service. It is generated internally using the *getDocumentHtml*. If the <div> does not exist, it is created.

The parameters are the same as those of the previous functions. Moreover, the Boolean *useExtUI* allows to invoke the new module of execution. An example of this function follows:

```
example3Function = function() {
    Sbi.sdk.api.injectDocument({
        documentLabel: 'DOCUMENT_LABEL'
        , executionRole: '/spagoBI/user'
        , parameters: {warehouse_id: 19}
        , displayToolbar: false
        , displaySliders: false
        , target: 'targetDiv'
        , height: '500px'
        , width: '800px'
        , iframe: {style: 'border: 0px;'}
    });
};
```

9. SpagoBI Applications

SpagoBI is a complete suite to develop Business Intelligence solutions. It is a ready-to-run product for BI developers who are well aware of end-users' needs and their data domain. Because Business Intelligence is an open subject, which in principle applies to all domains, SpagoBI does not provide by itself any pre-built analysis or a ready-to-use environment for end-users.

SpagoBI Applications is the module of the suite that is dedicated to the development of ready-to-run Business Intelligence Analytics, suitable for specific domains.

Goals and targets

SpagoBI Applications provides pre-built BI environments focused on a particular domain. Those applications exploit SpagoBI to build an analytical model on a specific market sector (e.g., e-government, financial risk management), subject (e.g., competitive intelligence, service intelligence, geo-marketing) or product, as a specific analytical component.

Applications basically consist of:

- a data model (DWH);
- a set of ETL processes to load data into the data model;
- several analytical documents available for end-users.

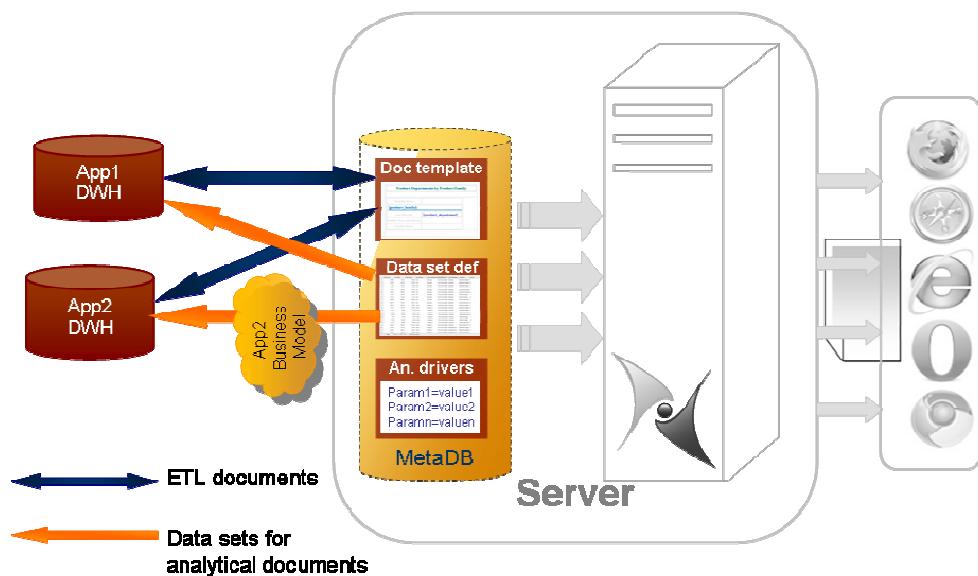


FIGURE 9.1 - Common components of a SpagoBI Application

This way, after installation and first set-up, the application is ready for use. End users can immediately start analyzing data, with no need for further developments.

SpagoBI Applications can be realized by SpagoBI team as well as by SpagoBI partners or by ISV/OEMs, who fully exploit the potentials of the suite to meet the market needs. They are very effective for end-user organizations because they can have a specific ready-to-run solution, eliminating any application development time.

SpagoBI Applications can be released both as open source products or as proprietary ones, because SpagoBI suite adopts a commercial-friendly open source license. This means that companies can realize and distribute also their proprietary applications based on SpagoBI, provided that:

- any SpagoBI suite derived work (i.e.: modifications/extensions to SpagoBI suite), is compliant with SpagoBI legal policy and licenses, as a consequence of the integration of new software code
- any distribution of the entire work (new software and SpagoBI suite) is compliant with SpagoBI legal policy and licenses (i.e.: SpagoBI package, including its source code, must be included in the distribution as a separate item with no elimination of its legal and licences texts).

Installation and configuration

As said above, applications can be released for free but also for fee, as proprietary modules. In that case, the delivery modality as well as the installation and configuration procedure depends from the single supplier. Here is explained the usual delivery modality and the installation and configuration procedure for the Applications released for free, as open source packages on the SpagoBI forge.

The first step is to check that your environment satisfies the following prerequisites:

- having the same version of a SpagoBI Server available
- having the required rdbms for the application data model (dwh), if a schema is released

If so, the second step is the download of the right package from the OW2 forge at:

- <http://forge.ow2.org/projects/spagobi> to look at latest releases
- http://forge.ow2.org/project/showfiles.php?group_id=204 to choose between all SpagoBI releases

Here you have to find the right package for the interested release.

<u>SpagoBI 2.4</u>				2009-12-17
<u>SpagoBISDK-bin-2.4.0_12182009.zip</u>	5,290.9	Any	.zip	
<u>SpagoBISDK-src-2.4.0_12182009.zip</u>	144.6	Any	Source .zip	
<u>SpagoBI 2.3</u>				2009-10-14
<u>SpagoBISDK-bin-2.3.0_10152009.zip</u>	5,265.7	Any	.zip	
<u>SpagoBISDK-src-2.3.0_10152009.zip</u>	109.4	Any	Source .zip	
<u>SpagoBI 2.2</u>				2009-06-30
<u>SpagoBISDK-bin-2.2.0_07022009.zip</u>	5,253.3	Any	.zip	
<u>SpagoBISDK-src-2.2.0_06302009.zip</u>	118.4	Any	Source .zip	
<u>SpagoBI 2.1</u>				2009-03-19
<u>SpagoBISDK-bin-2.1.0_03192009.zip</u>	2,517.2	Any	.zip	
<u>SpagoBISDK-src-2.1.0_03192009.zip</u>	38.6	Any	Source .zip	
f. SpagoBI Applications 				
<u>SpagoBI 3.3</u>				2011-12-22
<u>AuditingAndMonitoring_20111222.zip</u>	170.9	Any	.zip	
<u>SpagoBI 2.5</u>				2010-04-08
<u>BamDocuments-2.5.0_04092010.zip</u>	429.4	Any	.zip	
<u>SpagoBI 2.1</u>				2009-03-24
<u>AuditingAndMonitoringDocuments-2.1.0_03242009.zip</u>	154.2	Any	.zip	
Totali progetto				
67	303			23,066,467

FIGURE 9.2 – Download SpagoBI Applications from OW2 forge



SpagoBI Server and Applications versions

It is usually recommended to use the same version for SpagoBI Server and Applications. However, this does not mean that different versions cannot work together. Their compatibility usually depends the type of documents supported by the Application, as well as on the changes in the meta-base of the Server or on the import/export procedure (both are not so frequent).

The downloaded package usually contains:

- **<app_documents>.zip**: the archive containing all the analytical (report, chart, cockpits, etc.) and operational (ETL) documents, with all the related metadata (analytical drivers, data sets, etc.)
- **<app_dwh_schema>.zip**: the archive containing the scripts to create the DWH instance for the application data
- **Readme.txt**: the instruction file that explains all the specific installation and configuration steps for the application.

To install the a SpagoBI application, you should generally follow these steps:

- create the DWH schema, depending on the specific RDBMS used by the application. This process is normally described in the readme.txt file.
- configure a new data source on the SpagoBI Server instance installed in your environment, which references the schema created above
- load documents into SpagoBI Server. In detail, the **<app_documents>.zip** archives can be loaded into the Server using the import/export functionality.



Data source definition

Data source definition is a standard functionality available for the administrator. Please refer to section Administrator tools, at chapter 5 – SpagoBI Server.



