

Lycia Development Suite



Lycia BI

LyciaBI Getting Started
Version 7 – December 2013
Revision number 1.81



Lycia BI BIRT Integration Getting Started

Part Number: 020-070-181-013

Elena Krivtsova / Svitlana Ostnek / Daria Sakhno

Technical Writers

Last Updated 12 December 2013

LyciaBI Getting Started

Copyright © 2003-2012 Querix™ Ltd. All rights reserved.

Part Number: 020-060-101-012

Published by:

Querix Ltd, 50 The Avenue, Southampton, SO17 1XQ, UK

Printing History:

May 2011: Beta edition

January 2012: First Edition

December 2013: Last Edition

Last Updated

December 2013

Documentation written by:

Elena Krivtsova, Svitlana Ostnek, Daria Sakhno

Notices

The information contained within this document is subject to change without notice. If you find any problems in the documentation please submit your comments to documentation@querix.com . No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express permission of Querix Ltd.

Other products or company names used within this document are for identification purposes only, and may be trademarks of their respective owners.



Table of Contents

ABOUT THIS GUIDE	3
INTRODUCTION TO BIRT.....	4
Background	4
BIRT Report Design Files.....	5
<i>BIRT INTEGRATION</i>	5
THE BIRT Report Engine API (RE API)	5
BIRT Report Designer	6
THE BIRT Report Viewer	7
ENGINE CLASSES.....	8
Scripting.....	8
BIRT REPORTS.....	10
<i>Creating a Report File</i>	10
<i>The Report Design Perspective</i>	12
The Data Explorer view	13
The Palette view	13
The Outline view.....	16
<i>Database as a Data Source</i>	17
Setting the Data Source.....	17
Reusing a Data Source for other reports.....	19
Creating the Data Set	22
<i>The Report Design Perspective</i>	25
The Data Explorer view	25
The Palette view	26
The Outline view.....	29
<i>Composing a report</i>	30
<i>a complex bar chart</i>	33
<i>Creating Other types of charts</i>	38
Pie Chart	38
Meter Chart.....	39
<i>Data Cubes and Cross Tabs</i>	42
<i>Lists</i>	45
<i>Tables</i>	52
<i>Images</i>	56



INTEGRATING BIRT REPORTS INTO 4GL.....	59
EXAMPLES	59



About this Guide

This guide is intended for use, primarily, by market and business analysts who will create and maintain the reports. As such, the manual assumes the reader to have a basic knowledge of a query language (e.g. SQL, 4GL, etc..) and basic understanding of a database structure and functioning.

This manual describes the process of a report server management, reports creation, deployment and viewing. It also covers the aspects of integrating reports into 4GL. Creating graphical reports does not require the knowledge of 4GL aside from that needed for creating database queries.

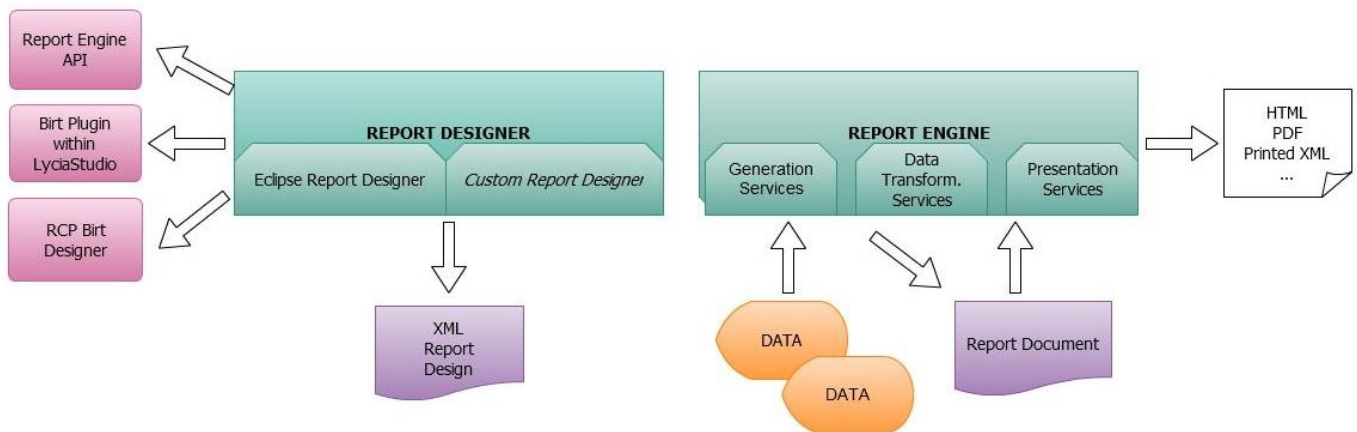
If you have any comments on how to improve this guide, please address them to documentation@querix.com.



Introduction to BIRT

Background

BIRT is an Eclipse-based open source reporting platform for server, desktop and web based applications. BIRT has two main components: a visual report designer (available as an Eclipse integrated plug-in and standalone designer), and a runtime components that you can deploy to your application server.



Application implementing in BIRT becomes possible with the report designer. The Eclipse-based interface provides an HTML page-oriented design model to create reports that are easy to create and integrate into web based applications.

BIRT reports are complex reports which include visual graphs, labels and other objects. The BIRT Engine enables you to produce professional looking reports. It supports the following operations:

- Adding a rich variety of report types to your application: lists, charts, grids, letter and documents, compound reports.
- Managing data sources and data sets.
- Running a report to produce HTML/Paginated HTML, WORD, XLS, PS, ODT, ODS, ODP or PDF format.
- Making editing quick and easy with the most commonly used properties in a convenient format.
- Exporting Report data to CSV.
- Generating and rendering report content.


The BIRT provides two types of designer: a plug-in for the Eclipse IDE and a Rich Client Platform (RCP) version.



BIRT Report Design Files

BIRT Report Design Files are XML files with the default extension .rptdesign. BIRT Reports can contain single or multiple files. The files are categorized as either library files or resource files.

- BIRT library files are also XML files with the extension .rptlibrary. BIRT library files contain code that is used multiple times for items such as font type, size, page numbers, etc. The individual files can be reused multiple times in BIRT report designs and templates.
- Resource Files contain items such as images or external files. Resource files can be used by either report design files or library files. The XML of the BIRT Report details which library files and resource files the report requires. Without these files, the BIRT report does not execute.

	<p>The report design file is the key designed "artifact" associated with a report. It contains the elements that make up a report, including visual design components (tables, images) and non-visual items (report parameters, data sources, etc.)</p>
---	---

BIRT INTEGRATION

THE BIRT Report Engine API (RE API)

The BIRT Report Engine API is the programming interface that allows you to integrate the run-time part of BIRT into your application as a set of POJOs (Plain Old Java Objects). With the engine, the wide range of actions are enabled:

- discovering the parameters defined for a report and get their default values,
- retrieving required data and transforming it as needed,
- design your own report, using the wide range of different items, including:
 - visual design components: tables, images, etc.
 - non-visual design components: report parameters, data sources, formatting styles.
- final rendering your report in HTML or PDF.

Using the Report Engine API (RE API) the engine can be embedded within any Java/Java EE application. The BIRT Web Viewer uses this API to execute and display reports.

You can find more detailed information regarding API Overview and its Configuration and Installation tips by following the link:

<http://www.eclipse.org/birt/phenix/deploy/reportEngineAPI.php#api>



BIRT Report Designer

Application development with BIRT starts with the report designer. This Eclipse-based set of plug-ins offers a variety of tools to build reports quickly. Some of these are listed below.

- **Data Explorer** - Organizes your data sources (connections) and data sets (queries). The data set editor allows you to test your data set to ensure the report receives the correct data. Within this view multi dimensional cubes can be created using existing data sets. Cubes are currently used when building dynamic cross tables. This view also is used to design report parameters.
- **Layout View** - WYSIWYG editor that provides drag & drop creation of the presentation portion of your report.
- **Palette** - Contains the standard BIRT visual report elements such as labels, tables, and charts and is used in conjunction with the Layout View to design reports.

Property Editor - Presents the most commonly used properties in a convenient format that makes editing quick and easy. BIRT also integrates with the standard Eclipse property view to provide a detailed listing of all properties for an item.

- **Report Preview** - You can test your report at any time with real data. The preview is a window directly within Eclipse.
- **Script Editor** - Scripting adds business logic to reports during data access, during report generation, or during viewing. The code editor provides standard Eclipse features for editing your scripts: syntax coloring, auto-complete and more. An interesting new feature, for BIRT 2.3 is the ability to debug scripts while the report is running.
- **Outline** - BIRT reports are organized as a tree structure with the overall report as the root, and separate categories for styles, report content, data sources, data sets, report parameters and more. The Outline view provides a compact overview of your entire report structure.
- **Cheat Sheets** - Learning a new tool is always a challenge, but Eclipse offers an innovative solution: cheat sheets. These are short bits of documentation that walk you through new tasks.
- **Resource Explorer** - BIRT allows the reuse of report objects, such as tables, data sources and styles. Objects created for reuse are stored in a library file. To browse the contents of report libraries BIRT supplies a Resource Explorer view. This view list all libraries within the resource folder, in addition other shared content such as images and JavaScript files.
- **Chart Builder** - Adding Charts to BIRT designs is expedited with the Chart Builder. Chart creation is separated into three phases: Select Chart Type, Select Data, and Format Chart.
- **Expression Builder** - BIRT expressions are really just simple scripts that return a value. Expressions are used for assigning data values to report elements, building image locations, hyperlinks, parameter



default values and many other places. Expressions are constructed within BIRT using the Expression Builder.

THE BIRT Report Viewer

The BIRT Viewer is the application used to preview reports within Eclipse. It illustrates how the BIRT engine can be used to create and render report content. It is used to preview reports within Eclipse.

This Web Viewer supports various interactive features, such as:

- using table of contents,
- exporting content to different formats,
- library of tags used to provide report functionality within an existing web application,
- report pagination, etc.

The BIRT Report Engine is embedded on Lycia system. When you install your application software, you also install the BIRT Report Engine.

BIRT includes an Apache Tomcat server invoked every time you preview your reports.

The Viewer Plug-in can also be embedded within a Rich Client Platform (RCP) application. BIRT provides web output as a single HTML document, paginated HTML, PDF, XLS, DOC, PPT, and Postscript. In addition, the Web Viewer allows exporting the data to CSV, printing, and Table of Contents functionality.



ENGINE CLASSES

Scripting

BIRT provides a complete set of JavaScript objects to access the Report Object Model: a representation of both the design and runtime aspects of a report, allowing complete control of the report to handle even the most complex report formatting tasks. Some of them are highlighted later in this chapter.

In this chapter, you can find the description of the basic Engine classes.

EngineConfig

The class is used to set global options for the report engine and to specify the location of engine plug-ins, data drivers. You can also use it to add application-wide scriptable objects.

```
EngineConfig config = new EngineConfig();  
  
config.setEngineHome( "put engine path here" );  
  
setLogConfig('String directoryName, Level level;)
```

This call sets the Log directory name and level (OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST and ALL)
Setting directoryName to null will log to stdout.

ReportEngine

The class represents the BIRT Report Engine. The report engine is created through a factory supplied by the Platform. Thus, before creating the engine, you should start the Platform. It will load the required plug-ins. To do it, you will need to call `Platform.startup(config)` function that takes an `EngineConfig` object as argument. After using the engine, call `Platform.shutdown()` to do clean-up, including unloading the extensions.



Each application should create just one `ReportEngine` instance and use to run multiple reports. Remember it while deploying the engine in a servlet as well.



IReportRunnable

As it was already mentioned, BIRT report designs are stored as XML files with the .rptdesign extension. To work with the report design in the engine, you need load the report using one of the `openDesign()` methods in the `ReportEngine` class. These methods return a `IReportRunnable` instance representing the engine's view of the report design.

IEngineTask

BIRT Engine tasks provide the framework for managing the scripting context, report locales and so on. For instance, if an operation requires a script context or a locale, a task class will represent the operation. It means that opening a design file or retrieving an image in the design file does not required setting up a scripting context. Other operations, such as retrieving default parameters or a dynamic selection list, running and rendering a report, etc. require a scripting context, and are represented as tasks.

Create tasks using the factory methods on the `ReportEngine` class. The supported Tasks are shown below:

- `engine.createDataExtractionTask();`
- `engine.createGetParameterDefinitionTask();`
- `engine.createRenderTask();`
- `engine.createRunTask();`
- `engine.createRunAndRenderTask();`

The detailed overview of these functions can be found in here:

<http://www.eclipse.org/birt/phenix/deploy/reportEngineAPI.php#api>

IRunTask

Use this task to run a report and generate a report document, which is saved to disk. The report document can then be used with the `IRenderTask` to support features such as paging.

IRunAndRenderTask

The task is used to run a report and output it to one of the supported output formats. `IRunAndRenderTask` does not create a report document which means you will need to create a new task for each report that you run. Reports may take different parameters, the default parameters values and the ones you can individually. The `IRunAndRenderTask` provides the `validateParameters()` method to validate the parameter values before you run the report.

The information regarding Web Viewer Settings, Parameters, Directory Structure and Tag Library is provided on Eclipse website, in the corresponding Chapter:

<http://www.eclipse.org/birt/phenix/deploy/viewerUsageMain.php>

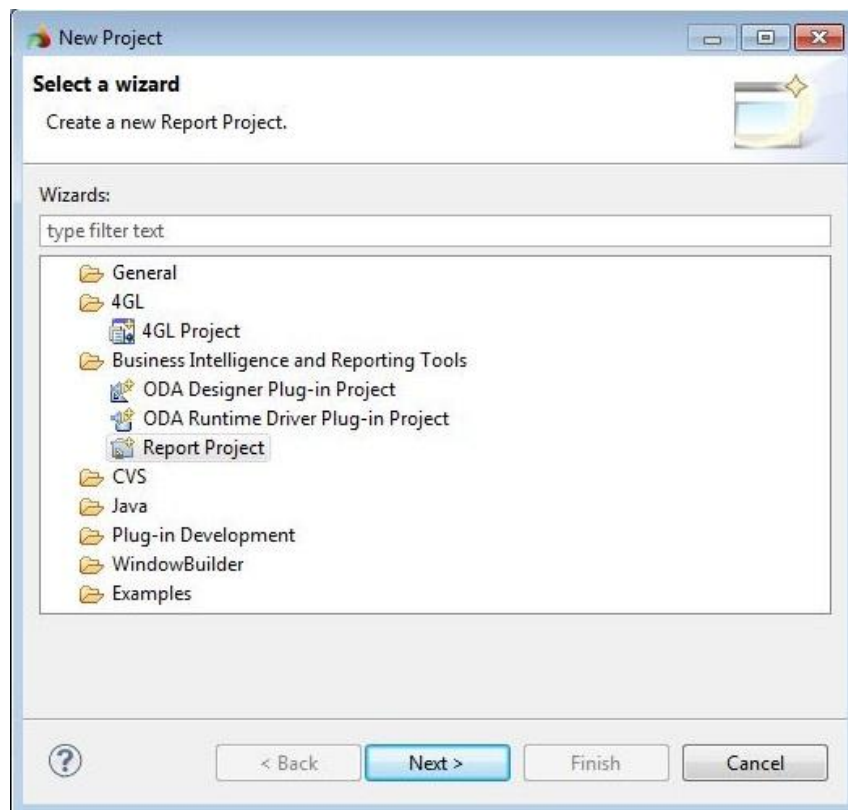


BIRT REPORTS

Creating a Report File

First you need to create a new report file you will later use while creating your 4G BIRT report. To do it:

1. Click on **File ->New ->Project ->Business Intelligence and Reporting Tools ->Report Project**



2. Specify a new project name.
3. Right click on the new project and select **New ->Report** in the context menu. Specify a new file name. To create a new report from scratch, press **Finish**.

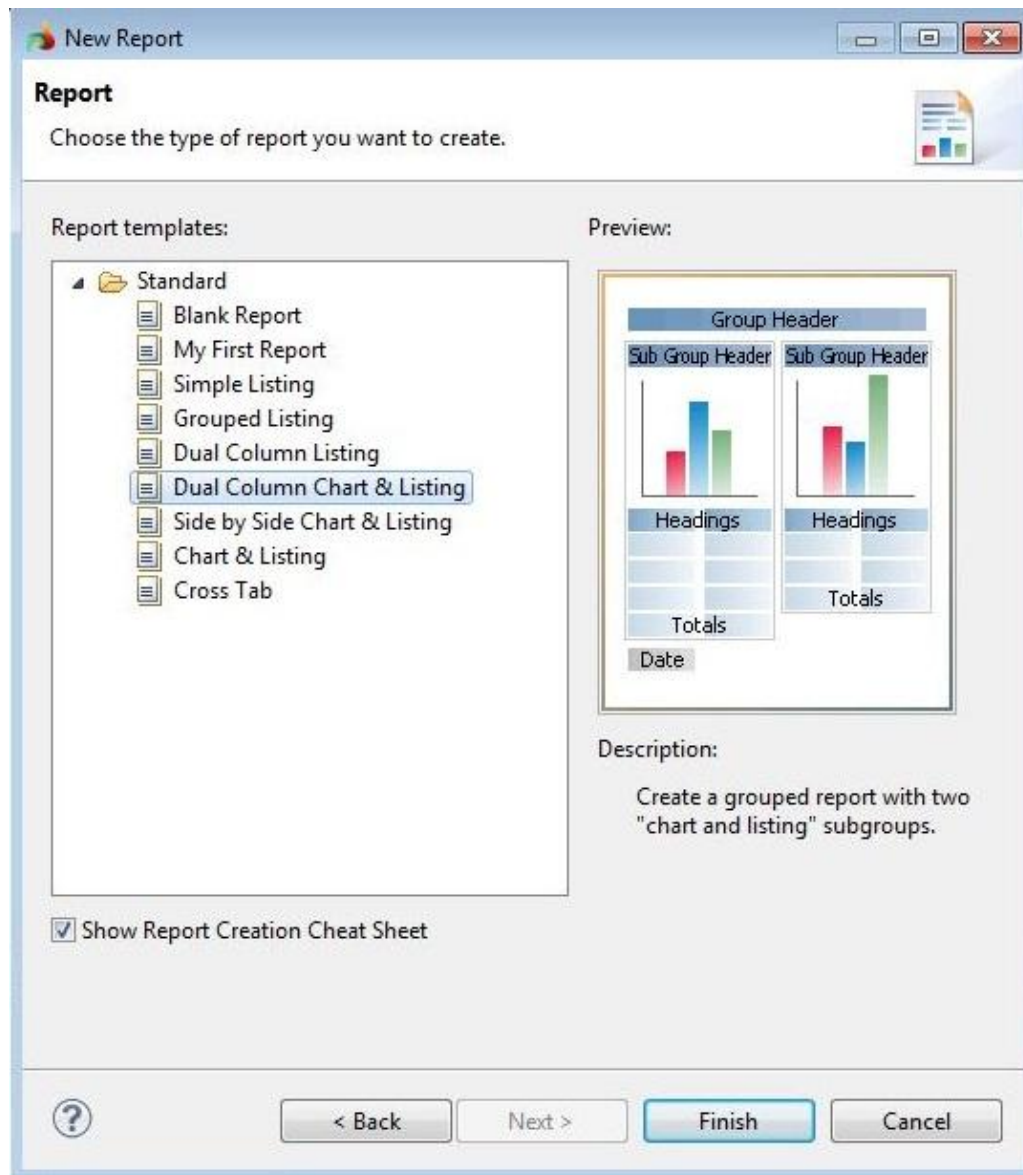
If you want to use a template report, choose **Next**. You will see the list of templates to use for your future report. Standard report templates include:

- Blank Report
- My First Report
- Simple Listing



- Grouped Listing
- Dual Column Listing
- Dual Column Chart and Listing
- Side by Side Chart and Listing
- Chart and Listing
- Cross Tab

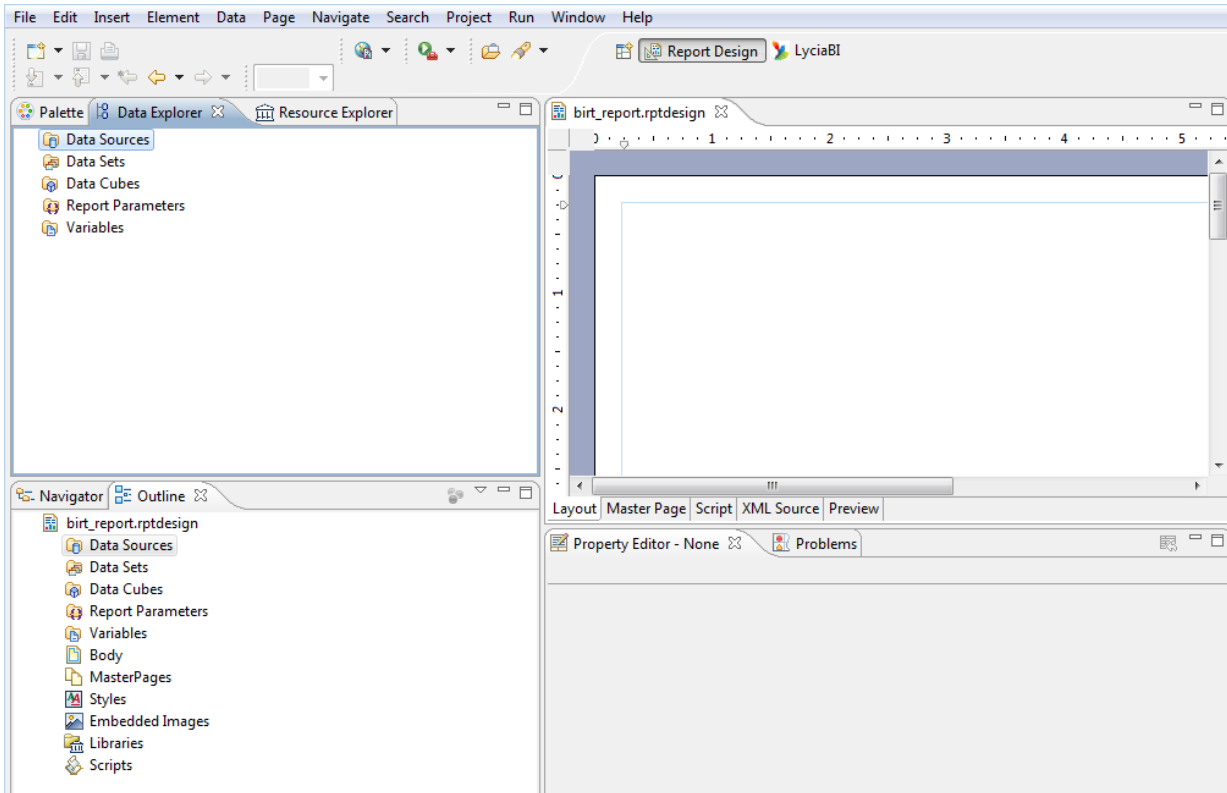
Clicking on each report template, you will see the preview of this template and its description.




