JASPERSOFT

JASPERSOFT STUDIO

USER GUIDE

RELEASE 5.5

http://www.jaspersoft.com

This is version 1013-JSP55-03 of the *Jaspersoft Studio User Guide*.

TABLE OF CONTENTS

Chapter 1 Getting Started with Jaspersoft Studio

Jaspersoft Studio's primary goal is to provide the features in the well-known Jaspersoft Report Editor, available as a port of Jaspersoft Studio. This is only the beginning - having its foundations on the Eclipse platform, Jaspersoft Studio will be a more complete solution allowing users to extend its capabilities and functionality.

This chapter contains the following sections:

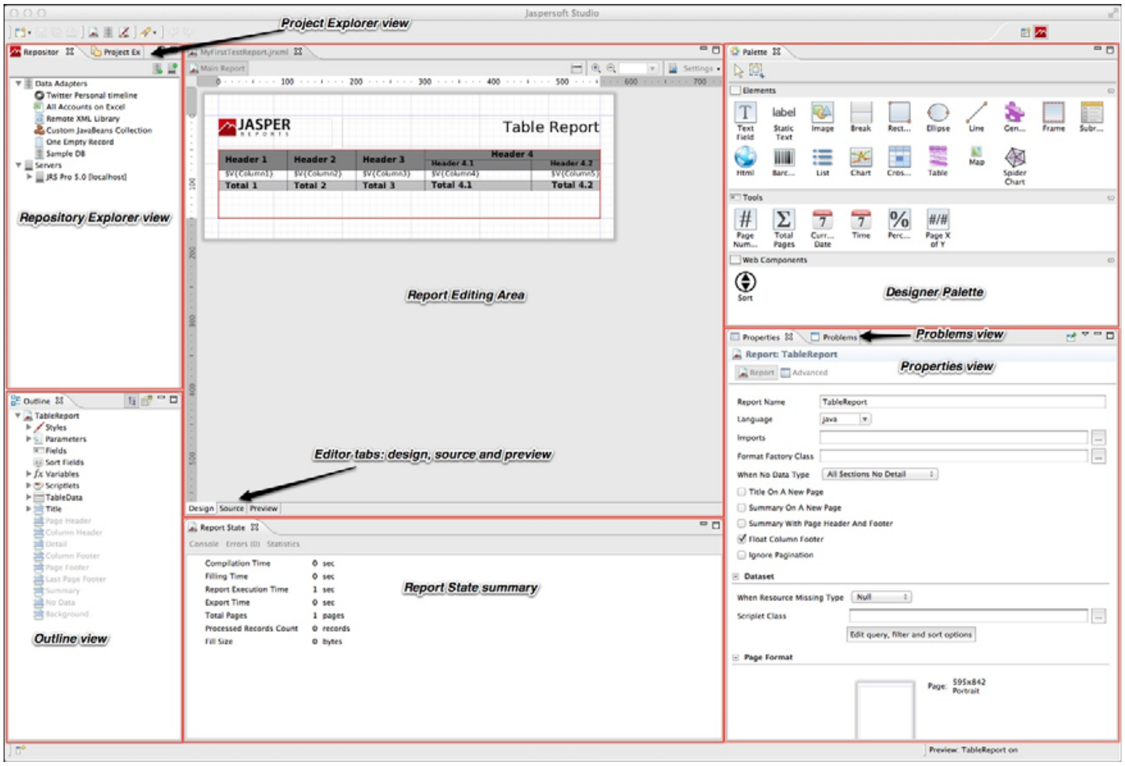
- **Introduction to Jaspersoft Studio**
- **User Interface**
- **Hardware Requirements**
- **Software Requirements**
- **Accessing the Source Code**
- **Report Structure in Jaspersoft Studio**
- **Measure Units in Jaspersoft Studio**
- **Exporting reports with Jaspersoft Studio**

1.1 Introduction to Jaspersoft Studio

Jaspersoft Studio (JSS) is the new Eclipse-based report designer for JasperReports and JasperReports Server. It is a full rewrite of iReport, available as Eclipse plug-in, and as a standalone application. Jaspersoft Studio allows you to create sophisticated layouts containing charts, images, subreports, crosstabs and much more. You can access your data through JDBC, TableModels, JavaBeans, XML, Hibernate, CSV, and custom sources, then publish your reports as PDF, RTF, XML, XLS, CSV, HTML, XHTML, text, DOCX, or OpenOffice.

1.2 User Interface

Jaspersoft Studio is offered in two different versions: a standalone RCP product, and an Eclipse plug-in version. People who have worked with Eclipse will be familiar with the user interface, while to new users - or those familiar only with iReport - the layout of the shown elements will appear quite different. Both standalone and plug-in version have a similar user interface. In **Figure 1-1**, you can see a preview of the Jaspersoft Studio interface, with the main areas highlighted:



Report: TableReport

Report Name TableReport
Language java
When No Data Type All Sections No Detail
□ Title On A New Page
□ Summary On A New Page
□ Summary With Page Header And Footer
☑ Float Column Footer
□ Ignore Pagination

Dataset

When Resource Missing Type Null

Page Format

Page: 595x842 Portrait

Report State

Compilation Time 0 sec
Filling Time 0 sec
Report Execution Time 1 sec
Export Time 0 sec
Total Pages 1 pages
Processed Records Count 0 records
Fill Size 0 bytes

Figure 1-1 Jaspersoft Studio User Interface

The main Report editing area is where you visually design the report by dragging, positioning, aligning and resizing report elements chosen from the Designer palette.

Jaspersoft Studio has a multi-tab editor. As you can see in the figure above, there are three different tabs: Design, Source and Preview:

- The Design tab is the main one selected when you open a report file and it allows you to graphically create your report.
- The Source tab contains the JRXML source code for your report.
- The Preview tab lets you run the report preview after having selected a data source and output format.

There are a number of views with which you can explore the data:

- The Repository Explorer view maintains the list of JasperServer connections and available data adapters.
- The Project Explorer view maintains the list of the projects in the current workspace, usually JasperReports projects.
- The Outline view shows the complete structure of the report in a tree-form way.
- The Properties view is usually a foundation one for any Eclipse based product/plug-in. It usually gets populated with properties information of the actual selected element. That's way when you select a report element from the main design area (i.e: a text field) or from the Outline, the view shows information on it. Some of these properties can be read-only, but most of them are editable and their modification will usually notify changes to the drawn elements (i.e: element width or height).
- The Problems view shows a list of problems and errors that can for example block the correct compilation of a report.
- The Report state summary gives the user useful information about statistics on report compilation/filling/execution. Errors are shown here as well.

This is a short comparison table that helps the users to see where the main areas are in iReport and Jaspersoft Studio

. **Table 1-1 Comparison of Features in iReport and Jaspersoft Studio**

iReport	Jaspersoft Studio
JasperServer Repository	Repository Explorer
Report Inspector	Outline view
Report Designer	Report Editing Area
Problems List	Problems view
Elements palette	Designer Palette
Formatting tools	Available via context menu on the element
Property sheet	Properties view
Styles library	---
---	Project Explorer
iReport Output window	Report State summary

1.3 Hardware Requirements

Jaspersoft Studio needs a 64bit or 32bit processor and at least 500MB of Hard Disk space. The amount of the RAM is really dependant from the complexity of the reports, a value of 1GB dedicated to Jaspersoft Studio is recommended, since in your system there are many other things that require RAM it's suggested a total value of 2GB.

1.4 Software Requirements

As any other Java project Jaspersoft Studio require the Java Runtime Environment (JRE), but to compile the report scriptlets a full distribution of Java is required. You need to download the Java Development Kit (JDK) 1.6 or newer from THIS. After clicking the link you will see a page like this:

You must accept the license agreement to proceed with the download and then click on the correct link for your Operative System. The systems outside the green rectangles are not supported by Jaspersoft Studio. At the moment Jaspersoft Studio support the most common operative systems:

- Windows XP/7/8 with 32 or 64 bit
- Linux with 32 or 64 bit
- MacOS X at 64 bit

The download of Jaspersoft Studio starts from http://community.jaspersoft.com/project/jaspersoft-studio/releases, a page with listed all the release of Jaspersoft Studio will appear, click the one on the top of the list (the last version) and a grid with many links will be shown. Here select from:

- jaspersoftstudio_x.x.x_i386.deb for the 32bit version for linux;
- jaspersoftstudio_x.x.x_amd64.deb for the 64bit version for linux;
- jaspersoftstudio_x.x.x_windows-installer-x86_64.exe for the 64bit version of windows;
- jaspersoftstudio_x.x.x_windows-installer-x86.exe for the 32bit version of windows;
- jaspersoftstudio-x.x.x-mac-x86_64.dmg for the 64bit version of MacOS X.

x.x.x represents the version number of Jaspersoft Studio. If your distribution of linux doesn't support the deb format there are also the tar versions. Note that first you must download and install the JDK and then download and install Jaspersoft Studio.

Be sure to download and install the correct version of Java and Jaspersoft Studio for your operative system. If you have any doubt on the architecture (number of bits) of your Operative System read the subsection below. In most of cases the installation requires only the download of the correct file and then double-clicking on in, this will show the installation wizard and here you can follow the on screen instructions.

Note that on a 64bit operative system you can install the 32bit version of Java and Jaspersoft studio (although in this case the 64bit is suggested), but you can't do the opposite. It's very important to install the same version of Java and Jaspersoft Studio. For example a Java environment with 32bit will not work with the 64bit version of Jaspersoft Studio, you must use a 32bit version of Java with a 32bit version of Jaspersoft Studio or a 64bit version of Java with a 64bit version of Jaspersoft Studio.

1.5 Accessing the Source Code

The last version of the source code is available from http://community.jaspersoft.com/project/jaspersoft-studio/releases by clicking “Browse Source Code”, in this way you will access directly the SVN repository (read only mode) where the most up-to-date version is available. You can download and compile this source code, but since it is a work in progress it might contain new, unreleased features and bugs. All the information necessary to download the Source Code, configure a develop environment on the Eclipse IDE and compile\run the source code are described in the tutorial Contributing to Jaspersoft Studio and building from sources.

1.6 Report Structure in Jaspersoft Studio

A report is defined by means of a type page. This is divided into different horizontal portions named bands. When the report is joined with the data generating the print, this section are printed many times according to their function (and according to the rules that the report author has set up). For instance, the page header is repeated at the beginning of every page, while the detail band i repeated for every single read record.

1.6.1 The Report Life Cycle

The report lifecycle starts with report design. Designing a report means creating some sort of template, such as a form containing blank space that can be filled with data. Some portions of a page defined in this way are reused, others stretch to fit the content, and so on.

This template is saved as an XML file sub-type called JRXML (“JR” for JasperReports). It contains all the basic information about the report layout, including complex formulas to perform calculations, an optional query to retrieve data out of a data source, and other functionality covered in later chapters.

The life cycle can be divided into two distinct action sets:

- Tasks executed during the development phase (design and planning of the report, and compilation of a Jasper file source, the JRXML).
- Tasks that must be executed in run time (loading of the Jasper file, filling of the report, and export of the print in a final format).

The main role of Jaspersoft Studio in the cycle is to design a report and create an associated JASPER file, though it is able to preview the result and export it in all the supported formats. Jaspersoft Studio provides support for a wide range of data sources and allows users to test their own data sources, thereby becoming a complete environment for report development and testing.

When you design a report using either iReport or Jaspersoft Studio, you are creating a JRXML file, which is an XML document that contains the definition of the report layout. The layout is completely visual, so you can ignore the underlying structure of the JRXML file. Before executing a report, the JRXML must be compiled in a binary object called a Jasper file. Jasper files are what you need to ship with your application in order to run the reports.

The report execution is performed by passing a Jasper file and a data source to JasperReports. There are many data source types. You can fill a Jasper file from an SQL query, an XML file, a .csv file, an HQL (Hibernate Query Language) query, a collection of JavaBeans, and others. If you don't have a suitable data source, JasperReports allows you to write your own custom data source. With a Jasper file and a data source, JasperReports is able to generate the final document in the format you want.

Jaspersoft Studio also lets you configure data sources and use them to test your reports. In many cases, data-driven wizards can help you design your reports much quicker. iReport includes the JasperReports engine itself to let you preview your report output, test, and refine your reports.

1.6.2 What is a Band

The type page is divided into nine predefined bands to which new groups are added. In addition, Jaspersoft Studio manages a heading band (group header) and a recapitulation band (group footer) for every group.

A band is always wide as the page width (right and left margins excluded). However, its height, even if it is established during the design phase, can vary during the print creation according to the contained elements; it can "lengthen" toward the bottom of a page in an arbitrary way. This typically occurs when bands contain subreports or text fields that have to adapt to the content vertically. Generally, the height specified by the user should be considered "the minimal height" of the band. Not all bands can be stretched dynamically according to the content, in particular the column footer, page footer and last page footer bands.

The sum of all band heights (except for the background) has to always be less than or equal to the page height minus the top and bottom margins.

1.6.3 Band Types

Following there is a brief descriptions of the available bands

Band Name	Description
Title	The title band is the first visible band. It is created only once and can be printed on a separate page. Regarding the allowed dimensions, it is not possible during design time to exceed the report page height (top and bottom margins are included). If the title is printed on a separate page, this band height is not included in the calculation of the total sum of all band heights, which has to be less than or equal to the page height, ass as mentioned previously.
Page Header	The page header band allows you to define a page header. The height specified during the design phase usually does not change during the creation process (except for the insertion of vertically resizable components, such as a text fields that contain long text and subreports). The page header appears on all printed pages in the same position defined during the design phase. Title and summary bands do not include the page header when printed on a separate page.
Column Header	The column header band is printed at the beginning of each detail column (The column concept will be explained in **1.6.5, "Columns," on page 15**). Usually, labels containing the column names of a tabular report are inserted in this band.
Group Header	A report can contains zero or more group bands, which permit the collection of detail records in real groups. A group header is always accompanied by a group footer (both can be independently visible or not). different properties are associated with a group. They determine its behavior from the graphic point of view. It is possible to always force a group header on a new page or in a new column and to print this band on all pages if the bands below it overflow the single page (as a page header, but at group level). It is possible to fix a minimum height required to print a group header: if it exceeds this height, the group header band will be printed on a new page (please note that a value too large for this property can create an infinite loop during printing).
Group Footer	The group footer band completes a group. Usually it contains fields to view subtotals or separation graphic elements, such as lines.
Column Footer	The column footer band appears on at the end of every column. Its dimension are not resizable at run time (not even if it contains resizable elements such as subreports or text fields with a variable number of text lines).
Page Footer	The page footer band appears on every page where there is a page header. Like the column footer, it is not resizable a run time.

Band Name	Description
Last Page Footer	If you want to make the last page footer different from the other footers, it is possible to use the special last page footer band. If the band height is 0, it is completely ignored, and the layout established for the common page will also used for the last page.
Summary	The summary band allows to insert fields concerning total calculations, means, or whatever you want to insert at the end of the report. In other systems, this band is often named report footer.
Background	The background band was introduced after insistent requests from many users who wanted to be able to create watermarks and similar effects (such as a frame around the whole page). It can have a maximum height equal to the page height.

1.6.4 Specifying Report Properties

Now that you have seen the individual parts that comprise a report, you will proceed to creating new one. Click on the **Edit page format** button in the properties view of the report, or right click on the report itself and select **Page Format** from the contextual menu, to see the page properties.

In the dialog that will appear you can change the page dimensions, that probably are the report's most important properties.

A list of standard measure is presented, The unit of measurement used by Jaspersoft Studio and JasperReports is the pixel. However, it is possible to specify report dimension using units of measurement that are more common, such as centimeters, millimeters, or inches. Note that because the dimensions management is based on pixels, some rough adjustments can take place when viewing the same data using different units of measurement. If you are interested in the argument check also the Measure Units Tutorial. In the following table there is a list of the standard measures and their dimensions in pixels.

Page Type	Dimensions in Pixels
Letter	612x792
Note	540x720
Legal	612x1008
A0	2380x3368
A1	1684x3368
A2	1190x1684
A3	842x1190
A4	595x842
A5	421x595
A6	297x421
A7	210x297
A8	148x210
A9	105X148
A10	74X105
B0	2836x4008
B1	2004x2836
B2	1418x2004
B3	1002x1418
B4	709x1002

Page Type	Dimensions in Pixels
B5	501x709
ARCH_E	2592x3456
ARCH_D	1728x2593
ARCH_C	1296x1728
ARCH_B	864x1296
ARCH_A	648x864
FLSA	612x936
FLSE	612x936
HALFLETTER	396x612
11X17	792x1224
LEDGER	1224x792

By modifying width and height, it is possible to create a report of whatever size you like. The page orientations options, Landscape or Portrait, in reality are not meaningful, because the page dimension are characterized by width and height independent of the sheet orientation. However, this property can be used by certain report exporters.The page margin dimensions are set by means of the four options on the Page Margin tab.

1.6.5 Columns

As you have seen, a report is divided into horizontal sections: bands. The page, one or more of which make up a report, presents bands are independent from the data (such as the title or the page footers) and other bands that are printed only if there are one or more data records to print (such as the group headers and the detail band). these last sections can be divided into vertical columns in order to take advantage of the available space on the page. In this context, the concept of a column can be easily confused with that of a field. In fact, a column does not concern the record fields, but it does concern the detail band. This means that if you have record with ten fields and you desire a table view, ten columns are not needed. However, the element will have to be placed correctly to have a table effect. Ten columns will result when long records lists (that are horizontally very narrow) are printed.

Now we will seen how to set up columns in a report with an example. Create a new report from File->New->Jasper Report. Choose as template “BlankA4” and as name “ColumnExample” and as data adapter we can use “Sample DB - Database JBDC Connection” with the following SQL query: select * from orders. Now some fields from the database will be discovered, for our purpose we need only the “SHIPNAME”, so double click on it to add to the report fields and hit “Next”, another time “Next” and “Finish”.

Now from the outline view drag the “SHIPNAME” field in the report in the detail band, resize the detail band and remove the unused band to obtain the result in the following image:

Now go to the Preview tab to see the report compiled:

By default the number of columns is 1, and its width is equal to the entire page, except the margins. The space between columns is zero by default. As you can see in the previous image most of the page is unused. If multiple columns are used, this report would look better. Set from the Page Format dialog the number of columns to two, compile the report and you will see the following result:

In this case, the columns field is set to two to specify the number of columns you want. Jaspersoft Studio will automatically calculate the maximum column width according to the margins and to the page width. If you want to increase the space between the columns, just increase the value of “Space” field. In the following figure there is the same report this time with three columns separated by some blank space:

The restricted area is used to mark every column after the first, to show that all the elements should be placed in the first column and in the others will be replicated automatically during the compilation process. Anyway you are not forced about this, if you want you can also put elements in the other columns, but in most of cases you need only the first. So you shouldn't use parts of the report as margins and columns after the first, which have to be considered as though they were a continuation of the first.

Multiple columns are commonly used for prints of very long lists (for example a phone directory). Functionally, it is important to remember that when you have more than one column, the width of the detail band and of linked bands is reduced to the width of the columns.

The sum of the margins, column widths, and space between columns has to be less or equal to the page width. If this condition is not met, the compilation could result in error. The example report with three columns should appear like in the following image

1.6.6 Advanced Options

From the properties tab of the report there are many other options for the report configuration, now we will see some of this properties. First of all select the report root note from the outline view, and in the properties tab you can see:

- **Report Name**: It is a logical name, independent form the source file's name, and is used only by the JasperReports library (e.g., to name the produced java file when a report is compiled);
- **Title on a new page**: This option specifies that the title band is to be printed on a new page, which forces a page break at the end of the title band. So in the first page only no other band than the title is printed, not event the page header or page footer. However this page is still counted in the total pages numeration;
- **Summary on a new page**: This option is similar to **"**Title on a new page**"** except that the summary band is printed as the last page. Now, if you need to print this band on a new page, the new page will only contain the summary band;
- **Summary with page header and footer**: This option specify if the summary band is accompanied by the page header and the page footer;
- **Floating column footer**: This option allow to force the printing of the column footer band immediately after the last detail band (or group footer) and not at the end of the column. This option is used, for example, when you want to create tables using the report elements;
- **When no data type**: When an empty data is supplied as the print number (or the SQL associated to the report give back no records), an empty file is created (or a a stream of zero byte is given back). This default behavior can be modified by specifying what to do in the case of absence of data. The possible values for this field are:
  - **NoPages**: This is the default value; the final result is an empty buffer;
  - **BlankPage**: This gives back an empty page;
  - **AlSectionNoDetails**: This gives back a page composed of all the bands except for the detail band.

1.7 Measure Units in Jaspersoft Studio

Jaspersoft Studio can handle a wide variety of measure units, including pixels, centimeters, millimeters and inches. To accomplish this, we included a measure component in Jaspersoft Studio. This component looks like a standard text box with a place to enter a measure unit to the right of the value. You can see an example of this widget in the following image:

This component can handle a different measure unit for each field, if needed.

1.7.1 Configuration

You can set two preferred (default) measure units, one at the field level, the other at the report level. The report level measure is used wherever there is not a preferred field measure unit. The report's default measure unit is the pixel.

To change the report level measure:

1. Select Window > Preferences >Jaspersoft Studio >Report Designer.
2. Find the Default Unit where you can set one of the supported measure units, as shown below:
3. Use the drop-down menu to select one of the following measure units:
   - Pixels
   - Inches

- Millimeters
- Centimeters
- Meters

1.7.2 Changing the Field Measure Unit

There are two ways to change a specific field's measure units:

You can insert a new value with the measure unit you want to use in that field. For example, in the figure below, there is an element with size “496 pixel”. By inserting a new value, such as “5 cm”, the new measure unit for this field will be switched to centimeter and saved for the current sections. This means that without closing the report and reopening it (or changing it manually) the measure unit for the width of that element will remain centimeters. If one value is provided without a measure unit it is assumed that it is the default measure of the field, or, if one isn't provided there, the default unit defined at report level. When a value is inserted manually into the field, no spaces are allowed in the numerical value or in the measure unit, but there can be any number of spaces between the value and the unit.

You can also change the local measure unit of a field by double clicking on the measure unit. A pop-up menu appears with all the available measure units listed. Select one of them to set the field's preferred measure, as shown in the following image:

1.7.3 Alias and Autocompletion

Jaspersoft Studio has included alias and autocompletion services for measure units. For instance, if you want to use inches, you can type “inch”, “inches”, or use quote marks. the table below shows the options for entering measure units:

Unit	Accepted Values
centimeter	centimeter, centimeters, cm
millimeter	millimeter, millimeters, mm
meter	meter, meters, m
pixel	pixel, pixels, px
inch	inch, inches, " (double quote)

In this way, it is simpler to provide a unit, and to help, an autocompletion system is provided. After the value is provided, and the first character of a measure unit is be typed, all the available units that start with that character will be suggested with a popup, and could be selected using the keyboard arrow keys or the mouse, as you can see in the image below:

If a typed value, or measure, is not understood, the background of its field will appear red, and the corresponding value or measure will not changed in the report until a correct value is inserted, as you can see in the following image:

For the width, in the figure above, we have an invalid number (the numerical value can have only digits and dot). The height has “cat” as an invalid measure unit.

1.7.4 Approximations

Even if Jaspersoft Studio handled many measure units, JasperReports works only with pixels. Because of this, pixels is the only measure unit that is allowed in the project file. Jaspersoft Studio will approximate measurement and convert them to pixels. For example, if you are using centimeters, “5 cm”, it will be converted to the nearest pixels value. In this case the 5 centimeters will be converted to 139 pixels (about 4.97 cm)

1.8 Exporting reports with Jaspersoft Studio

In addition to generating and viewing reports, Jaspersoft Studio allows you to export reports into many formats, including PDF, XLS, HTML and others. In this tutorial we will go over the steps necessary to export a generated report using Jaspersoft Studio. We'll use the report generated during the tutorial Designing a Report with Jaspersoft Studio.

1.8.1 Compiling the Report

When you switch on the Preview tab in the designer bottom bar, Jaspersoft Studio performs a set of operations to create the final report. The first operation compiles the JRXML source file in a Jasper file. This first step can fail if the elements are not correctly positioned (for example, if an element is placed outside of a band), or if an expression in the report has errors and cannot be compiled.

If the compilation runs successfully, the produced Jasper file is loaded and filled using the active connection or data source. This second operation can also lead to errors. This can happen if the referenced database is not active, an invalid query has been provided, or a null field produced an error in an expression during the filling process. If all operations complete without error, the report is displayed in the integrated viewer. Errors are shown in the Report State window, after clicking the Errors button, as shown in the following image:

If errors occur during the compilation, the tab changes from Preview to Design.

1.8.2 Preview and Exporting

If the compilation completes and there are no errors in the file, the preview is shown. From there you can browse the generated report and change its visualization, change the data source or export the report. Note that after changing the data source the report is recompiled automatically. You can also change the preview format and save the report in different formats.

In the image you can see that there is a green Play button. Clicking this button forces the report to be regenerated; it should be used when a subreport changes, or when you want to execute the report with different input parameters. When you set a preview format, the report is automatically regenerated in the chosen format, and the corresponding viewer application is opened - a PDF viewer or OpenOffice, for example.

CHAPTER 2 BASIC CONCEPTS OF JASPERREPORTS

This chapter illustrates JasperReports's base concepts for a better understanding of how Jaspersoft Studio works.

The JasperReports API, the XML syntax for report definition, and all the details for using the library in your own programs are documented in The JasperReports Ultimate Guide. This guide is available from Jaspersoft. Other information and examples are directly available on the official JasperReports site at http //jasperreports.sourceforge.net.

JasperReports is published under the LGPL license, which is less restrictive a GPL license. JasperReports can be freely used on commercial programs without buying very expensive software licenses and without remaining trapped in the complicated net of open source licenses. This is fundamental when reports created with Jaspersoft Studio have to be used in a commercial product; in fact, programs only need the JasperReports library to produce prints, which work something like a run time executable.

This chapter contains the following sections:

2.1 JRXML Sources and Jasper Files

JasperReports defines a report with an XML file. In previous versions, JasperReports defined the XML syntax with a DTD file (jasperreport.dtd). A JRXML file is composed of a set of sections, some of them concerning the report's physical characteristics, such as the dimension of the page, the positioning of the fields, and the height of the bands; and some of them concerning the logical characteristics, such as the declaration of the parameters and variables, and the definition of a query for data selection.