Import/Export

The import/export is a standard functionality available for the administrator. Please, refer to section Administrator tools, at chapter 5 – SpagoBI Server.

At the end, if some ETL processes are included in the application, the relative operational documents must be executed to load data into the dwh and the application will be ready to use.

Audit and monitoring

Audit&Monitoring is a SpagoBI application released to support the monitoring of a SpagoBI instance over time.

The application provides two types of documents:

- **Audit:** documents showing the links between the *analytical model* and the *behavioural model*. These models are the core of SpagoBI and audit

documents show their static structure, i.e., which documents work with which drivers and according to which modality.

- **Monitoring:** documents showing the dynamic usage of SpagoBI Server and its *analytical model*. These documents show the activity of each engine, including possible errors, frequency of usage and most active users for each document. This information is useful for the administrator to understand, for example, which documents need review, which are not particularly useful and so on.

The Audit&Monitoring application provides the `<app_documents>.zip` only because its data are already available, as SpagoBI meta-schema.

Once imported the Audit&Monitoring package into SpagoBI Server, the administrator finds two new folders in his functionalities sub tree. He may then decide to create new menus associated to the new functionalities, using the standard tools of menu configuration.

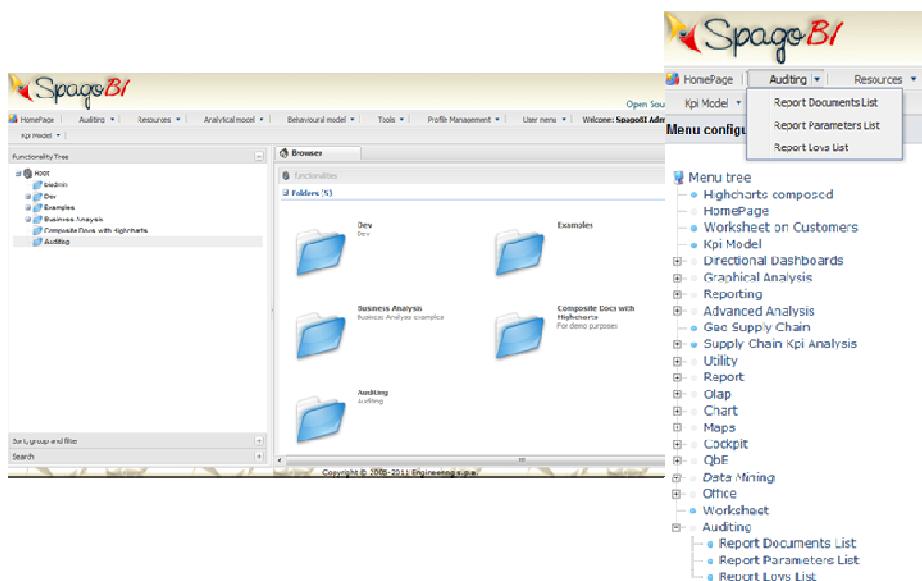


FIGURE 9.3 – Audit&Monitoring folders and menus

Audit documents

The audit model has three main reports as possible entry-point:

- **Document list:** the report shows the list of the available documents, according to some input filters
- **Driver list:** the report shows the list of available analytical drivers, according to some input filters
- **LOV list:** the report shows the list of available LOVs, according to some input filters.

From each entry point it is possible to navigate through details or other reports, as explained below.

Document List

The 'Document List' report shows the list of documents available on a SpagoBI Server instance, allowing the user to choose the filtering criteria to apply and how to show the results. Filtering criteria include:

- **Type:** one or more document types (report, chart, cockpit, etc.)
- **Engine:** one or more engines, according to the selected types (Birt, Jasper, QbE, etc.)
- **Document:** one or more documents, according to the selected types and engines
- **Order by:** the result column that drives the order of the list
- **Ordering type:** ascending or descending order.

The screenshot shows a web-based form titled 'Report Documents List'. At the top, there is a message: 'Fill the form below and click on the refresh button on the toolbar to re-execute the document'. Below this, there are five input fields arranged horizontally. From left to right: 'TYPE' (with a search icon), 'ENGINE' (with a search icon), 'DOCUMENT' (with a search icon), 'ORDER_BY' (with a search icon), and 'ORDERING_TYPE' (with a search icon). Each field has a small blue square icon with a white question mark inside it.

FIGURE 9.4 – Parameters form for Document List report

The report shows a flat list where each row contains a document information:

- **Label:** document label (alphabetic unique identifier)
- **Name:** document name
- **Type:** document type

- **Engine:** execution engine
- **Data source:** data source from which the document gets data
- **Data set:** data set used by the document to load data
- **State:** document state (DEV, TEST, REL, SUSP)
- **# Drivers:** number of drivers used by the document.

Analytical Document's List								
LABEL	NAME	TYPE	ENGINE	DATA SOURCE	DATA SET	STATE	# PARAMETER	
WEKA_CLUSTERS	Customer clusterization	DATA_MINING	WekaEngine	FoodMart		REL	2	
Gestione Colonne	Multitype parameters	REPORT	JasperReportEngine	SpagoBI		REL	2	
SBI_OBI_002	Parametric data mining	DATA_MINING	WekaEngine	FoodMart		DEV	2	
DOC_LIST	Report Documents List	REPORT	BirtEngine	SpagoBI		REL	6	
RetailStoreKpi	Retail Store	KPI	KpiEngine			REL	5	
SBI_OBI_003	Simple Data Mining	DATA_MINING	WekaEngine	FoodMart		DEV	2	
SupplierKpi	Supplier	KPI	KpiEngine			REL	5	
WarehouseKpi	Warehouse	KPI	KpiEngine			REL	5	
SupplyChainKpi	Supply Chain	KPI	KpiEngine			REL	4	
CHR_WATERFALL2	Chart Waterfall 2 for Composite Document	DASH	ChartEngine		Chart Waterfall brand name	REL	3	
SBI_OBI_025	Chart bottom right	DASH	ChartEngine		Chart Waterfall brand name	REL	3	
BPT_CST_CLUSTER	Cluster composition	REPORT	JasperReportEngine	FoodMart		REL	3	
testMultiValueAttrR	Multivalue lookup by query and script	REPORT	JasperReportEngine	SpagoBI		REL	3	
TEST_Correlazione	Related Parameters	REPORT	JasperReportEngine	SpagoBI		REL	3	
TopProductsForCountry	Top n Products for Country	REPORT	JasperReportEngine	FoodMart		REL	3	
ACC_PARQUERY	Accessible report with manual query	ACCESSIBLE_HTML	AccessibilityEngine	FoodMart		DEV	2	
RPT_AVG_SALES	Average sales	REPORT	JasperReportEngine	FoodMart		REL	2	
CHR_WATERFALLS	Chart Waterfall for Composite Document	DASH	ChartEngine		Chart Waterfall Subcategories	REL	2	
SBI_OBI_022	Chart bottom left	DASH	ChartEngine		Chart Waterfall Subcategories	REL	2	

FIGURE 9.5 – Document List

The underlined items are hyperlinks that allow the user to navigate toward details:

- **Label:** selecting this hyperlink, all details about the document are shown, as described below
- **#Drivers:** selecting this hyperlink, the user goes to the ‘Driver list’ document where he can find the drivers for the selected document only.

The detail view provides a summary section with all the main information of the selected document, and a list that shows the referred drivers.

Analytical Document's Information			
Label:	WEKA_CLUSTERS	Type:	DATA_MINING
Name:	Customer clusterization	Engine:	Weka
Description:	Customer clusterization by Data Mining		
DRIVER	NAME	DESCR	TYPE
SBI AD 037	AD_Foodmart_Clustering_Algorithms	Foodmart Clustering Algorithms	STRING
SBI AD 038	AD_Foodmart_Cluster_Num	Foodmart cluster number	STRING
SBI AD 040	AD_Foodmart_Versioning_Mode	Foodmart Versioning Mode	STRING
SBI AD 039	AD_Foodmart_Versioning_Y	Foodmart versioning true	STRING
SBI AD 041	AD_Foodmart_Versioning_Col	Foodmart versioning column names	STRING
SBI AD 042	AD_Foodmart_Versioning_Key	Foodmart versioning key	STRING
SBI AD 043	AD_Foodmart_Version	Foodmart version	STRING

FIGURE 9.6 – Document details

Selecting the **Driver** hyperlink you can reach to the ‘Driver list’ document showing the details for the selected driver only. For an overview of all drivers, you should directly access the Driver List, explained below.