Once you pick out an appropriate report templates, press **Finish**.



Now you can see the BIRT Report Perspective:



First of all we need to select the data source and data sets for the report.

	<p>You must create data sources and data sets for each Birt report file individually. However, they can be exported and then imported to facilitate the process.</p>
---	--

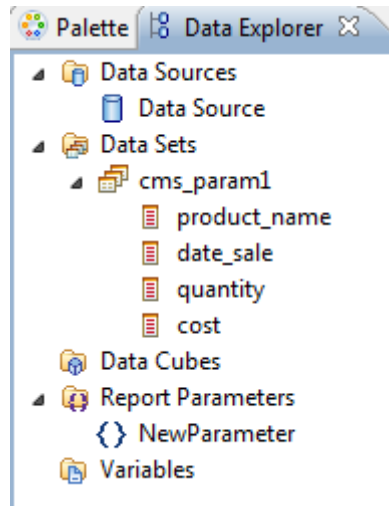
The Report Design Perspective

This perspective is used to create a BIRT report. It contains all the necessary tools needed for creating a report from beginning to end. The contents of the views linked to the perspective depends on the report file currently active in the editor area. The files are independent of one another, thus, for example, each one will have its own data sources and data sets, so the Data Explorer view will change each time you switch from file to file.



The Data Explorer view

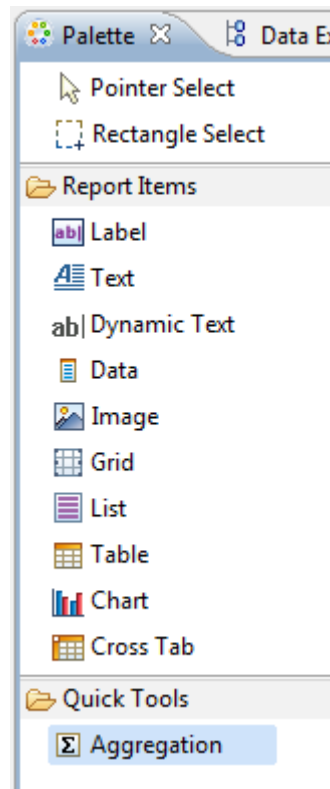
The Data Explorer view contains the references to the data sources and data sets, and to other sources of information used to create a report.



- The Data Sources element references a database, a csv file or other location from which the data for the report is retrieved. It is discussed later how to create a data source for a database and for a csv file.
- The Data Sets element contains a query which retrieves some set of data from the data source on the basis of which the report is actually built. A data set must reference a data source already defined for the file. The data source creation like a database query and a csv file subset are discussed later.
- The Data Cubes is a special subset of data which is based on the data of a data set and is used for Cross Tabs. A data cube consists of data groups and summary fields and is discussed later in this document.
- The Report Parameters element contains all the unknown parameters used in the data sets of a report. A parameter contains the information about the unknown parameter and a list of allowed parameters.
- The Variables element contains variables (for example a date variable which returns the current date) which can then be used in expressions of the report elements.

The Palette view

The Palette view contains all the elements you can add to a report to add an element drag it from the Palette view to the report page in the editor area.



- **Label** - adds a label element to the report. The label has vast formatting possibilities and is basically a formatted text field. It can be assigned a hyperlink, it can be hidden conditionally, etc. All the label properties are handled by the Property Editor view.

My Label

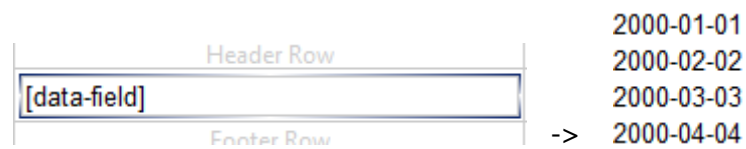
- **Text** - adds a text element to the report. The text element is represented by an area where text can be either non-formatted, or formatted using the HTML tags.

This is a text
This is a text
This is a text
This is a text

- **Dynamic Text** - adds a text which can contain variables, expressions and other changeable data.

new Date() -> Thu May 12 11:44:49 EEST 2011

- **Data** - adds a field which can contain a variable, an expression or a value retrieved from a data set. If it displays some data from the data set, it is typically used together with lists or tables, because otherwise you cannot control the information from which row from the query will be displayed. The usage of the Data element with the Table element is described later in this manual.



- Image** - adds an image to the report body. It can be an image located somewhere on the web, an embedded image, a dynamic image, etc. Images are discussed later in this document.



- Grid** - adds a column/row layout invisible to the user which helps to arrange different elements on the report page. Serves as the container for other elements, does not reference data sets directly.



- List** - adds a container for arranging data visually, like the grid it does not reference the data sets directly, but contains other elements which do. Unlike the grid it allows to group the data. Lists are discussed in detail later in this manual.

Header

Label

Detail

[product]

Footer

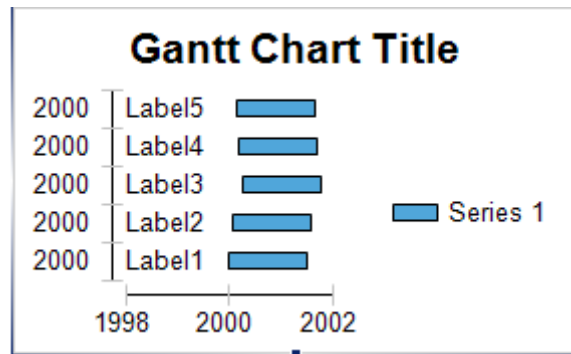
List

- Table** - adds a table which serves as a container for other elements, it also allows grouping and unlike the list has more than one column. Tables are discussed later in this manual.



	header		
	[cost]	[date]	[amount]
	Footer Row		

- Chart** - adds a graphical chart to the report. You can select among a number of different chart types. It has highly flexible creation interface which allows you to create complicated charts. The following chart types are described in detail later: bar chart, pie chart, meter chart.



- Cross Tab** - adds a cross tab to a report. It arranges the data so that to display aggregate data in table/column fashion. Cross Tabs are discussed later.

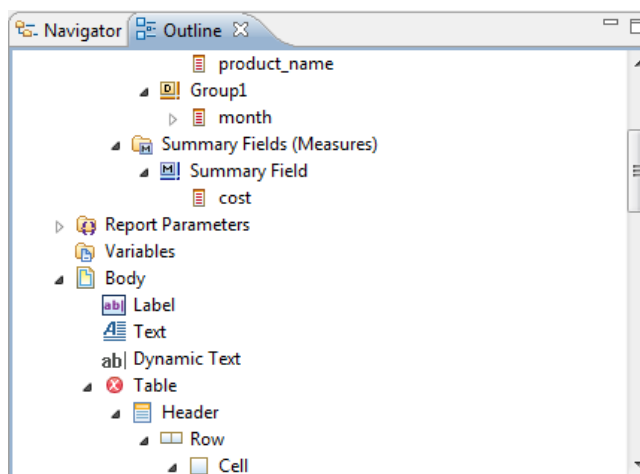
Label	[month]
	cost
[product_name]	Σ [cost_Group/produ...]

- Aggregation** - adds an aggregation based on a data set or on other data. It behaves similar to a data field, but allows you to use aggregate functions. It cannot be used independently, it can be used only within a table or a list. It can aggregate either on the whole table/list or on a group.

Σ [cost_Group/produ...]

The Outline view

The outline view displays all the elements present in the report page. The elements are gathered into groups. If you click on an element in the Outline view, it will be highlighted in the editor.



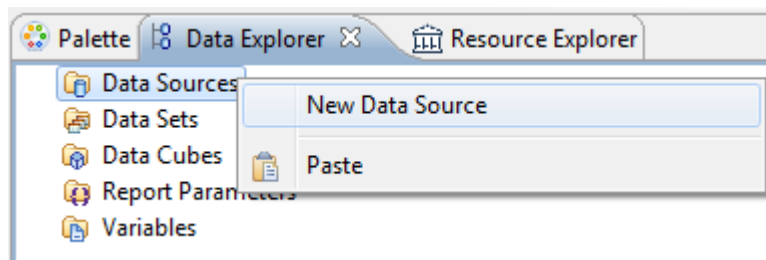
Database as a Data Source

BIRT can use either a database or a file with data (e.g. xml or csv file, etc.) as a data source. The section below explains how to create data sources and data sets which reference database tables.

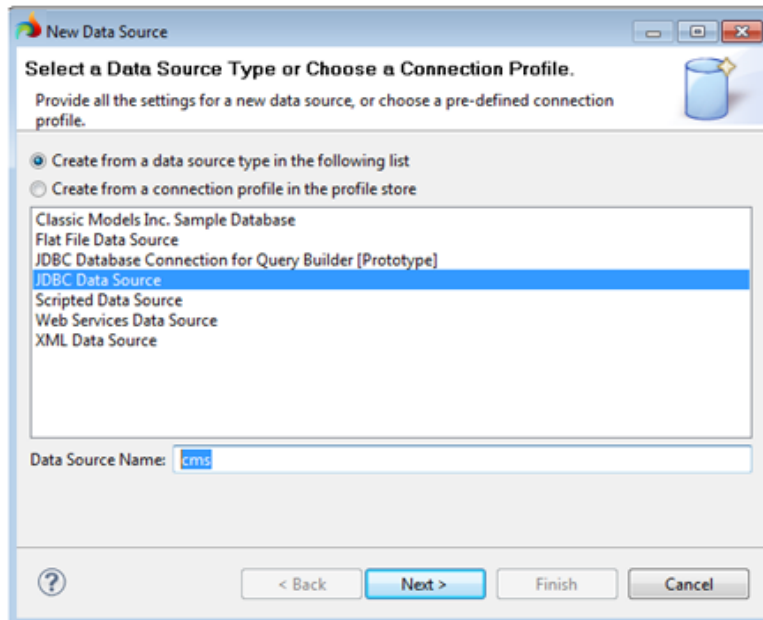
Setting the Data Source

To select the Data Source:

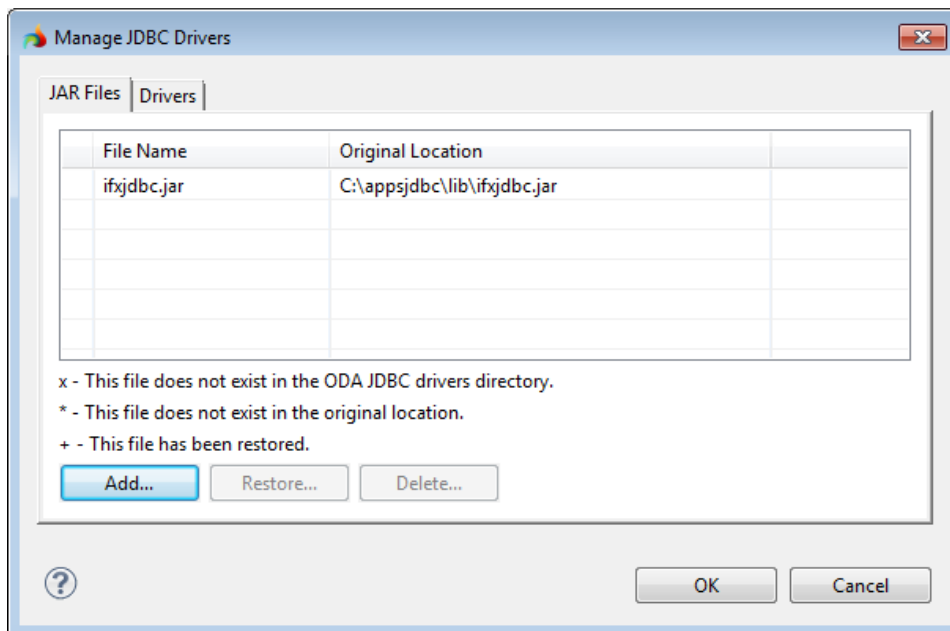
1. In the Data Explorer view right-click the Data Sources item and select **New Data Source**.



2. Select the JDBC Data Source option and enter the name of the data source below, then click **Next**:



3. In the next dialog you must first select a driver. For example, you may import an Informix driver.
 - a. If the driver of the database you need is not in the drop down list, click **Manage Drivers** button below.
 - b. In the JAR Files tab of the next dialog click **Add** button.
 - c. To set up the Informix driver, you need an Informix client installed on your machine. Select the Informix driver file called **ifxjdbc.jar**. Your Informix SDK may not include the driver. In this case you can download a jdbc driver from the IBM web site and install it.



- d. Click **OK**.
- e. Now you can select the installed driver from the list Driver Class.



- f. Enter the driver URL. It should look as follows *jdbc:informix-sqli://location:port-number/database_name:INFORMIXSERVER=server_name*.
- g. Enter the user name and the password for accessing the database.
- h. Leave JNDI URL field empty.

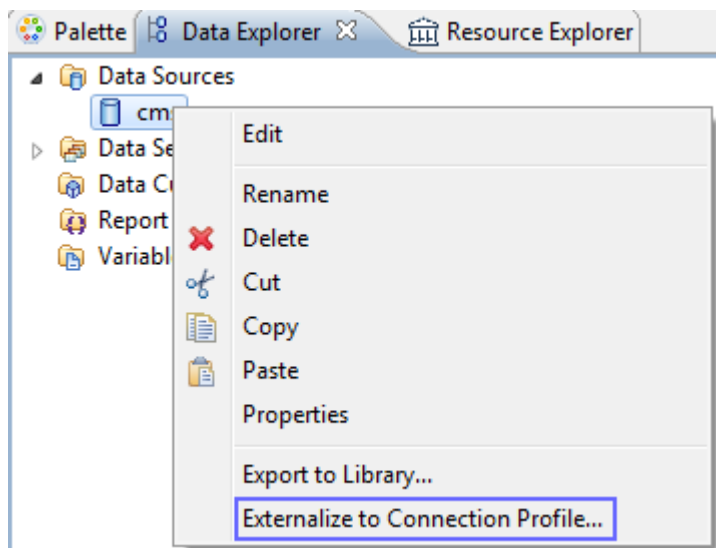
4. You can click **Test Connection** button to verify the connection settings. If the connection was successful, click **Finish**. The data source will appear in the Data Explorer.
5. You can edit the data source later.

Reusing a Data Source for other reports

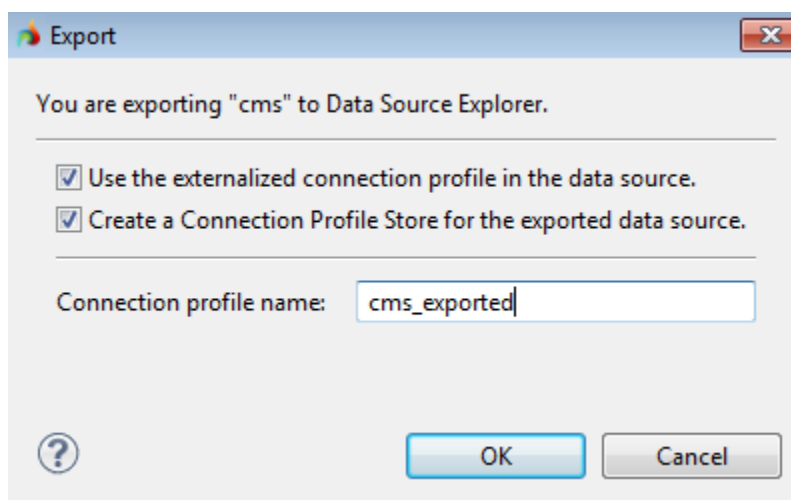
A Connection Profile allows you to save time and effort that might be spent on creating a new data source for each new Birt file. A connection profile stores the information about a data source in a file from which this data source can be set for any Birt file.

To create a connection profile:

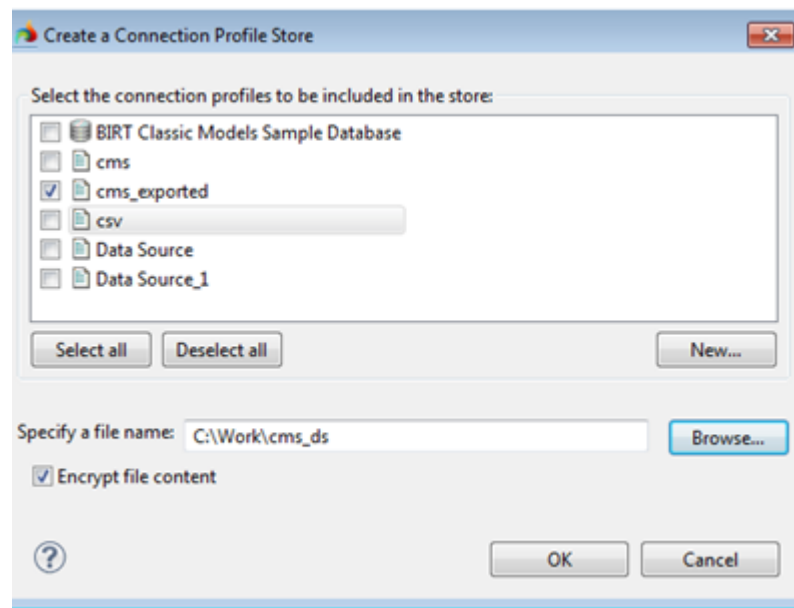
1. Open a BIRT file which already has a Data Source which you want to reuse for other reports.
2. In the Data Explorer view right-click this Data Source and select **Externalize to Connection Profile...** option from the context menu:



3. In the Export dialog check both check boxes, specify the name for the exported Data Source and click **OK**. If you check only the first option, the Data Source will be added to the list of externalized connection profiles, but no storage file for it will be created and the following steps will not take place.