The following figure shows sample report source code:

Code Example 2-1 A simple JRMXL file example

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
jasperreports.sourceforge.net/jasperreports http://jasperreports.sourceforge.net/
xsd/jasperreport.xsd" name="My first report" pageWidth="595" pageHeight="842"
columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
  <queryString language="SQL">
    <![CDATA[select * from address order by city]]>
  </queryString>
  <field name="ID" class="java.lang.Integer">
    <fieldDescription><![CDATA[]]></fieldDescription>
  </field>
  <field name="FIRSTNAME" class="java.lang.String">
    <fieldDescription><![CDATA[]]></fieldDescription>
  </field>
  <field name="LASTNAME" class="java.lang.String">
    <fieldDescription><![CDATA[]]></fieldDescription>
  </field>
```

Code Example 2-1 A simple JRMXL file example, continued

```
  <field name="STREET" class="java.lang.String">
    <fieldDescription><![CDATA[]]></fieldDescription>
  </field>
  <field name="CITY" class="java.lang.String">
    <fieldDescription><![CDATA[]]></fieldDescription>
  </field>
  <group name="CITY">
    <groupExpression><![CDATA[$F{CITY}]]></groupExpression>
    <groupHeader>
      <band height="27">
        <staticText>
          <reportElement mode="Opaque" x="0" y="0" width="139" height="27"
          forecolor="#FFFFFF" backcolor="#000000"/>
          <textElement>
            <font size="18"/>
          </textElement>
        <text><![CDATA[CITY]]></text>
      </staticText>
      <textField hyperlinkType="None">
        <reportElement mode="Opaque" x="139" y="0" width="416" height="27"
        forecolor="#FFFFFF" backcolor="#000000"/>
        <textElement>
          <font size="18" isBold="true"/>
        </textElement>
        <textFieldExpression class="java.lang.String"><![CDATA[$F{CITY}]]>
        </textFieldExpression>
      </textField>
    </band>
  </groupHeader>
  <groupFooter>
    <band height="8">
      <line direction="BottomUp">
        <reportElement key="line" x="1" y="4" width="554" height="1"/>
      </line>
    </band>
  </groupFooter>
</group>
<background>
  <band/>
</background>
<title>
  <band height="58">
    <line>
      <reportElement x="0" y="8" width="555" height="1"/>
    </line>
    <line>
        <reportElement positionType="FixRelativeToBottom" x="0" y="51" width="555"
        height="1"/>
      </line>
```

Code Example 2-1 A simple JRMXL file example, continued

```
      <staticText>
        <reportElement x=”65” y=”13” width ”424” height=”35”/>
        <textElement textAlignment=”Center”>
          <font size=”26” isBold=”true”/>
        </textElement>
      <text><![CDATE[Classic template]]> </text>
    </staticText>
  </band>
</title>
  <pageHeader>
    <band/>
  </pageHeader>
  <columnHeader>
    <band height="18">
      <staticText>
        <reportElement mode="Opaque" x="0" y="0" width="138" height="18"
        forecolor="#FFFFFF" backcolor="#999999"/>
        <textElement>
          <font size="12"/>
        </textElement>
        <text><![CDATA[ID]]></text>
      </staticText>
      <staticText>
        <reportElement mode="Opaque" x="138" y="0" width="138" height="18"
        forecolor="#FFFFFF" backcolor="#999999"/>
        <textElement>
          <font size="12"/>
        </textElement>
        <text><![CDATA[FIRSTNAME]]></text>
      </staticText>
      <staticText>
        <reportElement mode="Opaque" x="276" y="0" width="138" height="18"
        forecolor="#FFFFFF" backcolor="#999999"/>
        <textElement>
          <font size="12"/>
        </textElement>
        <text><![CDATA[LASTNAME]]></text>
      </staticText>
      <staticText>
        <reportElement mode="Opaque" x="414" y="0" width="138" height="18"
        forecolor="#FFFFFF" backcolor="#999999"/>
        <textElement>
          <font size="12"/>
        </textElement>
        <text><![CDATA[STREET]]></text>
      </staticText>
    </band>
  </columnHeader>
```

Code Example 2-1 A simple JRMXL file example, continued

```
<detail>
  <band height="20">
    <textField hyperlinkType="None">
      <reportElement x="0" y="0" width="138" height="20"/>
      <textElement>
        <font size="12"/>
      </textElement>
      <textFieldExpression class="java.lang.Integer"><![CDATA[$F{ID}]]>
      </textFieldExpression>
    </textField>
    <textField hyperlinkType="None">
      <reportElement x="138" y="0" width="138" height="20"/>
    </textField>
      <textElement>
        <font size="12"/>
      </textElement>
      <textFieldExpression class="java.lang.String"><![CDATA[$F{FIRSTNAME}]]>
      </textFieldExpression>
    <textField hyperlinkType="None">
      <reportElement x="276" y="0" width="138" height="20"/>
      <textElement>
        <font size="12"/>
      </textElement>
      <textFieldExpression class="java.lang.String"><![CDATA[$F{LASTNAME}]]>
      </textFieldExpression>
    </textField>
    <textField hyperlinkType="None">
      <reportElement x="414" y="0" width="138" height="20"/>
      <textElement>
        <font size="12"/>
      </textElement>
      <textFieldExpression class="java.lang.String"><![CDATA[$F{STREET}]]>
      </textFieldExpression>
    </textField>
  </band>
</detail>

<columnFooter>
  <band/>
</columnFooter>
<pageFooter>
  <band height="26">
    <textField evaluationTime="Report" pattern="" isBlankWhenNull="false"
    hyperlinkType="None">
      <reportElement key="textField" x="516" y="6" width="36" height="19"
      forecolor="#000000" backcolor="#FFFFFF"/>
      <textElement>
        <font size="10"/>
      </textElement>
```

Code Example 2-1 A simple JRMXL file example, continued

```
        <textFieldExpression class="java.lang.String"><![CDATA["" +
        $V{PAGE_NUMBER}]]></textFieldExpression>
      </textField>
      <textField pattern="" isBlankWhenNull="false" hyperlinkType="None">
        <reportElement key="textField" x="342" y="6" width="170" height="19"
        forecolor="#000000" backcolor="#FFFFFF"/>
        <box>
          <topPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
          <leftPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
          <bottomPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
          <rightPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
        </box>
        <textElement textAlignment="Right">
          <font size="10"/>
        </textElement>
        <textFieldExpression class="java.lang.String"><![CDATA["Page " +
        $V{PAGE_NUMBER} + " of "]]></textFieldExpression>
      </textField>

      <textField pattern="" isBlankWhenNull="false" hyperlinkType="None">
        <reportElement key="textField" x="1" y="6" width="209" height="19"
        forecolor="#000000" backcolor="#FFFFFF"/>
        <box>
          <topPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
          <leftPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
          <bottomPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
          <rightPen lineWidth="0.0" lineStyle="Solid" lineColor="#000000"/>
        </box>
        <textElement>
          <font size="10"/>
        </textElement>
        <textFieldExpression class="java.util.Date"><![CDATA[new Date()]]>
        </textFieldExpression>
      </textField>
    </band>
  </pageFooter>
  <summary>
    <band/>
  </summary>
</jasperReport>
```

During compilation of the JRXML file (using some JasperReports classes), the XML is parsed and loaded in a JasperDesign object, which is a rich data structure that allows you to represent the exact XML contents in memory. Regardless of which language is used for expressions inside the JRXML, JasperReports creates a special Java class that represents the whole report. The report is then compiled, instanced, and serialized in a JASPER file, ready for loading at any time.

JasperReports's speedy operation is due to all of a report's formulas being compiled into Java-native bytecode and the report structure being verified during compilation instead of at run time. The JASPER file does not contain extraneous resources, such as images used in the report, resource bundles to run the report in different languages, or extra scriptlets and external style definitions. All these resources must be provided by the host application and located at run time.

2.2 Data Sources and Print Formats

Without a means of supplying content from a dynamic data source, even the most sophisticated and appealing report would be useless. JasperReports allows you to specify fill data for the output report in two ways: parameters and data sources. Both kinds of data are presented by means of a generic interface named `JRDataSource`, as shown in **Figure 2-1**.

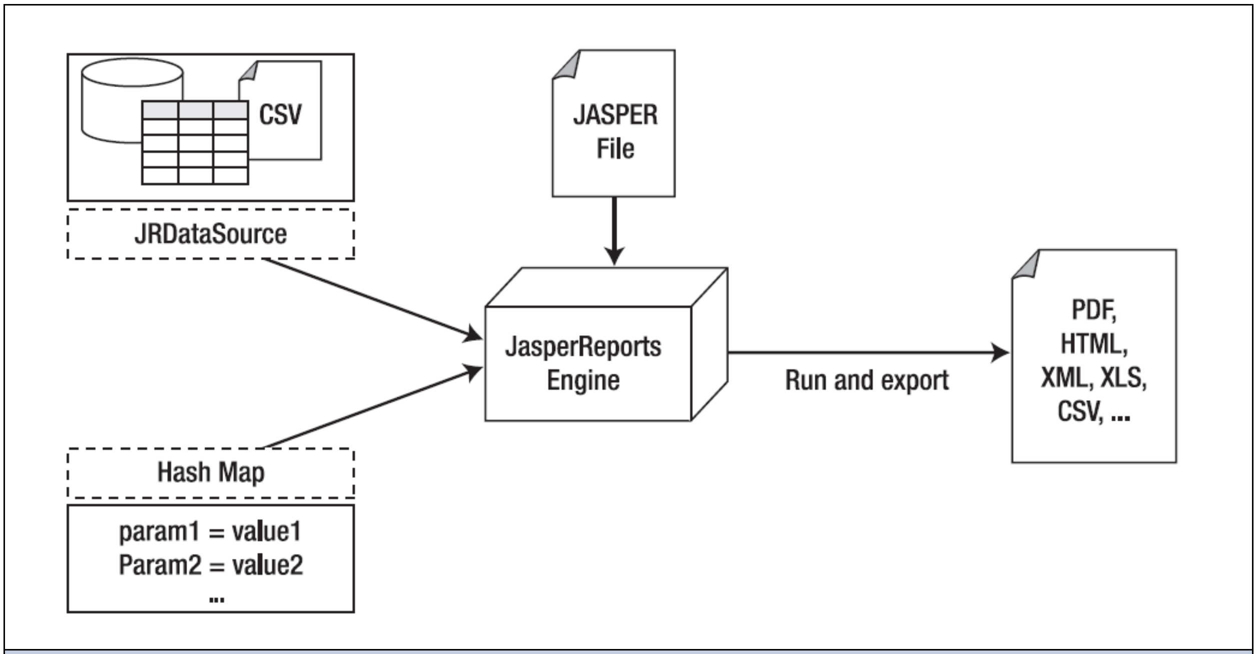


Figure 2-1 Data source and parameter Flows for Report Creation

`JRDataSource` allows a set of records that are organized in tables (rows and columns) to be read. It enables JasperReports to fill a report with data from an explicit data source, using a JDBC connection (already instanced and opened) to whichever relational database you want to run a SQL query on (which is specified in the report).

If the data (passed through a data source) don't meet the requirements of the user, that is, when it is necessary to specify particular values to condition the report's execution, it is possible to produce name/value pairs to "transmit" to the print engine. These pairs are named parameters, and they have to be "preventively declared" in the report. Through `fillManager`, it is possible to join a JASPER file and a data source in a JasperPrint object. This object is a meta-print that can create a real print after you have exported it in the desired format through appropriate classes that implement the `JRExporter` interface.

JasperReports puts at your disposal different pre-defined exporters, such as those for creating files in such formats as PDF, XLS, CVS, XML, RTF, ODF, text, HTML and SWF. Through the `JRViewer` class, you can view the print directly on the screen and print a hardcopy.

2.3 Compatibility Between Versions

When a new version of JasperReports is distributed, some classes usually change. These modified classes typically impact the XML syntax and the JASPER file structure.

Before JasperReports 1.1.0, this was a serious problem and a major upgrade deterrent, since it required recompiling all the JRXML files in order to be used with the new library version. Things changed after the release of Version 1.1.0, after which JasperReports assured backwards compatibility, that is, the library is able to understand and execute any JASPER file generated with a previous version of JasperReports.

With JasperReports 3.1, the JRXML syntax moved from a DTD-based definition to XML-based schema. The XML source declaration syntax now references a schema file, rather than a DTD. Based on what we said previously, this is not a problem since JasperReports assures backwards compatibility. However, many people have been used to designing reports with early versions of Jaspersoft Studio then generating the reports by compiling JRXML in JasperReports. This was always a risky operation, but it was still valid because the user was not using a new tag in the XML. With the move to an XML schema, the JRXML output of Jaspersoft Studio 3.1.1 and newer can only be compiled with a JasperReports 3.1.0 or later.

2.4 Expressions

Many settings in a report are defined using formulas such as conditions to hide an element, special calculations, or text processing, that require some knowledge of a scripting language.

Formulas can be written in at least three languages, two of which (JavaScript and Groovy) can be used without knowledge of programming methods.

All of the formulas in JasperReports are defined through expressions. The default expression language is Java, but if you are not a programmer, we recommend that you design your projects with JavaScript or Groovy because those languages hide a lot of the Java complexity. The language is a property of the document, so, to set it select the document root node in the Outline view and choose your language in the `Language` property in the Properties view.

An expression is a formula that operates on some values and returns a result, such as a formula in a spreadsheet cell. A cell can have a simple value or a complex formula that refers to other values. In a spreadsheet you refer to values contained in other cells, whereas in JasperReports you use the report fields, parameters, and variables. Whatever you have in your expression, when it is computed it gives a value as result (which can be null).

2.4.1 The Type of an Expression

The type of an expression is the nature of the value resulting from it; the type is determined by the context in which the expression is used. For example, if your expression is used to evaluate a condition, the type of the expression should be Boolean (true or false), or if you are creating an expression to display in a textfield, it will probably be a String or a number (Integer or Double). JasperReports requires precision when choosing an expression type.

Some of the most important Java types are:

`java.lang.Boolean`	Defines an Object that represents a boolean value such as true and false
`java.lang.Byte`	Defines an Object that represents a byte
`java.lang.Short`	Defines an Object that represents an short integer
`java.lang.Integer`	Defines an Object that represents integer numbers
`java.lang.Long`	Defines an Object that represents long integer numbers
`java.lang.Float`	Defines an Object that represents floating point numbers
`java.lang.Double`	Defines an Object that represents real numbers
`java.lang.String`	Defines an Object that represents a text
`java.util.Date`	Defines an Object that represents a date or a timestamp
`java.lang.Object`	A generic java Object

If an expression is used to determine the value of a condition that determines, for instance, whether an element should be printed, the return type will be `java.lang.Boolean`; to create it, you need an expression that returns an instance of a Boolean object. Similarly, if an expression shows a number in a textfield, the return type will be `java.lang.Integer` or `java.lang.Double`.

Neither JavaScript nor Groovy is particularly formal about types, since they are not typed languages; the language itself treats a value in the best way by trying to guess the value type or by performing implicit casts (conversion of the type).

2.4.2 Expression Operators and Object Methods

Operators in Java, Groovy and JavaScript are similar because these languages share the same basic syntax. Operators can be applied to a single operand (unary operators) or on two operands (binary operators).

Table 2-1 Expression operators

Operator	Description	Example
`+`	Sum (it can be used to sum two numbers or to concatenate two strings)	A + B
`-`	Subtraction	A - B
`/`	Division	A / B
`%`	Rest, it returns the rest of an integer division	A % B
`||`	Boolean operator OR	A || B
`&&`	Boolean operator AND	A && B
`==`	Equals*	A == B
`!=`	Not equals*	A != B
`!`	Boolean operator NOT	!A

* In Java the == operator can only be used to compare two primitive values. With objects, you need to use the special method “equals”; for example, you cannot write an expression like “test” == “test”, you need to write “test”.equals(“test”). != can only be used to compare two primitive values, as well.

Table 2-1 shows a number of operators, but it is not a complete list. For example, there is a unary operator to add 1 to a variable (++), but it is easier to use x + 1.

Within an expression, you can use the syntax that’s summarized in Table 2-2 to refer to the parameters, variables, and fields defined in the report.

Table 2-2 Syntax for referring to report objects

Syntax	Description
`$F{name_field}`	Specifies the `name_field` field ("F" means field).
`$V{name_variable}`	Specifies the `name_ variable` variable.
`$P{name_parameter}`	Specifies the `name_parameter` parameter.
`$P!{name_parameter}`	Special syntax used in the report SQL query to indicate that the parameter does not have to be dealt as a value to transfer to a prepared statement, but that it represents a little piece of the query.
`$R{resource_key}`	Special syntax for localization of strings.

In summary, fields, variables and parameters represent objects, and you specify their type when you declare them within a report.

Although expressions can be complicated, usually it is a simple operation that returns a value. There is a simple if-else expression that is very useful in many situations. An expression is just an arbitrary operation that any stage must represent a value. In Java, these operators can be applied only to primitive values, except for the sum operator (+). The sum operator can be applied to a String expression with the special meaning of “concatenate”. For example:

```
$F{city} + “, ” + $F{state}
```

will result in a string like this:

```
San Francisco, California
```

Any object in an expression can include methods. A method can accept zero or more arguments, and it can return or not a value. In an expression you can use only methods that return a value; otherwise, you would have nothing to return from your expression. The syntax of a method call is:

```
Object.method(argument1, argument2, <etc.>)
```

Some examples:

Expression	Result
`“test”.length()`	`4`
`“test”.substring(0, 3)`	`“tes”`
`“test”.startsWith(“A”)`	`false`
`“test”.substring(1, 2).startsWith(“e”)`	`true`

All the methods of each object are usually explained in a set of documents called Javadocs that are freely available on the Internet.

You can use parentheses to isolate expressions and make the overall expression more readable.

2.4.3 Using an If-Else Construct in an Expression

A way to create an if-else-like expression is by using the special question mark operator. For example:

```
(($F{name}.length() > 50) ? $F{name}.substring(0,50) : $F{name})
```

The syntax is `(<condition>) ? <value on true> : <value on false>`. It is extremely useful, and can be recursive, meaning that the value on `true` and `false` can be represented by another expression which can be a new condition:

```
(($F{name}.length() > 50) ?
(($F{name}.startsWidth(“A”)) ? “AAAA” : “BBB”)
    :
      $F{name})
```

This expression returns the String “AAAA” when the value of the field name is longer than 50 characters and starts with A, returns BBB if it is longer than 50 characters but does not start with A, and, finally, returns the original field value if neither of these conditions is true.

Despite the possible complexity of an expression, it can be insufficient to define a needed value. For example, if you want to print a number in Roman numerals or give back the name of the weekday of a date, it is possible to transfer the elaborations to an external Java class method, which must be declared as static, as shown in the following example:

```
MyFormatter.toRomanNumber( $F{MyInteger}.intValue() )
```

The function operand `toRomanNumber` is a static method of the `MyFormatter` class, which takes an `int` as argument (the conversion from Integer to `int` is done by means of the `intValue()` method; it is required only when using Java as language) and gives back the Roman version of a number in a lace.

This technique can be used for many purposes; for example, to read the text from a CLOB field or to add a value into a HashMap (a convenient Java object that represents a set of key/value pairs).

2.5 Using Java as a Language for Expressions

Java was the first language supported by JasperReports and is still the most commonly-used language as well as being the default.

Following are some examples of Java expressions:

- `“This is an expression”`
- `new Boolean(true)`
- `new Integer(3)`

- `(($P{MyParam}.equals("S")) ? "Yes" : "No")`

The first thing to note is that each of these expressions represents a Java Object, meaning that the result of each expression is a non-primitive value. The difference between an object and a primitive value makes sense only in Java, but it is very important: a primitive value is a pure value like the number 5 or the Boolean value true. Operations between primitive values have as a result a new primitive value, so the expression:

```
5+5
```

results in the primitive value 10. Objects are complex types that can have methods, can be null, and must be “instanced” with the keyword “new” most of the time. In the second example above, for instance (`new Boolean(true)`), we must wrap the primitive value `true` in an object that represents it.

By contrast, in a scripting language such as Groovy and JavaScript, primitive values are automatically wrapped into objects, so the distinction between primitive values and objects wanes. When using Java, the result of our expression must be an object, which is why the expression 5+5 is not legal as-is but must be fixed with something like this:

```
new Integer( 5 + 5 )
```

The fix creates a new object of type `Integer` representing the primitive value 10.

So, if you use Java as the default language for your expressions, remember that expressions like the following are not valid:

- `3 + 2 * 5`
- `true`
- `(($P{MyParam} == 1) ? "Yes" : "No")`

These expressions don’t make the correct use of objects. In particular, the first and the second expressions are not valid because they are of primitive types (`integer` in the first case and `boolean` in the second case) which do not produce an object as a result. The third expression is not valid because it assumes that the `MyParam` parameter is a primitive type and that it can be compared through the == operator with an `int`, but it cannot. In fact, we said that parameters, variables, and fields are always objects and primitive values cannot be compared or used directly in a mathematical expression with an object.

Since JasperReports is compiled to work with Java 1.4, the auto-boxing functionality of Java 1.5, that would in some cases solve the use of objects as primitive values and vice versa, is not leveraged.

2.6 Using Groovy as a Language for Expressions

The modular architecture of JasperReports provides a way to plug in support for languages other than Java. By default, the library supports two additional languages: Groovy and JavaScript.

Groovy is a full language for the Java 2 Platform. Inside the Groovy language you can use all classes and JARs that are available for Java. **Table 2-3** compares some typical JasperReports expressions written in Java and Groovy.

Table 2-3 Groovy and Java code samples

Expression	Java	Groovy
Field	`$F{field_name}`	`$F{field_name}`
Sum of two double fields	`new Double($F{f1}.doubleValue() + $F{f2}.doubleValue())`	`$F{f1} + $F{f2}`
Comparison of numbers	`new Boolean($F{f}.intValue() == 1)`	`$F{f} == 1`
Comparison of strings	`new Boolean($F{f} != null && $F{f}.equals("test"))`	`$F{f} == "test"`

The following is a correct Groovy expression:

```
new JREmptyDataSource($F{num_of_void_records})
```

`JREmptyDataSource` is a class of JasperReports that creates an empty record set (meaning with the all fields set to null). You can see how you can instance this class (a pure Java class) in Groovy without any problem. At the same time, Groovy allows you to use a simple expression like this one:

```
5+5
```

The language automatically encapsulates the primitive value 10 (the result of that expression) in a proper object. Actually, you can do more: you can treat this value as an object of type `String` and create an expression such as:

```
5 + 5+ ”my value”
```

Whether or not such an expression resolves to a rational value, it is still a legal expression and the result will be an object of type `String` with the value:

```
10 my value
```

Hiding the difference between objects and primitive values, Groovy allows the comparison of different types of objects and primitive values, such as the legal expression:

```
$F{Name} == “John”
```

This expression returns true or false, or, again:

`$F{Age} > 18`	Returns true if the `Age` object interpreted as a number is greater than 18.
`“340” < 100`	Always returns false.
`“340”.substring(0,2) < 100`	Always returns true (since the substring method call will produce the string “34”, which is less than 100).

Groovy provides a way to greatly simplify expressions and never complains about null objects that can crash a Java expression throwing a `NullPointerException`. It really does open the doors of JasperReports to people who don’t know Java.

2.7 Using JavaScript as a Language for Expressions

JavaScript is a popular scripting language with a syntax very similar to Java and Groovy. JavaScript has a set of functions and object methods that in some cases differ from Java and Groovy. For example, the method `String.startsWith(...)` does not exist in JavaScript. You can still use Java objects in JavaScript. An example is:

```
(new java.lang.String("test")).startsWith("t")
```

This is a valid JavaScript expression creating e a Java object (in this case a `java.lang.String`) and using its methods.

JavaScript is the best choice for users who have no knowledge of other languages. The other significant advantage of JavaScript is that it is not interpreted at run time, but instead generates pure Java byte-code. As a result, it produces almost the same performance as Java itself.

2.8 Using JasperReports Extensions in Jaspersoft Studio

JasperReports provides several ways to extend its functionality. In general, extensions (like components, fonts, query executors, chart themes, and so on) are packaged in JARs. To use these extensions in Jaspersoft Studio, just add the required JARs to the Jaspersoft Studio classpath. The Jaspersoft Studio classpath is composed of static and reloadable paths. Extensions must be set as static paths, while other objects which don’t require a proper descriptor or special loading mechanism (such as scriptlets and custom data sources) can be reloadable.

2.9 A Simple Program

In conclusion, following is an example of a simple program that shows how to produce a PDF file from a Jasper file using a data source named `JREmptyDataSource`, a utility data source that provides zero or more records without fields. The file test.jasper, referenced in the example, is the compiled version of the code in **Code Example 2-1**.

Code Example 2-2 JasperTest.java

```
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.export.*;
import java.util.*;

public class JasperTest
{
  public static void main(String[] args)
  {
    String fileName = "/devel/examples/test.jasper";
    String outFileName = "/devel/examples/test.pdf";
    HashMap hm = new HashMap();

    try
    {
      JasperPrint print = JasperFillManager.fillReport(
        fileName,
        hm,
        new JREmptyDataSource());

      JRExporter exporter =
        new net.sf.jasperreports.engine.export.JRPdfExporter();


      exporter.setParameter(
        JRExporterParameter.OUTPUT_FILE_NAME,
        outFileName);
      exporter.setParameter(
      JRExporterParameter.JASPER_PRINT,print);
      exporter.exportReport();
      System.out.println("Created file: " + outFileName);
    }
    catch (JRException e)
    {
      e.printStackTrace();
      System.exit(1);
    }
    catch (Exception e)
    {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

Chapter 3 Creating a Simple Report

Jaspersoft Studio is the new Eclipse-based report designer for JasperReports and JasperReports Server. It is a full rewrite of iReport, available as Eclipse plugin, and as a standalone application. Jaspersoft Studio allows you to create sophisticated layouts containing charts, images, subreports, crosstabs and much more. You can access your data through JDBC, TableModels, JavaBeans, XML, Hibernate, CSV, and custom sources, then publish your reports as PDF, RTF, XML, XLS, CSV, HTML, XHTML, text, DOCX, or OpenOffice.

Jaspersoft Studio's primary goal is to provide the features in the well-known Jaspersoft Report Editor, available as a port of iReport. This is only the beginning - having its foundations on the Eclipse platform, Jaspersoft Studio will be a more complete solution allowing users to extend its capabilities and functionality.