Driver List

As mentioned above, this report shows the list of the analytical drivers defined in the server instance. For each driver you can see the details and documents that are associated with that driver via a parameter.

The report can be executed directly or by setting the name of specific SpagoBI document in the filter section: in the latter case, you will only see drivers associated to the selected document.

Parameters List			
LABEL	NAME	TYPE	# DOCS
AD_Foodmart_Media	Media Types for Promotion in Foodmart	STRING	3
AD_Product_Family	Foodmart Product Families	STRING	3
AD_Status	Foodmart States	STRING	2
SBI_AD_003	AD_Foodmart_Product_Family	STRING	2
SBI_AD_006	AD_Foodmart_Job_Position	STRING	2
SBI_AD_007	AD_ManualInput	STRING	5
SBI_AD_008	AD_Text_Field	STRING	2
SBI_AD_009	AD_Date	DATE	2
SBI_AD_010	AD_Foodmart_Department	STRING	2
SBI_AD_012	AD_Foodmart_State_List	STRING	14
SBI_AD_014	AD_Foodmart_Gender_Age	STRING	2
SBI_AD_015	AD_Foodmart_Subcategories	STRING	3
SBI_AD_016	AD_Foodmart_Year	STRING	3
SBI_AD_017	AD_Foodmart_Prod_Category	STRING	1
SBI_AD_018	AD_Foodmart_Gender	STRING	3
SBI_AD_020	AD_Spagobi_Test_Alphanumeric	STRING	1
SBI_AD_021	AD_Spagobi_Test_Email	STRING	1
SBI_AD_022	AD_Spagobi_Test_FC	STRING	1
SBI_AD_023	AD_Spagobi_Test_Internet	STRING	1
SBI_AD_024	AD_Spagobi_Test_String	STRING	1
SBI_AD_025	AD_Spagobi_Test_Mandatory	STRING	1

FIGURE 9.7 - Analytical drivers document

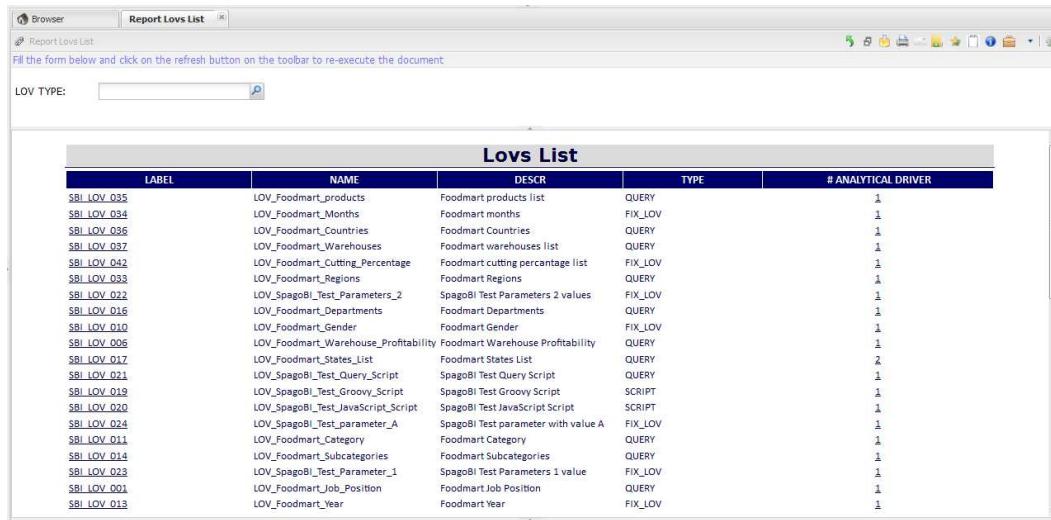
Clicking on the name of a parameter, you will be shown the details on the analysis report addressing the "Report Parameters Form" as shown in FIGURE 9.8. Here you can further explore by clicking on the name of the LOV associated with the current parameter.

Parameter's Information					
Label:	SBI_AD_037	Type:	STRING	# Associated Modalities:	1
Name:	AD_Foodmart_Clustering_Algorithms	# Associated Documents:	2	Description:	Foodmart Clustering Algorithms
LABEL	NAME	LOV LABEL	ROLE NAME	CHECK NAME	
ALL	ALL	SBI_LOV_026	/spagobi/dev /spagobi/user /spagobi/admin /spagobi/moni /spagobi/modeladmin /spagobi/test /spagobi /spagobi/user/demo /spagobi/model		

FIGURE 9.8 - Parameter details for a selected driver

LOV list

The last access point for the analysis is the document showing the list of LOVs. LOVs can be searched using a filter on the type of LOV, namely: query, fixed value, and script.



The screenshot shows a SpagoBI report titled "Report Lovs List". The interface includes a toolbar with various icons, a search bar labeled "LOV TYPE:" with a magnifying glass icon, and a message at the top stating "Fill the form below and click on the refresh button on the toolbar to re-execute the document". Below these, there is a table titled "Lovs List" with the following columns: LABEL, NAME, DESCRI, TYPE, and # ANALYTICAL DRIVER. The table contains approximately 30 rows of data, each representing a different LOV definition.

LABEL	NAME	DESCR	TYPE	# ANALYTICAL DRIVER
SBI_LOV_035	LOV_Foodmart_products	Foodmart products list	QUERY	1
SBI_LOV_034	LOV_Foodmart_Months	Foodmart months	FIX_LOV	1
SBI_LOV_036	LOV_Foodmart_Countries	Foodmart Countries	QUERY	1
SBI_LOV_037	LOV_Foodmart_Warehouses	Foodmart warehouses list	QUERY	1
SBI_LOV_042	LOV_Foodmart_Cutting_Percentage	Foodmart cutting percentage list	FIX_LOV	1
SBI_LOV_033	LOV_Foodmart_Regions	Foodmart Regions	QUERY	1
SBI_LOV_022	LOV_SpagoBI_Test_Parameters_2	SpagoBI Test Parameters 2 values	FIX_LOV	1
SBI_LOV_016	LOV_Foodmart_Departments	Foodmart Departments	QUERY	1
SBI_LOV_010	LOV_Foodmart_Gender	Foodmart Gender	FIX_LOV	1
SBI_LOV_006	LOV_Foodmart_Warehouse_Profitability	Foodmart Warehouse Profitability	QUERY	1
SBI_LOV_017	LOV_Foodmart_Status_List	Foodmart Status List	QUERY	2
SBI_LOV_021	LOV_SpagoBI_Test_Query_Script	SpagoBI Test Query Script	QUERY	1
SBI_LOV_019	LOV_SpagoBI_Test_Groovy_Script	SpagoBI Test Groovy Script	SCRIPT	1
SBI_LOV_020	LOV_SpagoBI_Test_JavaScript_Script	SpagoBI Test JavaScript Script	SCRIPT	1
SBI_LOV_024	LOV_SpagoBI_Test_parameter_A	SpagoBI Test parameter with value A	FIX_LOV	1
SBI_LOV_011	LOV_Foodmart_Categories	Foodmart Categories	QUERY	1
SBI_LOV_014	LOV_Foodmart_Subcategories	Foodmart Subcategories	QUERY	1
SBI_LOV_023	LOV_SpagoBI_Test_Parameter_1	SpagoBI Test Parameters 1 value	FIX_LOV	1
SBI_LOV_001	LOV_Foodmart_Job_Position	Foodmart Job Position	QUERY	1
SBI_LOV_013	LOV_Foodmart_Year	Foodmart Year	FIX_LOV	1

FIGURE 9.9 - LOV details document

Again, you can further explore the document by clicking on the label of a LOV. This will show a detail window with name, label, description and number of parameters associated with that LOV, as well as the script that implements the LOV in SpagoBI (see FIGURE 9.10).