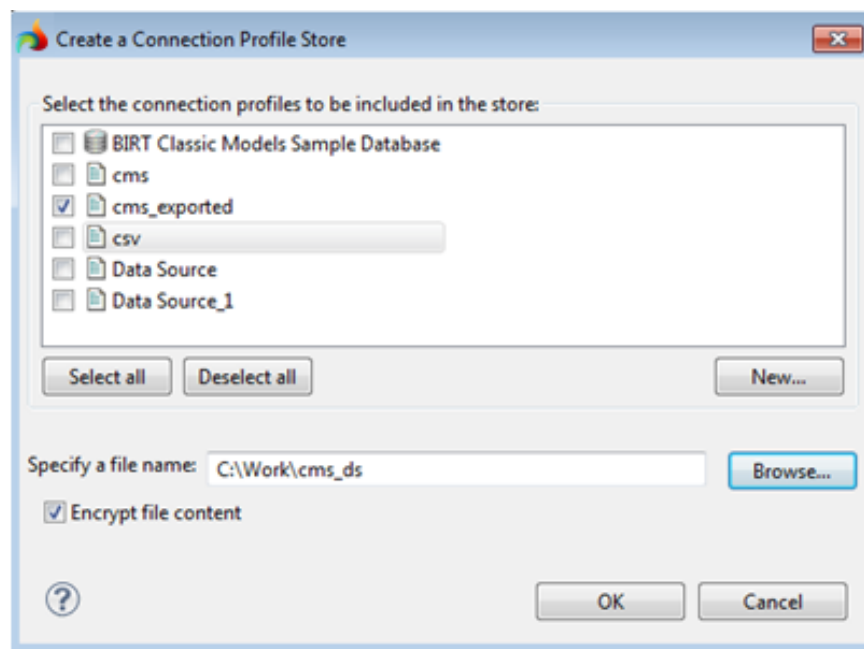
4. In the next dialog check the newly added connection profile from the list of stored profiles. Then click **Browse** to specify the name and location of the file to store the Data Source settings. Click **OK**.



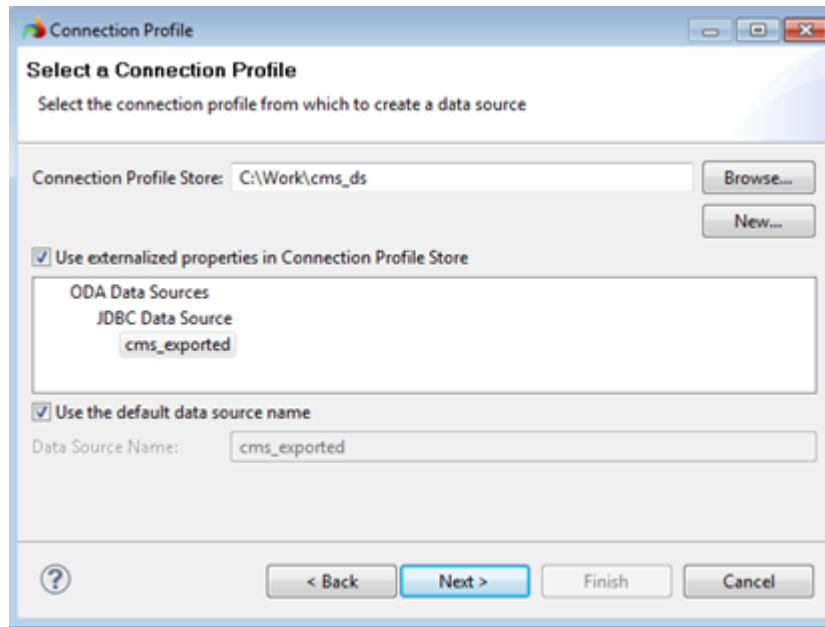
5. The connection profile file will be created in the specified directory.

To import a connection profile to another BIRT file, follow these steps:

1. Create a new BIRT file.
2. In the Data Explorer view right-click the Data Sources section and select **New Data Source** option.
3. In the New Data Source dialog select the option to create a data source from a connection profile and click **Next**.



4. In the next dialog click **Browse** and select the connection profile file created before.



5. Click **Next**. If it is a JDBC connection profile, you will see the connection details. Click **Finish** to import the Data Source. It will be added to the list of the data sources of the file.

Creating the Data Set

Unlike individual charts and dashboard templates the data sets for the Birt reports can be created directly within the LyciaStudio.

The following sections of this document will use the same table and the same data set based on the table. Basically, you can use any table you have at your disposal to create a data set. For the sake of convenience we suggest that you add to your database and use from now on the following demonstration table:

```
CREATE TABLE product
(
  product_name varchar(20),
  date_sale date,
  quantity int,
  cost money
)
```

You can populate the table with any values or load the following values into it:

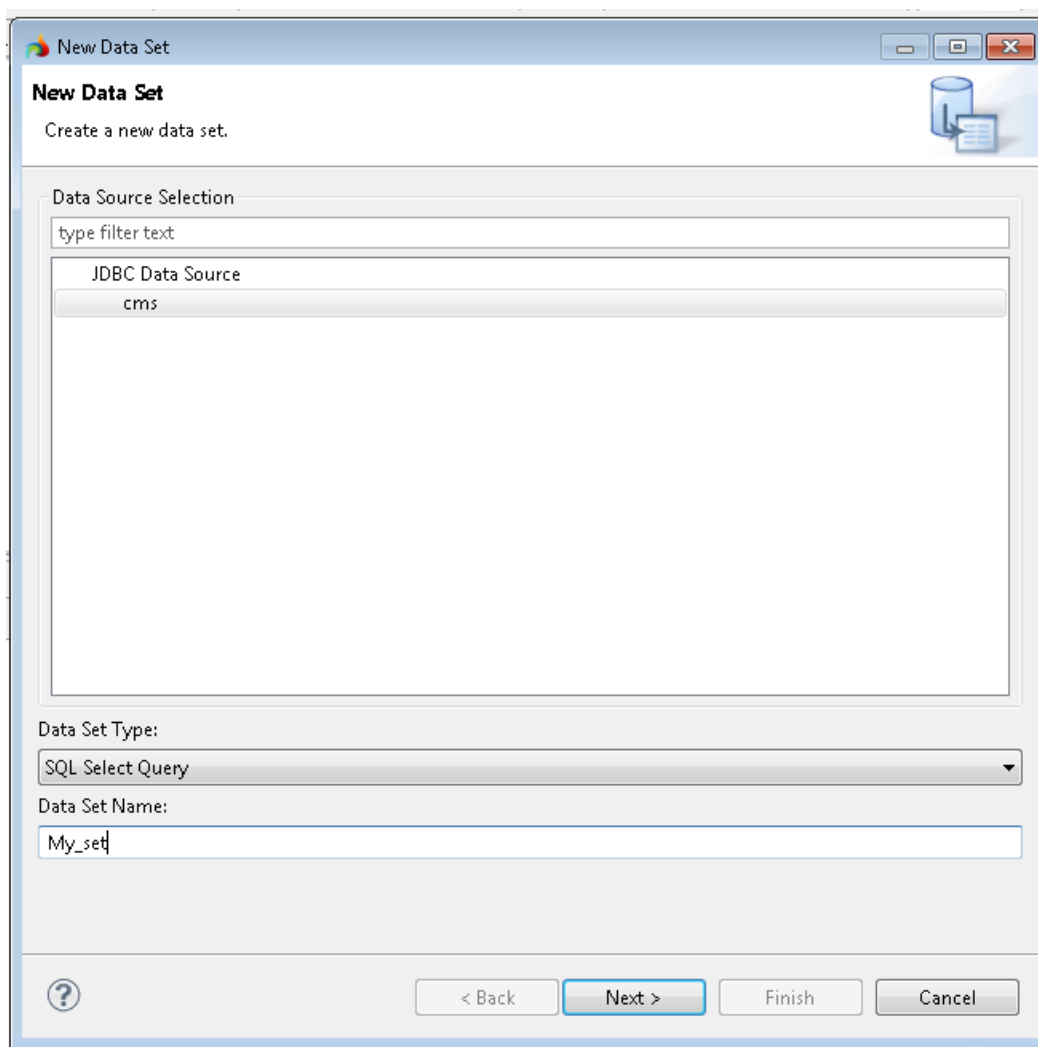
```
product_1|01/01/2000|105|53.54|
product_1|02/02/2000|87|64.00|
product_1|03/03/2000|123|49.70|
product_1|04/04/2000|45|60.87|
product_1|05/05/2000|99|34.46|
product_1|06/06/2000|130|35.79|
product_1|07/07/2000|145|40.56|
product_1|08/08/2000|120|55.78|
product_1|09/09/2000|60|90.67|
product_1|10/10/2000|71|92.45|
product_1|11/11/2000|35|100.40|
```



```
product_1|12/12/2000|20|120.00|  
product_2|01/01/2000|24|250.54|  
product_2|02/02/2000|13|345.00|  
product_2|03/03/2000|20|400.70|  
product_2|04/04/2000|9|390.87|  
product_2|05/05/2000|19|410.46|  
product_2|06/06/2000|15|287.79|  
product_2|07/07/2000|27|300.56|  
product_2|08/08/2000|21|326.78|  
product_2|09/09/2000|18|402.67|  
product_2|10/10/2000|10|378.45|  
product_2|11/11/2000|29|321.40|  
product_2|12/12/2000|24|279.00|
```

To create a data set on the above table:

1. In the Data Explorer view right-click the Data Sets item and select **New Data Set**.
2. Select the data source for which the data set is created, give it a name and click **Next**.

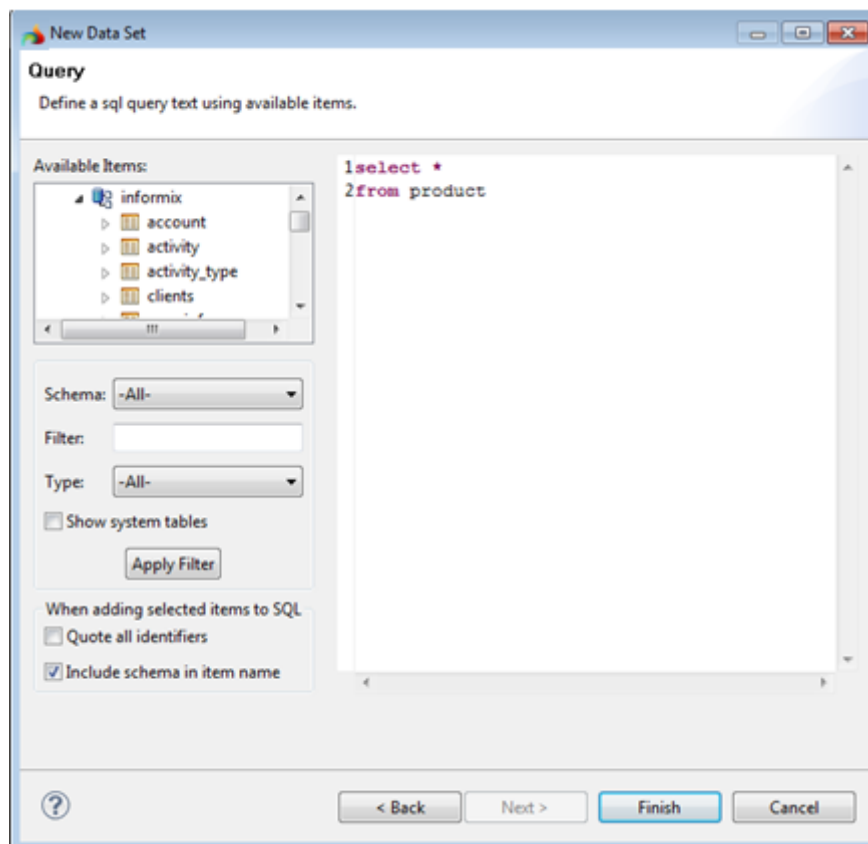


The image shows a 'New Data Set' dialog box with the following fields and controls:

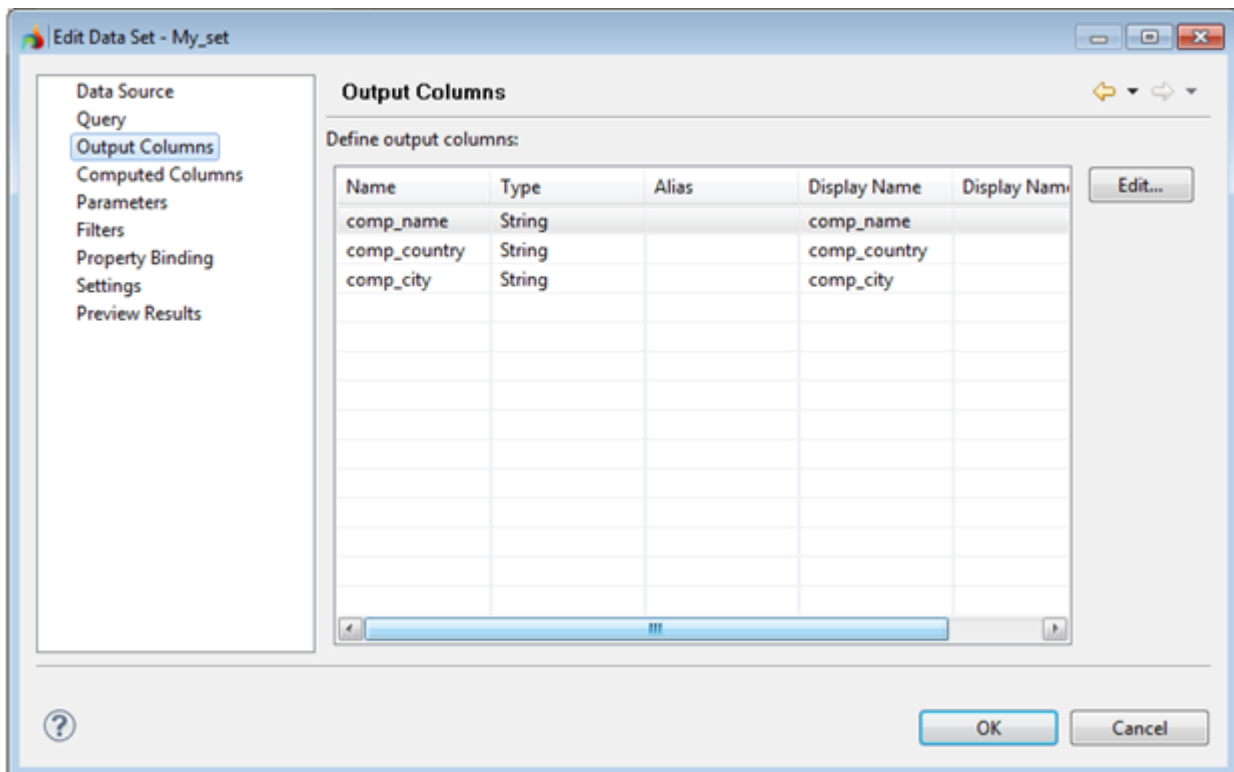
- Data Source Selection:** A search box labeled 'type filter text' and a list box containing 'JDBC Data Source' and 'cms'.
- Data Set Type:** A dropdown menu currently showing 'SQL Select Query'.
- Data Set Name:** A text input field containing 'My_set'.
- Navigation:** Buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted.
- Help:** A question mark icon in the bottom left corner.



3. You will see the database structure on the left and will be able to create the query text. To add an element (i.e. a column name or a table name) at the position of the cursor, double-click it in the list on the left. You can also type the text of the query manually:



4. Click Finish to save the data set. The edit dialog will open. Here you can change the display names of the columns and values, set parameters, etc.



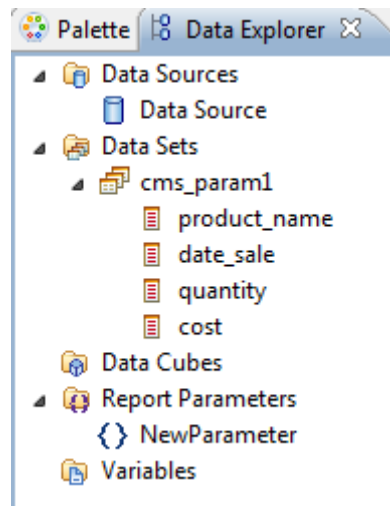
5. You can check whether the query actually returns some data by selecting the **Preview Results** from the list on the left.
6. After you click **OK**, the data source will appear in the Data Explorer.

The Report Design Perspective

This perspective is used to create a BIRT report. It contains all the necessary tools needed for creating a report from beginning to end. The contents of the views linked to the perspective depends on the report file currently active in the editor area. The files are independent of one another, thus, for example, each one will have its own data sources and data sets, so the Data Explorer view will change each time you switch from file to file.

The Data Explorer view

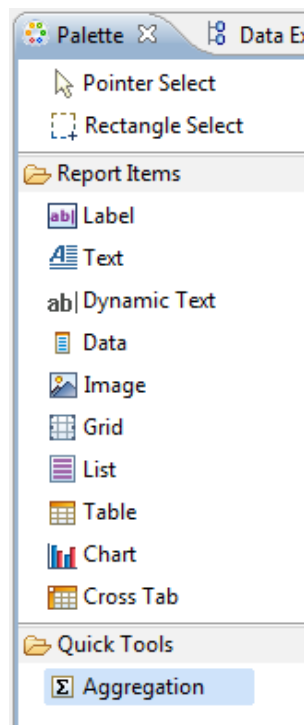
The Data Explorer view contains the references to the data sources and data sets, and to other sources of information used to create a report.



- The Data Sources element references a database, a csv file or other location from which the data for the report is retrieved. It is discussed later how to create a data source for a database and for a csv file.
- The Data Sets element contains a query which retrieves some set of data from the data source on the basis of which the report is actually built. A data set must reference a data source already defined for the file. The data source creation like a database query and a csv file subset are discussed later.
- The Data Cubes is a special subset of data which is based on the data of a data set and is used for Cross Tabs. A data cube consists of data groups and summary fields and is discussed later in this document.
- The Report Parameters element contains all the unknown parameters used in the data sets of a report. A parameter contains the information about the unknown parameter and a list of allowed parameters. They are discussed later together with the unknown parameters queries.
- The Variables element contains variables (for example a date variable which returns the current date) which can then be used in expressions of the report elements.

The Palette view

The Palette view contains all the elements you can add to a report to add an element drag it from the Palette view to the report page in the editor area.



- **Label** - adds a label element to the report. The label has vast formatting possibilities and is basically a formatted text field. It can be assigned a hyperlink, it can be hidden conditionally, etc. All the label properties are handled by the Property Editor view.

My Label

- **Text** - adds a text element to the report. The text element is represented by an area where text can be either non-formatted, or formatted using the HTML tags.

This is a text
This is a text
This is a text
This is a text


- **Dynamic Text** - adds a text which can contain variables, expressions and other changeable data.

`new Date()` -> Thu May 12 11:44:49 EEST 2011


- **Data** - adds a field which can contain a variable, an expression or a value retrieved from a data set. If it displays some data from the data set, it is typically used together with lists or tables, because otherwise you cannot control the information from which row from the query will be displayed. The usage of the Data element with the Table element is described later in this manual.

	Header Row	2000-01-01
		2000-02-02
[data-field]		2000-03-03
	Footer Row	-> 2000-04-04




-  **Image** - adds an image to the report body. It can be an image located somewhere on the web, an embedded image, a dynamic image, etc. Images are discussed later in this document.



-  **Grid** - adds a column/row layout invisible to the user which helps to arrange different elements on the report page. Serves as the container for other elements, does not reference data sets directly.

-  **List** - adds a container for arranging data visually, like the grid it does not reference the data sets directly, but contains other elements which do. Unlike the grid it allows to group the data. Lists are discussed in detail later in this manual.

Header


Label

Detail

[product]

Footer

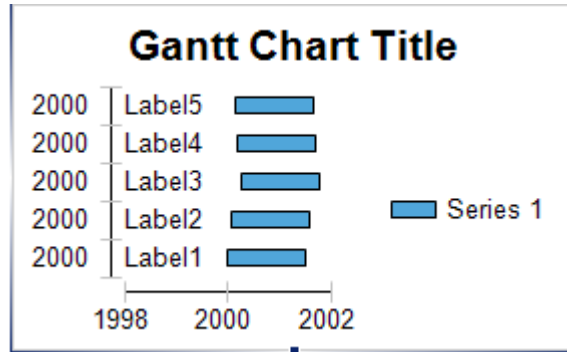
List

-  **Table** - adds a table which serves as a container for other elements, it also allows grouping and unlike the list has more than one column. [Tables](#) are discussed later in this manual.

	header		
	[cost]	[date]	[amount]
	Footer Row		



- **Chart** - adds a graphical chart to the report. You can select among a number of different chart types. It has highly flexible creation interface which allows you to create complicated charts. The following chart types are described in detail later: [bar chart](#), [pie chart](#), [meter chart](#).



- **Cross Tab** - adds a cross tab to a report. It arranges the data so that to display aggregate data in table/column fashion. [Cross Tabs](#) are discussed later.