This chapter contains the following sections:

- **Creating a New Report**
- **Adding and Deleting Report Elements**
- **Previewing a Report**
- **Creating a Project Folder**

3.1 Creating a New Report

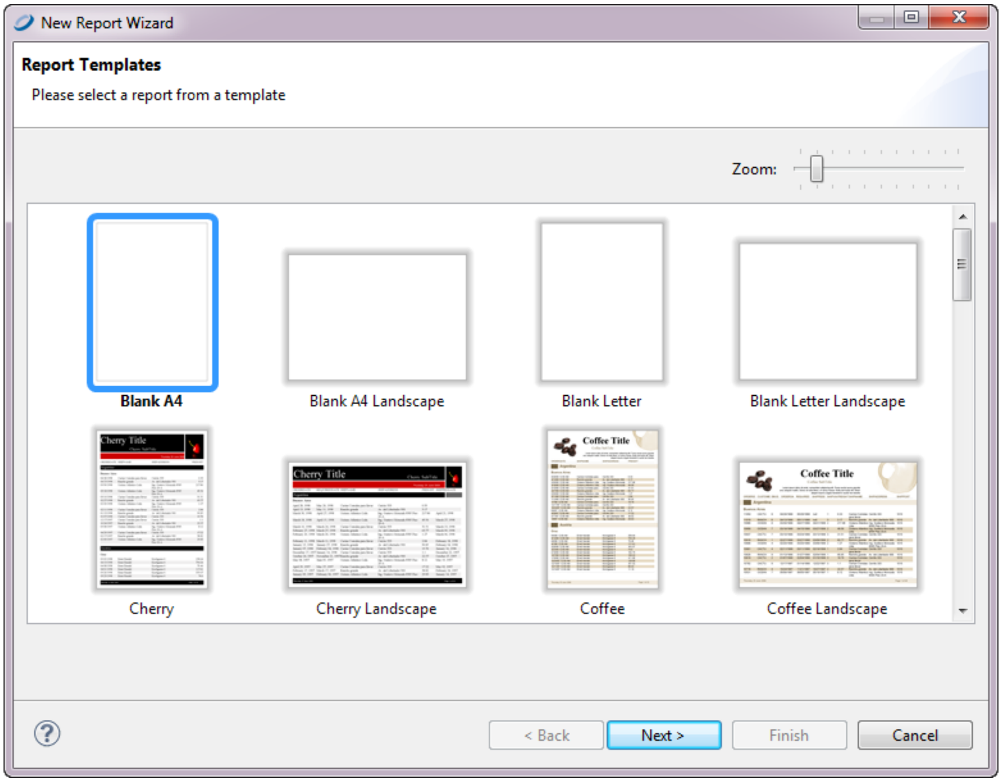
Jaspersoft Studio provides the option to create a report from scratch, add your own templates, or use one of the supplied templates. All of these options allow you to specify your data source. You have three data source options:

- **One Empty Record - Empty rows**: This is default data adapter used to create a report without any data. It can be used to define the layout of a report before connecting it to a data source. A data source can be added or changed at a later time, after the creation of a report.
- **Sample DB - Database JBDC Connection**: This is a SQL database provided with the Jaspersoft Studio installation.
- **Creating a new data adapter**: Jaspersoft Studio supports a wide variety of data sources. Using the **New** button allows you to browse and connect to an existing data source.

To create a new report:

1. Open the **File** menu, select **New**, and then click **Jasper Report**.

   The **New Report Wizard > Report Templates** window appears.



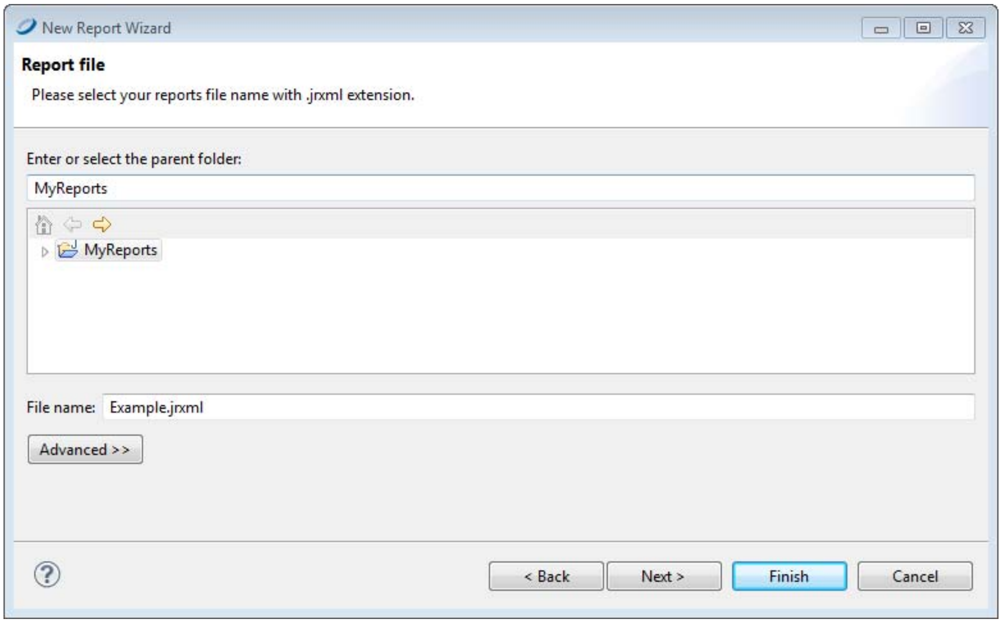
Report Templates

Please select a report from a template

Figure 3-1 New Report Wizard

2. Select Coffee and click **Next**.

The **New Report Wizard > Report file** window appears.



New Report Wizard

Report file

Please select your reports file name with .jrxml extension.

Enter or select the parent folder: MyReports

File name: Example.jrxml

Figure 3-2 New Report Wizard > Report file

3. Select the folder in the workspace where you want to put the report, and name the new report.

   You can use the default folder **MyReports**, name the report use Example,.

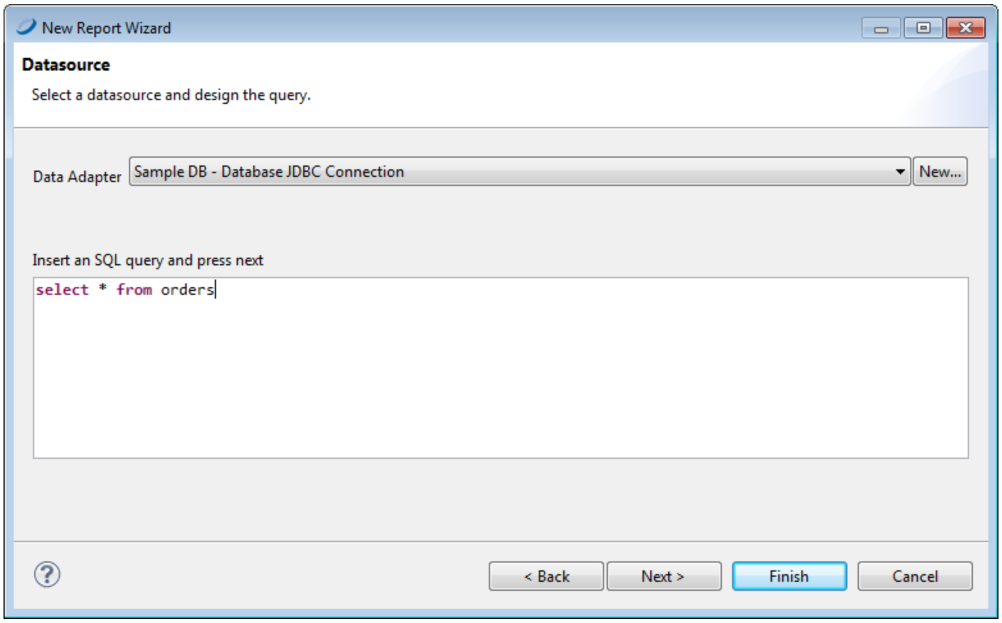
   If you want to create a new project folder for the report, see **"Creating a Project Folder" on page 43**.

4. Click **Next**.

   The **New Report Wizard > Datasource** window appears.

5. Choose **Sample DB - Database JDBC Connection**.

   Enter the query `select * from orders` and click **Next**.



New Report Wizard

Datasource

Select a datasource and design the query.

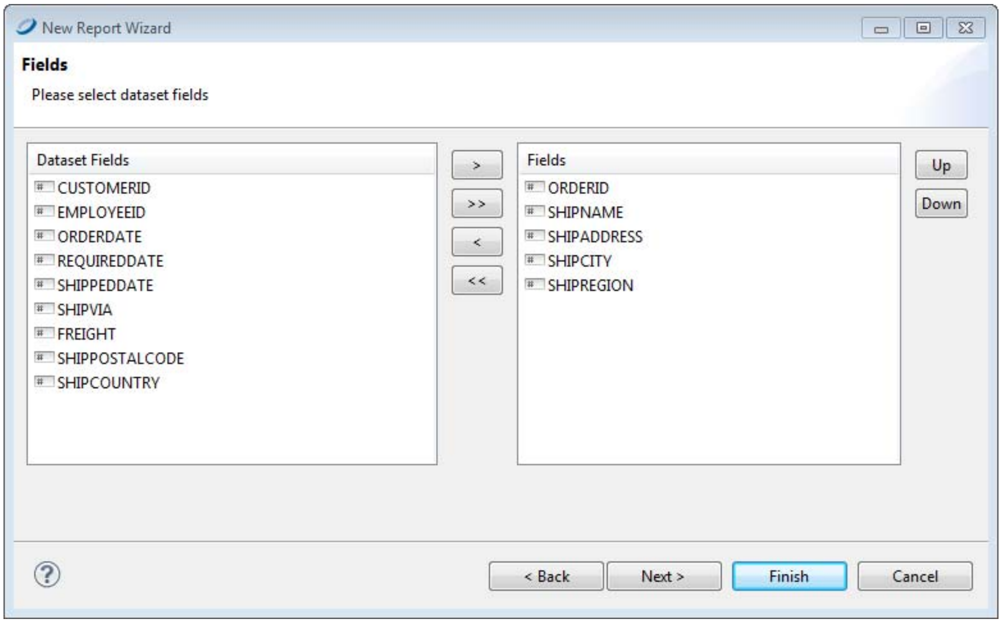
Data Adapter: Sample DB - Database JDBC Connection

Insert an SQL query and press next

```
select * from orders
```

Figure 3-3 New Report Wizard > Datasource

The **New Report Wizard > Fields** window appears.



New Report Wizard

Fields

Please select dataset fields

Dataset Fields
- CUSTOMERID
- EMPLOYEEID
- ORDERDATE
- REQUIREDDATE
- SHIPPEDDATE
- SHIPVIA
- FREIGHT
- SHIPPOSTALCODE
- SHIPCOUNTRY

Fields
- ORDERID
- SHIPNAME
- SHIPADDRESS
- SHIPCITY
- SHIPREGION

Figure 3-4 New Report Wizard > Fields

6. On the left there are all the discovered fields, and on the right the ones that will be added to the report. Select the following fields, then click the right arrow to add them to your report.
   - ORDERID
   - SHIPNAME
   - SHIPADDRESS
   - SHIPCITY
   - SHIPREGION
7. After selecting the fields click **Next**.

   The **New Report Wizard > Grouping** window appears.
8. Click **Next**.
9. Click **Finish**.

   Jaspersoft Studio now builds the report layout with the selected fields included, as shown in **Figure 3-5**:

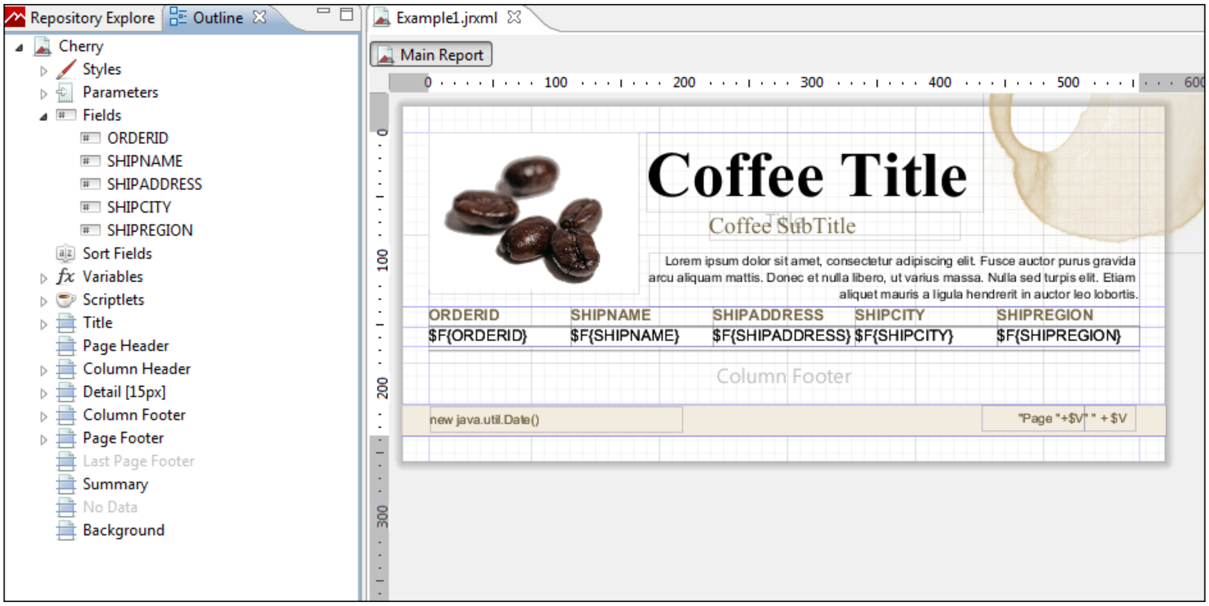


Figure 3-5 New Report in Design View

3.2 Adding and Deleting Report Elements

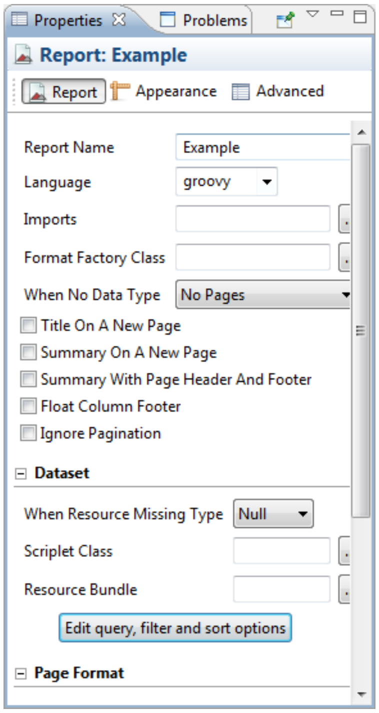
After a report is created, you can add and delete fields and other elements without creating a new report.

3.2.1 Adding Fields to a Report

To add fields to an already created report:

1. Select the main node of the report from the **Outline** view.

2. Select the **Properties** tab.



Report Name: Example

Language: groovy

Imports

Format Factory Class

When No Data Type: No Pages

□ Title On A New Page

□ Summary On A New Page

□ Summary With Page Header And Footer

□ Float Column Footer

□ Ignore Pagination

Dataset

When Resource Missing Type: Null

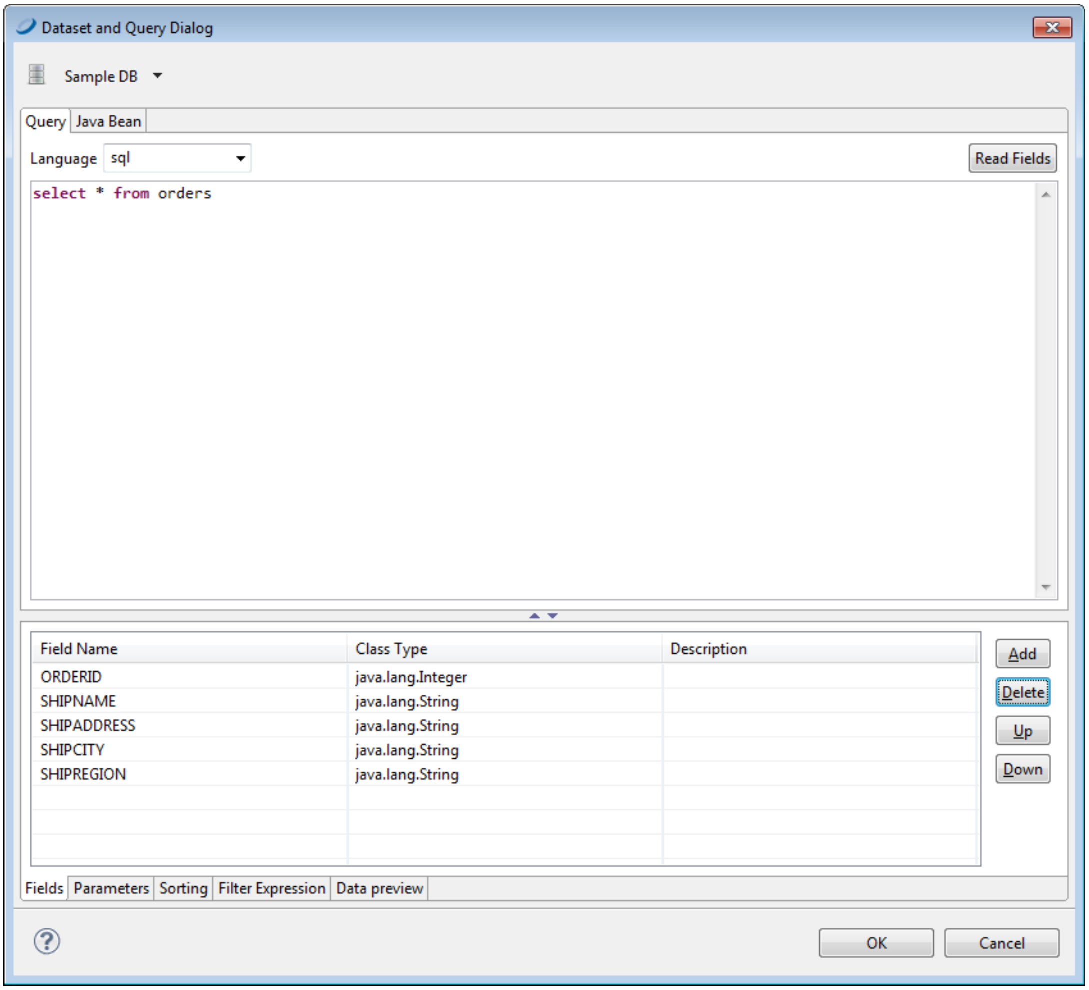
Scriplet Class

Resource Bundle

Figure 3-6 Properties Tab

3. On the **Properties** tab click the **Report** button.
4. Press the **Edit query, filter and sort option** button.

   The **Dataset and Query** dialog opens.



Language: sql

```
select * from orders
```

Field Name	Class Type	Description
ORDERID	java.lang.Integer	
SHIPNAME	java.lang.String	
SHIPADDRESS	java.lang.String	
SHIPCITY	java.lang.String	
SHIPREGION	java.lang.String	

Figure 3-7 Dataset and Query Dialog

5. Add more fields by clicking the **Read Fields** button.

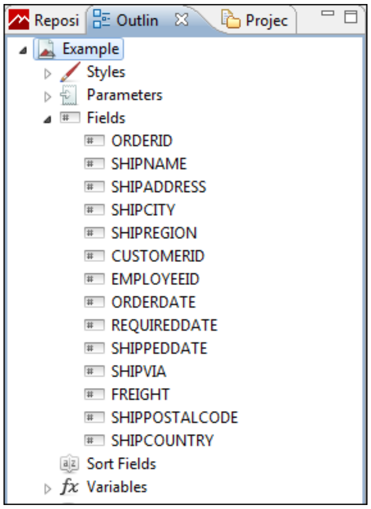
   All the fields discovered are added as new fields in the report.

You can also edit or change your query in the same dialog. If a new query discovers fewer fields than used in the existing report, the fields that are not included the new query will be removed from your report.

6. Click **OK**.

   You return to the **Design** view.

7. Expand **Fields** in the **Outline** view to see all the fields now available for your report.



ORDERID
SHIPNAME
SHIPADDRESS
SHIPCITY
SHIPREGION
CUSTOMERID
EMPLOYEEID
ORDERDATE
REQUIREDDATE
SHIPPEDDATE
SHIPVIA
FREIGHT
SHIPPOSTALCODE
SHIPCOUNTRY

Figure 3-8 Fields

8. To add fields to your report, click the field in the **Outline** view and drag it to the **Design** view.

   When the field object is dragged inside the detail band, Jaspersoft Studio creates a text field element and sets the text field expression for that element.

3.2.2 Deleting Fields

To delete a field from a report: in the **Design** view, right-click the field you wish to remove, and select **Delete**.

3.2.3 Adding Other Elements

To add other elements, such as lines, images or charts, drag the element from the palette shown in the following figure into the **Design** view, then resize and arrange it as desired.

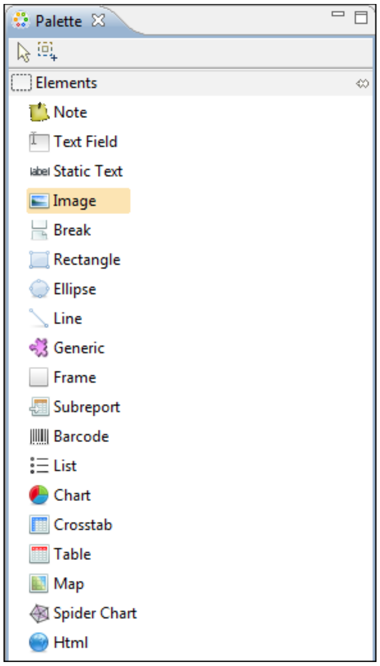
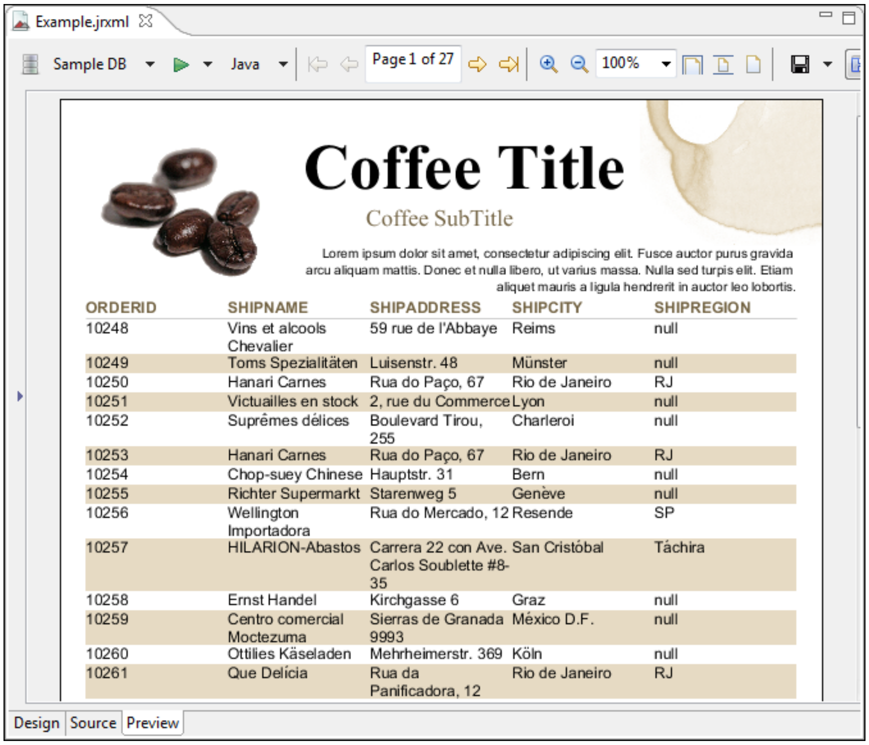


Figure 3-9 Elements Palette

3.3 Previewing a Report

To preview a report, switch to Preview mode by clicking the **Preview** tab (towards the bottom of the screen in **Design** view). The preview compiles the report in the background and fills it with data retrieved by the query through your JDBC connection. As shown in **Figure 3-10**, the Detail band repeats for every row in the query results, creating a simple table report:



Coffee Title

Coffee SubTitle

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce auctor purus gravida arcu aliquam mattis. Donec et nulla libero, ut varius massa. Nulla sed turpis elit. Etiam aliquet mauris a ligula hendrerit in auctor leo lobortis.

ORDERID	SHIPNAME	SHIPADDRESS	SHIPCITY	SHIPREGION
10248	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	null
10249	Toms Spezialitäten	Luisenstr. 48	Münster	null
10250	Hanari Carnes	Rua do Paço, 67	Rio de Janeiro	RJ
10251	Victuailles en stock	2, rue du Commerce	Lyon	null
10252	Suprêmes délices	Boulevard Tirou, 255	Charleroi	null
10253	Hanari Carnes	Rua do Paço, 67	Rio de Janeiro	RJ
10254	Chop-suey Chinese	Hauptstr. 31	Bern	null
10255	Richter Supermarkt	Starenweg 5	Genève	null
10256	Wellington Importadora	Rua do Mercado, 12	Resende	SP
10257	HILARION-Abastos	Carrera 22 con Ave. Carlos Soublette #8-35	San Cristóbal	Táchira
10258	Ernst Handel	Kirchgasse 6	Graz	null
10259	Centro comercial Moctezuma	Sierras de Granada 9993	México D.F.	null
10260	Ottilies Käseladen	Mehrheimerstr. 369	Köln	null
10261	Que Delícia	Rua da Panificadora, 12	Rio de Janeiro	RJ

Figure 3-10 Report Preview

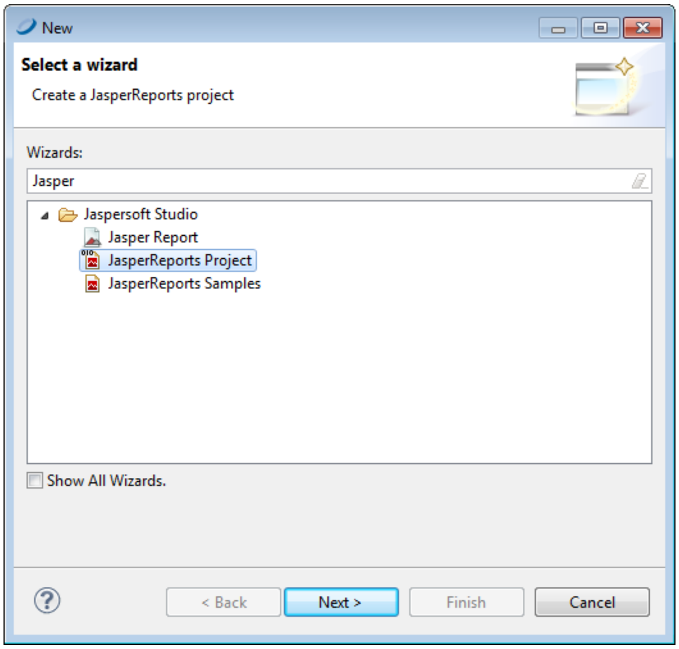
3.4 Creating a Project Folder

Project folders help you organize your reports.

To create a project folder:

1. Choose **File > New > Project**.

   The **Select a wizard** dialog appears.



New

Select a wizard

Create a JasperReports project

Wizards: Jasper

Figure 3-11 Select a Wizard

2. Enter **Jasper** in the Wizards bar to filter all actions. showing only the ones related to Jaspersoft Studio
3. Select **JasperReports Project**. Click **Next**.

   The **New JasperReports Project** wizard appears.
4. Enter the name for your project, and click **Finish**.