The screenshot shows a SpagoBI detail window titled "Lov's Information" for the LOV labeled "SBI_LOV_049". The window displays the following information:

- Name: SBI_LOV_049
- Type: QUERY
- Label: LOV Supply Chain Resources
- Used By # Parameters: 1
- Description: Supply Chain Resources

Below this, there is a section titled "SCRIPT" containing the following SQL code:

```
<QUERY><CONNECTION>SpagoBI</CONNECTION><STMT>SELECT s.resource_id, r.RESOURCE_DESCR FROM sbi_kpi_model_resources s, sbi_resources r, sbi_kpi_model_inst k where s.RESOURCE_ID = r.RESOURCE_ID and k.KPI_MODEL_INST = s.KPI_MODEL_INST and k.name = 'Supply Chain';</STMT><VALUE-COLUMN>resource_id</VALUE-COLUMN><DESCRIPTION-COLUMN>RESOURCE_DESCR</DESCRIPTION-COLUMN><VISIBLE-COLUMNS>RESOURCE_DESCR</VISIBLE-COLUMNS><INVISIBLE-COLUMNS>resource_id</INVISIBLE-COLUMNS></QUERY>
```

FIGURE 9.10 - LOV details window

The following table summarizes all audit documents. Documents whose label is in bold represent the access points to the application; other documents can be reached via cross navigation only.

Doc label	Doc name	Accessibility	Drivers	LOVs
DOC_LIST	Report Document s List	Entry point	TYPE - SBI_AD_AM_004	SBI_LOV_AM_04
			ENGINE - SBI_AD_AM_005	SBI_LOV_AM_05
			DOCUMENT - SBI_AD_AM_003	SBI_LOV_AM_03
			ORDERBY - SBI_AD_AM_002	SBI_LOV_AM_02
			ORDERING TYPE - SBI_AD_AM_001	SBI_LOV_AM_01
PAR_LIST	Report Parameters List	Entry point	DOCUMENT - SBI_AD_051	SBI_LOV_038
LOV_LIST	Report LOV List	Entry point	LOV TYPE - SBI_AD_AM_006	SBI_LOV_AM_06
RPT_DOC_FO RM	Report Document s Form	From DOC_LIST via cross-navigation	DOCUMENT - SBI_AD_051	SBI_LOV_038
RPT_PAR_FO RM	Report Parameters Form	From PAR_LIST via cross-navigation	PARAMETERS - SBI_AD_052	SBI_LOV_039
RPT_LOV_FO RM	Report LOV Form	From LOV_LIST via cross-navigation	LOV - SBI_AD_053	SBI_LOV_040

TABLE 9.1 - Summary of SpagoBI audit documents

Monitoring documents



Work in progress

SpagoBI monitoring documents are currently under review. They will be soon available for download on SpagoBI forge.

Other SpagoBI Applications

This section focuses on some vertical applications that have been implemented using SpagoBI. As you will see, they cover different domains, which shows the flexibility of the suite for the implementation of various analytics.

These applications are:

- **Spago4Q:** a platform for the assessment of quality in software development process.
- **SpagoBI for AREAS:** the analytics of an ERP system in the healthcare domain.
- **SpagoBI for ELISA:** the analytics of a tax recovery system for local municipalities, in the domain of public administration (*proposta*: public administration domain).
- **SpagoBI for Adaxa Suite:** the analytics of an open source ERP solution based on the ADempiere project.

Spago4Q

Spago4Q is a platform for the assessment of the maturity and effectiveness of software development processes and application services as well as for the quality inspection of the released software. This goal is achieved by evaluating data and measures, collected from the project management, application and development tools through non-invasive techniques.

Spago4Q goals are:

- high adaptability to various organizational contexts (constraints on internal procedures vs. flexibility of corporate environment)
- measurement processes not bound to the adopted software
- development processes and tools
- automatic data collection from a set of tools
- support to complex evaluation systems
- set of ready-to-use quality models to satisfy the end-users' needs, providing a low cost out-of-the-box solution
- open system and compliance to the *de facto* standards.

Spago4Q has been released as a SpagoWorld project, because it also provides some new software components and above all a specific metamodel for all aspects of the process measurement activity. Moreover, it has its own community and is open to contributions that are not directly oriented to SpagoBI.

Spago4Q has been realized and managed by Spago4Q team (Engineering Group), in collaboration with SpagoBI Competency Center.



Spago4Q

Read more about Spago4Q project at www.spago4q.org.

SpagoBI for Areas

SpagoBI for Areas is the analytical capacity of AREAS, the first Italian ERP designed and developed for the healthcare domain (H-ERP).

In the electronic healthcare era, the expectations on the information technologies are increasing continuously. The clinic and administrative

processes produce a large amount of data from which the healthcare institutions can obtain information that guarantees a conscious and efficient governance, according to the users' needs and to the regional/central rules.

AREAS supports the development of specific processes for all the health institutions, collecting all data generated during their elaboration in the database, while SpagoBI for Areas is the basic instrument allowing to deduce the necessary information to support the strategic, tactical and operational decisions that the health centres have to make every day.

Therefore SpagoBI for Areas realizes the analytical component that provides the different applicative areas with an extended intelligence layer, able to support not only the decision-making managerial processes (Business Intelligence) but also the administrative and clinical managerial processes (Clinical Intelligence, HR Intelligence).

SpagoBI for Areas has been realized and managed by Engineering Group's Healthcare Business Division in collaboration with SpagoBI Competency Center.



SpagoBI for Areas

Read more about SpagoBI for Areas on:
<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/ApplicationsAreas>.

SpagoBI for Elisa

According to Italian laws and tax regulations, the local taxes collection as well as the management and supply of local services to citizens is in charge of Italian local municipalities. Specifically, local taxes include local tributes and revenue taxes.

Within ELISA project, which aims at innovating the local taxation management on the Italian territory, an analytical application based on SpagoBI supports the recovery of tax evasion at the Italian municipalities.

The product allows to perform complete cross analyses on data related to the recovery of local taxation, according to the information collected from quality and certified sources, owned by local municipalities.

SpagoBI for ELISA is used by the administrative staff and is characterized by:

- High flexibility and user-friendliness
- Thorough use of open standards and open source technology
- Compatibility with various software platforms and RDBMS
- Management of multi-entity installations.

SpagoBI for Elisa is realized and managed by Engineering Group's Public Administration Business Division in collaboration with SpagoBI Competency Center.



SpagoBI for Elisa

Read more about SpagoBI for Elisa at

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/SpagoBI-for-ELISA>.

SpagoBI for Adaxa Suite

Adaxa Suite is a collection of open source business applications, addressing the main functional areas of most business domains, released as a complete open source ERP solution.

Adaxa Suite supports the idea that the configuration and customization of the solution according to end-users' business requirements is crucial to obtain the greatest value in business applications. Adaxa Suite with SpagoBI is designed to create strategic advantage for enterprises adopting a flexible and agile approach, allowing them to gain a competitive edge.

More in detail, SpagoBI has been integrated with Adaxa Suite as the analytical engine. SpagoBI provides a high level of intelligence and reporting across all the data sources of the suite, including external data sources, providing business

views that would be too costly or too complex to create with traditional, proprietary systems.

Adaxa Suite and SpagoBI are complementary tools. Adaxa Suite supports the logging of transactions from many business sources and SpagoBI is able to identify the shape and trends of business by relying on its analytical capabilities.³¹

Standard reports provided by the application include:

- Order Stream
- Back Orders
- Sales Margin
- Inventory performance
- Stocking effectiveness
- Manufacturing recoveries
- Vendor delivery performance
- Sales and Orders to Budget
- Monthly Target
- Pipeline.

SpagoBI for Adaxa is realized and managed by Adaxa, Open Source ERP Solutions³¹.



SpagoBI for Adaxa Suite

Read more about SpagoBI for Adaxa Suite at

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/SpagoBI-for-Adaxa>.

³¹ <http://www.adaxa.com/>

Reference and resources

SpagoBI Competency Center offers public resources for developers and analysts wishing to stay tuned and participate to SpagoBI growth. In the following, the main references to SpagoBI resources are listed by category.