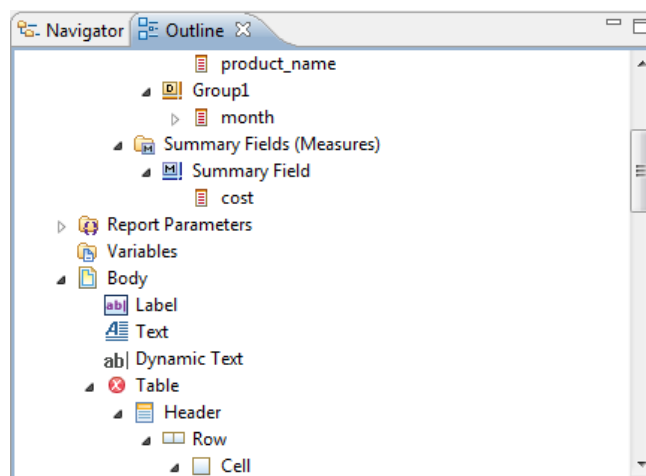
Label	[month]
	cost
[product_name]	Σ[cost_Group/produ...]

- **Aggregation** - adds an aggregation based on a data set or on other data. It behaves similar to a data field, but allows you to use aggregate functions. It cannot be used independently, it can be used only within a table or a list. It can aggregate either on the whole table/list or on a group.

Σ[cost_Group/produ...]

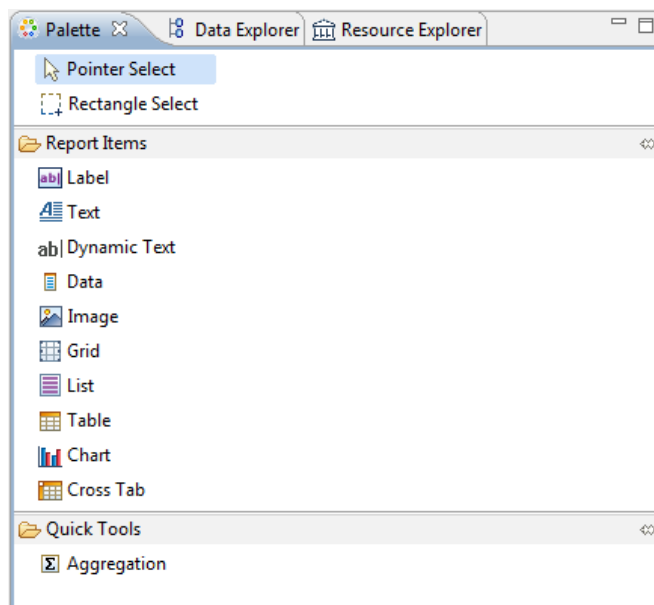
The Outline view

The outline view displays all the elements present in the report page. The elements are gathered into groups. If you click on an element in the Outline view, it will be highlighted in the editor.

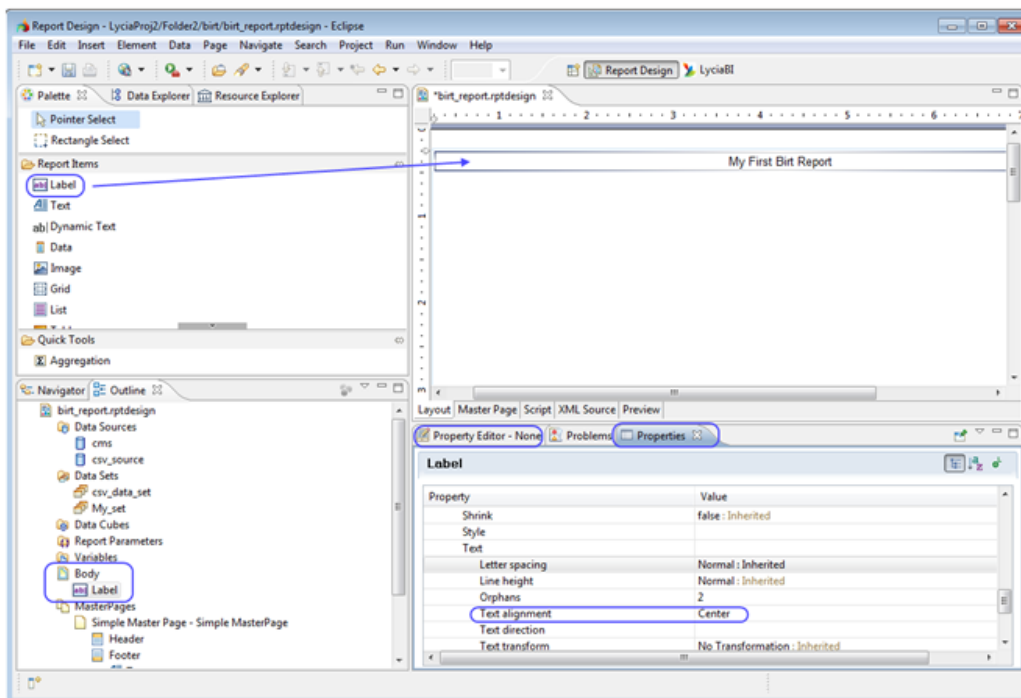


Composing a report


To populate the report page with graphs and data use the Palette view:

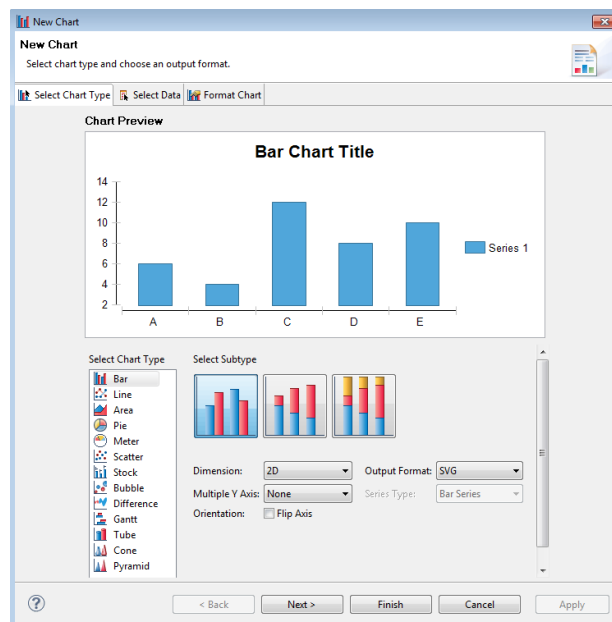


1. Create a label at the top of the report page by dragging it from the Palette View. Type the label text in the rectangular area. You can adjust the appearance and other properties of the label using the Properties view and Property Editor view:



2. The label will appear in the Outline view (Body section).

3. Now let's add a chart. Drag the chart  from the Palette to the report body. Select the Bar chart type and click **Next**.



4. Select the data set you've created for the csv data source:



Select Data

☐ Inherit Data from Container Inherit Columns only

☒ Use Data from prices

5. Now you should set X, Y and Y grouping. To set them you can click buttons next to them or type values manually. Y series specify which column will be used for the Y axis, X group specifies the column to be used for the X axis. Y grouping specifies the column to be used for the graph legend and thus different colouring of the different values if the group.

6.

New Chart

Select the data to display in the chart and bind it to the series.

Select Chart Type **Select Data** **Format Chart**

Chart Preview

Bar Chart Title

Value (Y) Series:*
Series 1
Σ row["Highest"]

Optional Y Series Grouping:
row["Product"]

Category (X) Series:* row["Product"]

Select Data

☐ Inherit Data from Container Inherit Columns only

☒ Use Data from prices

Data Preview

Use the right-click menu or drag the column into series fields.

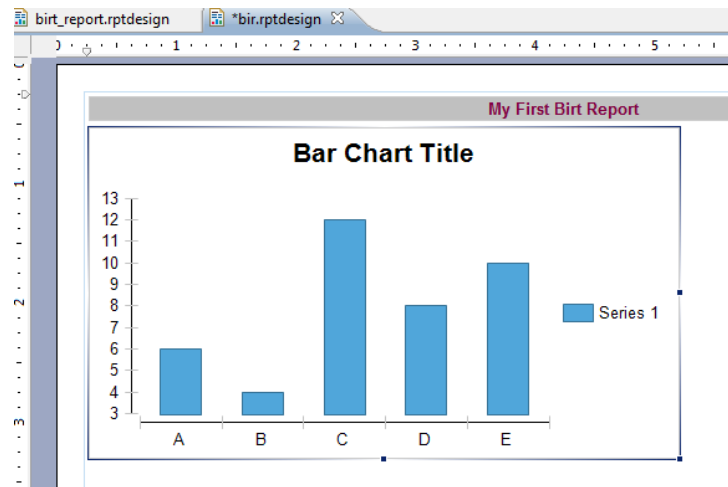
☒ Show data preview

Average	Highest	Lowest	Product
85	100	53	product1
112	123	78	product2
70	122	65	product3
78	98	54	product4

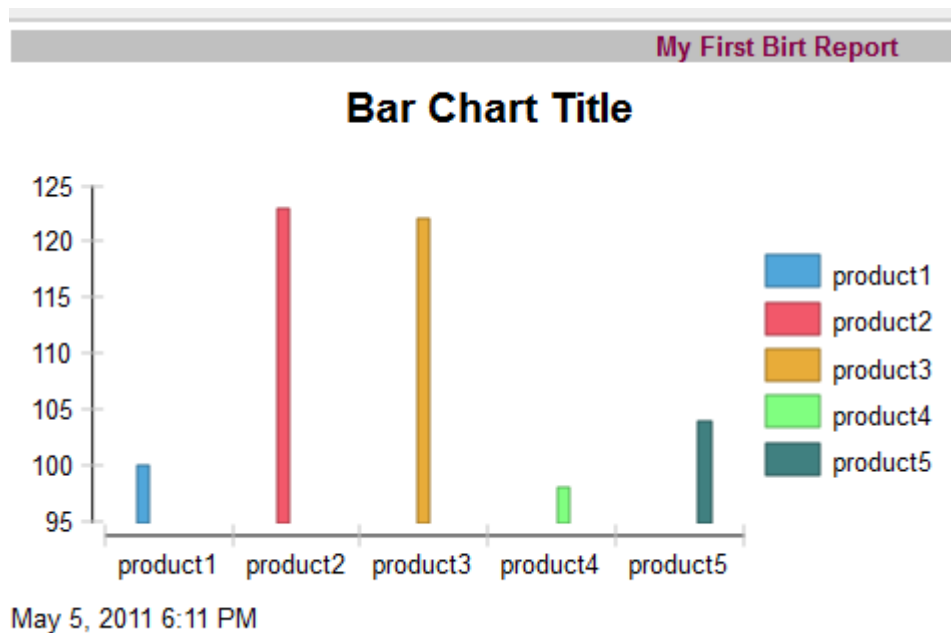
Filters...
Parameters...
Data Binding...

Finish **Cancel** **Apply**

7. Click **Finish**. You can resize the chart on the document.



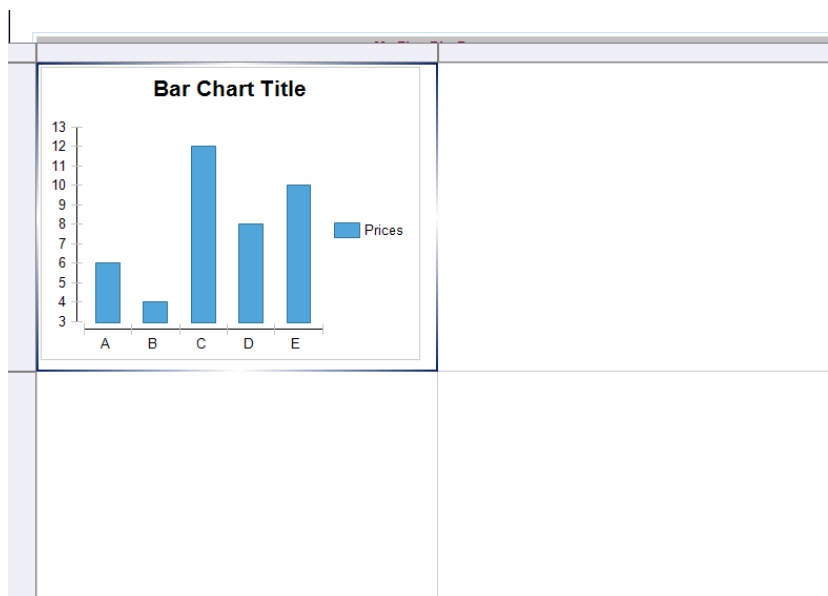
8. Now you can deploy the report to the server where it will look as follows:



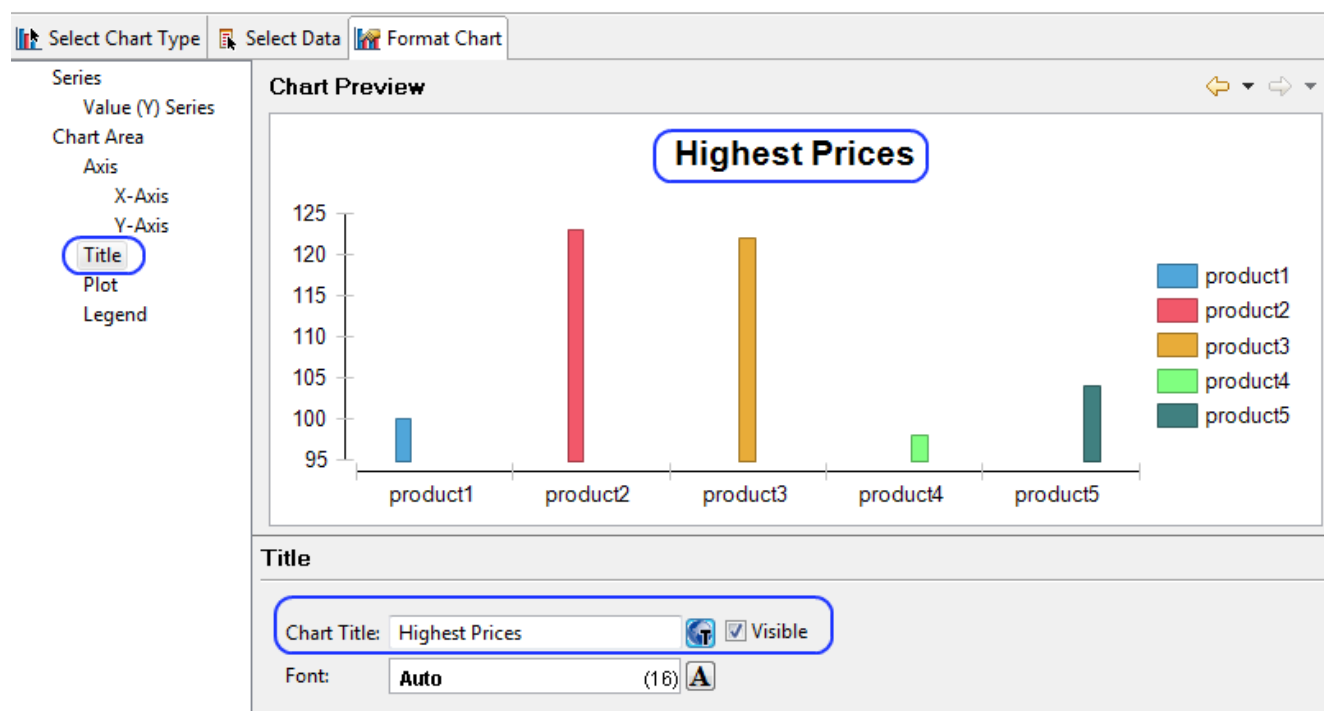
a complex bar chart

The above chart features only the highest prices for each produce. It is possible to have a separate column for all three prices for a product in one chart or have several different charts on one page. We will show how you can include a number of charts into one report page both complex and simple and how to arrange them.

1. From the Palette view add a Grid item Grid to the report body which has 2 rows and 2 columns and move the existing chart into the top left corner.



2. Change the name of the Bar chart. To do so right click it and select **Format Chart** option. Select the Title option from the list on the left and enter the new title.



3. Place another chart in the top right corner of the grid. Do not forget to select the working set in the field 'Use Data From' first. This time set ' row["Lowest"]' in the Value(Y) Series field in the Select Data tab. The rest of the fields should be the same as for the first chart.

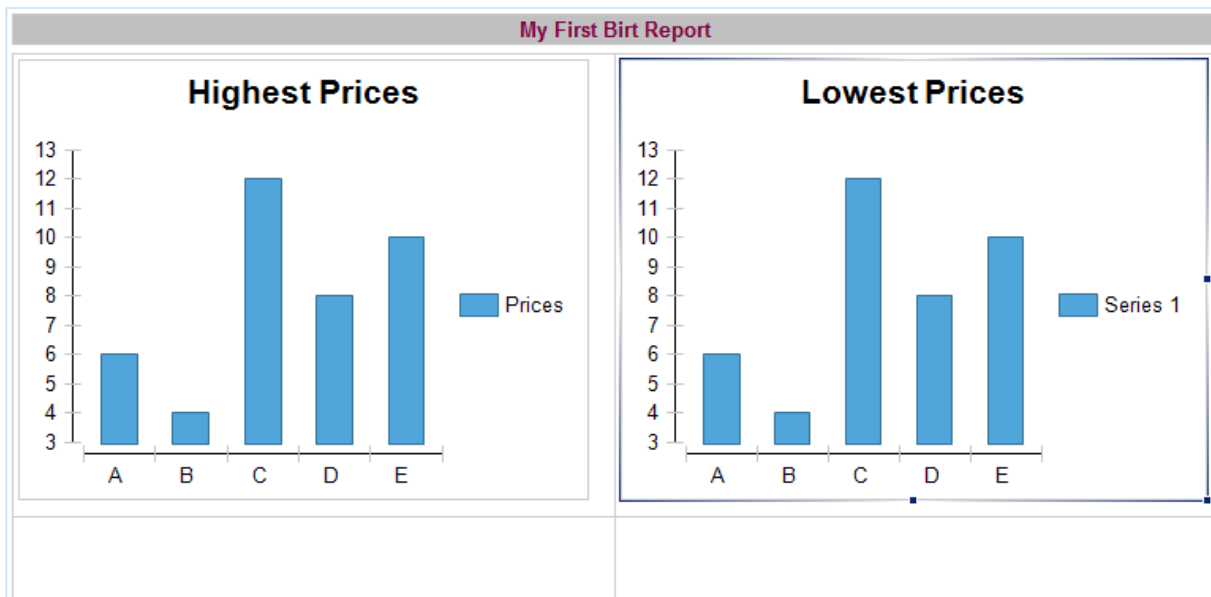


Value (Y) Series:*
Series 1
Σ row["Lowest"] f_x

Optional Y Series Grouping:
row["Product"] f_x

Category (X) Series:* row["Product"] f_x

4. In the Format Chart tab set the chart title to "Lowest Prices" and click **Finish**.
5. Now you will have two charts in the report body.
- 6.



7. Add another chart into the bottom left corner, call it 'Average Prices'. Select the same data set and set 'row["Average"]' in the Value(Y) Series field in the Select Data tab. Now you will have three charts.
8. To create a complex chart containing all types of prices add another chart of the same type to the bottom right corner. Call this chart "Prices Overview"
9. In the Select Data tab select the data set.
10. Category X and Optional Y fields should contain 'row["Product"]' value.
11. Now we will work mainly with the Values(Y) series field:
 - a. First set it as 'row["Highest"]'

Value (Y) Series:*
Series 1
Σ row["Highest"] f_x

- b. Then from the combo box select 'New Series':

Value (Y) Series:*
Series 1
Series 1
<New Series...>

- c. Now for Series 2 enter 'row["Lowest"]'



Value (Y) Series:*

Series 2

Σ row["Lowest"] f_x

- d. Create another Series in the same way and enter 'row["Average for it"]'

Value (Y) Series:*

Series 3

Σ row["Average"] f_x

- e. You will see now that your chart will have 3 columns for each product indicating different prices.
- f. To adjust the appearance of the chart go to Format Chart tab.
- g. Select the Series option from the list on the left and change the names for more appropriate ones. You can also check the 'Stacked' option to make the visual groups more distinct:

Select Chart Type

Select Data

Format Chart

Series

Value (Y) Series - 1

Value (Y) Series - 2

Value (Y) Series - 3

Chart Area

Axis

X-Axis

Y-Axis

Title

Plot

Legend

Chart Preview

Prices

Series

Color By: Value Series

Series	Title	Type	Z Order	Visible	Stacked	Translucent
Value (Y) Series - 1	Highest	Bar Series	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Value (Y) Series - 2	Lowest	Bar Series	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Value (Y) Series - 3	Average	Bar Series	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Series Palette

?