   You will see your new project in the **Project Explorer**.

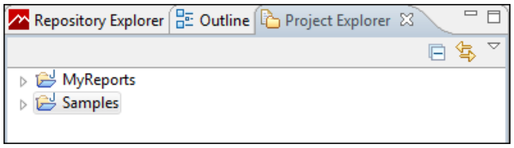


Figure 3-12

Chapter 4 Report Elements

The basic unit of reports is the element. An element is a graphical object, such as a text string or a rectangle. In Jaspersoft Studio, the concept of line or paragraph does not exist, as it does in word processing programs. Everything is created by means of elements, which can contain text, create tables when they are opportunely aligned, and so on. This approach follows the model used by the majority of report authoring tools.

Nine basic elements are offered by the JasperReports library:

- Line
- Rectangle
- Ellipse
- Static text
- Text field (or simply Field)
- Image
- Frame
- Subreport
- Crosstab
- Chart
- Break

Through a combination of these elements, it is possible to produce every kind of report. JasperReports also allows developers to implement their own generic elements and custom components for which it is possible to add support in Jaspersoft Studio to create a proper plug-in.

All elements have common properties, such as height, width, position, and the band to which they belong. Other properties are specific to the type of element (for example, font or, in the case of a rectangle, thickness of the border). There are several types; graphic elements are used to create shapes and display images (they are line, rectangle, ellipse, image); text elements are used to print text strings such as labels or fields (they are static text and text field); the frame element is used to group a set of elements and optionally draw a border around them. Subreports, charts and crosstabs are more complex elements, so I will touch briefly on them later in the this chapter and discuss them in more detail in separate chapters. Finally, there is a special element used to insert a fixed-in-place page or column break.

Elements are inserted into bands, and every element is associated indissolubly with its band. If an element is not completely contained within the band that it is part of, the report compiler will return a message that informs you about the position of the element; the report will be compiled despite such an error, and in the worst case, the out-of-band element will not be printed.

This chapter contains the following sections:

- **Basic Element Attributes**
- **Inserting, Selecting, and Positioning Elements**

- Formatting Elements
- Graphic Elements
- Text Elements
- Using Frames
- Inserting Page and Column Breaks
- Anchors, Bookmarks, and Hyperlinks
- Subreports
- Advanced Elements and Custom Components

4.1 Basic Element Attributes

All the elements have a set of common properties, or attributes; they are presented in the element properties view. These attributes concern information about element positioning on the page. Jaspersoft Studio positions an element relative to the band to which the element belongs (or, more broadly, to its container). The band width is always equal to the document page width minus the left and right margins; its height can change depending on the type of band and the contained elements.

The element types are presented in a palette, usually located in the top right portion of the main window:

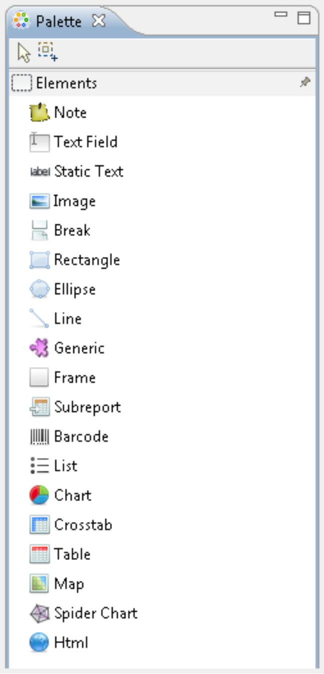


Figure 4-1 Elements in the Designer Palette

Element attributes are divided into categories, visible via tabs in the Properties view. The attributes available depend on the element type.

- The **Appearance** tab allows you to set the location, size, color, and text style of the field.
- The **Borders** tab allows you to set the padding and border style, color, and width of the field.
- The **Static Text** tab allows you to define unchangeable text for a field, and control its appearance.
- The **Text Field** tab allows you to place text in an element.
- The **Inheritance** tab allows you to view any attributes inherited from a lower level, and override those attributes when possible.
- The **Hyperlink** tab allows you to define a hyperlink in an element.
- The **Advanced** tab displays detailed information about the element.

Appearance and Borders are shared among many elements and contain the attributes that define the graphical appearance of the elements, such the width of its borders, the background color, and so on. The Image category is a special one, because it is present only for the elements of the type "Image."

Commonly, the value of an attribute is undefined, and it has a common default value. This means that the element does not have a specific behavior defined, but gets a behavior from somewhere else. For example, the default value of the attribute "background color" is undefined in most of cases, but when a non-transparent element is added to the designer, we can see that it has a white background. The value of the background color attribute is inherited from a lower level.

4.2 Inserting, Selecting, and Positioning Elements

4.2.1 Inserting Elements

To insert an element in a report, drag the element from the palette into a report band. The new element is created and appears in the Report Editing Area.

4.2.2 Selecting Elements

To select the element, click it in the Report Editing Area or in the Outline view. You can adjust the element position by dragging it; to modify its size, select it then drag a corner of the selection frame.

It is possible to select several elements at the same time by drawing a rectangle around the elements with the arrow tool. Depending on the direction in which the rectangle is drawn, elements can be selected only if fully contained in the selected area or partially selected.

Alternatively, it is possible to select more than one element at the same time by holding down the Shift key and clicking all the desired elements.

Specifying a value for a particular property applies that value to all selected elements. However, if two or more elements are selected, only their common properties are displayed in the Properties view. If the values of the properties are different, the value fields will be blank (usually the field is shown empty). To edit properties unique to one element, select only that element.

4.2.3 Positioning Elements

Jaspersoft Studio offers a number of ways to place the elements in your report with precision.

4.2.3.1 Using the Grid

If you need reference points to position and align elements in the page, you can turn on the grid in the design pane by selecting the menu item **View > Show Grid.**

To force the elements to snap to the grid, select **View > Snap to Grid**

4.2.3.2 Using Bands

The top and left values that define the element’s position are always relative to the parent band, or, to be more accurate, to the parent container, which is usually a band but could be a frame element.

If you want to move an element from its initial band to another band or a frame, or vice versa, drag the element node from the Outline view to the new band (or frame) node.

As general rule, an element must remain in its parent band; it should not be moved even partially out of the band. A design error will be displayed in the Report Problems view and the report will not run.

In the Report Editing Area, you can drag an element from one band to another band, but the element’s parent band will not change. Although in general an element must stay in its band, there are several exceptions to this rule. Jaspersoft Studio allows you to move an element anywhere in the report without changing or updating the parent band.

4.2.3.3 Guidelines

When dragging or resizing an element, Jaspersoft Studio suggests places to align it, based on the elements already in the design pane, the band bounds, and (if present) guidelines. When the element you are moving or resizing is in line with another element in the report, a guideline appears, allowing you align the elements.

You can also create guidelines without dragging an element. With the guidelines' magnetic effect, it is easy to place the elements in the right position. To create a guideline, just click a ruler (vertical or horizontal) and drag the resulting guideline to the wanted position.

You can drag and change the position of a guideline at any time; this will have no effect on the element's position.

To remove a guideline, drag it to the top/left corner of the Report Editing Area.

4.2.3.4 The Properties View

You can use the Properties view to edit an element's properties; it is usually located on the right side of the Jaspersoft Studio window. The Properties view is not used just for elements; it can be used to edit the properties of all the components that make up the report, including the page format, the band options, parameters, variables and fields options, and so on. When something is selected in the designer or in the Outline view, the Properties view shows the options for the selected object.

4.3 Formatting Elements

To better organize the elements in the Report Editing Area, a comprehensive set of tools is provided. To access the formatting tools, right-click the element you want to work on and select the needed tool from the context menu.

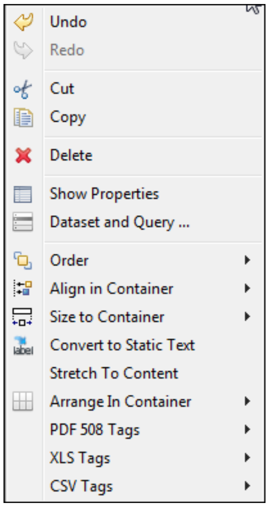


Figure 4-2 Formatting Tools Menu

Each tool is enabled only when the selection matches its minimum requirements (single or multiple selection). The following tables list the available tools, specifying what kind of selection each tool requires (single or multiple selection) and briefly explaining what each tool in each menu option does.

Table 4-1 Order Tools

Icon	Tool Name	Description	Multiple Select?
⧉	Send Backward	Moves the element behind its current layer.	Yes
⧉	Send to Back	Moves the element to the bottom layer.	Yes

Table 4-2 Align in Container Tools

Icon	Tool Name	Description	Multiple Select?
	Align to Left	Aligns the left sides to that of the primary element.	Yes
	Align to Center	Aligns the centers to that of the primary element.	Yes
	Align to Right	Aligns the right sides to that of the primary element.	Yes
	Align to Top	Aligns the top sides (or the upper part) to that of the primary element.	Yes
	Align to Middle	Aligns the middles to that of the primary element.	Yes
	Align to Bottom	Aligns the bottom sides (or the lower part) to that of the primary element.	Yes

Table 4-3 Size Components Tools

Icon	Tool Name	Description	Multiple Select?
	Match Width	Adjusts width to that of primary element.	Yes
	Match Height	Adjusts height to that of primary element.	Yes
	Match Size	Resizes to that of primary element.	Yes

Table 4-4 Size to Container Tools

Icon	Tool Name	Description	Multiple Select?
	Fit to Width	Adjusts elements to fill width of container.	Yes
	Fit to Height	Adjusts elements to fill height of container.	Yes
	Fit to Both	Adjusts elements to fill width and height of container.	Yes

Table 4-5 Arrange in Container Tools

Icon	Tool Name	Description	Multiple Select?
	Horizontal Layout	Centers selected elements vertically.	Yes
	Vertical Layout	Centers selected elements horizontally	Yes

Table 4-6 Miscellaneous Tools

Tool Name	Description	Multiple Select?
Maximize Band Height	Increases band height to the maximum possible.	N/A
Stretch to Content	Resizes element to fit the content	N/A

Tool Name	Description	Multiple Select?
PDF 508 Tags		
XLS Tags		

4.4 Graphic Elements

Graphic elements are drawing objects such as line and rectangle; they do not show data generally, but they are used to make prints more readable and agreeable from an aesthetic point of view. Graphic elements are added, like most elements, by dragging from the Designer palette to the Report Editing Area.

4.4.1 Line

In Jaspersoft Studio, a line is defined by a rectangle for which the line represents the diagonal. By default, the foreground color is used as the default color and a normal 1-pixel-width line is used as the line style.

You can customize the look, style, and direction of the line in the element's Properties view.

4.4.2 Rectangle and Ellipse

The rectangle element is usually used to draw frames around other elements. By default, the foreground color setting is used and a normal 1 pixelwidth

The ellipse is the only element that has no attributes specific to it. The ellipse is drawn in a rectangle that defines the maximum height and width. By default, the foreground color is used, and a normal 1-pixel-width line is used as line style. The background is filled with the background color setting if the element has not been defined as transparent.

4.4.3 Images

An image is the most complex of the graphic elements. It can be used to insert raster images (such as GIF, PNG and JPEG images) in the report, but it can be also used as a canvas object to render, for example, a Swing component, or to leverage some custom rendering code.

When you drag an image element from the Palette into the Report Editing Area, the **Create new image element** dialog appears. This is the most convenient way to specify an image to use in the report. Jaspersoft Studio will not save or store the selected image anywhere, it will just use the file location, translating the absolute path of the selected image into an expression to locate the file when the report is executed. The expression is then set as the value for the Image Expression property.

You can add an image by explicitly defining the full absolute path of the image file in your expression. This is an easy way to add an image to the report, but, overall, it has a big impact on the report's portability, since the file may not be found on another machine (for instance, after deploying the report on a web server or running the report on a different computer).

4.4.4 Padding and Borders

For the image and text elements you can visualize a frame or define a particular padding (the space between the element border and its content) for the four sides. Border and padding are specified by selecting the element in the Report Editing Area, and using the Properties view.

In the Properties view, click the Borders option. This includes the following controls:

- Padding, which allows you to define padding widths for each of the four sides, or to apply the same value to all sides.
- Borders, which allows you to select the color, style, and width of the border, as well as choose where the border should appear.

As always, all the measurements are set in pixels.

4.5 Text Elements

There are two elements specifically designed to display text in a report: static text and text field. Static text is used for creating labels or to print static text set at design time, that is not meant to change when the report is generated. That said, in some cases you will still use a text field to print labels too, since the nature of the static text elements prevents the ability to display text dynamically translated in different languages when the report is executed with a specific locale and it is configured to use a resource bundle leveraging the JasperReports internationalization capabilities.

A text field is similar to a static text string, but the content (the text in the field) is provided using an expression (which can be a simple static text string itself). That expression can return several kinds of value types, allowing the user to specify a pattern to format that value. Since the text specified dynamically can have an arbitrary length, a text field provides several options about how the text must be treated regarding alignment, position, line breaks and so on. Optionally, the text field is able to grow vertically to fit the content when required.

By default both text elements are transparent with no border, with a black text color. The most used text properties can be modified using the text tool bar displayed when a text element is selected. Text element properties can also be modified using the Properties view.

Text fields support hyperlinks as well. See **“” on page 55** for more information.

4.5.1 Static Text

The static text element is used to show non-dynamic text in reports. The only parameter that distinguishes this element from a generic text element is the Text property, where the text to view is specified: it is normal text, not an expression, and so it is not necessary to enclose it in double quotes in order to respect the conventions of Java, Groovy, or JavaScript syntax.

4.5.2 Text Fields

A text field allows you to print an arbitrary section of text (or a number or a date) created using an expression. The simplest case of use of a text field is to print a constant string (`java.lang.String`) created using an expression like this:

```
"This is a text"
```

A text field that prints a constant value like the one returned by this expression can be easily replaced by a static field; actually, the use of an expression to define the content of a text field provides a high level of control on the generated text (even if it’s just constant text). A common case is when labels have to be internationalized and loaded from a resource bundle. In general, an expression can contain fields, variables and parameters, so you can print in a text field the value of a field and set the format of the value to present. For this purpose, a text field expression does not have to return necessarily a string (that’s a text value): the `text field expression class name` property specifies what type of value will be returned by the expression. It can be one of the following:

Valid Expression Types		
`java.lang.Object`	`java.sql.Time`	`java.lang.Long`
`java.lang.Boolean`	`java.lang.Double`	`java.lang.Short`
`java.lang.Byte`	`java.lang.Float`	`java.math.BigDecimal`
`java.util.Date`	`java.lang.Integer`	`java.lang.String`
`java.sql.Timestamp`	`java.io.InputStream`	

An incorrect expression class is frequently the cause of compilation errors. If you use Groovy or JavaScript you can choose `String` as expression type without causing an error when the report is compiled. The side effect is that without specifying the right expression class, the pattern (if set) is not applied to the value.

Let’s see what properties can be set for a text field:

`Blank when null`	If set to true, this option will avoid to print the text field content if the expression result is a null object that would be produce the text `"null"` when converted in a string.
`Evaluation time`	Determines in which phase of the report creation the `Text field Expression` has to be elaborated.
`Evaluation group`	The group to which the evaluation time is referred if it is set to `Group`.
`Stretch with overflow`	When it is selected, this option allows the text field to adapt vertically to the content, if the element is not sufficient to contain all the text lines.
`Pattern`	The pattern property allows you to set a mask to format a value. It is used only when the expression class is congruent with the pattern to apply, meaning you need a numeric value to apply a mask to format a number, or a date to use a date pattern.

4.6 Using Frames

A frame is an element that can contain other elements and optionally draw a border around them. Since a frame is a container of other elements, in the document outline view the frame is represented as a node containing other elements.

A frame can contain other frames, and so on recursively. To add an element to a frame, just drag the new element from the palette inside the frame. Alternatively you can use the outline view and drag elements from a band into the frame and so on. The position of an element is always relative to the container position. If the container is a band, the element position will be relative to the top of the band and the left margin. If the container (or element parent) is a frame, the element coordinates will be relative to the top left corner of the frame. Since an element dragged from a container to another does not change its top/left properties, when moving an element from a container to another its position is recalculated based on the new container location.

The advantages of using a frame to draw a border around a set of elements, with respect to using a simple rectangle element, are:

- When you move a frame, all the elements contained in the frame will move in concert.
- While using a rectangle to overlap some elements, the elements inside the rectangle will not treated as overlapped (respect to the frame), so you will not have problems when exporting in HTML (which does not support overlapped elements).
- Finally, the frame will automatically stretch accordingly to its content, and the element `position type` property of its elements will refer to the frame itself, not to the band, making the design a bit easier to manage.

4.7 Inserting Page and Column Breaks

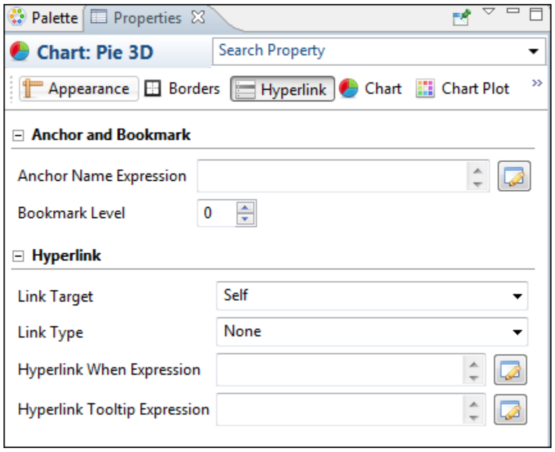
Page and column breaks are used to force the report engine to make a jump to the next page or column. A column break in a single column report has the same effect as a page break.

In the design view they are represented as a small line. If you try to resize them, the size will be reset to the default, this because they are used just to set a particular vertical position in the page (or better, in the band) at which Jaspersoft Studio forces a page or column break.

The type of break can be changed in the Properties view.

4.8 Anchors, Bookmarks, and Hyperlinks

Image, textfield, and chart elements can be used both as anchors into a document and as hypertext links to external sources or other local anchors. To set anchor, bookmark, or hyperlink properties, go to the **Hyperlink** tab in the **Properties** view.



Bookmark Level: 0

Link Target: Self

Link Type: None

Figure 4-3 Anchor, Bookmark, and Hyperlink Properties

4.8.1 Anchors and Bookmarks

An anchor is a kind of label that identifies a specific position in a document. The **Hyperlink** tab includes a text area in which to enter an expression to be the name of the anchor. This name can be referenced by other links. Clicking the ✎ button opens the **Expression Editor**, in which you write or build the expression.

If you plan to export your report as a PDF, set a bookmark level to populate the bookmark tree, making the final document navigation much easier. To make an anchor available in a bookmark, simply choose a bookmark level greater than 1. The use of a greater level makes possible the creation of nested bookmarks.

4.8.2 Hyperlinks

Some types of datasets provide a way to assign a hyperlink to the value represented in the chart, allowing you to open a web page or navigate through the report by clicking the chart.

The click-enabled area depends on the chart type. For example, in pie charts, the hyperlink is linked to each slice of the pie, and in bar charts, the click-enabled areas are the bars themselves.

Image, text field, and chart elements can be used both as anchors into a document and as hypertext links to external sources or local anchors.

To create hyperlinks

1. Click the **Hyperlink** tab in the **Properties** view.
2. In the **Link Target** drop-down, choose one of the following target types:
   - **Self**: This is the default setting. It opens the link in the current window.
   - **Blank**: Opens the target in a new window. Used for output formats such as HTML and PDF.
   - **Top**: Opens the target in the current window but outside eventually frames. Used for output formats such as HTML and PDF.
   - **Parent**: Opens the target in the parent window (if available). Used for output formats such as HTML and PDF.
3. In the **Link Type** drop-down, choose whether the link type is None, Reference, LocalAnchor, LocalPage, RemoteAnchor, RemotePage, or ReportExecution.

   See **"Hyperlink Types" on page 54** for an explanation of the different choices.

4. Click the ✎ button next to **Hyperlink When Expression** to create
5. Click the ✎ button next to **Hyperlink Tooltip Expression** if you want to create a tooltip for your hyperlink.
6. Save your report.

4.8.3 Hyperlink Types

Jaspersoft Studio provides six types of built-in hypertext links: Reference, LocalAnchor, LocalPage, RemoteAnchor and RemotePage. Other types of hyperlinks can be implemented and plugged into JasperReports.

The following table lists the hyperlink types:

Reference	The Reference link indicates an external source that is identified by a normal URL. This is ideal to point, for example, to a servility to manage a record drill-down tools. The only expression required is the hyperlink reference expression.
LocalAnchor	To point to a local anchor means to create a link between two locations into the same document. It can be used, for example, to link the titles of a summary to the chapters to which they refer. To define the local anchor, it is necessary to specify a hyperlink anchor expression, which will have to produce a valid anchor name.
LocalPage	If instead of pointing to an anchor you want to point to a specific current report page, you need to create a LocalPage link. In this case, it is necessary to specify the page number you are pointing to by means of a hyperlink page expression (the expression has to return an Integer object)
RemoteAnchor	If you want to point to a particular anchor that resides in an external document, you use the RemoteAnchor link. In this case, the URL of the external file pointed to will have to be specified in the Hyperlink Reference Expression field, and the name of the anchor will have to be specified in the Hyperlink Anchor Expression field.
RemotePage	This link allows you to point to a particular page of an external document. Similarly, in this case the URL of the external file pointed to will have to be specified in the Hyperlink Reference Expression field, and the page number will have to be specified by means of the hyperlink page expression. Some export formats have no support for hypertext links.
ReportExecution	This type of hyperlink is used to implement JasperServer's drill-down feature.

❗ Some export formats have no support for hypertext links.

4.8.4 Hyperlink Parameters

Sometimes you will need to define some parameters that must be attached to the link. The Link parameters table provides a convenient way to define them. The parameter value can be set using an expression. The parameter expression is supposed to be a string (since it will be encoded in the URL). But when using custom link types it makes sense to set different types for parameters.

4.8.5 Hyperlink Tooltips

The tooltip expression is used to set a text to display as tooltip when the mouse is over the element that represents the hyperlink (this only works when the document is exported in a format that supports this type of interactive use).

4.9 Subreports

The Subreport element is used to include inside a report another report represented by an external Jasper file and feed using the

same database connection used by the parent report or thought a data source that is specified in the subreport properties. `Subreport Expression` This identifies the expression that will return a subreport expression class object at run time. According to the return type, the expression is evaluated in order to recover a Jasper object to be used to produce the subreport. In case the expression class is set to `java.lang.String`, JasperReports will look for a file following the same approach explained for the `Image Expression` of the Image element.

The following briefly describes the characteristics of subreports:

`Subreport Expression`	This identifies the expression that will return a subreport expression class object at run time. According to the return type, the expression is evaluated in order to recover a Jasper object to be used to produce the subreport. In case the expression class is set to `java.lang.String`, JasperReports will look for a file following the same approach explained for the `Image Expression` of the Image element.
`Subreport Expression Class`	This is the class type of the expression; there are several options, each of one subtends to a different way to load the JasperReport object used to fill the subreport.
`Using cache`	This specifies whether to keep in memory the report object used to create the subreport in order to avoid to reload it all the times it will be used inside the report. It is common that a subreport element placed, for instance, into the Detail band is printed more than once (or once for each record in the main dataset). The cache works only if the subreport expression is of type `String` since that string is used as key for the cache.
`Connection/ Datasource Expression`	This identifies the expression that will return at run time a JDBC connection or a JRDataSource used to fill in the subreport. Alternatively the user can choose to avoid to pass any data. This last option is possible and many times it is very useful, but requires some expedient in order to make the subreport to work. Since a subreport (like a common report) will return an empty document if no data are provided, the subreport document should have the document property `When No Data Type` set to something like `All Sections, No Detail`.
`Parameters Map Expression`	This optional expression can be used to produce at run time an object of type `java.util.Map`. The map must be contain a set of coupled names/objects that will be passed to the subreport in order to set a value for its parameters. Nothing disallows to use this expression in order to pass as parameters map to the subreport a map previously passed as parameter to the parent report.
`Subreport parameters`	This table allows you to define some coupled names/expressions that are useful for dynamically set a value for the subreport parameters by using calculated expressions.
`Subreport return values`	This table allows you to define how to store in local variables values calculated or processed in the subreport (such as totals and record count).

To create a subreport:

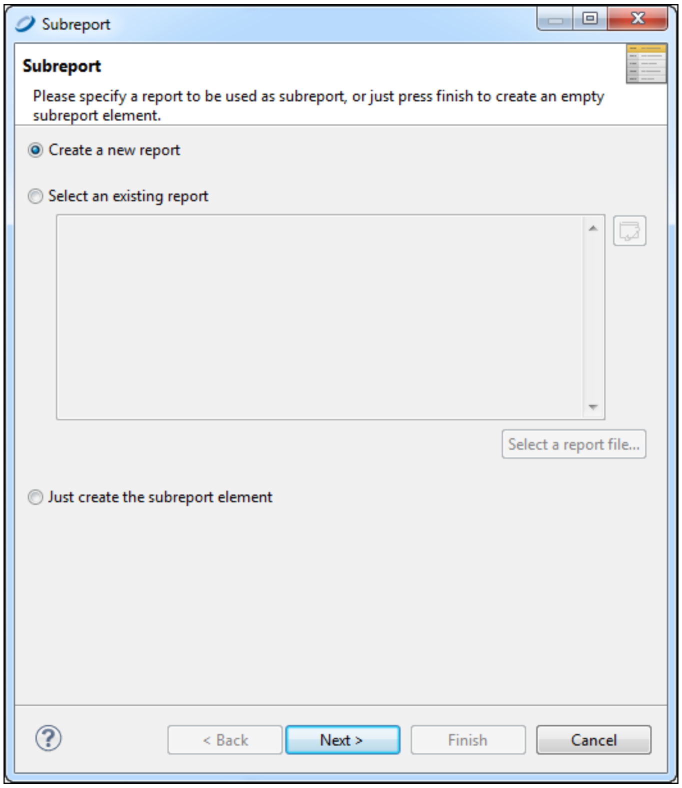
Subreports are useful when your chart does not need all the same data as the main report.

1. Drag the **Subreport** element from the **Palette** to the area of your report in which you will use it.

   The **Subreport** wizard opens. You have three options:

   - **Create a new report** - Use this option when you need to use different data or a new query from your main report.
   - **Select an existing report** - Use this option when you already have a report using the data and query you need.

- **Just create the subreport element** - Use this option to create a placeholder to be used later.



Subreport

Subreport

Please specify a report to be used as subreport, or just press finish to create an empty subreport element.

● Create a new report

○ Select an existing report

Select a report file...