General information

Resources to discover or keep up to date on SpagoBI.

SpagoBI web site:

<http://www.spagobi.org>

On-line demo:

<http://spagobi.eng.it/SpagoBI>

Information kit:

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/Presentations>

Mailing lists:

http://forge.ow2.org/mail/?group_id=204

Newsletters:

<http://www.spagoworld.org/xwiki/bin/view/SpagoWorld/Newsletter>

Webinar center:

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/WebinarCenter>

Success stories:

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/SuccessStories>

Technical resources

For developers and system integrators that use SpagoBI on projects, interacting with the community.

Download:

http://forge.ow2.org/project/showfiles.php?group_id=204

Wiki for public technical documentation:

<http://wiki.spagobi.org/xwiki/bin/view/Main/>

Forum:

<http://www.spagoworld.org/jforum/forums/list.page>

Tracker for bugs management:

<http://www.spagoworld.org/jira/browse/SPAGOBI>

Professional services

For users looking for a guaranteed, high-quality support directly from the SpagoBI core team.

Support:

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/Support>

Consulting:

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/Consulting>

Training:

<http://www.spagoworld.org/xwiki/bin/view/SpagoBI/Training>

On-line Shop:

<http://spagoshop.spagoworld.org/spagoshop/>

Figures and Tables

Index of figures

FIGURE 1.1 – SpagoBI history.....	4
FIGURE 2.1 – SpagoBI suite main modules	15
FIGURE 2.2 – Interaction between SpagoBI main modules.....	15
FIGURE 3.1 – Download SpagoBI Meta package from OW2 forge	23
FIGURE 3.2 – Run SpagoBI Meta by selecting the workspace.....	24
FIGURE 3.3 – SpagoBI Meta welcome page	25
FIGURE 3.4 – Create a SpagoBI project	25
FIGURE 3.5 – SpagoBI Meta folders tree	26
FIGURE 3.6 – SpagoBI Meta towards SpagoBI Servers.....	27
FIGURE 3.7 – Connection to a SpagoBI Server from SpagoBI Meta	28
FIGURE 3.8 – Data sources on SpagoBI Meta.....	29
FIGURE 3.9 – Step 1 and 2 : Create a new connection profile selecting the right RDBMS type.....	30
FIGURE 3.10 – Step 3 and 4 : Choose or add the right JDBC driver and set its properties.....	31
FIGURE 3.11 – Step 5 : Test and save the new connection profile	31
FIGURE 3.12 – Create a new business model	33
FIGURE 3.13 – Create a physical model importing all tables from the database	34
FIGURE 3.14 – Create a business model with an initial selection of tables from the physical model	35
FIGURE 3.15 – Add a business class to the business model.....	36
FIGURE 3.16 – Edit attribute and class properties	37
FIGURE 3.17 – Create a calculated field in the business model	38
FIGURE 3.18 - Foreign keys in SpagoBI meta model.....	39
FIGURE 3.19 – Add an outbound relation to a business class	40
FIGURE 3.20 – Business classes with BM relations	40
FIGURE 3.21 – Create a query on a business model.....	41
FIGURE 3.22 – Tree representation of the business model, with symbols	42
FIGURE 3.23 – SpagoBI Meta graphical query editor	43
FIGURE 3.24 – Deployment of a BM query into SpagoBI Server.....	46
FIGURE 3.25 – Upload the business model onto SpagoBI Server	47
FIGURE 3.26 - Save the BM at different locations in the functionality tree	48
FIGURE 3.27 – Extend visibility on documents describing the BM to other users	49
FIGURE 4.1 – Relation between SpagoBI Studio and SpagoBI Server	51
FIGURE 4.2 – Manual management of document templates and data sets.....	52
FIGURE 4.3 – SpagoBI features, engines and designers.....	54
FIGURE 4.4 – Download SpagoBI Studio package from OW2 forge.....	55

FIGURE 4.5 – Run SpagoBI Studio and select the workspace	56
FIGURE 4.6 – SpagoBI Studio welcome page	57
FIGURE 4.7 – Create a SpagoBI project	58
FIGURE 4.8 – SpagoBI Studio folder tree	59
FIGURE 4.9 – SpagoBI Studio towards SpagoBI Servers	59
FIGURE 4.10 – Connection to a SpagoBI Server from SpagoBI Studio.....	60
FIGURE 4.11 – Direct access to RDBMS from SpagoBI Studio	62
FIGURE 4.12 – Access the RDBMS from SpagoBI Studio by means of a metamodel.....	62
FIGURE 4.13 – Download the document template from a Server through SpagoBI Studio.....	64
FIGURE 4.14 – Deploy the document template from SpagoBI Studio to a Server.....	64
FIGURE 4.15 – Document metadata.....	66
FIGURE 4.16 – Create a new document choosing type and engine.....	67
FIGURE 4.17 – Step 1: Create a BIRT report.....	68
FIGURE 4.18 – Switch to the report designer perspective.....	69
FIGURE 4.19 – Step 2: Creation of a data source	70
FIGURE 4.20 – Setting the connectionName parameter in the Data Source editor	71
FIGURE 4.21 – Step 3: Data Set definition.....	71
FIGURE 4.22 – Dataset editor, with preview.....	72
FIGURE 4.23 – BIRT Property Editor	73
FIGURE 4.24 – BIRT Designer preview	73
FIGURE 4.25 – Definition of a SpagoBI Server Data Source.....	76
FIGURE 4.26 – Creation of a new parameter in a BIRT report	77
FIGURE 4.27 – Insert parameters into the dataset definition.....	78
FIGURE 4.28 -- Add a hyperlink to a BIRT report for cross-navigation.....	80
FIGURE 4.29 – Configuration of the iReport editor within SpagoBI Studio	81
FIGURE 4.30 – Step 1: Create a Jasper report	82
FIGURE 4.31 – Creation of a JDBC data source in a Jasper report.....	83
FIGURE 4.32 – Step 3: Data Set definition.....	84
FIGURE 4.33 – iReport graphical editor	85
FIGURE 4.35 - Linking external dataset to Jasper report	88
FIGURE 4.36 – Deploy a Jasper report with an external dataset.....	89
FIGURE 4.37 – Add a hyperlink for cross-navigation to a Jasper report	91
FIGURE 4.38 – Chart creation and chart wizard	95
FIGURE 4.39 – Chart information, dimension and style settings.....	97
FIGURE 4.40 – Configuration Settings and Series Intervals	99
FIGURE 4.41 – Dial charts: a) Simple Dial b) Meter Dial c) Speedometer d) Bullet Dial and e) Thermometer.....	102
FIGURE 4.42 – Simple bar chart with months as categories and two value series	103
FIGURE 4.43 – Series labels configuration	105
FIGURE 4.44 – Set cross navigation from a bar chart document.....	107
FIGURE 4.45 – Stacked bar charts: a) basic and b) with cumulate and percentage options.....	108
FIGURE 4.46 – Stacked bar group with sub-categories.....	109
FIGURE 4.47 – Overlaid bar line chart with double Y axis	111
FIGURE 4.48 – Simple pie chart in 3D	112
FIGURE 4.49 – Drill parameters section for a linkable pie chart	113
FIGURE 4.50 – Simple cluster chart with one series selected (red).....	114