< Back

Next >

Finish

Cancel

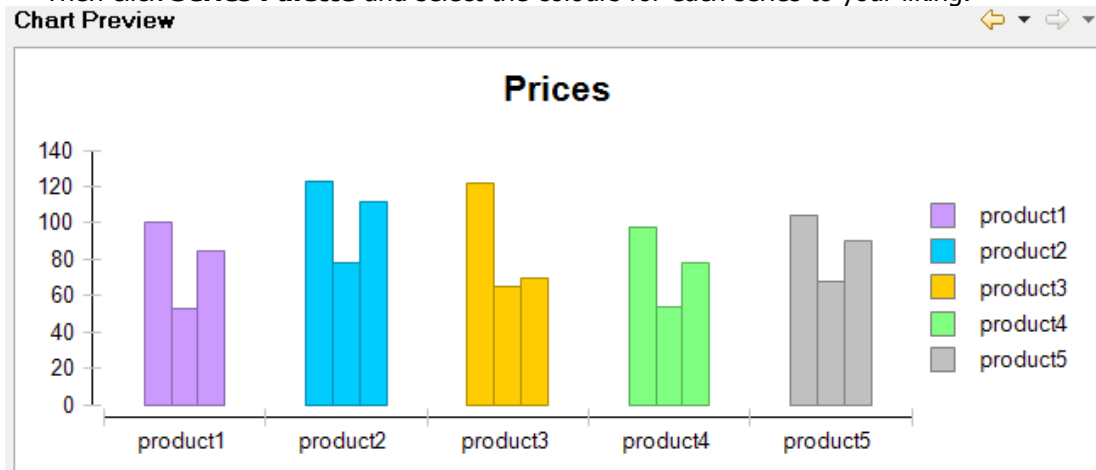
Apply



- h. Now we need the colouring that will reflect the product groups more distinctly. Select 'Categories' in the Colour by field.

Color By: Categories

Then click **Series Palette** and select the colours for each series to your liking.



- i. Now to make the scaling more understandable we will name our X and Y axes. IN the same tab select Y Axis from the list on the left. check the 'Visible' check box and enter the name for the axis:

Select Chart Type | Select Data | **Format Chart**

Series
Value (Y) Series - 1
Value (Y) Series - 2
Value (Y) Series - 3

Chart Area
Axis
X-Axis
Y-Axis
Title
Plot
Legend

Chart Preview

Y-Axis

Title: Prices (\$) ☒ Visible

Type: Linear

Origin: Min

Value: 0

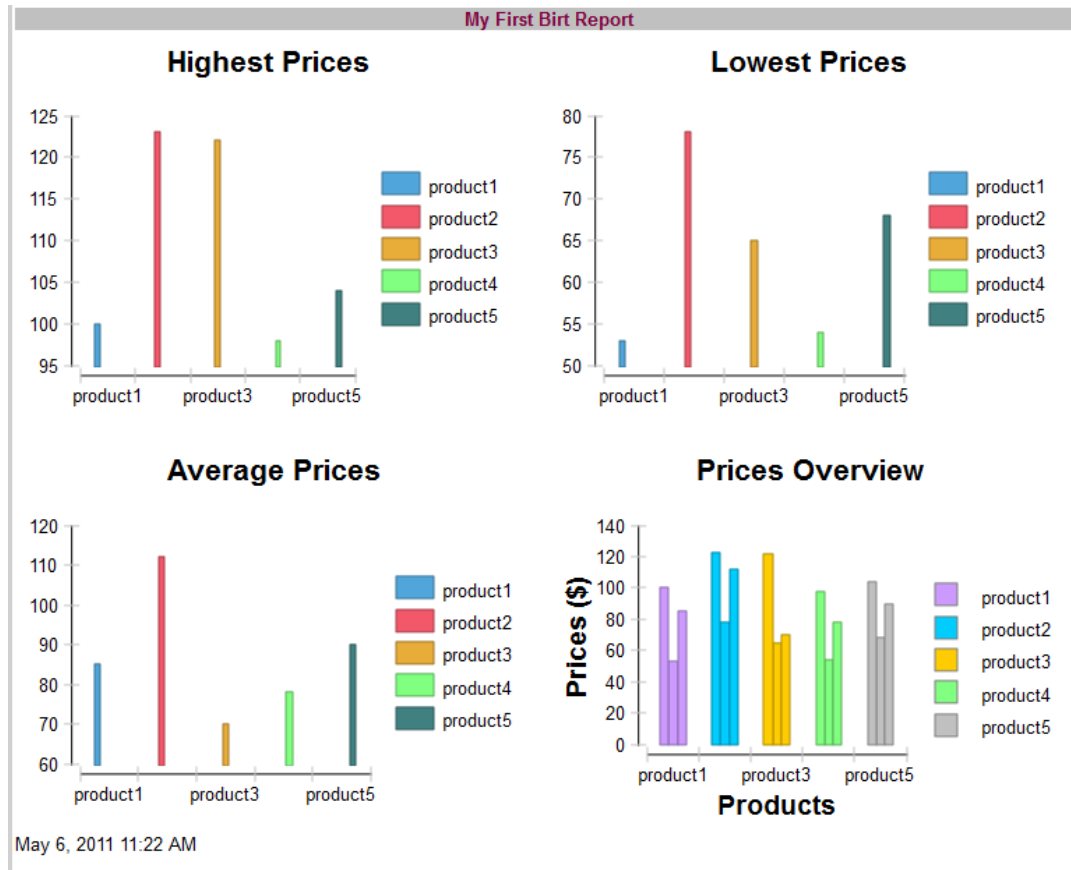
Labels: Auto (Auto) ☒ Visible ☐ Stagger

Label Span:* 0 (Points) ☐ Fixed



- j. Do the same for the X axis.
- k. Now click **Finish**.

12. You will have 4 different graphs on one report page. Save the file and deploy it to the server. After it is deployed, it will look as follows:



Creating Other types of charts


Birt offers a number of different chart types, this section will dwell upon some of them.

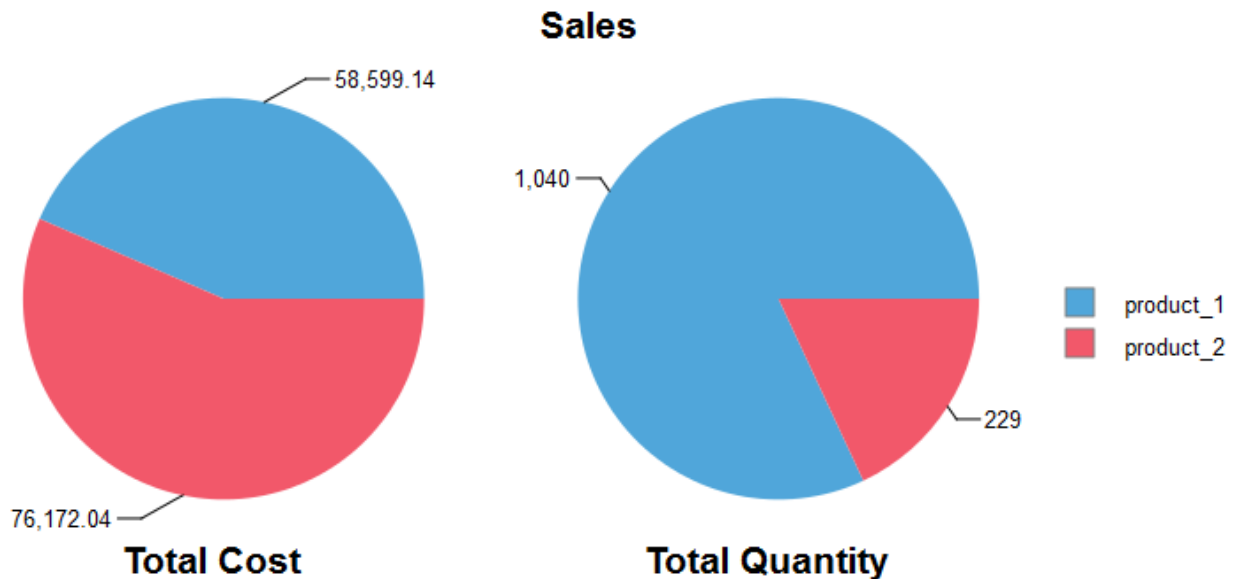
Pie Chart

We have a data set which selects all the rows from a table which contains the product name, the date of sale performed, the quantity of goods sold, and the price of goods. We want to create a graph showing the dynamics of the sales over the year. We will use the data set described earlier in this document.

1. Create a BIRT report.
2. Create a data source for it which references a database



3. Create a data set (i.e. select * from product).
4. Add a Pie type chart  to the report body and select the dataset for it.
5. In the Select Data tab:
 - a. For the Slice Size Definition field enter 'row["cost"]*row["quantity"]' - you may need to change the names of the columns used here, if the names of your columns differ. Basically the size of the slice of the pie chart will be based on the total cost of all the sold goods for each product.
 - b. For the Category Definition field enter 'row["product_name"]' - replace the column name for the one you use, if needed.
6. Add new Series to the chart.
7. For the Slice Size Definition field of the new series enter 'row["quantity"]' - this chart will display the amount of items sold for each product.
8. Name the chart and the series accordingly.
9. Save the file and deploy it to the server:

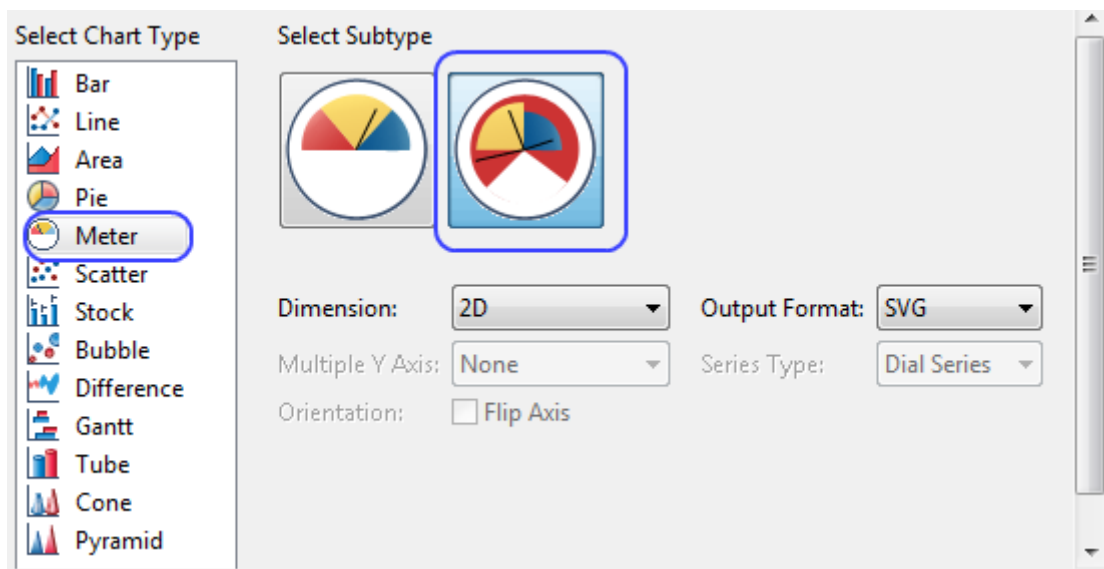


May 6, 2011 1:44 PM

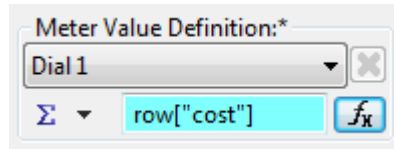
Meter Chart

To illustrate the Meter chart we will use the same query from the same table as for the Pie chart above. The Meter chart will display maximum, minimum and average prices per product.

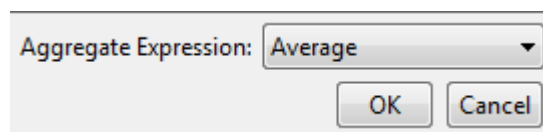
1. Add a Meter chart of the second subtype which allows to have several dials combined together:



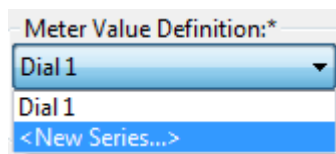
2. In the Category Definition field select the product name column.
3. In the Optional Grouping field also select the product name column.
4. In the Meter Value Definition select the cost value for the first Dial:



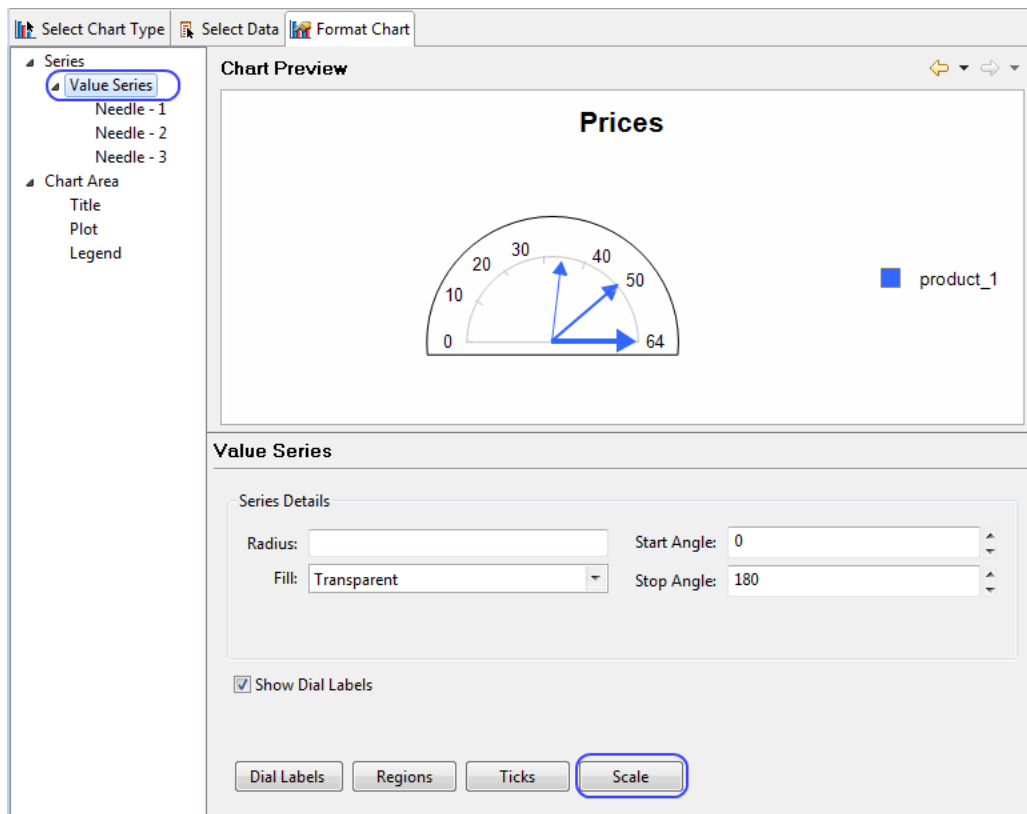
5. Now click Σ button Σ under the Dial to select the aggregate function. By default the Sum aggregate function is used. It sums all the row values for the selected column. Select Average aggregate function from the list to display the average cost value per product.



6. Create another Dial:



7. For the new Dial select the cost column and the **Maximum** aggregate function.
8. Create one more Dial. Select the cost column and the **Minimum** aggregate function.
9. To make the graph more clear you can change the colours and types of arrows and make other adjustments in the Format Chart tab.
10. You can define the scale of the dial using the Format Chart Tab, **Value Series** option, **Scale** button.



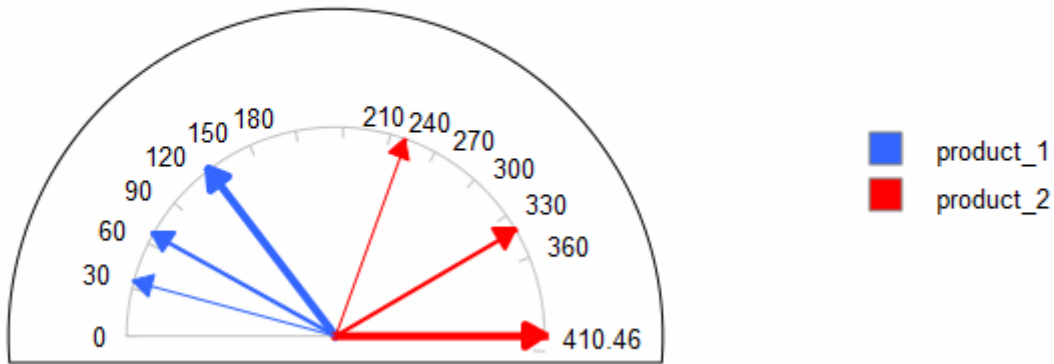
11. You should set the step for the scale. Otherwise it will display only the initial and the rightmost value:

12. Select 'Categories' in the Colour by field in the Format Chart Tab.



Color By: Categories

13. Save the file and deploy it to the server:



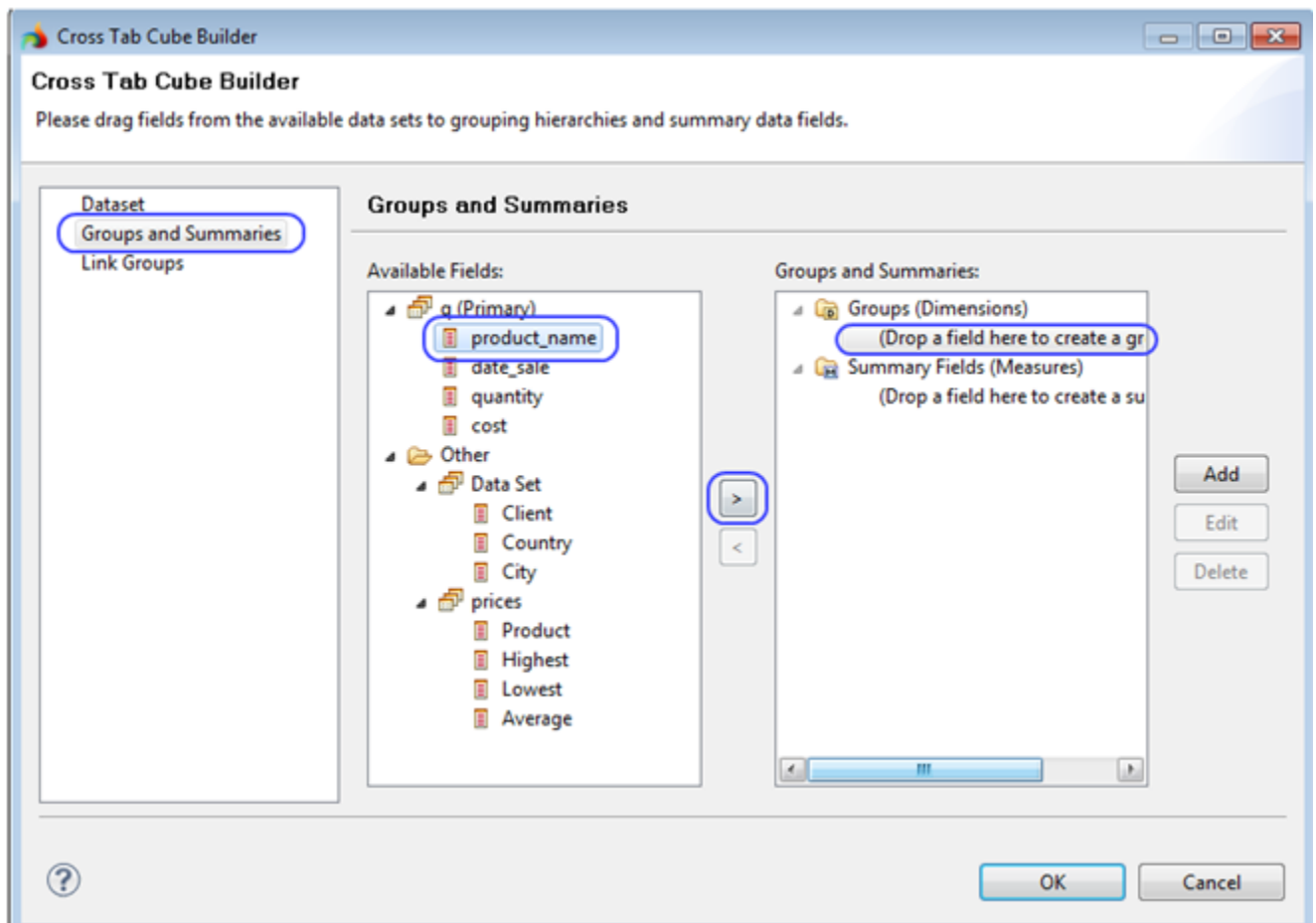
Data Cubes and Cross Tabs

A Cross Tab is another element which can be added to a report. A Cross Tab displays aggregate data in column and row format. To create a Cross Tab:

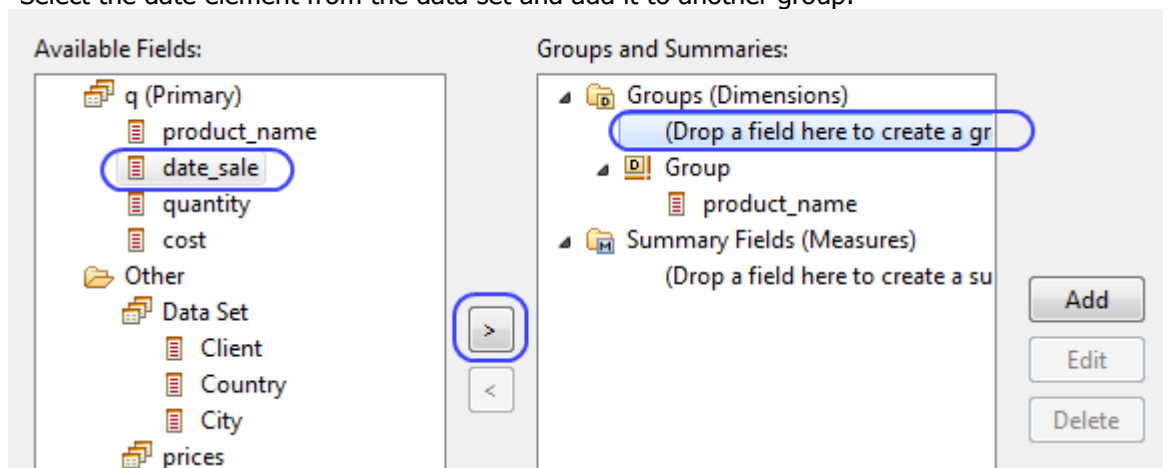
1. Drag and drop the Cross Tab item from the Palette to the report body. You will see an empty grid:

	Drop data field(s) to define columns here
Drop data field(s) to define rows here	Drop data field(s) to be summarized here

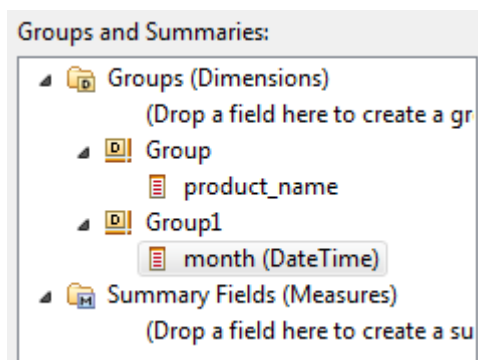
2. Now to populate a Cross Tab we need A Cube Set. To create a cube set, in the Data Explorer right-click Data Cubes element **Data Cubes** and select **New Data Cube** option.
3. In the New Data Cube dialog type the name for the cube and select the primary data set. It can be the same data set that was used for the previous examples.
4. Go to the Groups and Summaries page. Select the product name from the data set on the left and the Groups section on the right. Then use the arrow button to add the data set element to a group.



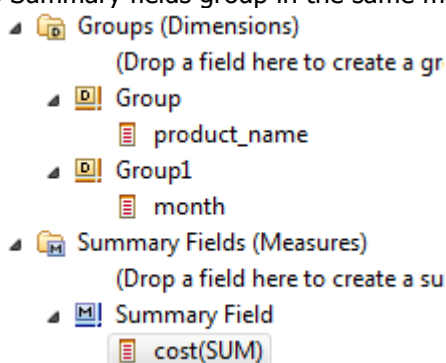
5. Select the date element from the data set and add it to another group:



6. You will be presented with the dialog there you can select the date value to be used. Select Month and click **OK**:



7. Then select cost and add it to Summary fields group in the same manner:



8. Click **OK**. The data cube will appear in the Data Explorer.
9. Unfold it. Select the product name cube element and drag it into the bottom left cell of the Cross Tab.
10. Drag the date cube element to the top right cell.
11. Drag the cost element to the bottom right cell.



12. As a result you will see the following. You can click Preview button to see how the Cross Tab will look like:

[month]

cost

[product_name]

Σ[cost_Group/produ...]

[out](#)
[Master Page](#)
[Script](#)
[XML Source](#)
[Preview](#)

	1	2	3	4	5	6	7	8	9	10	11	12
	cost	cost	cost	cost	cost	cost	cost	cost	cost	cost	cost	cost
product	53.54	64	49.7	60.87	34.46	35.79	40.56	55.78	90.67	92.45	100.4	120
product	250.54	345	400.7	390.87	410.46	287.79	300.56	326.78	402.67	378.45	321.4	279

Lists

A list object does not contain any data by itself. Like the Grid it serves as a container for some information. Unlike the Grid, a List arranges the information vertically in groups assigning a header and a footer and a description to all the groups included. The steps below will illustrate how you can create a list with several groups:

1. Create a BIRT report document. The data source and data set can be the same as used for the Pie chart explained above.
2. Add the List item **List** from the Palette to the report body in the editor area. You will see the empty list structure.

Header

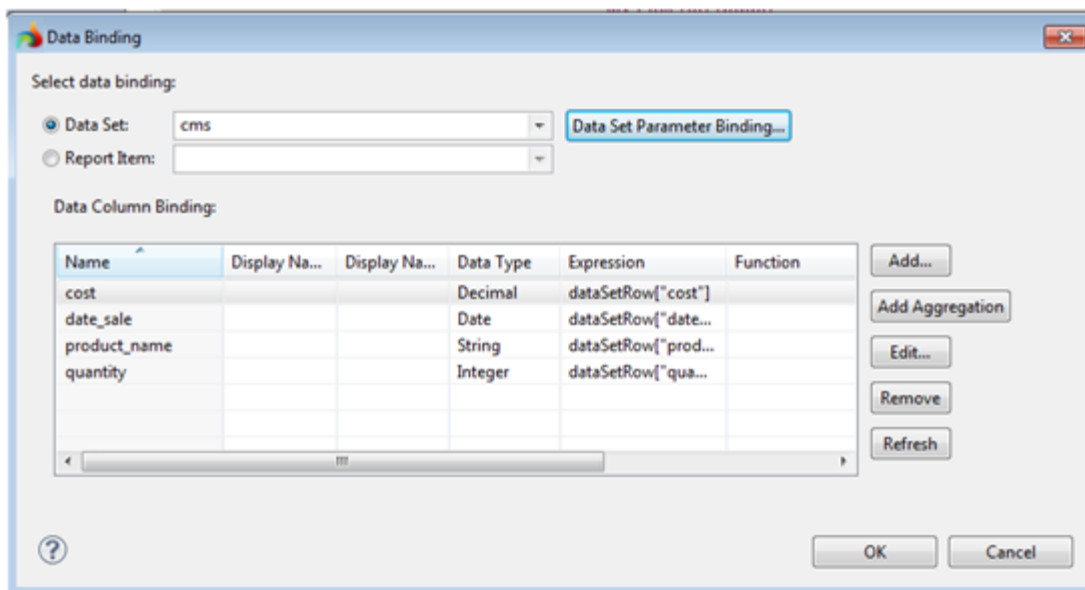
Detail

Footer

3. Now we need to bind the List with a data set. Right-click anywhere on the List and select **Edit Data Binding...** from the context menu.



4. Select the Working set from the list. The table below will display the columns available in the data set. Click **OK**.

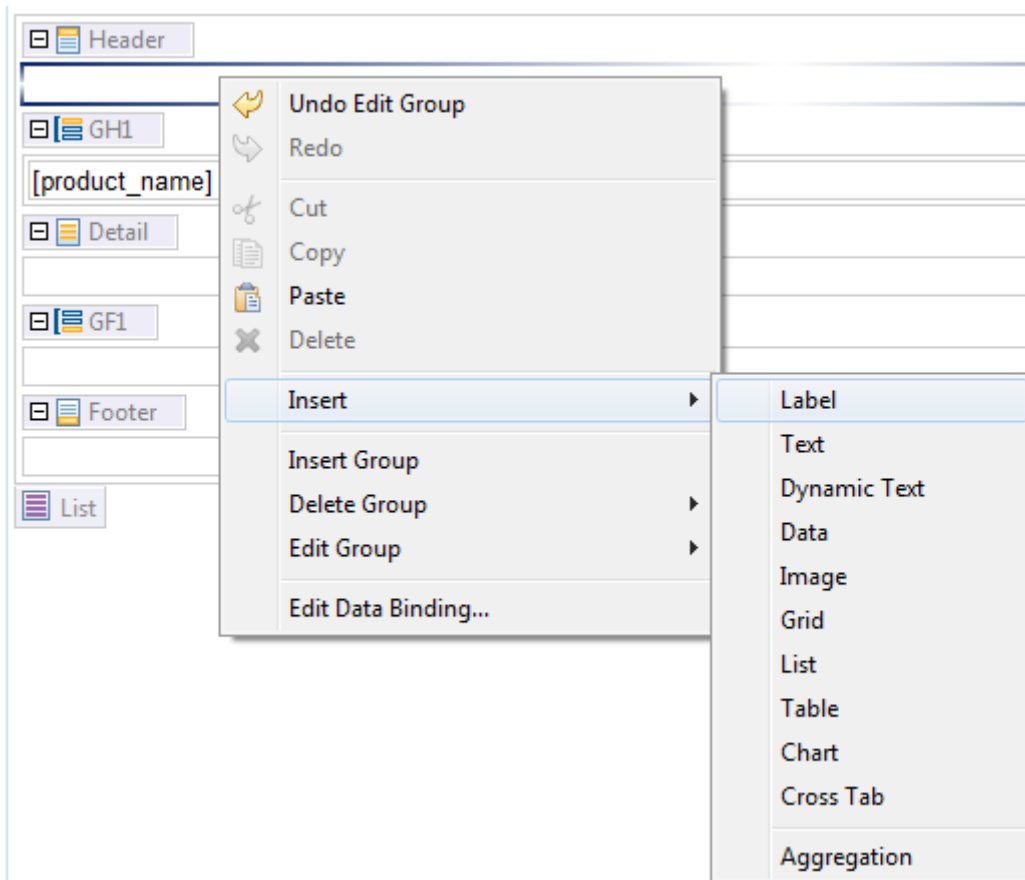


5. Now the List is bound to a data set.
6. Now you need to add a group. Right-click the List area and select **Insert Group** option from the context menu.
7. In the New Group dialog enter the name for the group and select the column by which the items will be grouped. Click **OK**.

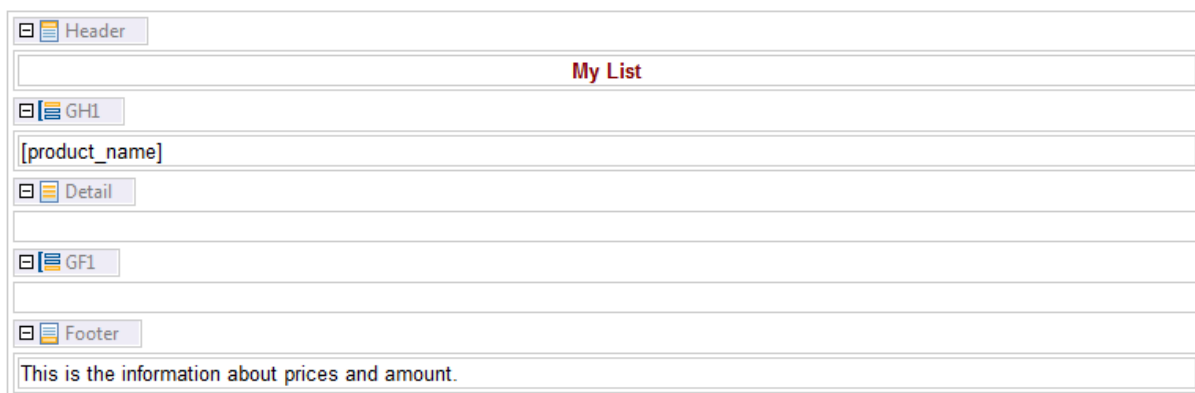


8. The list will now contain the group header (GH1) and the Group Footer (GF1):

9. Now to add the header to the list, right-click the general Header field and select **Insert -> Label** from the context menu:

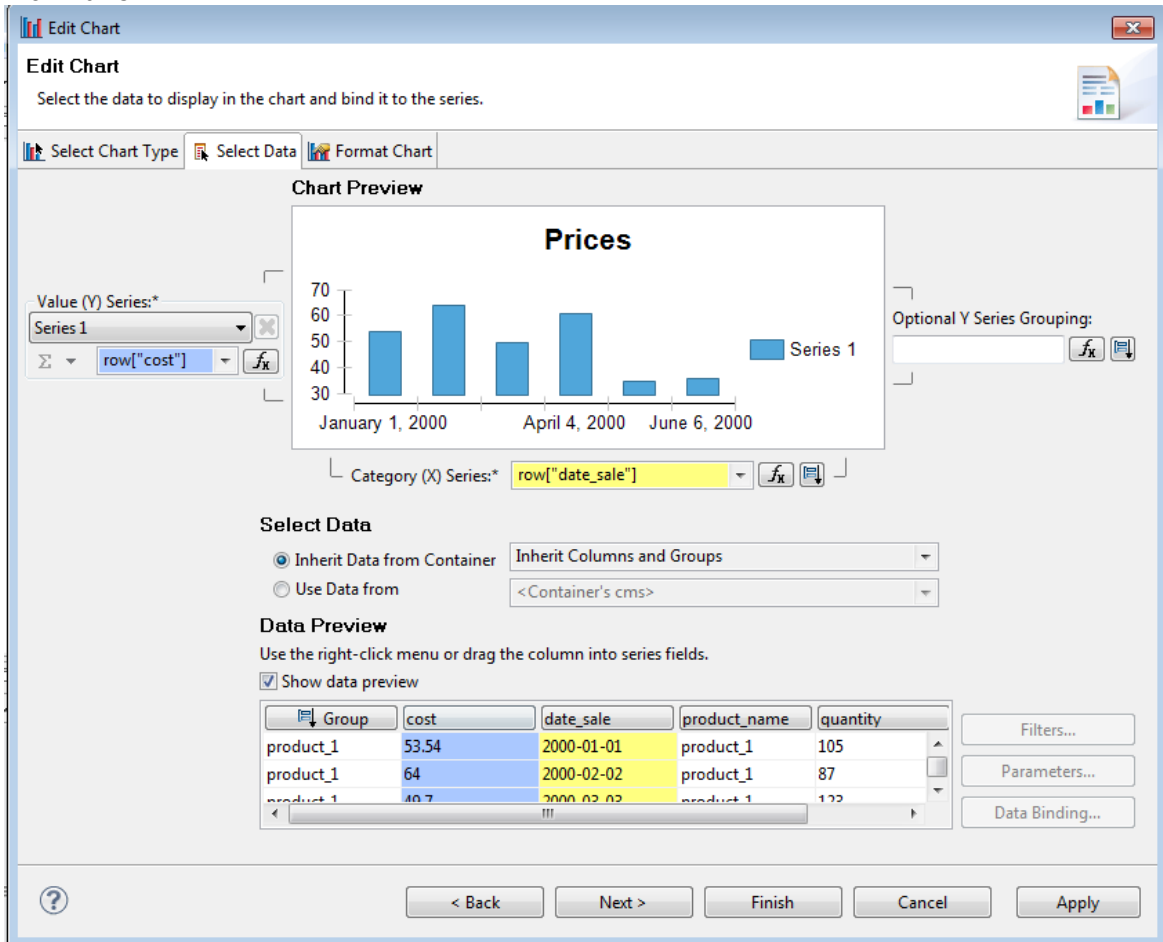


10. Now we have the Label object included to the list. To type the label name double-click this object. It has the same properties as any other Label outside a list. Type the Label name; you can change the font, colour, etc. using the Property Editor view below the editor. You can also add the objects to the list by dragging them from the Palette view.
11. You can also add some Text to the general Footer. Right-click the general Footer field and select **Insert -> Text** from the context menu. You can also include a Dynamic text there. A Dynamic text can include some variables and/or data set elements which value will change depending on the circumstances.
12. Now you will have the following:





13. Add a Chart there by right-clicking the GF1 field and selecting **Insert -> Chart** from the context menu. We are adding the cart to the group footer and not to the Details section, because in this case the results will be grouped by the product and we will receive only two charts - one per product. If we add the chart to the Details section, we will get a number of charts - one for each row.
14. IN the chart settings select the Bar chart. Then select the cost column for Y axis and the date column for X axis.



15. In the Format Chart tab change the name of the chart. Also remove the Legend, for each chart will have only one group of bars. To remove the legend select **Legend** option on the left and remove the check mark from the Visible option.

Legend

Legend

☐ Visible

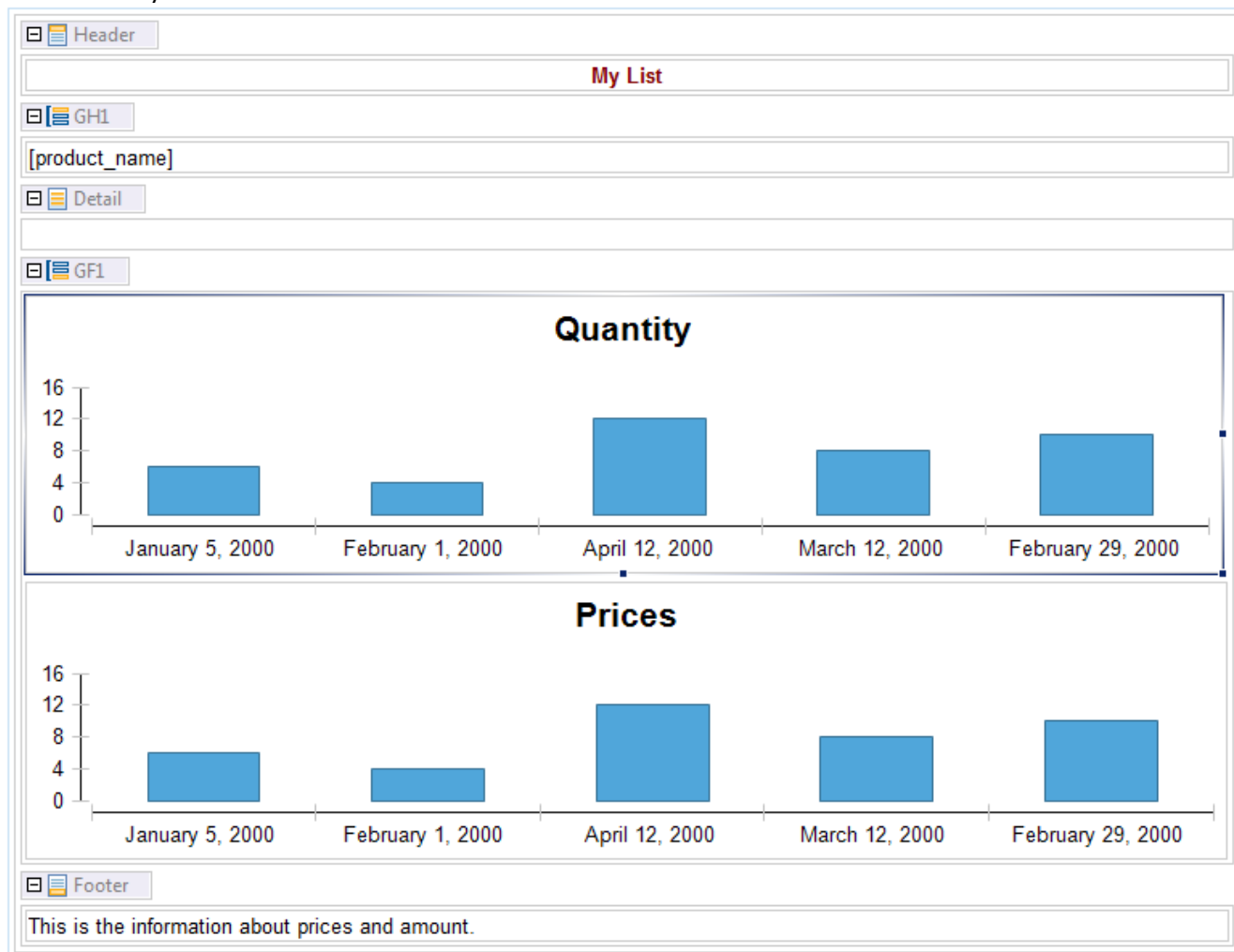
Title: ☐ Visible

Behavior: **None**

Value: ☐ Show Value



16. From the Palette view drag another Chart to the GF1 field. It also should be a bar chart with invisible legend. The Y axis should reference the quantity table, the rest should be the same as for the previous chart.
17. Now your list will look as follows:



GH1

```
[product_name]
```

My List

The following Graphs are for product:
product_1

Quantity

Date	Quantity
January 1, 2000	105
February 1, 2000	85
March 3, 2000	120
April 1, 2000	45
May 5, 2000	100
June 1, 2000	125
July 7, 2000	140
August 1, 2000	120
September 9, 2000	65
October 1, 2000	75
November 1, 2000	35
December 12, 2000	25

Prices

Date	Prices
January 1, 2000	50
February 1, 2000	60
March 3, 2000	45
April 1, 2000	55
May 5, 2000	35
June 1, 2000	35
July 7, 2000	40
August 1, 2000	55
September 9, 2000	90
October 1, 2000	90
November 1, 2000	100
December 12, 2000	120

The following Graphs are for product:
product_2

Quantity

Date	Quantity
January 1, 2000	24
February 1, 2000	13
March 3, 2000	20
April 1, 2000	9
May 5, 2000	19
June 1, 2000	15
July 7, 2000	27
August 1, 2000	21
September 9, 2000	18
October 1, 2000	10
November 1, 2000	29
December 12, 2000	24

Prices

Date	Prices
January 1, 2000	250
February 1, 2000	350
March 3, 2000	400
April 1, 2000	390
May 5, 2000	410
June 1, 2000	290
July 7, 2000	300
August 1, 2000	330
September 9, 2000	400
October 1, 2000	380
November 1, 2000	320
December 12, 2000	280


This is the information about prices and amount.