○ Just create the subreport element

< Back | Next > | Finish | Cancel

Figure 4-4 Subreport Wizard

2. Select **Create a new report** and click **Next**.

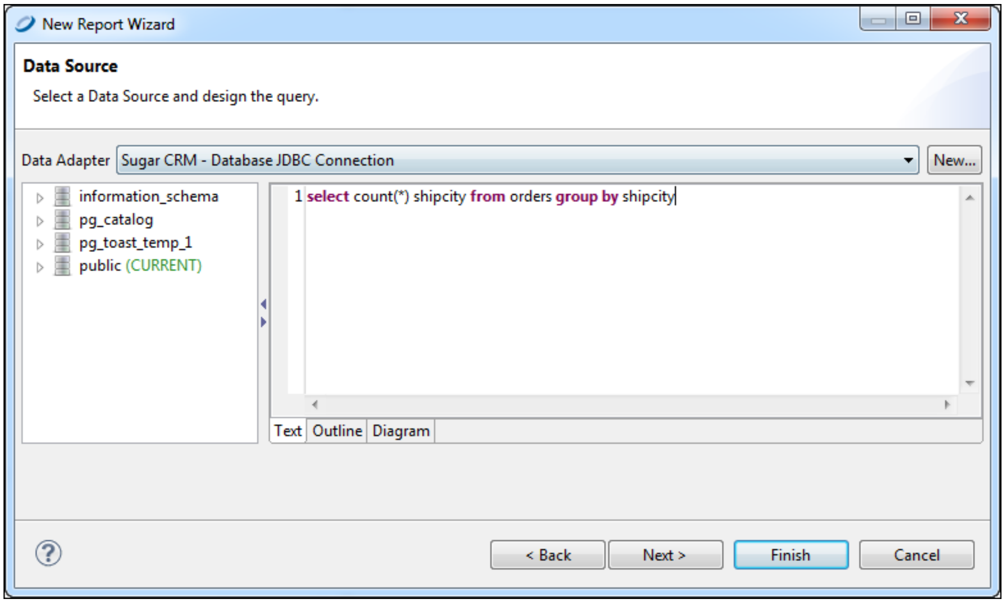
   The **New Report Wizard > Report Templates** window appears.

3. Select a template for your subreport. For this example, select one of the blank templates. Click **Next**.

   The **New Report Wizard > Report file** window opens.

4. Select a location for your subreport, and name it. Click **Next**.

The **Data Source** window opens.



Data Source

Select a Data Source and design the query.

Data Adapter: Sugar CRM - Database JDBC Connection

```
select count(*) shipcity from orders group by shipcity
```

Figure 4-5 Data Source and Query

5. In this step you can choose to use the same data adapter as the main report, or a different data adapter.

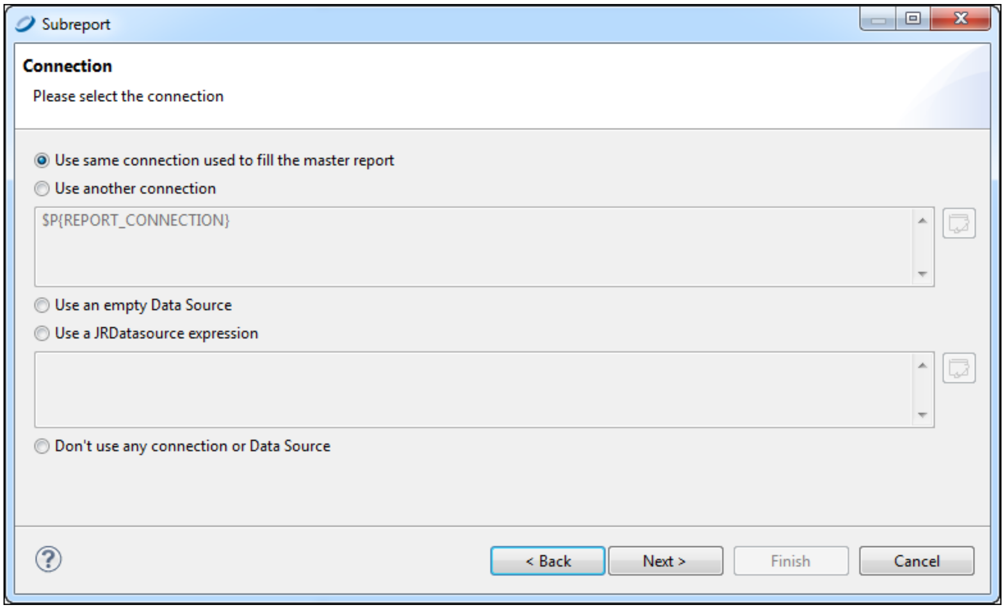
   For this example, choose the same adapter (Sugar CRM). Enter the following SQL statement:

   ```
   select count (*), shipcity from orders group by shipcity
   ```

   Add all the fields to the list on the right. Click **Next**.

6. Click **Next** to skip the **Group By** step.

   The **Subreport > Connection** window opens.



Connection

Please select the connection

● Use same connection used to fill the master report

○ Use another connection

$P{REPORT_CONNECTION}

○ Use an empty Data Source

○ Use a JRDatasource expression

○ Don't use any connection or Data Source

Figure 4-6 Subreport > Connection Window

7. In this window you have a choice of whether to connect to the same database as the main report or to a different database. For this example, click **Use same connection used to fill the master report**. Click **Next**.

The **Subreport Parameters** window opens.

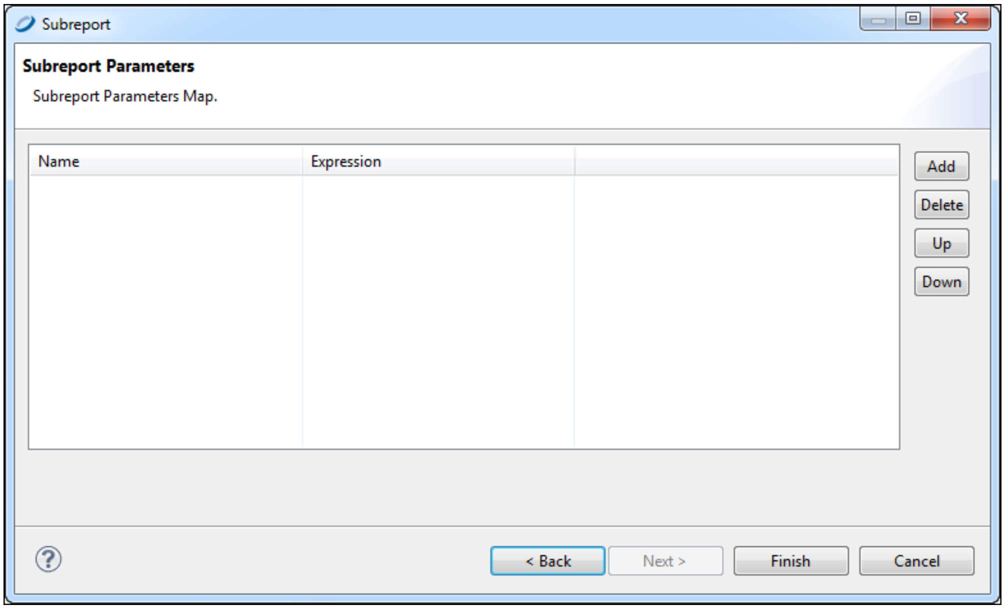


Figure 4-7 Subreport Parameters Window

8. In this window you can add any parameters needed in your subreport. For this example, skip this window and click **Finish**.

   A new report opens, containing all bands.

9. Delete all bands but the Title or Summary band, because those two bands only print once per report. Deleting the other bands eliminates extra white space in your report.

   You now have a location into which to place your table, chart, or other element attached to the new subreport.

4.10 Advanced Elements and Custom Components

Besides the built-in elements seen up to now, JasperReports supports two technologies that enable you to plug-in new JasperReport objects respectively called “custom components” and “generic elements.” Both are supported by Jaspersoft Studio. Without a specific plug-in offered by the custom element provider, there is not much you can do with it; you can just set the common element properties. Therefore, a custom element developer should provide a plug-in for Jaspersoft Studio through which you can, at least, add the element to a report (maybe adding a palette item) and modify the element properties (implementing what is required to display the additional properties in the Properties view when the element is selected.

Chapter 5 Connecting Jaspersoft Studio to JasperReports Server

Connecting Jaspersoft Studio to JasperReports Server works much the same as it did for iReport.

This chapter contains the following sections:

- **Prerequisites**
- **Connecting Jaspersoft Studio to JasperReports Server**
- **Publishing a Report to JasperReports Server**

This section describes functionality that can be restricted by the software license for JasperReports Server. If you don't see some of the options described in this section, your license may prohibit you from using them. To find out what you're licensed to use, or to upgrade your license, contact Jaspersoft.

5.1 Prerequisites

In order to connect Jaspersoft Studio to JasperReports Server, you need:

- URL of the server
- Credentials to access the server

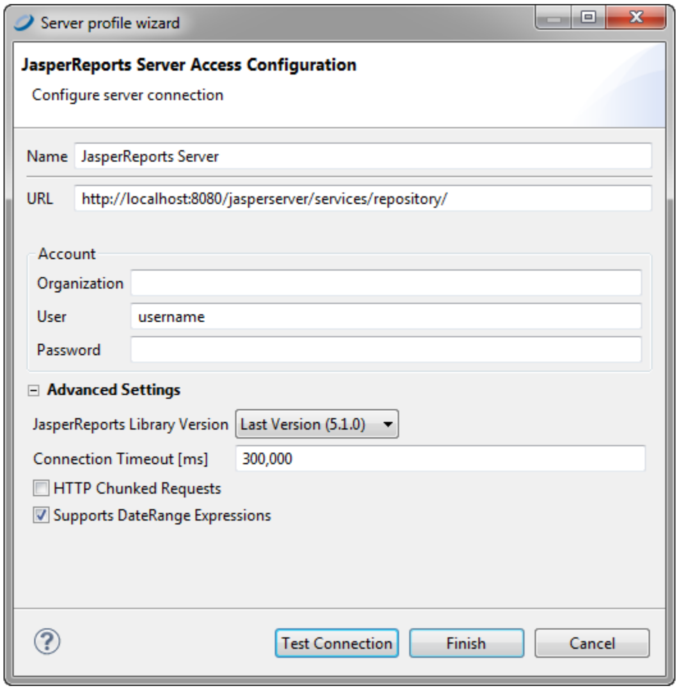
If you do not have both of these things, you can install JasperReports Server locally. You can find the installers on the Community Site.

5.2 Connecting Jaspersoft Studio to JasperReports Server

To connect Jaspersoft Studio to the server:

1. Start Jaspersoft Studio.
2. Click the **Repository Explorer** tab, then click the **Create a JasperReports Server Connection** icon .

The **Server profile wizard** appears.



Server profile wizard

JasperReports Server Access Configuration

Configure server connection

Name: JasperReports Server

URL: http://localhost:8080/jasperserver/services/repository/

Account

Organization:

User: username

Password:

Advanced Settings

JasperReports Library Version: Last Version (5.1.0)

Connection Timeout [ms]: 300,000

☐ HTTP Chunked Requests

☑ Supports DateRange Expressions

Figure 5-1 Server Profile Wizard

3. Enter the URL, user name and password for your server.

   The defaults are:

   - User name: `jasperadmin`
   - Password: `jasperadmin`

4. Click **Test Connection**.

5. If the connection is successful, click **Finish**.

   You will see the server in the **Repository Explorer**.

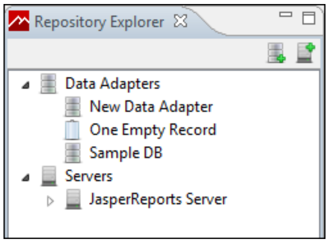


Figure 5-2 Repository Explorer

5.3 Publishing a Report to JasperReports Server

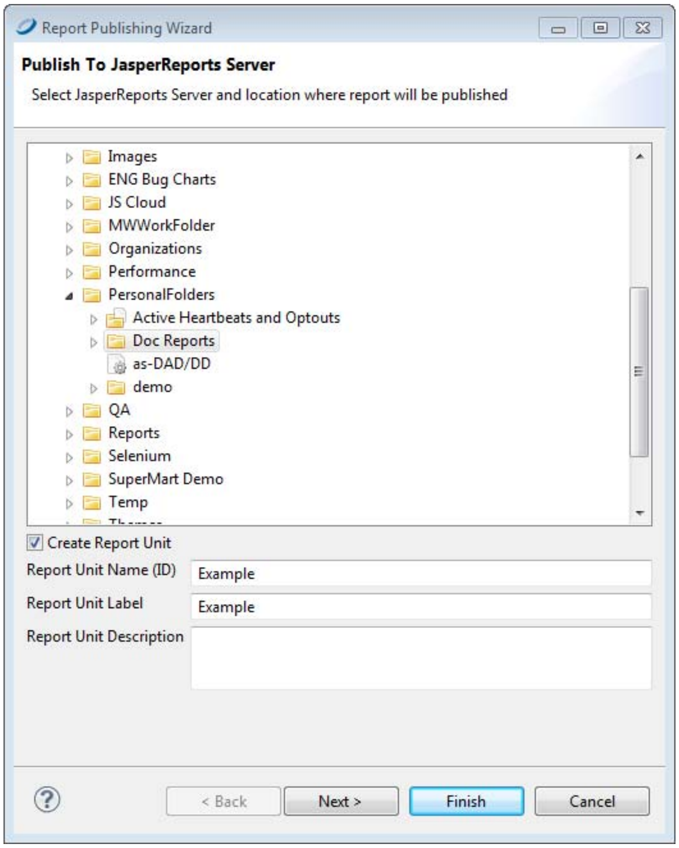
Exporting a report is quite simple, but exporting the data source used to fill the report it isn't so trivial. For exporting the data source we have three options:

- Data source from repository: the server will use a data source already there it to fill the report, so we will only need to browse the server until we find the wanted data source.
- Local data source

- Don't use any data source: only the report is exported without any data source connection

To publish a report to the server:

1. Open a report.
2. Click the **Publish Report** button , located in the upper-right hand corner of the Designer. The **Report Publishing Wizard** opens.



Report Publishing Wizard

Publish To JasperReports Server

Select JasperReports Server and location where report will be published

☑ Create Report Unit

Report Unit Name (ID): Example

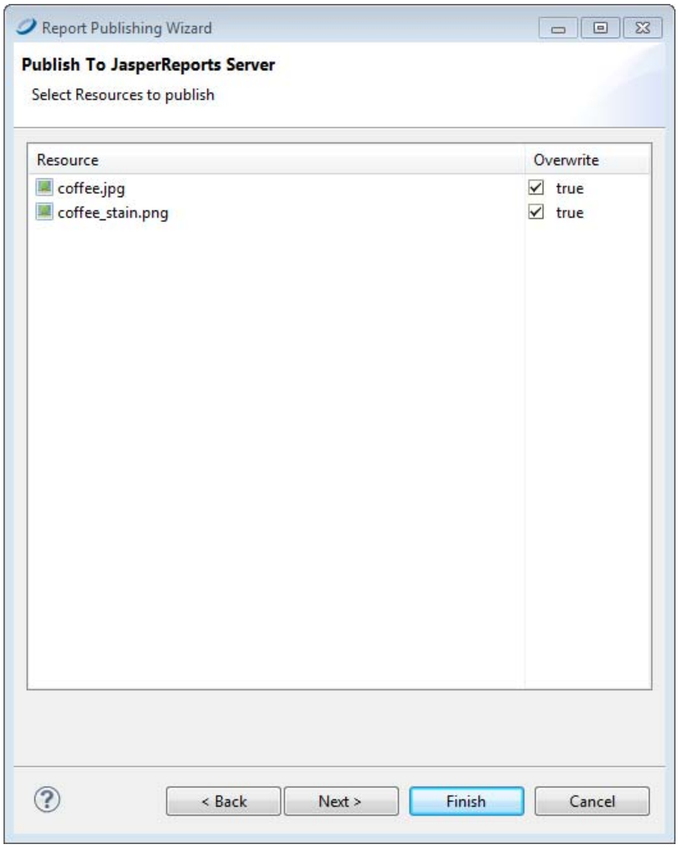
Report Unit Label: Example

Report Unit Description:

Figure 5-3 Report Publishing Wizard

3. Navigate the directory structure and choose the directory where you want to store your report.
4. Name the report unit. The report unit contains all report files. Click **Next**.

The **Select Resources** window opens.



Publish To JasperReports Server

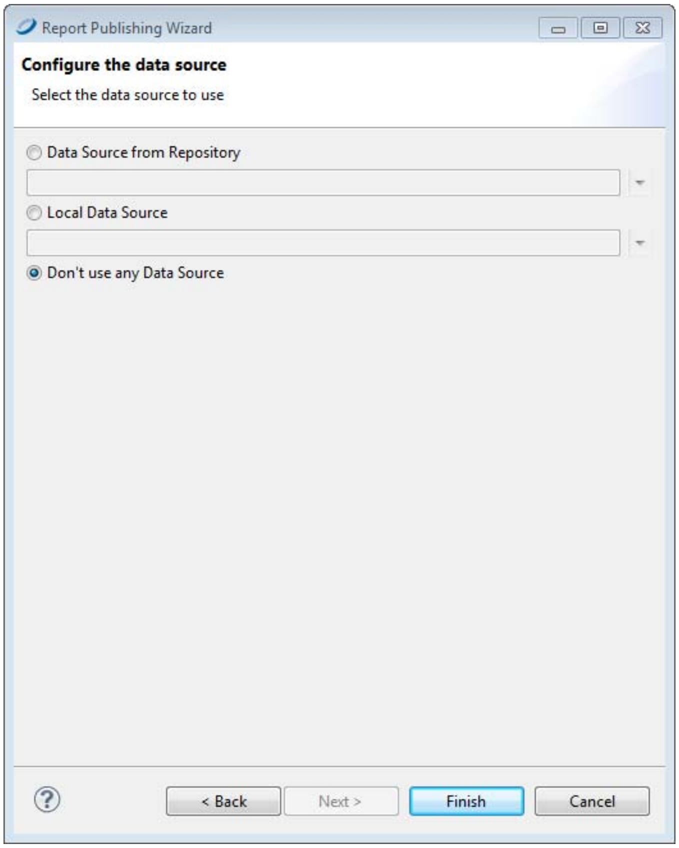
Select Resources to publish

Resource	Overwrite
coffee.jpg	☑ true
coffee_stain.png	☑ true

Figure 5-4 Select Resources

5. Select any resources you want to upload with your report and check the box if you want to overwrite previous versions of those resources. Click **Next**.

The **Configure the data source** window opens.



Report Publishing Wizard

Configure the data source

Select the data source to use

○ Data Source from Repository

○ Local Data Source

● Don't use any Data Source

Figure 5-5 Configure Data Source

6. Select a data source, or no data source. Click **Finish**.

   The report is uploaded to the server. If there are no errors, an appropriate message will be shown.

Chapter 6 Working with Fields

In a report, there are three groups of objects that can store values:

- Fields
- Parameters
- Variables

Jaspersoft Studio uses these objects in data source queries. In order to use these objects in a report, they must be declared with a discrete type that corresponds to a Java class, such as `String` or `Double`. After they have been declared in a report design, the objects can be modified or updated during the report generation process.

This chapter contains the following sections:

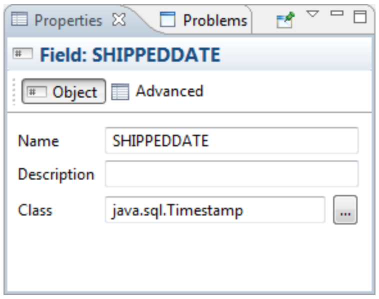
- **Understanding Fields**
- **Registration of Fields from a SQL Query**
- **Registration of JavaBean Fields**
- **Fields and Textfields**

6.1 Understanding Fields

A print is commonly created starting from a data source that provides a set of records composed of a series of fields. This behavior is exactly like obtaining the results of an SQL query.

Jaspersoft Studio displays available fields as children of the **Fields** node in the document outline view. To create a field, right-click the **Fields** node and select **Create Field**. The new field will be included as an undefined entry on the **Properties** tab. You can configure the field properties by selecting it.

Press the **Object** button to name your field, enter a description, and choose a class.



Field: SHIPPEDDATE

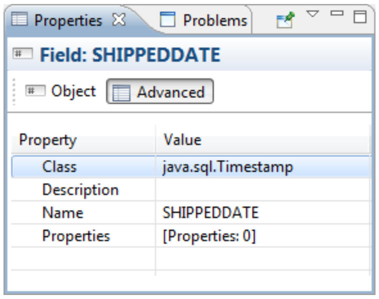
Name: SHIPPEDDATE

Description:

Class: java.sql.Timestamp

Figure 6-1 Field Properties Tab - Object

Press the **Advanced** button to enter advanced properties for the field.



Field: SHIPPEDDATE

Property	Value
Class	java.sql.Timestamp
Description	
Name	SHIPPEDDATE
Properties	[Properties: 0]

Figure 6-2 Field Properties Tab - Advanced

A field is identified by a unique name, a type, and an optional description. You can also define a set of name/value pair properties for each field. These custom properties are not generally used by JasperReports, but they can be used by external applications or by some custom modules of JasperReports (such as a special query executor).

Jaspersoft Studio determines the value for a field based on your data source. For example, when using a SQL query to fill a report, Jaspersoft Studio assumes that the name of the field is the same as the name of a field in the query result set. You must ensure that the field name and type match the field name and type in the data source. You can systematically declare and configure large numbers of fields using the tools provided by Jaspersoft Studio. Since the number of fields in a report can be quite large (possibly reaching the hundreds), Jaspersoft Studio provides different tools for handling declaration fields retrieved from particular types of data sources.

Inside each report expression (like the one used to set the content of a textfield) Jaspersoft Studio specifies a field object, using the following syntax:

```
$F{<field name>}
```

where *<field name>* must be replaced with the name of the field. When using a field expression (for example, calling a method on it), keep in mind that it can have a value of `null`, so you should check for that condition. An example of a Java expression that checks for a `null` value is:

```
($F{myField} != null) ? $F{myFiled}.doSomething() : null
```

This method is generally valid for all the objects, not just fields. Using Groovy or JavaScript this is rarely a problem, since those languages handle a null value exception in a more transparent way, usually returning an empty string.

In some cases a field can be a complex object, like a JavaBean, not just a simple value like a String or an Integer. A trick to convert a generic object to a String is to concatenate it to a empty string this way:

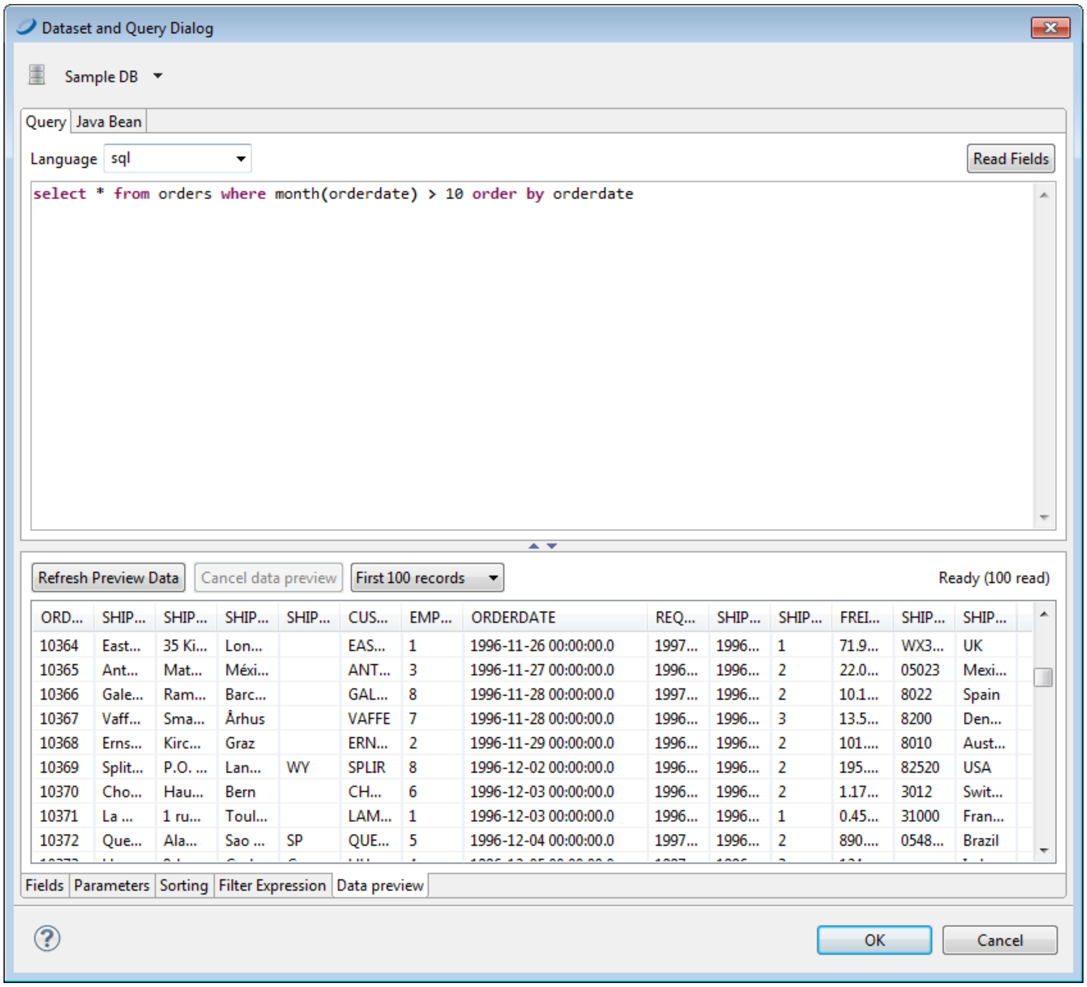
```
$F{myfield}+ “”
```

All Java objects can be converted to a string; the result of this expression depends on the individual object implementation (specifically, by the implementation of the `toString()` method). If the object is null, the result will return the literal text string “`null`” as a value.

6.2 Registration of Fields from a SQL Query

The most common way to fill a report is by using a SQL query. Jaspersoft Studio provides several tools to work with SQL, including a query designer and a way to automatically retrieve and register the fields derived from a query in the report.