FIGURE 4.51 – Simple box chart	116
FIGURE 4.52 – Scatter charts: a) simple and b) with markers	119
FIGURE 4.53 – Configuration of markers and range for a scatter chart	120
FIGURE 4.54 – Block chart.....	121
FIGURE 4.55 – Create a new real time dashboard.....	125
FIGURE 4.56 – Dashboard editor: fixed sections (on the left) and specific settings (on the right)	126
FIGURE 4.57 – Dashboard components: a) Rotation b) Live line and c) Live table	128
FIGURE 4.58 – Dashboard settings for: a) live line and b) live table.....	129
FIGURE 4.59 – Create a composed document in SpagoBI Studio	132
FIGURE 4.60 – Step 1: Create a container in the editor area	133
FIGURE 4.61 - Internal navigation path in a composed document	134
FIGURE 4.62 - Document Parameters tab showing parameters associated to a document.....	136
FIGURE 4.63 - Setting a navigation path between a source and a destination document of the cockpit.....	137
FIGURE 4.64 – Create a GEO document using SpagoBI Studio.....	140
FIGURE 4.65 – Geo Designer	142
FIGURE 4.66 – Measure configuration in the GEO deisgner	144
FIGURE 4.67 – Creation of a hierarchy for a GEO document	146
FIGURE 4.68 – GUI Settings for Windows.....	148
FIGURE 4.69 – GUI Settings section for: a) Parameters and b) Labels	150
FIGURE 5.1 – Conceptual model of SpagoBI Server	154
FIGURE 5.2 – SpagoBI Server architecture	156
FIGURE 5.3 – Download SpagoBI Server ready-to-run package from OW2 forge	164
FIGURE 5.4 – SpagoBI Server login page	165
FIGURE 5.5 – Data source definition.....	166
FIGURE 5.6 – User roles in SpagoBI.....	171
FIGURE 5.7 – Profile attribute management.....	173
FIGURE 5.8 – Role management	174
FIGURE 5.9 – User management	175
FIGURE 5.10 – Create a new folder and assign permissions	177
FIGURE 5.11 – Analytical driver schema	179
FIGURE 5.12 – Analytical driver example.....	180
FIGURE 5.13 – Analytical drivers at work	181
FIGURE 5.14 – LOV Management.....	183
FIGURE 5.15 – LOV creation, with different possible types.....	184
FIGURE 5.16 – Test results of a LOV.....	185
FIGURE 5.17 – Create a validation rule.....	186
FIGURE 5.18 – Analytical driver management.....	188
FIGURE 5.19 – Analytical driver creation	189
FIGURE 5.20 – LOV Lookup list	190
FIGURE 5.21 – Profiled filters based on the use of analytical drivers: general managers (left) and product managers (right).....	191
FIGURE 5.22 – Create or modify a menu item.....	192
FIGURE 5.23 – Assign roles to a menu containing a static page	193
FIGURE 5.24 – Define a menu item that contains a document	194
FIGURE 5.25 - Linking a subobject from a JPivot document to a menu item	195

FIGURE 5.26 –Define a menu item that contains a functionality	196
FIGURE 5.27 – The components of the analytical documents in SpagoBI	198
FIGURE 5.28 -- Document development: tree view (left) and flat list view (right).....	201
FIGURE 5.29 - Detail panel of SpagoBI analytical document	202
FIGURE 5.30 – Select type of document and engine for a new analytical document.....	202
FIGURE 5.31 - Selecting a dataset for the new document	205
FIGURE 5.32 - Template versioning for analytical documents.....	206
FIGURE 5.33 - Managing visibility of an analytical document via a) visibility conditions and b) functionalities tree.....	208
FIGURE 5.34 - Adding parameters to an analytical document	209
FIGURE 5.35 - Drivers lookup window	210
FIGURE 5.36 – Correlation between analytical drivers.....	211
FIGURE 5.37 – Multiple correlations between analytical drivers	212
FIGURE 5.38 – Labels displayed during the runtime.....	212
FIGURE 5.39 – Document browser.....	213
FIGURE 5.40 – View on document metadata.....	217
FIGURE 5.41 – Execution panel in SpagoBI document browser	218
FIGURE 5.42 –Document rating.....	221
FIGURE 5.43 – “Remember me” section of the interface dedicated to <i>Hot links</i> in the user menu	222
FIGURE 5.44 –Hot links visualization window	223
FIGURE 5.45- Note editor for analytical documents.....	224
FIGURE 5.46 – Document business metadata.....	225
FIGURE 5.47 – Business metadata icon in the document thumbnail from the document browser	226
FIGURE 5.48 – Create a new distribution list	228
FIGURE 5.49 – Enter details to subscribe to a Distribution list.....	229
FIGURE 5.50 - Massive export of worksheet document.....	230
FIGURE 5.51 - Wizard for manual massive export.....	231
FIGURE 5.52 - Wizard for scheduled export of worksheet documents	232
FIGURE 5.53 – Selecting documents for a scheduled activity	233
FIGURE 5.54 - Setting document parameters in a scheduled activity.....	234
FIGURE 5.55 - Schedule configuration: a) trigger and b) dispatch.....	235
FIGURE 5.56 - Check snapshots on Scheduled executions panel.....	236
FIGURE 5.57 - Save scheduled execution as a document in SpagoBI functionalities tree	237
FIGURE 5.58 - Dataset to dispatch execution results to different folders	238
FIGURE 5.59 - Static configuration of email recipients of a scheduled execution	243
FIGURE 5.60 - Example of mapping dataset for dynamic distribution list.....	244
FIGURE 5.61 - Dynamic configuration of distribution list: a) with mapping dataset and b) wth script.....	244
FIGURE 5.62 - Send execution results to distribution list.....	245
FIGURE 5.63 - Import/export graphical interface.....	247
FIGURE 5.64 - Download exported packages.....	247
FIGURE 5.65 - Upload imported package.....	248
FIGURE 5.66 – Engine list.....	249
FIGURE 5.67 – Engine detail.....	250
FIGURE 5.68 – Add a new map	255
FIGURE 5.69 –Save a new map	256
FIGURE 5.70 – Data Source List	257

FIGURE 5.71 – Detail panel of a SpagoBI data source	258
FIGURE 5.72 – Dataset editor in SpagoBI.....	262
FIGURE 5.73 - Dataset of type query	263
FIGURE 5.74 - Script editor of a dataset.....	265
FIGURE 5.75 – File dataset creation.....	266
FIGURE 5.76 – QbE dataset creation.....	269
FIGURE 5.77 – Pivot transformation	274
FIGURE 5.78 – Parameter prompt window in the dataset preview	275
FIGURE 6.1 - Example of SpagoBI accessible report.....	282
FIGURE 6.2 - Architecture of : a) JPivot and b) JPalo OLAP Engines.....	287
FIGURE 6.3 - Creation of an OLAP analytical document	288
FIGURE 6.4 - JPivot template building wizard of an OLAP query.....	292
FIGURE 6.5 – Creation of an OLAP document	297
FIGURE 6.6 - JPivot configuration toolbar.....	298
FIGURE 6.7 - Cross navigation from a member of a dimension in a JPivot document.....	307
FIGURE 6.8 – Cross navigation with a drill through column in JPivot	309
FIGURE 6.9 - Creation of an OLAP document with JPalo on SpagoBI Server.....	315
FIGURE 6.10 - Creation of a JPalo view using the graphical interface	316
FIGURE 6.11 - Creation of a customized view in JPalo	318
FIGURE 6.12 – SpagoBI JPXMLAEngine web application main page.....	321
FIGURE 6.13 – Dial charts: a) Simple Dial b) Meter Dial c) Speedometer d) Bullet Dial and e) Thermometer.....	327
FIGURE 6.14 – Simple bar chart with months as categories and two value series	333
FIGURE 6.15 – Stacked bar charts: a) basic and b) with cumulate and percentage options.....	341
FIGURE 6.16 – Overlaid bar line chart with double Y axis	345
FIGURE 6.17 – Simple pie chart in 3D	346
FIGURE 6.18 – Simple cluster chart with one series selected (in red)	349
FIGURE 6.19 – Simple box chart	352
FIGURE 6.20 – Simple box chart configuration options.....	354
FIGURE 6.21 – Scatter charts: a) simple and b) with markers.....	355
FIGURE 6.22 – Block chart.....	360
FIGURE 6.23 – SpagoBI ExtJS line chart	372
FIGURE 6.24 - SpagoBI ExtJS bar and line chart	375
FIGURE 6.25 - SpagoBI ExtJS bar chart.....	377
FIGURE 6.26 - SpagoBI ExtJs grouped bar chart	380
FIGURE 6.27 - SpagoBI ExtJS sacked bar chart	382
FIGURE 6.28 - SpagoBI ExtJS area chart	384
FIGURE 6.29 - SpagoBI ExtJS pie chart	386
FIGURE 6.30 - SpagoBI ExtJS gauge chart.....	389
FIGURE 6.31 - SpagoBI ExtJS radar chart.....	392
FIGURE 6.32 - SpagoBI ExtJs scatter chart.....	394
FIGURE 6.33 - Cross navigation between two SpagoBI ExtJS charts	397
FIGURE 6.34 - Cross navigation parameters definition for SpagoBI ExtJS charts	398
FIGURE 6.35 - Export feature for SpagoBI ExtJS charts	398
FIGURE 6.36 - Example of SpagoBI composed document.....	401
FIGURE 6.37 – SpagoBI KPI Engine modules	405