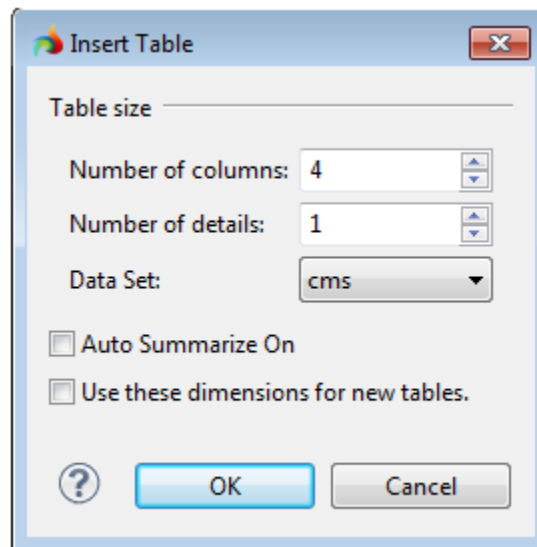


Tables

A table element is similar to the List element, it can include groups with headers and footers and serves for arranging data. Table cells can contain charts, labels, text and other objects. The major difference between a list and a table is that a table can have more than one column.

To create a table:

1. Create a BIRT file with the usual data source (e.g. cms database)
2. Create a data set for the file (i.e. select * from product). You can use the same sources as in the previous examples.
3. Add a table  **Table** to the report body from the Palette.
4. In the dialog of the new table creation choose the number of columns like the number of columns retrieved by the query in the data set, set the number of rows, select the data set to be referenced by the table. Click **OK**.



5. You will see an empty table:

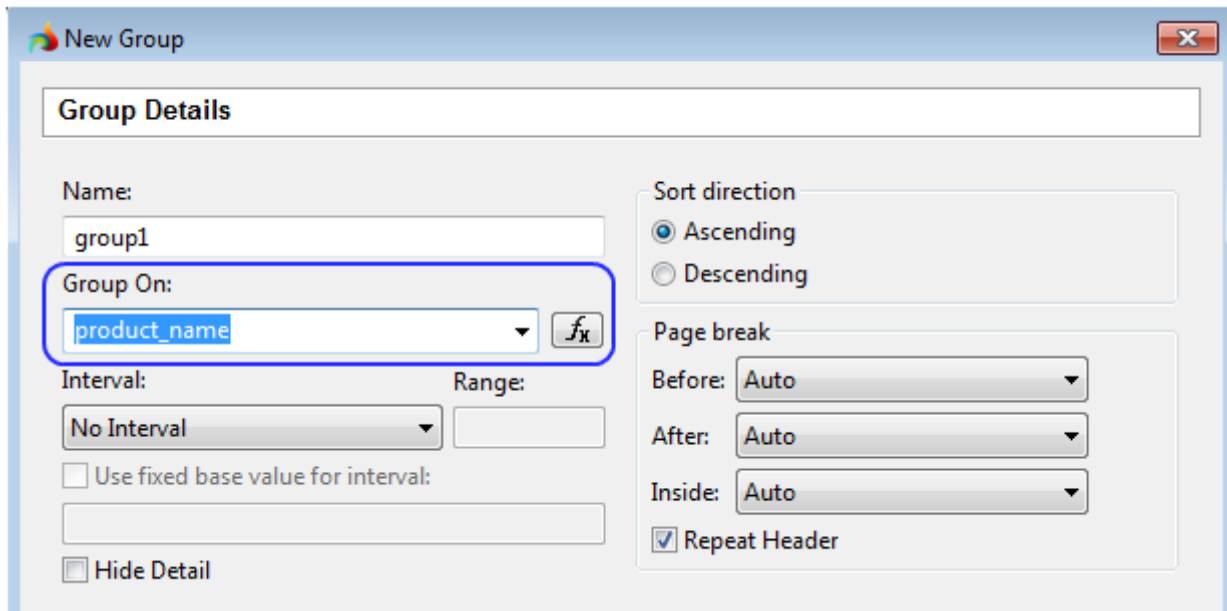
	Header Row			
	Detail Row			
	Footer Row			

6. In the Header Row add labels which would correspond to the column names by dragging the labels from the Palette. Double-click the label to enter the column name. You can format them to improve the design.

	Product Name	Date of Sale	Quantity	Price
	Detail Row			



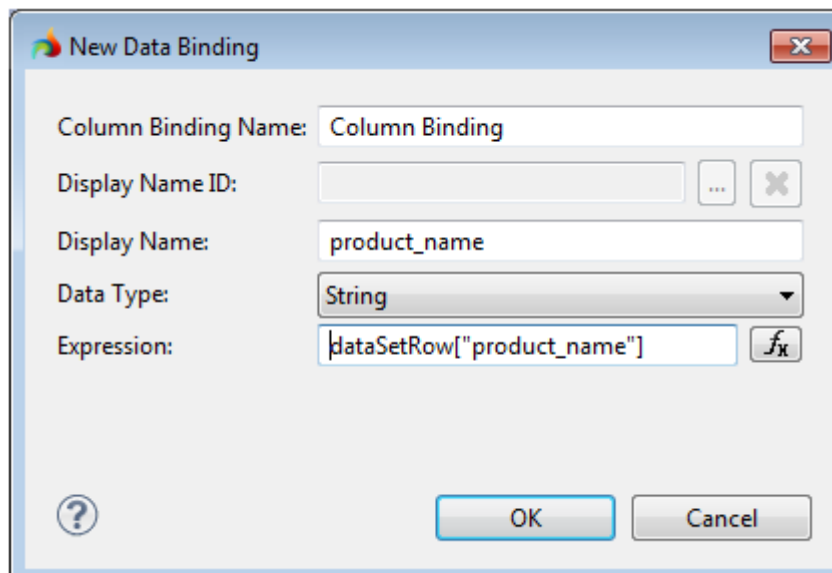
- Now you need to add a group to the table. We will use the group to sort the displayed data by the product. Right-click on a free cell and select **Insert Group** from the list. Select the product_name column for grouping and click **OK**.



The 'New Group' dialog box is shown. It has a title bar with a close button. The 'Group Details' section contains the following fields:

- Name:** group1
- Group On:** product_name (highlighted with a blue box)
- Interval:** No Interval
- Range:** (empty)
- ☐ Use fixed base value for interval:
- ☐ Hide Detail
- Sort direction:** ☒ Ascending, ☐ Descending
- Page break:** Before: Auto, After: Auto, Inside: Auto
- ☒ Repeat Header

- Now in the detail row into each cell add a Data item referencing the corresponding column of the data set:



The 'New Data Binding' dialog box is shown. It has a title bar with a close button. The fields are:

- Column Binding Name:** Column Binding
- Display Name ID:** (empty)
- Display Name:** product_name
- Data Type:** String
- Expression:** dataSetRow["product_name"]

At the bottom, there is a help icon, an OK button, and a Cancel button.

- Now your table will look as follows:



	Product Name	Date of Sale	Quantity	Price
1	[product_name] - Group header			Labels in the header row
	[product_name]	[date_sale]	[quantity]	[cost]
1	Group Footer Row (product_name)			The data row
	Footer Row			

- You can adjust the appearance of your table, add a footer, etc.; for example we can add an Aggregation to the footer of the cost column which counts the total amount. To add an aggregation click on the last cell of the footer row and select **Insert -> Aggregation** from the context menu. It can also be dragged from the Palette view.
- In the Aggregation dialog enter the name for the binding and the display name. Select SUM as the aggregate function. In the expression field enter 'dataSetRow["quantity"]*dataSetRow["cost"]' - thus we will calculate the total cost of all the table rows. Click **OK**.

The Aggregation Builder dialog box is shown with the following settings:

- Column Binding Name: Total
- Display Name ID: (empty)
- Display Name: Total
- Data Type: Float
- Function: SUM
- Expression: dataSetRow["quantity"]*dataSetRow["cost"]
- Filter Condition: (empty)
- Aggregate On: ☒ Table, ☐ Group (group1)

Buttons: OK, Cancel

- You can also add a label to indicate that the aggregation is about:

Product Name	Date of Sale	Quantity	Price
[product_name]			
[product_name]	[date_sale]	[quantity]	[cost]
Group Footer Row (product_name)			
Footer Row		Total Cost:	Σ[Total]



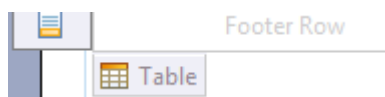
Now you can preview the result and deploy the saved file to the server. The result will look as follows:

Product Name	Date of Sale	Quantity	Price
product_1			
product_1	2000-01-01	105	53.54
product_1	2000-02-02	87	64
product_1	2000-03-03	123	49.7
product_1	2000-04-04	45	60.87
product_1	2000-05-05	99	34.46
product_1	2000-06-06	130	35.79
product_1	2000-07-07	145	40.56
product_1	2000-08-08	120	55.78
product_1	2000-09-09	60	90.67
product_1	2000-10-10	71	92.45
product_1	2000-11-11	35	100.4
product_1	2000-12-12	20	120
product_2			
product_2	2000-01-01	24	250.54
product_2	2000-02-02	13	345
product_2	2000-03-03	20	400.7
product_2	2000-04-04	9	390.87
product_2	2000-05-05	19	410.46
product_2	2000-06-06	15	287.79
product_2	2000-07-07	27	300.56
product_2	2000-08-08	21	326.78
product_2	2000-09-09	18	402.67
product_2	2000-10-10	10	378.45
product_2	2000-11-11	29	321.4
product_2	2000-12-12	24	279
Total Cost:			134771.18000000002

May 12, 2011 10:37 AM

Though the table has only one details row Lycia displays all the data retrieved by the query in the form of a table and grouped by the product (due to the group inserted).


You can make the table look more table-like, if you select the whole table and in the Properties view choose the **Border** option. Here you can select which table borders should be visible and how they should look like. You can also set the border for a single field or a set of fields. You can modify only the outer border of the table, if you click on the Table tag at the bottom of the table and use the Properties view:

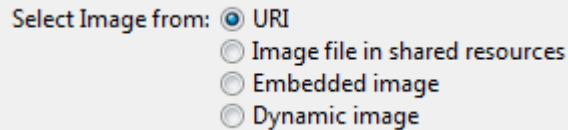


The table example above illustrates how you can display the data stored in a database table in the form of a table. The table element can be used for other display forms as well, it can include charts and other forms of visualization and become quite complex.

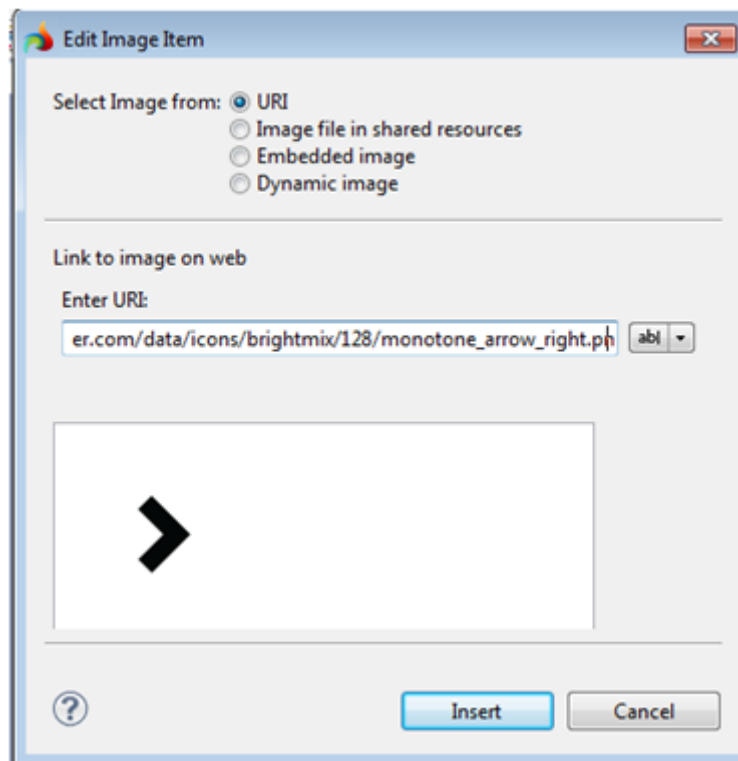



Images

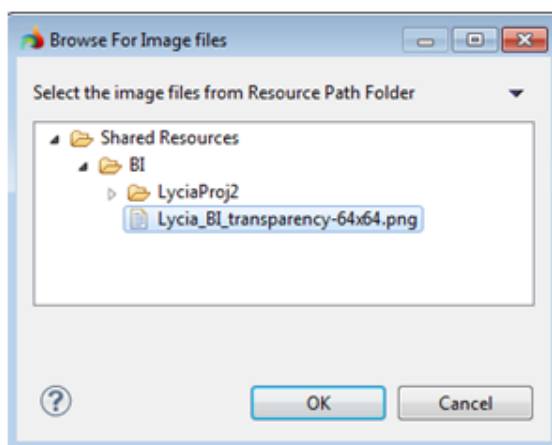
An image element can reference some data from a data set (e.g. display an image stored in a BYTE column of a database) or it can serve a purely ornamental purpose. To add an image  **Image**, drag it from the Palette to the report body. You will see the dialog where you can select what type of image you want to add:



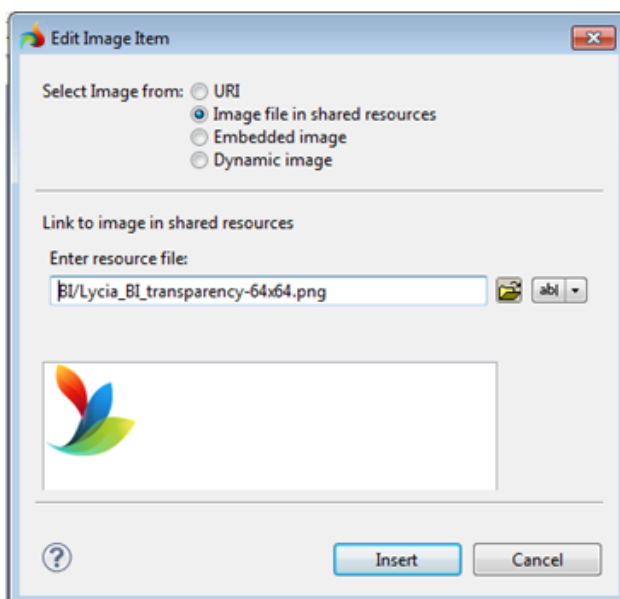
- Adding an URL image - the image is not copied to your report, only the link is added:
 1. Select an image you want to add on the web. To find out the URL of the image, right-click it in the browser and select **Copy the image URL** option.
 2. Select the **URI** option in the New Image dialog.
 3. Enter paste the image URL from the clipboard to the URL field:



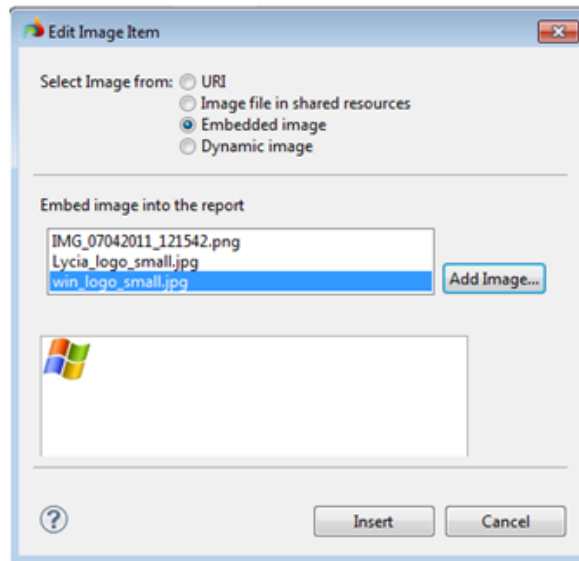
4. The image preview will be available in the field below. Click **Insert** to add the image.
- Adding an image included into the project sources - the image is not copied to your report, only the link is added:
 1. Select **Image file in shared resources** option.
 2. Click  button to browse the shared resources (the resources included into the current project):



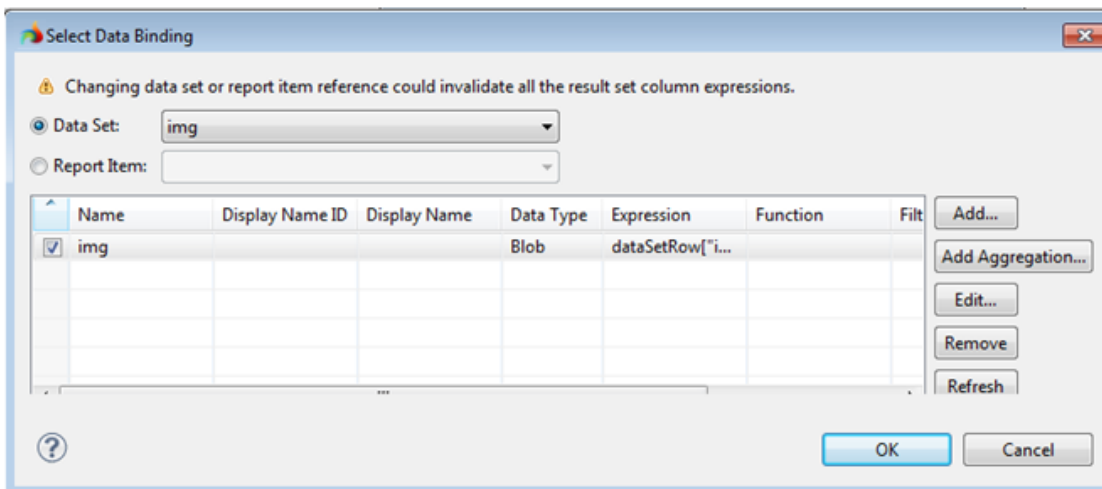
3. After you have selected the image, it will be shown for preview. Click **Insert** to add the image to the report.




- Adding an embedded image - the image is copied into the report and stored within it:
 1. Select **Embedded image** option.
 2. Click the **Add Image** button.
 3. Select the image you want to add from anywhere on your system. It will be added to the list of embedded images of the report and the preview will be displayed.



- Adding a dynamic image - the image is retrieved from a BYTE column of a database
 1. Create a data set which references a BYTE column with an image. If the data set is composed so that it can retrieve several images from the BYTE column, the first retrieved image will be displayed.
 2. Select **Dynamic image** option.
 3. Click **Select Image Data...** button.
 4. In the following dialog select the data set which retrieves a value from a BYTE column. Check the column from which the image is to be retrieved, Click **OK**.