Before you open the query dialog, pay attention to the active connection/data source. All operations performed by the tools in the query dialog will use that data source, so ensure that you select the correct connection/data source. Open the query dialog (**Figure 6-3**) by right-clicking the name of your report in the **Outline** view and choosing **Dataset and Query...**.



```
select * from orders where month(orderdate) > 10 order by orderdate
```

ORD...	SHIP...	SHIP...	SHIP...	SHIP...	CUS...	EMP...	ORDERDATE	REQ...	SHIP...	SHIP...	FREI...	SHIP...	SHIP...
10364	East...	35 Ki...	Lon...		EAS...	1	1996-11-26 00:00:00.0	1997...	1996...	1	71.9...	WX3...	UK
10365	Ant...	Mat...	Méxi...		ANT...	3	1996-11-27 00:00:00.0	1996...	1996...	2	22.0...	05023	Mexi...
10366	Gale...	Ram...	Barc...		GAL...	8	1996-11-28 00:00:00.0	1997...	1996...	2	10.1...	8022	Spain
10367	Vaff...	Sma...	Århus		VAFFE	7	1996-11-28 00:00:00.0	1996...	1996...	3	13.5...	8200	Den...
10368	Erns...	Kirc...	Graz		ERN...	2	1996-11-29 00:00:00.0	1996...	1996...	2	101....	8010	Aust...
10369	Split...	P.O. ...	Lan...	WY	SPLIR	8	1996-12-02 00:00:00.0	1996...	1996...	2	195....	82520	USA
10370	Cho...	Hau...	Bern		CH...	6	1996-12-03 00:00:00.0	1996...	1996...	2	1.17...	3012	Swit...
10371	La ...	1 ru...	Toul...		LAM...	1	1996-12-03 00:00:00.0	1996...	1996...	1	0.45...	31000	Fran...
10372	Que...	Ala...	Sao ...	SP	QUE...	5	1996-12-04 00:00:00.0	1997...	1996...	2	890....	0548...	Brazil

Figure 6-3 Query Dialog

Jaspersoft Studio does not need you to define a query in order to generate a report. It can obtain data records from a data source that is not defined by a query execution. JasperReports supports a number of query languages including:

- CQL
- HiveQL
- JSON
- MongoDBQuery
- PLSQL
- SQL
- XLS
- XPath

If the selected data source is a JDBC connection, Jaspersoft Studio will test the access connection to the data source as you define the query. This allows Jaspersoft Studio to identify the fields using the query metadata in the result set. The design tool lists the discovered fields in the bottom portion of the window. For each field, Jaspersoft Studio determines the name and the Java type specified for that field by the JDBC driver.

A query that accesses one or more tables containing a large amount of data may require a long delay while Jaspersoft Studio scans the data source to discover field names. You may want to disable the **Automatically Retrieve Fields** option in order to quickly finish your query definition. When you have completed the query, click the **Read Fields** button in order to start the fields discovery scan.

All fields used in a query must have a unique name. Use alias field names in the query for fields having the same name.

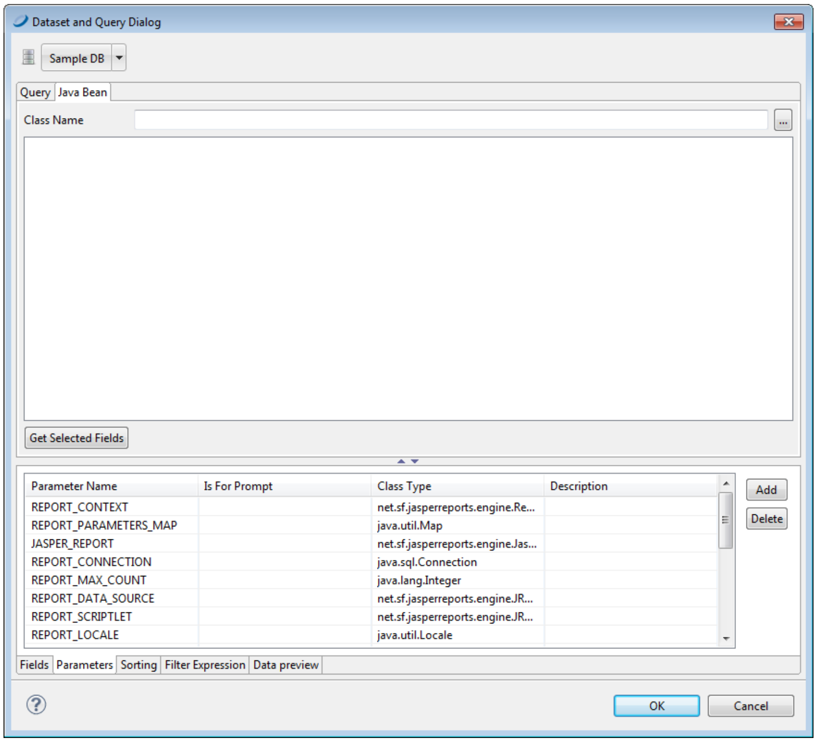
In case of an error during the query execution (due to a syntax error or to an unavailable database connection), an error message will be displayed instead of the fields list.

The field name scan may return a large number of field names if you are working with complex tables. I suggest that you review the list of discovered names and remove any fields that you are not planning to use in your report, in order to reduce unnecessary complexity.When you click the **OK** button all the fields in the list will be included in the report design. You can also remove them later in the outline view, but it’s a good idea at this point in the design process to remove any field names that you won’t be using.

6.3 Registration of JavaBean Fields

One of the most advanced features of JasperReports is the ability to manage data sources that are not based on simple SQL queries. One example of this is JavaBean collections. In a JavaBean collection, each item in the collection represents a record. JasperReports assumes that all objects in the collection are instances of the same Java class. In this case the “fields” are the object attributes (or even attributes of attributes).

By selecting the **Java Bean** tab in the query designer, you can register the fields which correspond to the specified Java classes. The concept here is that you will know which Java classes correspond to the objects that you will be using in your report.



Parameter Name	Is For Prompt	Class Type	Description
REPORT_CONTEXT		net.sf.jasperreports.engine.Re...	
REPORT_PARAMETERS_MAP		java.util.Map	
JASPER_REPORT		net.sf.jasperreports.engine.Jas...	
REPORT_CONNECTION		java.sql.Connection	
REPORT_MAX_COUNT		java.lang.Integer	
REPORT_DATA_SOURCE		net.sf.jasperreports.engine.JR...	
REPORT_SCRIPTLET		net.sf.jasperreports.engine.JR...	
REPORT_LOCALE		java.util.Locale	

Figure 6-4 JavaBeans Tab

Suppose that you are using objects of this Java class:

```
com.jaspersoft.ireport.examples.beans.PersonBean
```

To register fields for the class:

1. Put the class name in the name field and click **Read attributes**. Jaspersoft Studio will scan the class.
2. Check the scan results to make sure that Jaspersoft Studio has captured the correct object attributes for the class type.
3. Select the fields you want to use in your report and click **Add**.
4. Jaspersoft Studio creates new fields corresponding to the selected attributes and adhesion to the list. The description, in this case, will be used to store the method that the data source must invoke in order to retrieve the value for the specified field.

Jaspersoft Studio parses a description such as `address.state` (with a period character between the two attributes) as an attribute path. This attribute path will be passed to the function `getAddress()` in order to locate the target attribute, and then to `getState()` in order to query the status of the attribute. Paths may be arbitrary long, and iReport can recursively parse attribute trees within complex JavaBeans and in order to register very specific fields.

We have just discussed the two tools used most frequently to register fields, but we're not done yet. There are many other tools you can use to discover and register fields, for instance, the HQL and XML node mapping tools.

6.4 Fields and Textfields

To print a field in a text element, you will need to set the expression and the `textfield` class type correctly. You may also need to define a formatting pattern for the field.

To create a corresponding textfield, drag the field you wish to display from the Outline view into the design panel. Jaspersoft Studio will create a new textfield with the correct expression (e.g., `$F{fieldname}`) and assign the correct class name.

Chapter 7 Report Templates

One of the most useful tools of Jaspersoft Studio is the ability to use and create templates. The provided templates can form the base for new reports with no further changes, or they can be used as a model to which fields, text fields and groups can be added in the Report Wizard.

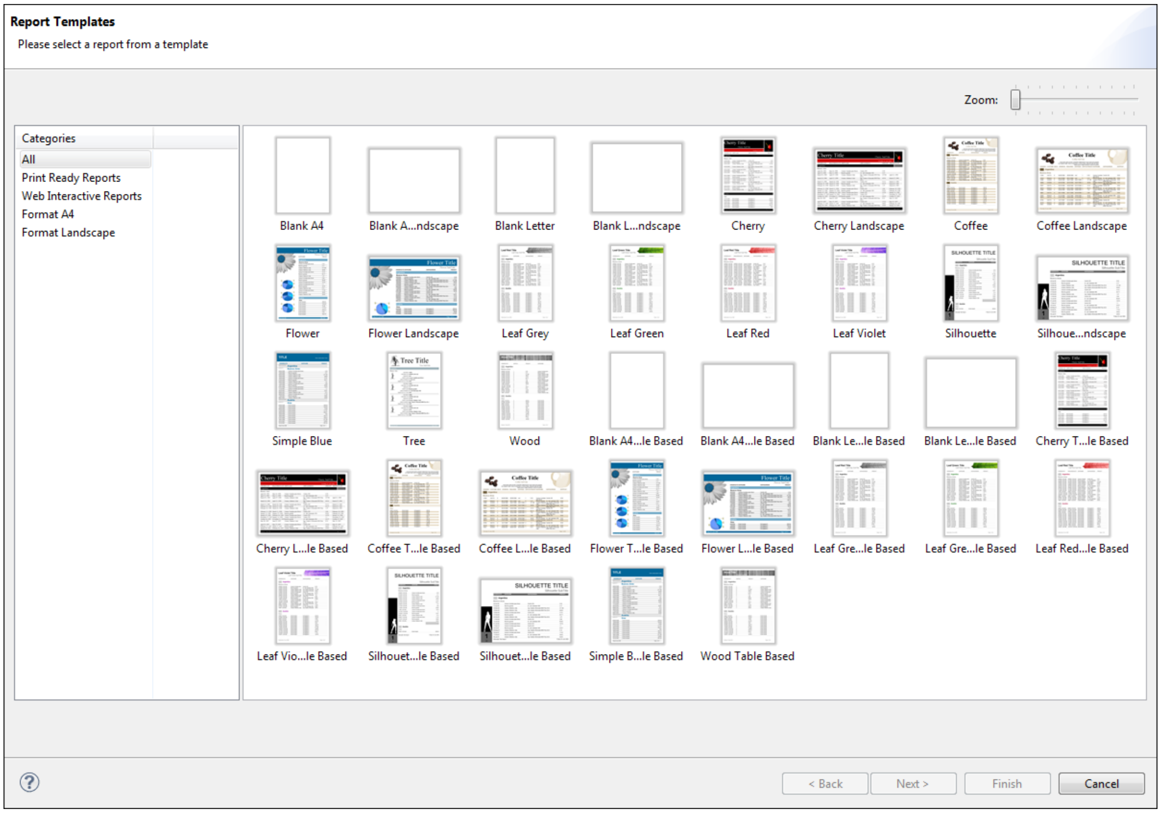
This chapter explains how to build a custom template and add them so that they appear in the Template Chooser. It has the following sections:

- Template Structure
- Creating and Customizing Templates
- Saving Templates
- Adding Templates to Jaspersoft Studio

7.1 Template Structure

A template is a JRXML file. When a new report is created, the JRXML file of the selected template is loaded and modified according to any options you have specified in the wizard steps. If the user chooses not to use the wizard, the selected template is just copied along with all the referenced images in the location the user has specified.

When the **Report Template** dialog box appears, it scans all paths specified as template directories. Any valid JRXML files found are included in the **Report Template** dialog box. If a template provides a preview image, the image will be displayed. Otherwise, you will see a white box.



Report Templates

Please select a report from a template

Figure 7-1 Default Report Templates

A template contains all or some of the same parts as a report. When creating a new template or editing an existing template, it is important to remember the following:

- Every band will have the same size in the template and in the report generated from the template. For example, if your template has a **Detail** band with a height of100 pixels and a **Summary** band with height of 10 pixels then every report generated with this template will have by default a **Detail** band with a height of 100 pixels and a **Summary** band with a height of 10 pixels.
- Page formatting will be the same in a report as it is in the template from which it is generated. For example, if your template is for A4 paper with half-inch margins, any report using that template will be for A4 paper with half-inch margins.
- The content of the **Summary**, **Title**, **Page Header** and **Page Footer** bands, as well as the background, is static. Every element that is placed in these bands in a template will appear in every report that uses this template.
- In the **Column Header** band there should be only a **Static Text** element, and its text content must be `Label`. The appearance, font and the other attributes of this label will be used to create every label that is inserted in this band.
- In the **Group Header** band there should be only a **Text Field** with the string `"GroupField"` (including the double quotes). As with the **Column Header** this will be taken as example to generate every field that goes in this band.
- In the **Detail** band there should be only a **Text Filed** with the string “`Field`” (including the double quotes). Again, this will be taken as example to generate every field that goes in this band.

When you group data using the wizard, the wizard creates all the necessary report structures to produce the requested groups. The Report Wizard permits the creation of up to four groups, and a group header and group footer is associated with each. If the template defines one or more groups and you group the data, the wizard tries to use any existing groups before creating new ones. By default, groups in the template are deleted if they are not used. For each group, the wizard sets the group expression and adds a label for the name and a text field showing the value of the group expression (which is always a field name, since the grouping criteria set using the wizard is one of the selected fields).

7.2 Creating and Customizing Templates

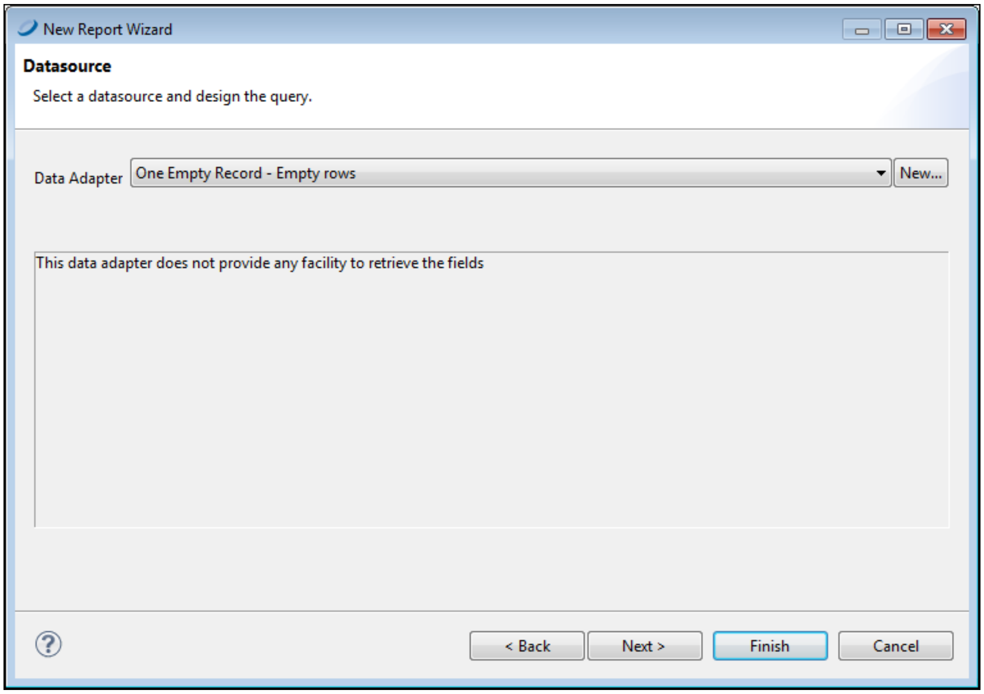
You can create templates from scratch or edit existing templates and save them as new templates.

7.2.1 Creating a New Template

If you want to start fresh with your template, create a new one.

To create a new template:

1. Go to **File > New > Jasper Report**.
2. From the New Report wizard, select a template from which to start. Click **Blank Letter**, and **Next**.
3. Choose where you want to store the file, and name the new template. Click **Next**.
4. For creating a template, there is no need to connect to a data source. Select **One Empty Record - Empty rows** and click **Finish**.



New Report Wizard

Datasource

Select a datasource and design the query.

Data Adapter: One Empty Record - Empty rows

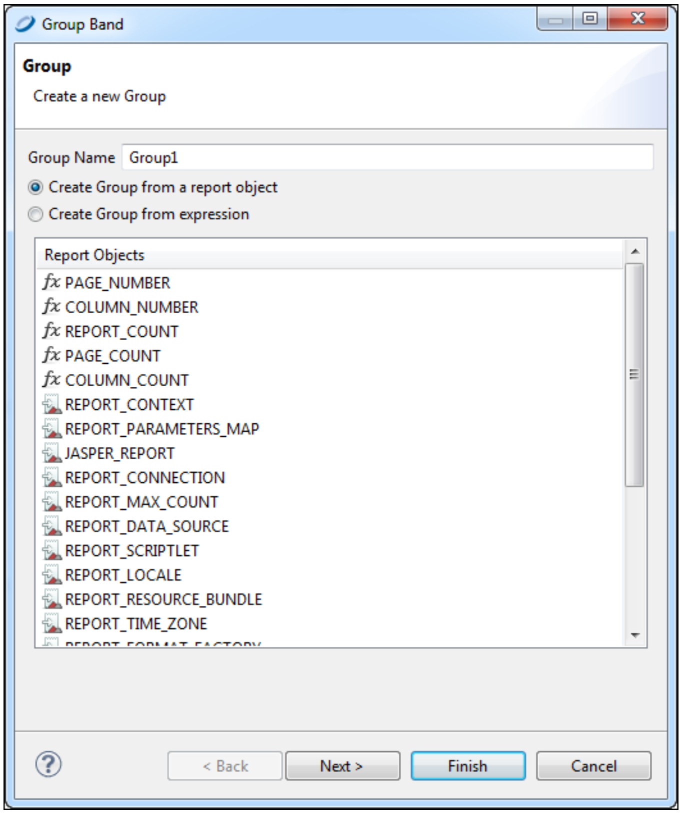
This data adapter does not provide any facility to retrieve the fields

Figure 7-2 One Empty Record Data Source

An empty report opens, containing the following bands:

- Title
- Page Header
- Column Header
- Detail
- Column Footer
- Page Footer
- Summary

5. Right-click on the report root node in the **Outline**, and select **Create Group**.



Group Band

Group

Create a new Group

Group Name: Group1

● Create Group from a report object

○ Create Group from expression

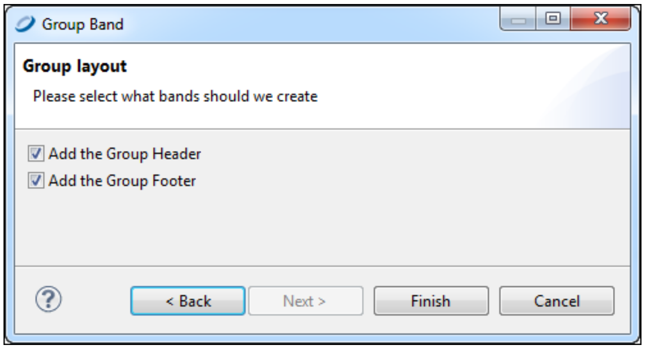
Report Objects

- *fx* PAGE_NUMBER
- *fx* COLUMN_NUMBER
- *fx* REPORT_COUNT
- *fx* PAGE_COUNT
- *fx* COLUMN_COUNT
- REPORT_CONTEXT
- REPORT_PARAMETERS_MAP
- JASPER_REPORT
- REPORT_CONNECTION
- REPORT_MAX_COUNT
- REPORT_DATA_SOURCE
- REPORT_SCRIPTLET
- REPORT_LOCALE
- REPORT_RESOURCE_BUNDLE
- REPORT_TIME_ZONE
- REPORT_FORMAT_FACTORY

Figure 7-3 Group Band Dialog Box

The **Group Band** dialog box appears.

6. Name your group, and click **Next**.



Group Band

Group layout

Please select what bands should we create

☑ Add the Group Header

☑ Add the Group Footer

Figure 7-4 Group Layout Dialog Box

The **Group Layout** dialog box appears.

7. Leave both **Add the Group Header** and **Add the Group Footer** checked, and click **Finish**.

Your report will look like the one in **Figure 7-5**.

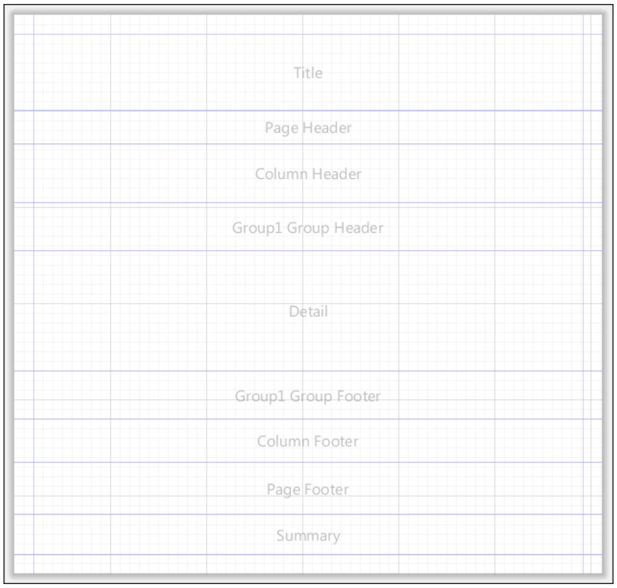


Figure 7-5 Report Containing Group Header and Footer

7.2.2 Customizing a Template

Now that you have a blank template, you can customize it to suit your preference. For example, you can add your company name and logo, page numbering, add a background for your report, and set band and column sizes. You also use this procedure to make changes to an existing template.

To customize a template:

1. To add a graphic, drag an **Image** element into the band where you want the image to appear. This is usually the Title band.

   For more information about the Image element, see **"Graphic Elements" on page 50**.
2. To add a title you want to see in your template, drag a **Static Text** element to the Title band. Style the text in the Properties tab. For more information about Static Text elements, see **"Text Elements" on page 51**.
3. If you want the background to cover the entire page, right-click the element in the **Outline** and choose **Maximize Band Height**. Otherwise, set the Background band to the size you want. Drag an **Image** element into the Background band to create your background.
4. You can add page numbering to the Page Footer band. Drag a **Page Number** element into the band, and place it where you want the page number to be. You can also add a **Page X of Y** element if you prefer.
5. If you want a label in the Column Header band, add a **Static Text** element with the text "Label".
6. To set styles for your report's text, add a **Text** field to the Group Header and a **Text** field to the Detail band. Set the text of the first **Text** field to `"GroupField"` and the text of the second **Text** field to `"Field"`. Format the text as you like.
7. Save your template file.

8. Click the **Preview** tab. Your template should like something like the one in **Figure 7-6**.

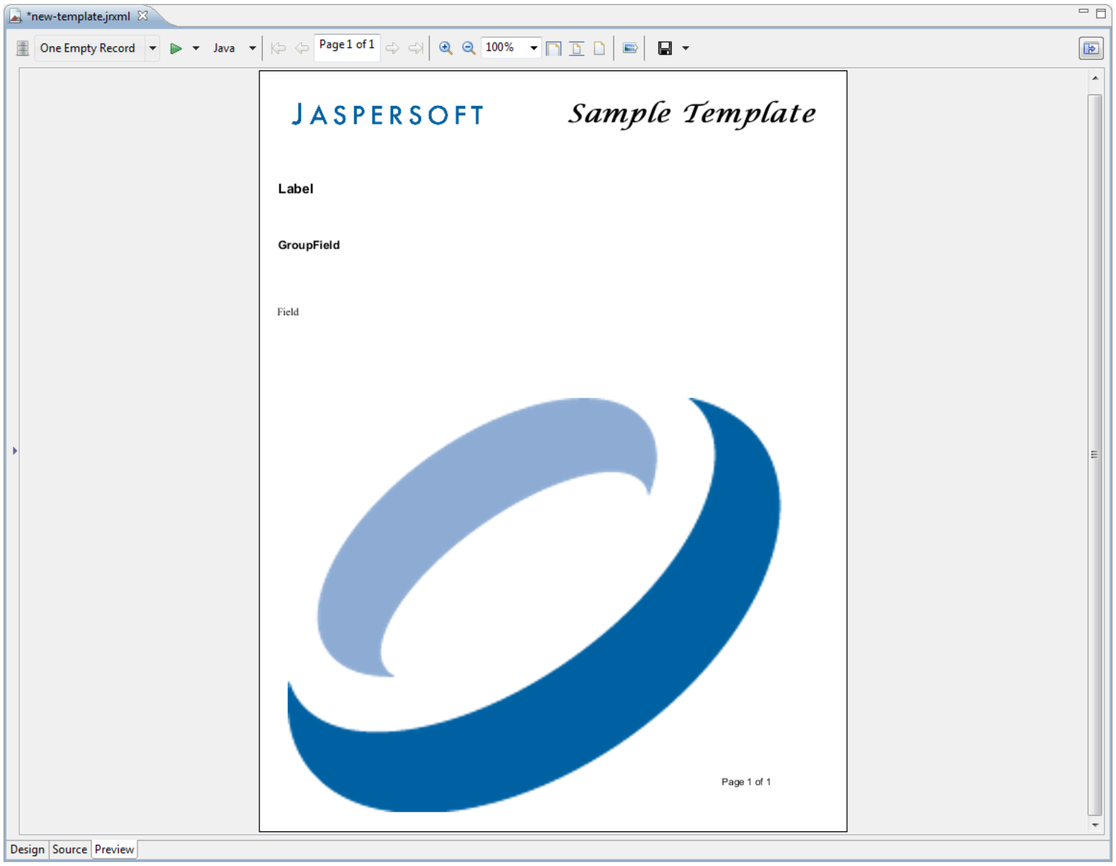


Figure 7-6 Template Preview

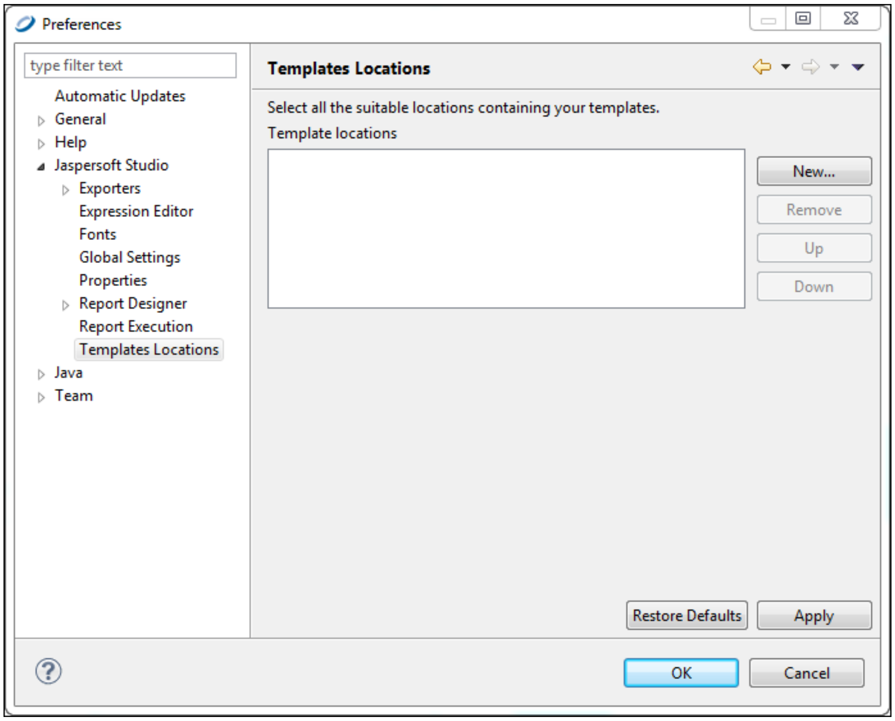
7.3 Saving Templates

Jaspersoft Studio templates require a flat folder structure (resources and report must be on the same folder), so when you export a template, the resources are moved and the paths in the report exported are changed to point to the same directory. Because if a flat structure one resource will overwrite another, if there are resources in multiple folders with the same name, an error message is shown.