FIGURE 6.38 - SpagoBI KPI Model functionalities	406
FIGURE 6.39 - Example of derived measure for KPI calculation.....	408
FIGURE 6.40 - KPI hierarchy.....	410
FIGURE 6.41 - KPI editor panel.....	411
FIGURE 6.42 - Creating a new KPI – dataset and threshold definition	412
FIGURE 6.43 - Composed KPI configuration – KPI links.....	413
FIGURE 6.44 - KPI Model editor – KPI model list and model configuration editor.....	414
FIGURE 6.45 - Create the root node of a KPI model	415
FIGURE 6.46 - Creating a KPI node in the model.....	416
FIGURE 6.47 - KPI instance definition editor.....	417
FIGURE 6.48 - KPI threshold configuration panel.....	420
FIGURE 6.49 - Resource definition panel.....	421
FIGURE 6.50 – KPI grants configuration panel.....	423
FIGURE 6.51 - Associate organizational units to KPI execution.....	424
FIGURE 6.52 - Goal model definition.....	425
FIGURE 6.53 - KPI visualization report.....	432
FIGURE 6.54 - Steps of a Knowledge Discovery in Database process	435
FIGURE 6.55 - DWH used in the KDD example with Weka	438
FIGURE 6.56 - Customer-department relational table.....	439
FIGURE 6.57 - New table after pre-processing.....	440
FIGURE 6.58 - New table after transformation.....	441
FIGURE 6.59 - Adding the database loader operator	442
FIGURE 6.60 – Configuration of the database loader	442
FIGURE 6.61 – Configuration of the database saver operator.....	443
FIGURE 6.62 - Configuration of the addcluster operator	444
FIGURE 6.63 - Remove unnecessary columns	444
FIGURE 6.64 - Data mining process once completed.....	445
FIGURE 6.65 – Access page to SpagoBI Weka engine.....	446
FIGURE 6.66 – Creating an analytical document using the kfml file	447
FIGURE 6.67 – Analytical document correctly created.....	447
FIGURE 6.68 – Event Monitor	448
FIGURE 6.69 – Associating the WEKA document to other documents	448
FIGURE 6.70 – Selecting the documents to be associated to the WEKA document.....	449
FIGURE 6.71 – Cluster composition report.....	449
FIGURE 6.72 – Average sales report	450
FIGURE 6.73 – Results of the document association	451
FIGURE 6.74 – Clusterer analytical driver.....	452
FIGURE 6.75 – ClusterNum analytical driver.....	453
FIGURE 6.76 – KDD process in data mining	454
FIGURE 6.77 - QbE document lifecycle	457
FIGURE 6.78 - QbE user interface	460
FIGURE 6.79 - Query ordering and grouping.....	462
FIGURE 6.80 - Functionalities in the contextual menu of the datamart schema panel	463
FIGURE 6.81 - Create a range for age with the band wizard	464
FIGURE 6.82 - Range instance creation	465

FIGURE 6.83 - Query creation : drag attributes and choose alias	466
FIGURE 6.84 - Select fields: query configuration functions.....	467
FIGURE 6.85 - Filter lookup for right operand selection.....	469
FIGURE 6.86 - QbE results tab with custom image and hyperlink.....	477
FIGURE 6.87 - QbE query: use of a subquery in a filter.....	480
FIGURE 6.88 - Smart filter logical architecture	481
FIGURE 6.89 - Grouping filter results after execution	481
FIGURE 6.90 - Smart filter selection form.....	482
FIGURE 6.91 - Choose the metamodel for the QbE query.....	483
FIGURE 6.92 - Smart filter form editor.....	484
FIGURE 6.93 - Static closed filters configuration editor	486
FIGURE 6.94 - Static closed filter editor	487
FIGURE 6.95 - Static open filter editor	489
FIGURE 6.96 - Add dynamic filter group.....	490
FIGURE 6.97 - Create a group of dynamic filters.....	490
FIGURE 6.98 - Grouping variables definition	491
FIGURE 6.99 - Worksheet designer	494
FIGURE 6.100 - Designing the first sheet.....	495
FIGURE 6.101 - Save the worksheet document	495
FIGURE 6.102 - Field configuration options	496
FIGURE 6.103 - Choosing the grouping function for charts	497
FIGURE 6.104 - Setting export options.....	498
FIGURE 6.105 - Pivot table in a worksheet document	498
FIGURE 6.106 - Pivot table details configuration	499
FIGURE 6.107 - Editing options on pivot tables.....	499
FIGURE 6.108 - Splitting filter in the worksheet designer	500
FIGURE 6.109 – A base layer in raster and vector format	503
FIGURE 6.110 – Overlapping layer.....	504
FIGURE 6.111 – Examples of feature	505
FIGURE 6.112 – Definition of GIS, BI, spatial data and business data.....	506
FIGURE 6.113 – Catalog-based integration of spatial and BI data.....	508
FIGURE 6.114 – Example of two layers included in the map	511
FIGURE 6.115 – Template structure definition.....	512
FIGURE 6.116 – Template definition	513
FIGURE 6.117 – Correspondence between dataset measures and SVG layers.....	517
FIGURE 6.118 – Document execution results.....	523
FIGURE 6.119 – Dynamic navigation through the geographic dimension	524
FIGURE 6.120 – Encoding the dataset.....	525
FIGURE 6.121 – Re-executing the analytical document and navigating through the product hierarchy.....	525
FIGURE 6.122 – Cross navigation starting from a geo document	527
FIGURE 6.123 – Windows configuration block	528
FIGURE 6.124 – Bridge integration architecture of the GEOResult engine	534
FIGURE 6.125 – Input elements to produce a thematic map	535
FIGURE 6.126 – The map uses different color intensity according to the value of the selected measures.....	539
FIGURE 6.127 – Defining the name of the map	540

FIGURE 6.128 – Comparison between the choropleth and propoportionalSymbols analysis.....	541
FIGURE 6.129 – Indicator attribute definition	542
FIGURE 6.130 – Measure definition.....	543
FIGURE 6.131 – Defining how target layer and input dataset shall be joined.....	544
FIGURE 6.132 – Creating an analytical document	547
FIGURE 6.133 – Contextual pop-up window.....	548
FIGURE 6.134 – Specifying the properties of the pop-up window.....	549
FIGURE 6.135 – Defining the detail document included in the pop-up window	550
FIGURE 6.136 - Rotation (a) and multi rotation (b) dashboardDataset structure.....	554
FIGURE 6.137 - Live line (a) and live table (b) dashboards	561
FIGURE 6.138 - SpagoBI real time console	564
FIGURE 6.139 - RT console graphical widgets: a) live line b) multi led c) speedometer d) semaphore	571
FIGURE 6.140 - Example of table layout in the console summary panel.....	578
FIGURE 6.141 - Cross navigation bar in the RT console.....	582
FIGURE 6.142 - Inline actions in the RT console	592
FIGURE 6.143 - Parameters to start process execution.....	595
FIGURE 6.144 - Error acknowledge popup window.....	597
FIGURE 6.145 – Alarm acknowledge popup window.....	599
FIGURE 6.146 - Setting the start and stop date for downloading log files.....	601
FIGURE 6.147 - Inline widgets in the RT console	606
FIGURE 6.148 - Parameters window for user input	612
FIGURE 6.149 - Final architecture of the SpagoBI Mobile module.....	621
FIGURE 6.150 - Current architecture of the SpagoBI Mobile module.....	621
FIGURE 6.151 - Installation and login of SpagoBI Mobile client on IPad.....	622
FIGURE 6.152 - Mobile navigator.....	624
FIGURE 6.153 – Mobile cockpit composed of a table and a chart	637
FIGURE 6.154 - Slider in a mobile cockpit	639
FIGURE 6.155 - SpagoBI Office document executed within the browser.....	641
FIGURE 6.156 - Dossier template configuration GUI	645
FIGURE 6.157 - Configuration page of the dossier document	645
FIGURE 6.158 – Starting a new execution of a dossier document.....	649
FIGURE 6.159 - List of add note activities for a user	650
FIGURE 6.160 - Add notes editor for the dossier document	651
FIGURE 6.161 - Approve the execution of a dossier document	651
FIGURE 7.1 - Creation process of an ETL analytical document.....	653
FIGURE 7.2 - Talend tMap operator	655
FIGURE 7.3 - Talend ETL job sample	655
FIGURE 7.4 – Setting connections through parameters and values.....	656
FIGURE 7.5 - Configuring connection to SpagoBI in Talend Open Studio.....	656
FIGURE 7.6 - Deploying a Talend ETL job on SpagoBI.....	657
FIGURE 7.7 - Starting an ETL process from SpagoBI document browser	659
FIGURE 7.8 - Talend ETL event management	660
FIGURE 7.9 - Scheduling job execution	661
FIGURE 7.10 - Save options in job scheduling.....	662
FIGURE 7.11 - ProcessEngine GUI for starting/stopping processes	669