5. The preview is not available for the images retrieved from a database. Click **Insert** to add the image.
6. In the report page the image referencing a table will be displayed as  which is normal. The image can be viewed normally when the document is deployed to the server or by clicking the **Preview** tab of the editor.



Integrating BIRT Reports into 4GL

BIRT reports can be incorporated into 4GL applications. The Apache Tomcat Web Application server is used to store the reports created and deployed by LyciaStudio. The Apache Tomcat Server allows users to launch these reports in their web browser, even if they do not have Lycia installed.

To make your reports visible on the Web Server, you need to put your reports into the corresponding "report" folder, using standard Windows tools. By default, the next path leads to this folder:

C:\ProgramData\Querix\Lycia6\apache-tomcat\webapps\birt\report. Once done, get back to the Studio, in 4GL Perspective right-click on your project, in the context menu choose **Add Requirements**, select **Other Files** filter and add the required .rptdesign files. Again, right-click on your updated project and select **Deploy to Server** option.

These are the predefined dynamic text samples which can prove useful.

EXAMPLES

This is the example of a simple report menu.

GLOBALS

```
DEFINE reports DYNAMIC ARRAY WITH dimension 1 OF STRING
```

```
DEFINE report_menu_n INTEGER
```

```
END GLOBALS
```

```
# convert string for using in command-line
```

```
FUNCTION escaped_url(str)
```

```
  DEFINE str STRING
```

```
  RETURN "\"" || str || "\""
```

```
END FUNCTION
```

```
# report_url creates url for report generation online web server
```

```
# see http://www.eclipse.org/birt/phoenix/deploy/viewerUsageMain.php
```

```
# for more options
```

```
FUNCTION report_url(fname, fformat)
```

```
  DEFINE fname STRING
```

```
  DEFINE fformat STRING
```

```
  DEFINE host_addr STRING
```

```
  DEFINE port_num STRING
```

```
  DEFINE url STRING
```

```
  DEFINE birt_command STRING
```

```
LET birt_command = "frameset"
```

```
#let birt_command = run
```

```
#let birt_command = preview
```

```
# with ajax application, parameter dialog and toolbars
```

```
# without navigation toolbars, with parameters dialog
```

```
# generate without parameters dialog
```



```
LET host_addr = "localhost"
LET port_num = "0000"      #indicate your port number
LET url = "http://" || host_addr || ":" || port_num || "/birt/" || birt_command || "?"
LET url = url || "__report=report/" || fname
IF NOT fformat IS NULL THEN
    LET url = url || "&__format=" || fformat
END IF
# below the set of the optional parameters is indicated
LET url = url || "&__showtitle=false"
LET url = url || "&__navigationbar=false"
LET url = url || "&__toolbar=false"
#let url = url || "&__svg=false"          # false value forces to use png image format
#let url = url || "&__parameterpage=false"
#let url = url || "&__pagerange=1-4,7"
RETURN escaped_url(url)
END FUNCTION
# generates the list of available reports (*.rptdesign files in birt server filesystem)
# returns chosen report name and generated format
FUNCTION create_report_list(submenu_id)
    DEFINE submenu_id,
        url STRING,
        freport_name STRING,
        fformat STRING,

        dir INTEGER,
        path STRING,
        i INTEGER

    LET path = "../apache-tomcat/webapps/birt/report"
    CALL os.Path.dirfmask(1 + 2 + 4) -- retrieves only regular files
    CALL os.Path.dirsort("name", 1) -- sorts by name
    LET dir = os.Path.diropen(path)

    LET i = 1

    #displays "designs from path: ", path
    WHILE TRUE
        DEFINE f STRING
        DEFINE ext STRING
        DEFINE fname STRING
        DEFINE tm STRING

        LET f = os.Path.dirnext(dir)
        IF f IS NULL THEN EXIT WHILE END IF
        IF f == "." OR f == ".." THEN CONTINUE WHILE END IF

        LET fname = os.Path.rootname(f)
        LET ext = os.Path.extension(f)
```




```
LET tm = os.Path.mtime(os.Path.join(path, f))
IF ext = "rptdesign" THEN
  LET reports[i] = f
  CALL menu_add_option(submenu_id, tm || " " || fname, i + report_menu_n)

  #display tm, ": ", f
  LET i = i + 1
END IF
END WHILE

CALL os.Path.dirclose(dir)
END FUNCTION

MAIN
DEFINE
  main_menu_id,      -- This is the main menu id that
                     -- will be referenced by the
                     -- functions to add options
                     -- and submenus

  submenu1_id,      -- These variables will store the ids of submenus
  submenu2_id,
  submenu3_id,
  action_id INT      -- And this variable will hold
                     -- the id of the menu option
                     -- activated by the user

LET report_menu_n = 100500
LET main_menu_id = create_menu()
LET submenu1_id = menu_add_submenu(main_menu_id, "Standard_Programs", 1)

CALL menu_add_option(submenu1_id, "Notepad", 2) -- action ID to reflect the user's choice
CALL menu_add_option(submenu1_id, "Paint", 3)
CALL menu_add_option(submenu1_id, "Calculator", 4)
CALL menu_add_option(main_menu_id, "4GL_Program", 7)

LET submenu2_id = menu_add_submenu(main_menu_id, "Open_Documents", 8)

CALL menu_add_option(submenu2_id, "Document_1", 9)
CALL menu_add_option(submenu2_id, "Document_2", 10)
LET submenu3_id = menu_add_submenu(main_menu_id, "Reports", 11)
CALL create_report_list(submenu3_id)
CALL menu_add_option(main_menu_id, "Exit", 12)
```



Publishing and using the menu. After the menu has been created and all the submenus and options have been added to it, we must execute the `menu_publish()` function. Otherwise, the menu will not be available. Now we cannot modify the menu in any way

CALL `menu_publish()`

To keep the menu on the screen until the user presses the Exit or X buttons,
the rest of the code is enclosed within the WHILE loop

WHILE TRUE

Next, we call the `execute_menu()` function. It returns the id of the menu
option activated by the user. This id is then passed to the CASE statement and
the corresponding action is executed

LET `action_id` = `execute_menu()`

CASE `action_id`

WHEN 2

CALL `winexec("notepad")`

-- These options in the first submenu
-- call standard programs: "Notepad",
-- "MSPaint""Calculator" and the default
-- mail client.

WHEN 3

CALL `exec_local("MSpaint","SYSTEM")` -- As the programs are standard, you don't
-- have to specify the path to their .exe
-- files or the extension itself.

WHEN 4

CALL `exec_local("calc","SYSTEM")` -- If you run this application on Linux,
-- you should replace them with default
-- Linux applications if required

WHEN 7

CALL `exec_program("hello","localhost","1889","SYSTEM")` -- indicate your current port

The following two options are used for opening documents on the local system.
These documents must exist on your OS and you must specify the correct
paths to them as the arguments. The paths below are given only for example

WHEN 9

CALL `winshellexecwait("C:\\ProgramData\\Querix\\Lycia 6\\progs\\1.txt")`

WHEN 10

CALL `winshellexec("C:\\ProgramData\\Querix\\Lycia 6\\progs\\logo.jpg")`

WHEN 12

EXIT PROGRAM

OTHERWISE

CALL `winshellexec(report_url(reports[action_id - report_menu_n], "html"))` --report_format))

END CASE

END WHILE

END MAIN



This example shows how to create the report with drop-in report list and the list of the available report formats.

```
#####
```

```
# convert string for using in command-line
```

```
FUNCTION escaped_url(str)
  DEFINE str STRING
  RETURN "\"" || str || "\""
END FUNCTION
```

```
# report_url creates url for report generation online web server
# see http://www.eclipse.org/birt/phoenix/deploy/viewerUsageMain.php
# for more options
```

```
FUNCTION report_url(fname, fformat)
  DEFINE fname STRING
  DEFINE fformat STRING
  DEFINE host_addr STRING
  DEFINE port_num STRING
  DEFINE url STRING
  DEFINE birt_command STRING
```

```
LET birt_command = "frameset"           # with ajax application, parameter dialog and toolbars
#let birt_command = run                 # without navigation toolbars, with parameters dialog
#let birt_command = preview             # generate without parameters dialog
```

```
LET host_addr = "localhost"
LET port_num = "0000" -- your local port
LET url = "http://" || host_addr || ":" || port_num || "/birt/" || birt_command || "?"
LET url = url || "__report=report/" || fname
IF NOT fformat IS NULL THEN
  LET url = url || "&__format=" || fformat
END IF
```

```
# you can use multiple optional parameters
```

```
LET url = url || "&__showtitle=false"
LET url = url || "&__navigationbar=false"
LET url = url || "&__toolbar=false"
```

```
#let url = url || "&__svg=false"           # false value forces to use png image format
```

```
#let url = url || "&__parameterpage=false"
```

```
#let url = url || "&__pagerange=1-4,7"
```

```
RETURN escaped_url(url)
END FUNCTION
```

```
# generate list of available reports (*.rptdesign files in birt server filesystem)
# returns chosen report name and generated format
```

```
FUNCTION choose_report()
```

```
  DEFINE url STRING
  DEFINE freport_name STRING
```



```
DEFINE fformat STRING
DEFINE flist ui.ComboBox

DEFINE dir INTEGER
DEFINE path STRING

OPEN FORM report_list_form FROM "online_report_list" --opens form containing your report list
DISPLAY FORM report_list_form

#fill combo box with report names
LET flist = ui.ComboBox.ForName("freport_name")

CALL flist.clear()
LET path = "../apache-tomcat/webapps/birt/report"

CALL os.Path.dirfmask(1 + 2 + 4) -- retrieves only regular files
CALL os.Path.dirsort("name", 1) -- sorts by name
LET dir = os.Path.diropen(path)

#display "designes from path: ", path
WHILE TRUE
  DEFINE f STRING
  DEFINE ext STRING
  DEFINE fname STRING
  DEFINE tm STRING

  LET f = os.Path.dirnext(dir)
  IF f IS NULL THEN EXIT WHILE END IF
  IF f == "." OR f == ".." THEN CONTINUE WHILE END IF

  LET fname = os.Path.rootname(f)
  LET ext = os.Path.extension(f)
  LET tm = os.Path.mtime(os.Path.join(path, f))
  IF ext = "rptdesign" THEN
    CALL flist.addItem(f, tm || " " || fname)
    #display tm, ":", f
  ENF IF
END WHILE

CALL os.Path.dirclose(dir)
LET freport_name = null
LET fformat = "html"
INPUT BY NAME freport_name, fformat
#without defaults
ATTRIBUTE (unbuffered)
#on change freport_name
# if not freport_name is null then
#   exit input
# end if
```



END INPUT

CLOSE FORM report_list_form

RETURN freport_name, fformat
END FUNCTION

MAIN

DEFINE report_name STRING
DEFINE report_format STRING

WHILE TRUE

CALL choose_report() RETURNING report_name, report_format
IF report_name IS NULL THEN
EXIT WHILE
END IF

CALL *winshellexec*(report_url(report_name, report_format))
END WHILE
END MAIN

This examples shows how to display your report online and work with it using different parameters.

GLOBALS

DEFINE browser_form WINDOW
DEFINE birt_host STRING
DEFINE birt_port STRING
DEFINE report_name STRING
DEFINE params STRING

END GLOBALS

FUNCTION display_browser()
DEFINE url STRING
DEFINE cbx ui.ComboBox

DEFINE dir INTEGER
DEFINE path STRING

OPEN FORM browser_form FROM "online_report"
DISPLAY FORM browser_form

#fill combo box with report names

LET cbx = ui.ComboBox.ForName("formonly.freport_list")

CALL cbx.clear()
LET path = "../apache-tomcat/webapps/birt/report"

CALL os.Path.dirfmask(1 + 2 + 4) *-- retrieves only regular files*
CALL os.Path.dirsort("name", 1) *-- sorts by name*
LET dir = os.Path.diropen(path)



```
DISPLAY "designs from path: ", path
```

```
WHILE TRUE
```

```
    DEFINE f STRING
```

```
    DEFINE ext STRING
```

```
    DEFINE fname STRING
```

```
    DEFINE tm STRING
```

```
    LET f = os.Path.dirnext(dir)
```

```
    IF f IS NULL THEN EXIT WHILE END IF
```

```
    IF f == "." OR f == ".." THEN CONTINUE WHILE END IF
```

```
    LET fname = os.Path.rootname(f)
```

```
    LET ext = os.Path.extension(f)
```

```
    LET tm = os.Path.mtime(os.Path.join(path, f))
```

```
    IF ext = "rptdesign" THEN
```

```
        CALL cbx.addItem(f, tm || " " || f)
```

```
        DISPLAY tm, ": ", f
```

```
    END IF
```

```
END WHILE
```

```
CALL os.Path.dirclose(dir)
```

```
    LET url = "http://" || birt_host || ":" || birt_port || "/birt/frameset?__report=report/" || report_name ||  
"&" || params
```

```
    # choose report and parameters
```

```
    DISPLAY url TO fbrowser
```

```
    DISPLAY url TO fparams
```

```
END FUNCTION
```

```
FUNCTION main_loop()
```

```
    DEFINE url STRING
```

```
    DEFINE rpt STRING
```

```
    OPTIONS
```

```
    PROMPT LINE FIRST,
```

```
    MESSAGE LINE LAST -1,
```

```
    COMMENT LINE LAST,
```

```
    HELP KEY F1,
```

```
    DELETE KEY F9,
```

```
    INSERT KEY F10,
```

```
--    NEXT KEY "prevpage",
```

```
--    PREVIOUS KEY "nextpage",
```

```
    ACCEPT KEY "RETURN"
```



DEFER INTERRUPT

CALL *fgl_putenv*("DBDATE=dmy4/")

LET url = "http://" || birt_host || ":" || birt_port || "/birt/frameset?__report=" || report_name || "&" ||
params

```
#display url
#call winshellexec("\"" || url || "\"")
#call winexec("explorer.exe \"\" || url || "\"")
```

CALL display_browser()

WHILE TRUE

MENU "Contact"

BEFORE MENU

COMMAND KEY (F5) "New" "New" HELP 1 -- "New" "Create a new contact record" Help 202

HIDE OPTION ALL

```
--LET pend_contact_id = contact_create(0,FALSE) --false is don't open window
--IF pend_contact_id IS NOT NULL THEN
-- CALL contact_show(pend_contact_id)
--ELSE
-- CALL contact_show(l_contact_id)
--END IF
```

```
--IF exist(pend_contact_id) THEN
-- EXIT MENU
--END IF
--CALL contact_sales_populate_grid_panel(l_contact_id, NULL,NULL,NULL)
```

SHOW OPTION ALL

COMMAND KEY (F8) "Find" "Find" HELP 1 -- "Find" "Find a contact record" Help 207

HIDE OPTION ALL

```
#LET pend_contact_id = query_contact()
#IF exist(pend_contact_id) THEN
# EXIT MENU
#END IF
SHOW OPTION ALL
```

COMMAND KEY (F9) "Report" "Report" HELP 1 -- "Find" "Browse through reports"

HIDE OPTION ALL

{input rpt from freport_list

LET url = "http://" || birt_host || ":" || birt_port || "/birt/frameset?__report=" || rpt

DISPLAY url TO fbrowser

--CALL browse_reports()



```
}  
SHOW OPTION ALL  
  
COMMAND KEY ("CONTROL-F","KEY_HOME",F91) --First Record  
#FETCH FIRST c_cont_scroll  
#INTO l_contact_id, tmp_cont_company  
#CALL contact_show(l_contact_id)  
#CALL contact_sales_populate_grid_panel(l_contact_id, NULL,NULL,NULL)  
  
COMMAND KEY ("CONTROL-P","PREVIOUS",UP,F93) --Previous Record  
#FETCH PRIOR c_cont_scroll  
#INTO l_contact_id, tmp_cont_company  
#IF sqlca.sqlcode THEN  
#FETCH FIRST c_cont_scroll  
#INTO l_contact_id, tmp_cont_company  
#END IF  
#CALL contact_show(l_contact_id)  
#CALL contact_sales_populate_grid_panel(l_contact_id, NULL,NULL,NULL)  
  
COMMAND KEY ("CONTROL-N","NEXT",DOWN,F94) --Next Record  
#FETCH NEXT c_cont_scroll  
#INTO l_contact_id, tmp_cont_company  
#IF sqlca.sqlcode THEN  
#FETCH LAST c_cont_scroll  
#INTO l_contact_id, tmp_cont_company  
#END IF  
#CALL contact_show(l_contact_id)  
#CALL contact_sales_populate_grid_panel(l_contact_id, NULL,NULL,NULL)  
  
COMMAND KEY ("CONTROL-L","KEY_END",F96) --Last Record  
#FETCH LAST c_cont_scroll  
#INTO l_contact_id, tmp_cont_company  
#CALL contact_show(l_contact_id)  
#CALL contact_sales_populate_grid_panel(l_contact_id, NULL,NULL,NULL)  
  
COMMAND KEY (F12,"R") "Exit" "Exit" HELP 1 -- "Main Menu" "Close Window and return to main  
menu/window" Help 51  
EXIT WHILE  
  
END MENU --CLOSE c_cont_scroll  
  
IF int_flag THEN  
  DISPLAY "int_flag was received"  
  LET int_flag = FALSE  
  EXIT WHILE  
END IF  
END WHILE  
  
CLOSE FORM browser_form
```




END FUNCTION

MAIN

```
LET birt_host = "localhost"
LET birt_port = "0000"
LET report_name = "report_table.rptdesign"
LET params = "sample=my+parameter"
```

```
CALL main_loop()
```

END MAIN

This is the example of a running HTML task. The content and data location are output in the # application.

```
#import java java.util.logging.Level
import java org.eclipse.birt.core.framework.Platform
import java org.eclipse.birt.report.engine.api.EngineConfig
import java org.eclipse.birt.report.engine.api.HTMLRenderOption
import java org.eclipse.birt.report.engine.api.IReportEngine
import java org.eclipse.birt.report.engine.api.IReportEngineFactory
import java org.eclipse.birt.report.engine.api.IReportRunnable
import java org.eclipse.birt.report.engine.api.IRunAndRenderTask
import java java.lang.Object
import java java.lang.Integer
```

```
FUNCTION mybirt (param)
  DEFINE param STRING
```

```
#define log_level Level
  DEFINE config EngineConfig
  DEFINE obj Object
  DEFINE factory IReportEngineFactory
  DEFINE engine IReportEngine
  DEFINE design IReportRunnable
  DEFINE task IRunAndRenderTask
  DEFINE html_opts HTMLRenderOption
  DEFINE birt_home_dir STRING
  DEFINE par_i1 java.lang.Integer
```

```
LET birt_home_dir = fgl_getenv("LYCIA_DIR") || "\\ReportEngine"
#configure the Engine
LET config = EngineConfig.create()
# set BIRT root directory
CALL config.setBIRTHome(birt_home_dir)
#let log_level = Level.parse("FINEST")
#call config.setLogConfig("../logs", log_level)
```



CALL Platform.startup(config) -- before creating the Engine, you should start the Platform, which will load the appropriate plug-ins.; the function takes an EngineConfig object as argument

```
#set parameters for the Engine
#create Factory Object - used to create Report Engine
LET obj = Platform.createFactoryObject("org.eclipse.birt.report.engine.ReportEngineFactory")
#define public interface
LET factory = cast(obj AS IReportEngineFactory)
LET engine = factory.createReportEngine(config)
LET design = engine.openReportDesign("report/report_table.rptdesign")
#create a run and render task object
LET task = engine.createRunAndRenderTask(design)
#set parameters for the Report
CALL task.setParameterValue("prod_name", param)
#validate the stated parameters
CALL task.validateParameters()
#set render options
LET html_opts = HTMLRenderOption.create()
#set output location
CALL html_opts.setOutputFileName("../apache-tomcat/webapps/birt/report/report_table.html")

#set rendering options
CALL task.setRenderOption(html_opts)
DISPLAY "HTML render task is running"
CALL task.run()
CALL task.close()
CALL engine.destroy() -- destroy the Engine
CALL Platform.shutdown() -- the function is called to clean up work, including unloading extensions

DISPLAY "application completed"
END FUNCTION

MAIN
CALL mybirt("param")
END MAIN
```

BIRT SERVLETS

More information regarding BIRT servlets and how they can be used you will find on the Eclipse website by the following link:

<http://www.eclipse.org/birt/phoenix/deploy/viewerUsage2.2.php>

The detailed overview of the BIRT reportlets is also presented in the BIRT official blog:

<http://birtworld.blogspot.com/2008/01/birt-reportlets.html>