7.3.1 Creating a Template Directory

You can specify one ore more directories in which to place custom templates.

1. Go to **Window > Preferences > Jaspersoft Studio > Templates Locations**.



Select all the suitable locations containing your templates.

Template locations

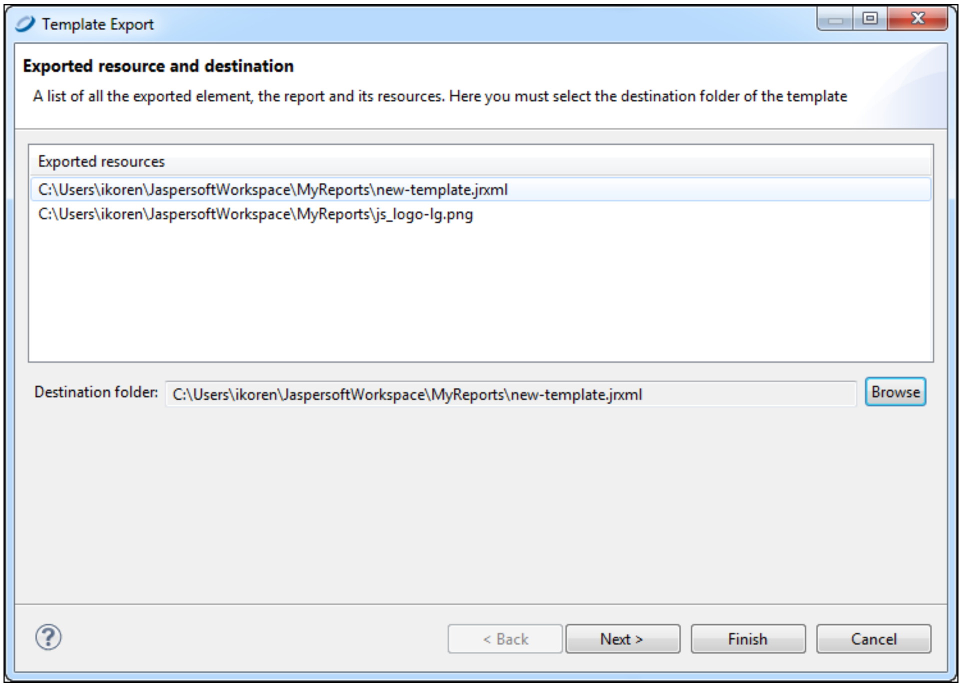
Figure 7-7 Template Location Preferences

2. Click the **New...** button and navigate to the directory in which you want to store your template.
3. Click the **Apply** button.
4. Click **OK**.

7.3.2 Exporting a Template

After you have created a template, save it for future use.

1. Go to **File > Export as Report Template**.



Template Export

Exported resource and destination

A list of all the exported element, the report and its resources. Here you must select the destination folder of the template

Exported resources

C:\Users\ikoren\JaspersoftWorkspace\MyReports\new-template.jrxml
C:\Users\ikoren\JaspersoftWorkspace\MyReports\js_logo-lg.png

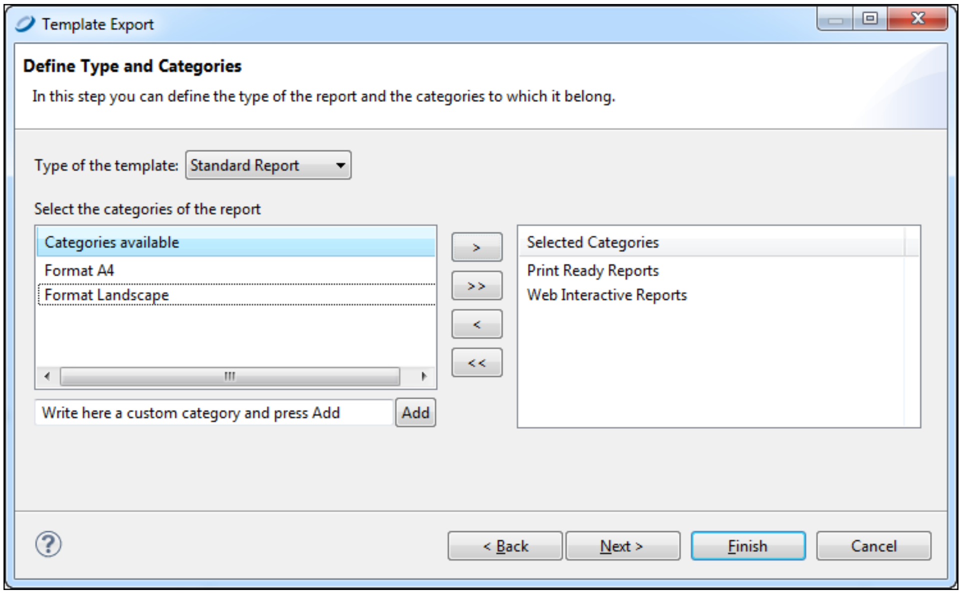
Destination folder: C:\Users\ikoren\JaspersoftWorkspace\MyReports\new-template.jrxml

Figure 7-8 Template Export Dialog

The **Template Export** dialog opens.

2. Click the **Browse** button and navigate to the directory where you wish to save your template. Click **Next**.

   The **Define Type and Categories** dialog opens.



Template Export

Define Type and Categories

In this step you can define the type of the report and the categories to which it belong.

Type of the template: Standard Report

Select the categories of the report

Categories available
Format A4
Format Landscape

Selected Categories
Print Ready Reports
Web Interactive Reports

Write here a custom category and press Add

Figure 7-9 Define Type and Categories Dialog

3. In the drop-down, choose whether the template type is a Standard Report or a Table-Based report.

   This selection is used to validate the report. For example. by selecting **Standard Report**, the validation process will search for the group field with the text group or for the column header label with the text label. If any of the required fields is not found, an error message is displayed.

4. Select the categories in which you want to make the template available, and using the arrow button, add them to the **Selected Categories** box.
5. Click **Finish**.

7.3.3 Creating a Template Thumbnail

Saving a thumbnail is not mandatory, but it is helpful, especially if your template will be used by more than one person.

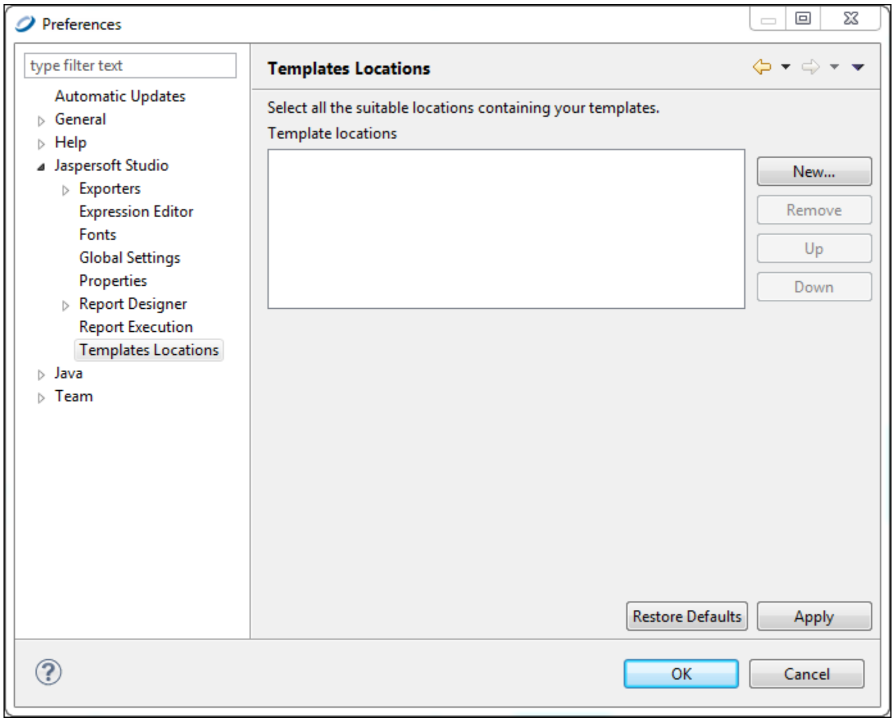
1. Go to the **Preview** tab.
2. Click the Export Image button.
3. Save the image in the same directory and with the same name as your template.

7.4 Adding Templates to Jaspersoft Studio

Once you have created a custom template, you need to add it to Jaspersoft Studio in order to use it.

To add a template to Jaspersoft Studio:

1. Go to **Window > Preferences > Jaspersoft Studio > Templates Locations**.



Templates Locations

Select all the suitable locations containing your templates.

Figure 7-10 Template Location Preferences

2. Navigate to the directory in which you stored your template.
3. Click the **Apply** button.
4. Click **OK**.

When you go to **File > New > Jasper Report**, your new template will appear along with the default templates.

Chapter 8 Using Parameters

Parameters are values usually passed to the report from the application that originally requested it. They can be used for configuring features of a particular report during report generation (such as the value to use for a condition in a SQL query) and to supply additional data to the report that cannot properly provided by the data source (e.g., a custom title for the report, an application-specific path to search for images, or even a more complex object like an image).

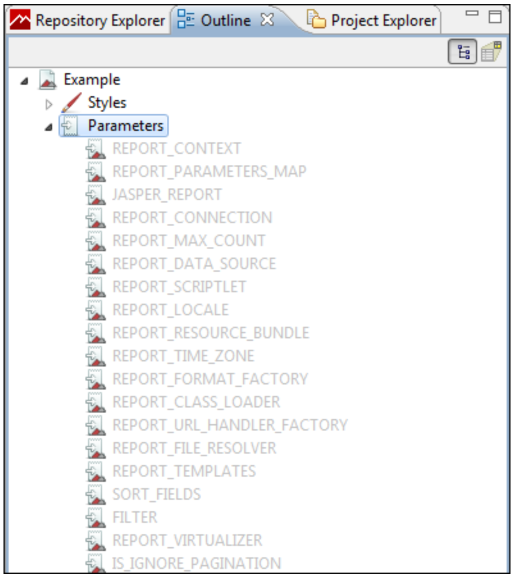
Report parameters are used in many different ways inside a report. They can be used in the where condition of an SQL query, or to provide additional data to the report (i.e. the a value for a title or the name of user that executed the report). A parameter is defined by a name and a Class, which is a Java class type like java.lang.String or java.lang.Integer. Any Java class is a valid parameter class. In example a parameter of type java.sql.Connection may be used to populate a subreport, while a simple java.lang.Boolean parameter may be used to show or hide a section of the report.

This chapter contains the following sections:

8.1 Managing Parameters

Parameters are the best communication channel between the report engine and the execution environment (your application).

A parameter can have a default value which is defined by means of the default expression property. This expression is evaluated by JasperReports only when a value for the parameter has not been provided by the user at run time.

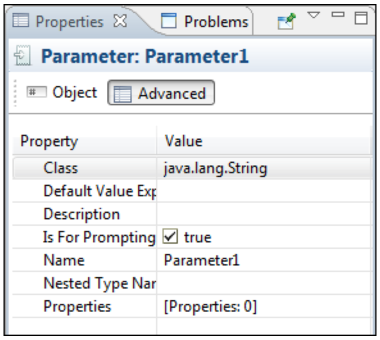


- Example
  - Styles
  - Parameters
    - REPORT_CONTEXT
    - REPORT_PARAMETERS_MAP
    - JASPER_REPORT
    - REPORT_CONNECTION
    - REPORT_MAX_COUNT
    - REPORT_DATA_SOURCE
    - REPORT_SCRIPTLET
    - REPORT_LOCALE
    - REPORT_RESOURCE_BUNDLE
    - REPORT_TIME_ZONE
    - REPORT_FORMAT_FACTORY
    - REPORT_CLASS_LOADER
    - REPORT_URL_HANDLER_FACTORY
    - REPORT_FILE_RESOLVER
    - REPORT_TEMPLATES
    - SORT_FIELDS
    - FILTER
    - REPORT_VIRTUALIZER
    - IS_IGNORE_PAGINATION

Figure 8-1 Parameters in Outline View

To manage parameters, use the outline view. Add a parameter by right-clicking on the item **Parameters** and choosing **Create Parameter**. To delete a parameter from the outline view right click on it and select **Delete**. The parameters with light gray names are created by the system and can not be deleted or edited.

Right-click any parameter and choose **Show Properties** to view and edit the properties of the parameter.



Parameter: Parameter1

Property	Value
Class	java.lang.String
Default Value Exp	
Description	
Is For Prompting	☑ true
Name	Parameter1
Nested Type Nar	
Properties	[Properties: 0]

Figure 8-2 Parameters - Advanced Properties

The **Default Value Expression** field allows you to set a pre-defined value for the parameter. This value will be used if no value is provided for the parameter from the application that executes the report. The type of value must match the type declared in the **Class** field.

You may legally define another parameter as the value of **Default Value Expression**, but this method requires careful report design. Jaspersoft Studio parses parameters in the same order in which they are declared, so a default value parameter must be declared before the current parameter.

The parameter **Description** is not used directly by JasperReports, but like the **Is for Prompting**, may be passed to an external application.

As with fields, you may specify pairs of type name/value as properties for each parameter. This is a way to add extra information to the parameter for use by external applications. For example, the designer can use properties to include the description of the parameter in different languages or perhaps add instructions about the format of the input prompt.

8.2 Default Parameters

JasperReports provides some built-in parameters (they are internal to the reporting engine) that you may read but cannot modify. Some the most important are the "REPORT_CONNECTION", which holds the JDBC connection used to run the SQL query of the report (if the report is filled using a JDBC connection), the "REPORT_DATA_SOURCE" which contains, if available, the data source used to fill the report, the "REPORT_LOCALE" which contains the Locale used to fill the report and so on.

Some built-in parameters are specific of some query languages, in example when using the Hibernate query language, the reports automatically includes the parameter "HIBERNATE_SESSION" that holds the Hibernate session used to run the HQL query. As said, the built-in parameters can not be modified or deleted.

The built-in parameters are:

Table 8-1 JasperReports Default Parameters

Parameter	Description
REPORT_CONTEXT	
REPORT_PARAMETERS_MAP	This is the java.util.Map passed to the fillReport method; it contains the parameter values defined by the user.
JASPER_REPORT	
REPORT_CONNECTION	This is the JDBC connection passed to the report when the report is created through a SQL query.
REPORT_MAX_COUNT	This is used to limit the number of records filling a report. If no value is provided, no limit will be set.
REPORT_DATA_SOURCE	This is the data source used by the report when it is not using a JDBC connection.
REPORT_SCRIPTLET	This represents the scriptlet instance used during creation. If no scriptlet is specified, this parameter uses an instance of net.sf.jasperreports.engine.JRDefaultScriptlet.
REPORT_LOCALE	This is used to set the locale used to fill the report. If no locale is provided, the system default will be used.
REPORT_RESOURCE_BOUNDLE	This is the resource bundle loaded for this report.
REPORT_TIME_ZONE	This is used to set the time zone used to fill the report. If no value is provided, the system default will be used.
REPORT_FORMAT_FACTORY	This is an instance of a net.sf.jasperreports.engine.util.FormatFactory. The user can replace the default one and specify a custom version using a parameter. Another way to use a particular format factory is by setting the report property format factory class.
REPORT_CLASS_LOADER	This parameter can be used to set the class loader to use when filling the report.
REPORT_URL_HANDLER_FACTORY	Class used to create URL handlers. If specified, it will replace the default.
REPORT_FILE_RESOLVER	This is an instance of net.sf.jasperreports.engine.util.FileResolver used to resolve resource locations that can be passed to the report in order to replace the default implementation.

Table 8-1 JasperReports Default Parameters

Parameter	Description
REPORT_TEMPLATES	This is an optional collection of styles (JRTemplate) that can be used in the report in addition to the ones defined in the report.
SORT_FIELDS	
FILTER	
REPORT_VIRTUALIZER	This defines the class for the report filler that implements the JRVirtualizer interface for filling the report.
IS_IGNORE_PAGINATION	You can switch the pagination system on and off with this parameter (it must be a Boolean object). By default, pagination is used except when exporting to HTML and Excel formats.

8.3 Using Parameters in Queries

Generally, parameters can be used in the query associated with a report even if not all the languages support them.

JasperReports executes queries, passing the value of each parameter used in the query to the statement.

This approach has a major advantage with respect to concatenating the parameter value to the query string—you do not have to take care of special characters or sanitize your parameter, since the database will do it for you. At the same time, this method places limits on the control you have on the query structure. For example, you cannot specify a portion of a query with a parameter.

8.3.1 Using Parameters in a SQL Query

Parameters can be used in SQL queries to filter records in a where condition or to add/replace pieces of raw SQL or even to pass the entire SQL string to execute.

In the first case the parameters are used as standard SQL parameters, in example:

```
SELECT * FROM ORDERS WHERE ORDER_ID = $P{my_order_id}
```

In this example `my_order_id` is a parameter that contains the ID of the order to read. This parameter can be passed to the report from the application that is running it to select only a specific order. Please note that the parameter here is a valid SQL parameter, meaning that the query will be executed using a prepared statement like:

```
SELECT * FROM ORDERS WHERE ORDER_ID = ?
```

and the value of the parameter `my_order_id` will then passed to the statement.

In this query:

```
SELECT * FROM ORDERS ORDER BY $P!{my_order_field}
```

`my_order_field` cannot be treated as a SQL parameter. JasperReports will consider this parameter to be a placeholder (note the special syntax `$P!{})` which will be replaced with the text value of the parameter (in this case "`ORDERDATE DESC`").

With the same logic, a query can be fully passed using a parameter. The query string would look like:

```
$P!{my_query}
```

The number of parameters in a query is arbitrary. When passing a value using the `$P!{}` syntax, the value of the parameter is taken as is, the user is responsible of the correctness of the passed value (SQL escaping is not performed by JasperReports in this case). When using a parameter in a query, a default value must be set for the parameter to allow Jaspersoft Studio to execute the query to retrieve the available fields.

8.3.2 Using Parameters with Null Values

The parameter form `$P{parametername}` will not work correctly with null values. In an operation in which your value could be null, use the form `$X{EQUAL,fieldname,parametername}`.

For example:

1. `$P{param}: "select * where num_column > $P{num_param}"`

   In this case $P should be used, because we don't have $X{GREATER,..}, and Null has no meaning for the operation “greater than”.

2. `$X{EQUAL, column_name, param_name}`

   Let's compare two expressions:

   `"select * where num_column = $P{num_param}"`

   and

   `"select * where $X{EQUAL, num_column, num_param}"`

   Both will generate the same output if parameter has not Null value: `"select * where num_column = 1"`

   However, if the parameter has a Null value the outputs will be different:

   - `$P`: `"select * where num_column = null"`
   - `$X`: `"select * where num_column IS null"`

   The key difference that databases don't understand `"= null"`, but `"is null"`. So if you want your query with the condition "=" to work with null values, you need to use `$X{EQUAL/NOTEQUAL, column, parameter}`.

8.3.3 IN and NOTIN Clauses

JasperReports provides a special syntax to use with a `where` condition: the clause `IN` and `NOTIN`.

The clause is used to check whether a particular value is present in a discrete set of values. Here is an example:

```
SELECT * FROM ORDERS WHERE SHIPCOUNTRY IS IN ('USA','Italy','Germany')
```

The set here is defined by the countries USA, Italy and Germany. Assuming we are passing the set of countries in a list (or better a `java.util.Collection`) or in an array, the syntax to make the previous query dynamic in reference to the set of countries is:

```
SELECT * FROM ORDERS WHERE $X{IN, SHIPCOUNTRY, myCountries}
```

where `myCountries` is the name of the parameter that contains the set of country names. The `$X{}` clause recognizes three parameters:

- Type of function to apply (`IN` or `NOTIN`)
- Field name to be evaluated
- Parameter name

JasperReports will handle special characters in each value. If the parameter is `null` or contains an empty list, meaning no value has been set for the parameter, the entire `$X{}` clause is evaluated as the always true statement “`0 = 0`”.

8.3.4 Relative Dates

The relative dates feature enables you create reports that to filter information based on a date range relative to the current system date. To do this, use the following template:

`<Keyword> <+/-> <N>` where:

- `<Keyword>` indicates the time span you want to use. Options include: `DAY`, `WEEK`, `MONTH`, `QUARTER`, `SEMI`, and `YEAR`.
- `<+/->` indicates whether the time span occurs before or after the chosen date.
- `<N>` indicates the number of the above-mentioned time spans you want to include in the filter.

For example, if you want to look at Sales for the prior month, your expression would be `MONTH - 1`.

Relative dates are sensitive to time zones. The relative date expression is calculated in the time zone of the logged-in user.

Only one property is configurable: the start day of the week does not depend on locale.

The `class` attribute of a JasperReports Parameter of type Relative Date should have one of the following values:

- `net.sf.jasperreports.engine.rd.DateRange` – if you want to use `java.util.Date` (Date only).
  For example: `<parameter name=”myParameter” class="net.sf.jasperreports.engine.rd.DateRange”>`
- `net.sf.jasperreports.engine.rd.TimestampRange` – if you want to use `java.sql.Timestamp` (Date and Time).
  For example: `<parameter name=”myParam” class="net.sf.jasperreports.engine.rd.TimestampRange”>`

Following are two examples of when and how to use relative dates:

Problem	Solution
Find all purchases made previous to this quarter	Relative data parameter called STARTDATE takes this value: `QUARTER` "`QUARTER`" evaluates to the first day (the first instant, really) of this quarter. `SQL: select * from orders where order_date < $P{STARTDATE}`

Find all purchases made in this quarter	Valid answer:
	`select * from orders where order_date >= QUARTER and order_date < QUARTER + 1`
	Better answer using DateRanges:
	`select * from orders where $X{EQUAL, order_date, QUARTER}`

If you want to set a relative date as a default value expression of a JasperReports parameter, use the following relative date-builder pattern:

- `new DateRangeBuilder("DAY-1").toDateRange()`
- `new DateRangeBuilder("WEEK").set(Timestamp.class).toDateRange()`
- `new DateRangeBuilder("2012-08-01").toDateRange()`
- `new DateRangeBuilder("2012-08-01 12:34:56").toDateRange();`

For queries, Relative dates are not supported by the `$P{}` function because it can handle only a limited number of classes (standard Java classes Integer, String etc.). Instead, a new implementation of pluggable functions and parameters in JasperReports supports relative dates with `$X{}` functions.

The following is a JRXML example that shows data from the previous day:

```
<parameter name=“myParameter” class=“net.sf.jasperreports.engine.rd.DateRange”>
<defaultValueExpression>
<![CDATA[
new DateRangeBuilder("DAY-1").toDateRange()
]]>
</defaultValueExpression>
</parameter>

<queryString>
<![CDATA[
Select * from account where $X{EQUAL, OpportunityCloseDate, SelectedDateRange}
]]>
</queryString>
```

Older reports which have date parameters will work just as before, and Relative Dates functionality will be unavailable. In order to make older reports support relative dates, you will need to modify the JRXML: change the parameter class, and, if needed, set a default value expression. In addition, remember to use the $X{} function instead of $P{}.

Relative dates currently do not support keywords like “Week-To-Date” (from the start of the current week to the end of the current day). However, you can emulate that by using `IS_BETWEEN`:

- In JRXML, the query use is:`$X{IS_BETWEEN, column, startParam, endParam}` where `startParam` has value `WEEK` and `endParam` has value `DAY`.
- Likewise, you can do the same for other time ranges: Year-To-Week, Year-To-Month, etc.

8.3.5 Passing Parameters from a Program

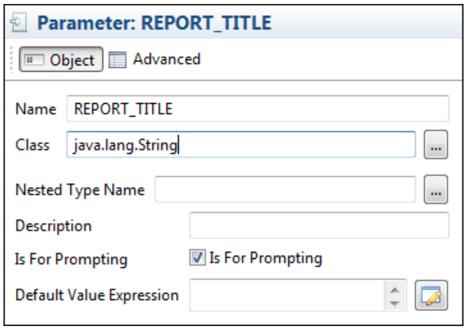
Jaspersoft Studio passes parameters from a program “caller” to the print generator using a class that extends the `java.util.Map` interface. For example:

```
...
  HashMap hm = new HashMap();
    ...
  JasperPrint print = JasperFillManager.fillReport(
    fileName,
    hm,
  new JREmptyDataSource());
...
```

`fillReport` is a key method that allows you to create a report instance by specifying the file name as a parameter, a parameter map, and a data source. (This example uses a dummy data source created with the class `JREmptyDataSource` and an empty parameter map created using a `java.util.HashMap` object.)

Let’s see how to pass a simple parameter to a reporting order to specify the title of a report.

The first step is to create a parameter in the report to host the title (that will be a String). We can name this parameter `REPORT_TITLE` and the class will be `java.lang.String` (**Figure 8-3**).

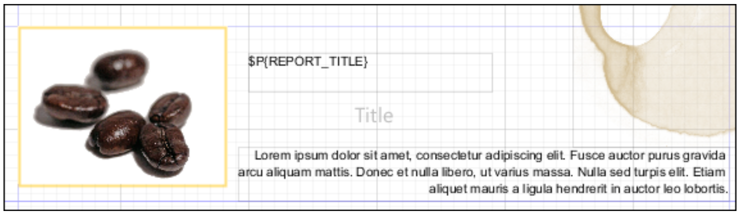


Parameter: REPORT_TITLE

Name: REPORT_TITLE
Class: java.lang.String
Nested Type Name:
Description:
Is For Prompting: ☑ Is For Prompting
Default Value Expression:

Figure 8-3 Definition of `REPORT_TITLE`

All the other properties can be left as they are. Drag the parameter into the Title band to create a text field to display the `REPORT_TITLE` parameter.



$P{REPORT_TITLE}
Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce auctor purus gravida arcu aliquam mattis. Donec et nulla libero, ut varius massa. Nulla sed turpis elit. Etiam aliquet mauris a ligula hendrerit in auctor leo lobortis.