FIGURE 7.12 - SpagoBI Registry document.....	671
FIGURE 8.1 - SpagoBI SDK towards external applications	676
FIGURE 8.2 – Download SpagoBI SDK package from OW2 forge.....	678
FIGURE 8.3 - Access the SDK test pages.....	679
FIGURE 8.4 - Test page for web services.....	680
FIGURE 8.5 - Java API example use case	681
FIGURE 8.6 - SDKMap.....	707
FIGURE 8.7 - SDKFeature	708
FIGURE 9.1 - Common components of a SpagoBI Application	716
FIGURE 9.2 – Download SpagoBI Applications from OW2 forge.....	718
FIGURE 9.3 – Audit&Monitoring folders and menus.....	720
FIGURE 9.4 – Parameters form for Document List report.....	721
FIGURE 9.5 – Document List	722
FIGURE 9.6 – Document details.....	723
FIGURE 9.7 - Analytical drivers document.....	724
FIGURE 9.8 - Parameter details for a selected driver.....	724
FIGURE 9.9 - LOV details document.....	725
FIGURE 9.10 - LOV details window	725

Index of tables

TABLE 4.1 - Drill parameters.....	98
TABLE 4.2 - Chart configuration options.....	105
TABLE 4.3 - Stacked bar configuration options	108
TABLE 4.4 - Stacked bar group configuration options	109
TABLE 4.5 - Overlaid bar line configuration options	110
TABLE 4.6 - Pie chart configuration options.....	111
TABLE 4.7 - Simple cluster chart configuration options	115
TABLE 4.8 - Simple box chart configuration options	116
TABLE 4.9 - Scatter chart configuration options.....	118
TABLE 4.10 - Markers scatter chart configuration options	119
TABLE 4.11 - Block chart configuration options	122
TABLE 4.12 - Dashboard configuration options	125
TABLE 4.13 - Rotation dashboard configuration options	127
TABLE 4.14 - Live line configuration options	128
TABLE 4.15 - Live table configuration options	130
TABLE 4.16 - Measure Configuration Parameters.....	144
TABLE 4.17 - Special parameters for cross navigation	147
TABLE 4.18 - GUI Windows configuration parameters.....	149
TABLE 4.19 - GUI Params configuration parameters	150
TABLE 5.1 - SpagoBI role types	170
TABLE 5.2 - Execution panel top toolbar	220

TABLE 5.3 – Configuration settings for all SpagoBI engines	254
TABLE 6.1 Cube configuration options in the Mondrian/JPivot Engine	298
TABLE 6.2 - Single column ordering options	299
TABLE 6.3 - Style configuration options for a simple dial chart	330
TABLE 6.4 - Style configuration options for a simple bar chart	336
TABLE 6.5 - Configuration parameters for a simple bar chart	337
TABLE 6.6 - Drill parameters for charts.....	340
TABLE 6.7 - Additional configuration parameters for stacked bar charts.....	343
TABLE 6.8 - Configuration options for overlaid bar line chart	345
TABLE 6.9 - Pie configuration options.....	347
TABLE 6.10 - Simple cluster chart configuration option.....	351
TABLE 6.11 - Configuration parameters of simple scatter chart template	357
TABLE 6.12 - Marker definition parameters	359
TABLE 6.13 - Block chart configuration options	362
TABLE 6.14 - Parameter definition for a composing document	403
TABLE 6.15 - KPI-specific system parameters.....	409
TABLE 6.16 - KPI system variables	430
TABLE 6.17 - Configuration parameter attributes.....	432
TABLE 6.18 - Analytical drivers for Weka document versioning	455
TABLE 6.19 - Select fields toolbar	467
TABLE 6.20 - Possible combinations of filters in the QbE.....	470
TABLE 6.21 - Static closed filters configuration options.....	486
TABLE 6.22 - Editing actions on pivot tables.....	501
TABLE 6.23 - Heuristic for value interval partition.....	520
TABLE 6.24 - Heuristics supporting color interval definition.....	521
TABLE 6.25 - Configuration parameters for GEO Engine GUI window.....	531
TABLE 6.26 - List of parameters to be used during the GUI setting process.....	533
TABLE 6.27 - Configuration attributes for rotation dashboard template	556
TABLE 6.28 - Configuration attributes for multi rotation dashboard template	559
TABLE 6.29 - Configuration attributes in the live line template	560
TABLE 6.30 - Configuration attributes in the live table template.....	562
TABLE 6.31 - Dataset column driving console actions.....	566
TABLE 6.32 - Dataset configuration properties.....	569
TABLE 6.33 - Live line widget configuration properties in the RT console.....	572
TABLE 6.34 - Multi led widget configuration properties in the RT console	575
TABLE 6.35 - Speedometer widget configuration options in the RT console	576
TABLE 6.36 - Semaphore widget configuration options in the RT console.....	578
TABLE 6.37 - Table layout configuration properties for the RT console summary panel.....	580
TABLE 6.38 - Table layout widget configuration properties for the RT console summary panel	580
TABLE 6.39 - Column configuration options in the detail panel of the console	586
TABLE 6.40 - Single column filter configuration properties	587
TABLE 6.41 - Generic actions in the RT console.....	588
TABLE 6.42 - Item-specific actions in the RT console	593
TABLE 6.43 - Configuration options for parameters management in the RT console template	616
TABLE 6.44 - Options for column headers with internationalization.....	618

TABLE 6.45 - Column attributes for mobile tables.....	627
TABLE 6.46 - Condition attributes for mobile reports	627
TABLE 6.47 - Values for parameter type attribute in mobile reports.....	629
TABLE 6.48 - Series attributes for mobile charts.....	630
TABLE 6.49 - Axis attributes for mobile charts.....	631
TABLE 6.50 - Values for parameter type attribute for mobile charts.....	635
TABLE 6.51 - Slider attributes for mobile documents	639
TABLE 7.1 - Parameters to be used in the configuration panel.....	661
TABLE 8.1 – SDKDocument structure.....	685
TABLE 8.2 - SDKFunctionality structure.....	686
TABLE 8.3 - SDKConstraint structure	686
TABLE 8.4 - SDKDocumentParameter structure	687
TABLE 8.5 - SDKExecutedDocumentContent structure.....	687
TABLE 8.6 - SDKTemplate structure	688
TABLE 8.7 - Constraint types	692
TABLE 8.8 - SDKDataSet.....	701
TABLE 8.9 - SDKDataStoreMetadata.....	701
TABLE 8.10 - SDKDataStoreFieldMetadata.....	702
TABLE 8.11 - SDKDataSetParameter.....	702
TABLE 8.12 - SDKDataSource.....	704
TABLE 8.13 - SDKEngine	706
TABLE 9.1 - Summary of SpagoBI audit documents.....	726