Figure 8-4 Design Panel with `REPORT_TITLE` in the Title Band

To set the value of the `REPORT_TITLE` parameter in our application, modify the code of the previous source code example by adding:

```
 ...
HashMap hm = new HashMap();
hm.put(“REPORT_TITLE”,”This is the title of the report”);
...
JasperPrint print = JasperFillManager.fillReport(
fileName,
hm,
new JREmptyDataSource());
...
```

We have included a value for the `REPORT_TITLE` parameter in the parameter map. You do not need to pass a value for all the parameters. If you don’t provide a value for a certain parameter, JasperReports will assign the value of `Default Value Expression` to the parameter with the empty expression evaluated as null.

Printing the report, Jaspersoft Studio includes the String `This is the title of the report` in the Title band. In this case we just used a simple String. However, it is possible to pass much more complex objects as parameters, such as an image (`java.awt.Image`) or a data source instance configured to provide a specified subreport with data. The most important thing to remember is that the object passed in the map as the value for a certain parameter must have the same type (or at least be a super class) of the type of the parameter in the report. Otherwise, Jaspersoft Studio fails to generate the report, returning a `ClassCastException` error.

8.4 Parameters Prompt

If a parameter is set to be used as a prompt, when a report is executed, Jaspersoft Studio asks for the value of the parameter.

To create a parameter prompt:

Create a simple report with the template **Blank A4**, name `ParameterExample` and data adapter **One Empty Record - Empty Rows**.

3. In this report create a parameter and rename it (from its properties tab) to `MESSAGE`, with type `java.lang.String` set to be used as a prompt (check the box next to the property **Is For Prompting**).
4. Drag the parameter from the outline view inside the **Title** band and Jaspersoft Studio will create a text field to display the parameter value.

   You should have something like the following image:

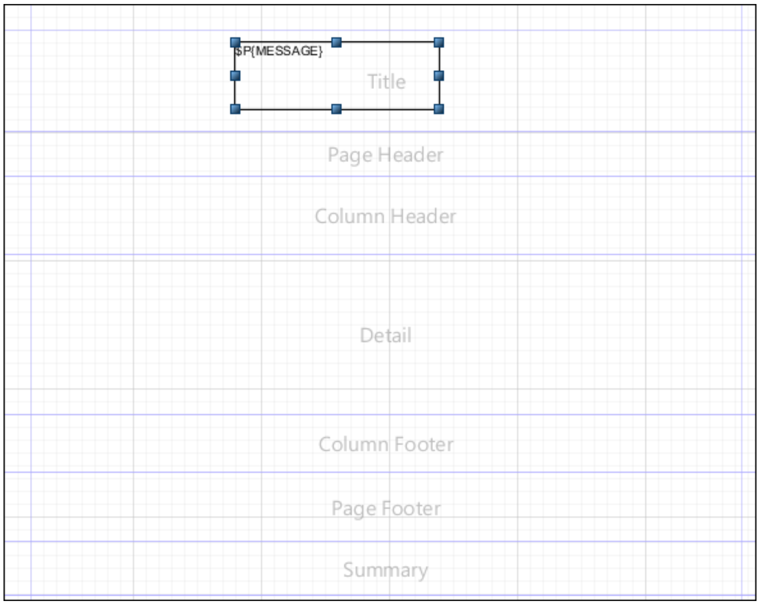


Figure 8-5 Parameter in Title Band

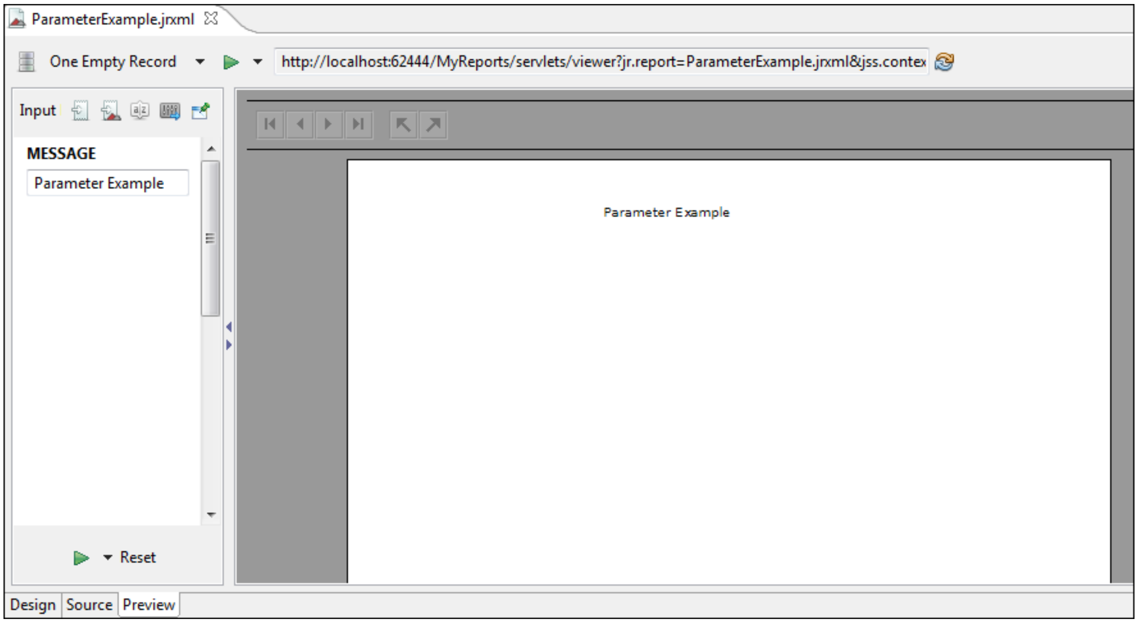
To compile and preview the report:

1. Click the **Preview** tab.

   Be sure the **Input Parameter** window is open. If not, click the grey right-arrow to the left of the **Preview** screen.
2. Add a value for the `MESSAGE` parameter.

   For this example, type `Parameter Example`.

3. Press the **Play** button. The message will be printed in the title band.



ParameterExample.jrxml

One Empty Record

http://localhost:62444/MyReports/servlets/viewer?jr.report=ParameterExample.jrxml&jss.contex

Input

MESSAGE

Parameter Example

Reset

Parameter Example

Design Source Preview

Figure 8-6 Preview Tab with Parameter Value

Jaspersoft Studio provides input dialogs for parameters of type String, Date, Time, Number, and Collection.

CHAPTER 9 VARIABLES

Variables can be used to store partial results and do complex calculations with the data extracted from data source. These values can then be used in other parts of the report, including other variables.

This chapter contains the following sections:

9.1 Defining a New Variable or Editing an Existing One

As with many other elements, all defined variables are visible in the Outline view under the item called "Variables". From there you can create a new variable and edit its properties in the Properties view.

To define a new variable:

1. In the Outline view, right click the **Variables** item and select **Create Variable**. A new variable is added to the list of variables.
2. Click to select the new variable. Information for this variable opens in the Properties view.
3. In the Properties view, click the **Object tab**.
4. Update the variable properties. See **"Base Properties of a Variable" on page 92** for information on these options.

There are some built-in variables on Jaspersoft Studio, which present in every report. You can identify these variables because their name has a light gray color, instead of a black color, and the properties of these variables can not be edited. For information on these variables, see **"Built-In Variables" on page 94**

9.2 Base Properties of a Variable

At a minimum, all variables must have the following defined:

- **Name**: A string used to refer to a variable. It is necessary to use this variable inside other expressions (such the valorization of a Text Field or the computation of another variable). To refer to a variable the following syntax is used: `$V{variable_name}`.
- **Type**: Necessary because a variable is an object that probably will be used in other expressions, so its type must be known to be manipulated correctly.
- **Expression**: The function used to define the variable value, it can be composed of more fields and variables, and could be logic operators, math operators and so on. To easily define the expression an expression editor is provided in Jaspersoft Studio. This can be opened using the button to the right of the text Field used to write the expression. The expression is evaluated on each iteration, every time a record is readed from the data source. If there isn't a calculation function defined, the result of the expression will be assigned to the variable. Thus it's important that the result has a type compatible with the one in the variable.
- **Initial Value**: The value assumed from the variable at the beginning, before the first computation of its expression. The initial value is an expression itself, so it can be defined through the expression editor. It's not mandatory, but it is good practice to define an initial value. For example, if you have a variable called `variable1` with the expression `new Integer(5)`. At every record read the variable will be assigned the integer value 5, so the initial value it isn't important in this context. However, if you change the expression to `$V{variable1}+5`, this means that at every iteration the variable is incremented by 5. In this case an initial value is necessary because if at the first iteration the `variable1` is undefined it will break all future evaluations. So an initial value is not mandatory, but undefined variables can be dangerous.

9.3 Other Properties of a Variable

The most complex property of a variable is its temporal value. Since its expression is evaluated at every iteration, it is important to understand which value has a variable, and at which time. This can be complicated, considering that a variable can use other variables inside its expression. For these reasons there are mechanisms that can be used to simplify the evaluation or the reading of the variable value during the iterations.

9.3.1 Evaluation Time

The evaluation time is not an attribute of the variable but of the elements that can use variables in their expressions (like a Text Field) and define the "time" when the value of the variable should be read. A variable can potentially change value at every iteration, so a value read at one time may be different from the one obtained during another time. So for every element that can define an expression is possible to say when evaluate it. And since in the expression could be defined one or more variables this parameter also influences also when these variables are read.

The possible values are:

- **Report**: The value of the expression is evaluated ad the end of the report.
- **Page**: The value of the expression is evaluated at the end of every page of the report.
- **Column**: The value of the expression is evaluated at the end of each column (if the report is composed only of a column this is equivalent to Page).
- **Group**: The value of the expression is evaluated after the break of the specified group. This option is visible only if at least one group is defined.
- **Band**: The value of the expression is evaluated after the end of the band where the element with this evaluation time is placed. This is a very particular case, introduced to wait that the other elements in the band are completely created. Typically the value of the variables are read at the start of the band, but for example suppose to have a subreport with an output parameter to print in the main report. To print this parameter it must be read when the subreport was already computed, so the value could be printed when the band is completely created, in this cases the Band evaluation time is necessary.
- **Auto**: This is used when in the expression of the element there are more variables and fields, that need to be evaluated at different times. The variables are evaluated at a time corresponding to their Reset Type (see below for more information),

instead the fields are always evaluated at time -now. This type is useful when report elements have expressions that combine values evaluated at different times (e.g. percentage out of a total).

- **Now**: The value of the expression is evaluated after the read of every record, so at every iteration, this is the default behavior.

9.3.2 Calculation Function

A calculation function is an attribute of a variable that defines when the variable can be used in association with the expression to determinate the value of the variable. When using a calculation function, the value of the variable is not determinate directly by its expression. Instead, it is passed to the calculation function that will use it (depending on the function) to calculate the variable value.

There are many functions built-in to Jaspersoft Studio:

- **Sum**: At every iteration the value of the expression is taken it will be summed at the value of the variable. This is one of the cases where the initial value is really important.
- **Count**: At every iteration the variable value is incremented by one unit. This only if the expression is different from null, in that case the value of the variable will be left unchanged.
- **Distinct Count:** At every iteration the variable value is incremented by one unit, but only if the value of the expression was never returned before.
- **Average**: The value of the variable is the arithmetic average of all values received in input from the expression.
- **Lowest**: The variable take the value of the lowest element received from the expression.
- **Highest**: The variable take the value of the highest element received from the expression.
- **Standard Deviation**: The standard deviation of all the value received from the expression.
- **First**: The variable take the value from the first value returned by the expression.
- **System**: No calculation is done and the expression is not evaluated, the value of this variable will be the last value set on it. Useful to store partial results or the final result of a computation.

9.3.3 Increment Type

As stated above, when a calculation function is defined, the value of the expression is passed to the function that will do the calculation for the variable. The default behavior is to do this for every record read, but sometimes a different behavior is desired. Using the increment type parameter is possible to change the "time" on which the calculation function is used.

The possible values for this attribute are:

- **Report**: The Calculation Function is called only at the end of the report, passing to it the value of the expression at that moment.
- **Page**: The Calculation Function is called at the end of each page, passing to it the value of the expression at that moments.
- **Column**: The Calculation Function is called at the end of each column (if the report is composed only of a column this is equivalent to Page).
- **Group**: The Calculation Function is called at the start of every occurrence of the specified group. This option is visible only if at least one group is defined.
- **None**: The Calculation Function is called after the read of every record, this is the default behavior.

Remember that the expression is evaluated at every record read, independently from the increment type selected, but the calculation function will be used only when the times match those defined in the increment type.

9.3.4 Reset Type

The reset type defines when a variable should be reset to the initial value, or to null if an initial value is undefined. This is useful when the variable is used to compute a partial value, such a sum or an average of only some of the records read.

The possible values for this attribute are:

- **Report**: The variable is initialized only one time at the beginning of the report creation.
- **Page**: The variable is initialized on each page.

- **Column**: The variable is initialized again in each new column (if the report is composed only of a column this is equivalent to Page).
- **Group**: The variable is initialized at the start of every occurrence of the specified group. This option is visible only if at least one group is defined.
- **None**: The variable will never be initialized, so the initial value expression is ignored.

9.3.5 Incrementer Factory Class Name

The calculation functions are useful but are also generics and limited at the numeric types. You may have a case where something more specific is needed. Suppose you have a field of type String and you want to concatenate the value read. You can do this by defining a new Incrementer. An incrementer is a piece of java code that extends the interface JRIncrementerFactory, and can build a personalized calculation function to do what you need. Every calculation function receives the expression value and the variable value and returns the result of the increment, so there is everything needed to do the calculation and return the right value.

9.4 Built-In Variables

Jaspersoft Studio offers a number of built-in variables for your use, listed in the table below.

You can identify these variables because their name has a light gray color, instead of a black color, and the properties of these variables cannot be edited.

Variable Name	Description
PAGE_NUMBER	Contains the current number of pages in the report at report time.
COLUMN_NUMBER	Contains the current number of columns.
REPORT_COUNT	Contains the number of records processed.
PAGE_COUNT	Contains the current number of records processed in the current page.
COLUMN_COUNT	Contains the current number of records processed during the current column creation.
CITY_COUNT	
ID_COUNT	

9.5 Tips & Tricks

Pay attention at the type of the variable, many time a bad result is due to this. For example if your expression return a number but the variable is type string (that it's also the default type) then its value will be always zero.

The form of the expression it's really important for the computation of a value, especially when in the expression is used the variable itself. Consider the following example:

A field with name `Money_Gained`, read from the data source, that has an integer value and could be null.

A variable `Total1` with the expression
`IF(EQUALS($F{Money_Gained}, null), $V{Total1}, $V{Total1}+$F{Money_Gained})`
initial value zero, and no calculation function;

A variable `Total2` with the expression
`$V{Money_Gained} == null ? $V{Total2} : $V{Total2}+$F{Money_Gained}`
initial value zero, and no calculation function;

The two expressions seem equivalent: they both add the money gained to the variable when it is not null (remember that if there isn't a calculation function then the value of the expression is assigned to the variable). The check if the `Money_Gained` has value null it's necessary because the sum of a number with the value null is null. So adding null to Total1 or Total2 will change the variable to null. But even with this check when `Money_Gained` will became null for the first time even `Total1` will be null, instead `Total2` will have the correct value.

This happens because this two expressions have different interpreters, the first is interpreted by Groovy, the second by Java. The Java behavior is to evaluate the condition and then select the correct branch. On the other hand, Groovy computes the two branches, then evaluates the expression, and finally it returns the correct branch. Doing this it will add the null value to `Total1` before doing the check, and this will made `Total1` to became null. A trick to avoid this is to use the variable only in the main branch, for example `Total1` could be rewritten as:

```
$V{Total1}  + IF(EQUALS($F{Money_Gained}, null),0,F{Money_Gained}).
```

The syntax is still interpreted by Groovy, but now the variable is out of the `IF` branches, so even if they are both evaluated, the variable maintains its value.

Chapter 10 Data Sources

JasperReports can fill a report with data in a number of ways. For example, you can put an SQL query directly inside a report and provide a connection to a database against which to execute the query and read the resulting record set, or you can use more sophisticated custom technology to provide a table-like set of values.

Jaspersoft Studio provides direct support for a rich set of query languages, including SQL, HQL, EJBQL, and MDX, and supports other languages like XPath (XML Path Language). If you don’t want to use a query language or put your query inside your report, you can use a JasperReports data adapter. Basically, a JasperReports data adapter is an object that iterates on a record set that is organized like a simple table.

All the data adapters implement the `JRDataSource` interface. JasperReports provides many ready-to-use implementations of data sources to wrap generic data structures, such as arrays or collections of JavaBeans, result sets, table models, CSV and XML files. This chapter documents some of these data sources.

Jaspersoft Studio provides support for all these things: you can define JDBC (Java Database Connectivity) connections to execute SQL queries, set up Hibernate connections using Spring, and test your own `JRDataSource` or your custom query language.

This chapter has the following sections:

- **Understanding Data Adapters and Connections in Jaspersoft Studio**
- **Creating and Using Database JDBC Connections**
- **Working with Your JDBC Connection**
- **Understanding the JRDataSource Interface**
- **Data Source Types**
- **Importing and Exporting Data Sources**

10.1 Understanding Data Adapters and Connections in Jaspersoft Studio

In Jaspersoft Studio, a data adapter is an XML file that contains the information about where your data is located. This information includes, URL, user, password, paths, etc. Data adapters also contain the logic to prepare all parameters for JasperReports to run the query and iterate data. A data adapter does not contain any data itself, the data is located in the JRDataSource.

Data adapters in Jaspersoft Studio are used to:

- Define the data connection type. The connection is a SQL connection rather than a JasperReports or Jaspersoft Studio object.
- Bind information to connect your data to Jaspersoft Studio.

- Create the connection and necessary parameters for JasperReports library.

Inside Jaspersoft Studio, data adapters can be stored in your Eclipse settings or in files. Files are the preferred method, because it's easier to deploy them later.

For report developers, Jaspersoft Studio data adapters serve the same purpose as what iReport calls data sources.

The most frequently used connection in Jaspersoft Studio is the JDBC data adapter. Other connection types provided by Jaspersoft Studio include:

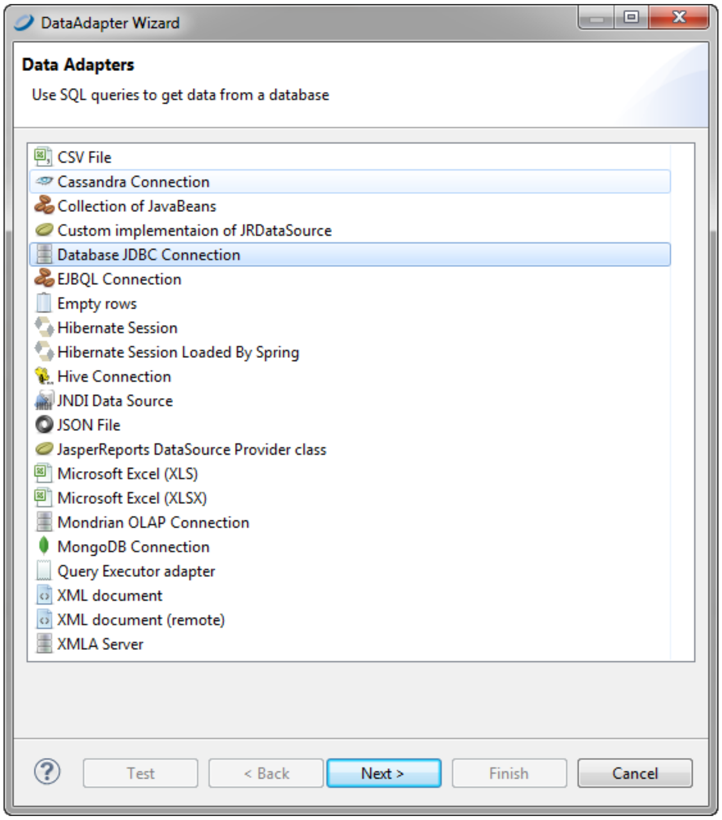
- JDBC connection
- JavaBean collection data source
- XML data source
- CSV data source
- Hibernate connection
- Spring-loaded Hibernate connection
- Hadoop Hive data source
- JRDataSourceProvider
- Custom data source
- Mondrian OLAP connection
- XMLA connection
- EJBQL connection
- Empty data source

All the connections are opened and passed directly to JasperReports during report generation. For many connections, JasperReports provides one or more built-in parameters that can be used inside the report for several purposes (for example, to fill a subreport that needs the same connection as the parent).

- The XML data source allows you to take data from an XML document.
- A CSV (comma-separated values) data adapter allows you to open a CSV file for use in a report.
- The JavaBean set data source, custom data adapter, and JRDataSourceProvider allow you to print data using purposely written Java classes.
- The Hibernate connection provides the environment to execute HQL (Hibernate Query Language) queries (this connection can be configured using Spring, as well).
- EJBQL (Enterprise JavaBean Query Language) queries can be used with an EJBQL connection.
- MDX queries can be used with a native direct connection to a Mondrian server or using the standard XML/A interface to interrogate a generic OLAP database.

An empty data adapter is something like a generator of records having zero fields. Use this for testing or to achieve specific needs, such static content reports or subreports.

Connections and data adapters are managed through the **DataAdapter Wizard**. To set up a new data source, right-click **Data Adapters** from the Repository view and choose **Create Data Adapter**.



DataAdapter Wizard

Data Adapters

Use SQL queries to get data from a database

Figure 10-1 DataAdapter Wizard

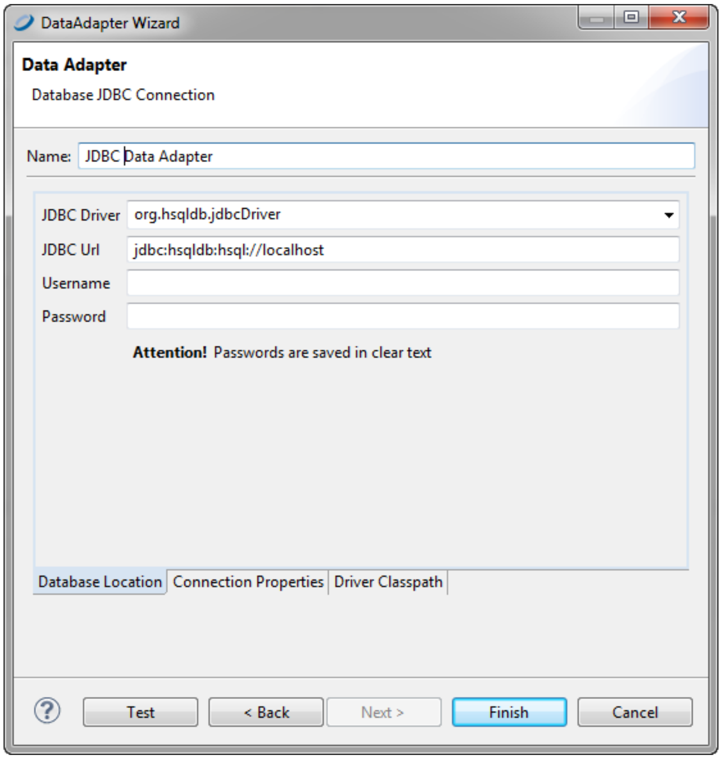
A connection and a data adapter are different objects. However, this document uses the two terms interchangeably because their functions are so similar.

You can set the active data adapter in several ways. The easiest and most intuitive way is to select the data adapter from the Repository view.

If no data adapter is selected, it is not possible to fill a report with data, therefore, when Jaspersoft Studio starts for the first time, a pre-configured empty data adapter is defined and selected by default. Data adapters can also be used in conjunction with the Report Wizard. That’s why configuring the connection to your data is usually the first step when starting with Jaspersoft Studio.

10.2 Creating and Using Database JDBC Connections

A JDBC connection allows you to use a relational DBMS (or, in general, whatever databases are accessible through a JDBC driver) as a data source. To set a new JDBC connection, click the **New** button in the Connections/Datasources dialog box (shown earlier in Figure 10-1) to open the interface for creation of a new connection (or data source). From the list, select **Database JDBC connection** to bring up the window shown in Figure 10-2.



DataAdapter Wizard

Data Adapter

Database JDBC Connection

Name: JDBC Data Adapter

JDBC Driver: org.hsqldb.jdbcDriver

JDBC Url: jdbc:hsqldb:hsql://localhost

Username:

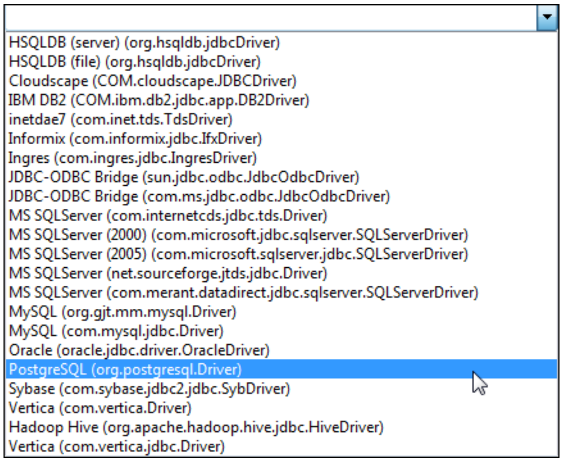
Password:

Attention! Passwords are saved in clear text

Figure 10-2 Configuring a JDBC Connection

The first thing to do is to name the connection (possibly using a significant name, such as `Mysql – Test`). Jaspersoft Studio will always use the specified name to refer to this connection.

In the **JDBC Driver** field, you specify the name of the JDBC driver to use for the connection to the database. The combo box proposes the names of the most common JDBC drivers (see Figure 10-3).



HSQLDB (server) (org.hsqldb.jdbcDriver)
HSQLDB (file) (org.hsqldb.jdbcDriver)
Cloudscape (COM.cloudscape.JDBCDriver)
IBM DB2 (COM.ibm.db2.jdbc.app.DB2Driver)
inetdae7 (com.inet.tds.TdsDriver)
Informix (com.informix.jdbc.IfxDriver)
Ingres (com.ingres.jdbc.IngresDriver)
JDBC-ODBC Bridge (sun.jdbc.odbc.JdbcOdbcDriver)
JDBC-ODBC Bridge (com.ms.jdbc.odbc.JdbcOdbcDriver)
MS SQLServer (com.internetcds.jdbc.tds.Driver)
MS SQLServer (2000) (com.microsoft.jdbc.sqlserver.SQLServerDriver)
MS SQLServer (2005) (com.microsoft.sqlserver.jdbc.SQLServerDriver)
MS SQLServer (net.sourceforge.jtds.jdbc.Driver)
MS SQLServer (com.merant.datadirect.jdbc.sqlserver.SQLServerDriver)
MySQL (org.gjt.mm.mysql.Driver)
MySQL (com.mysql.jdbc.Driver)
Oracle (oracle.jdbc.driver.OracleDriver)
PostgreSQL (org.postgresql.Driver)
Sybase (com.sybase.jdbc2.jdbc.SybDriver)
Vertica (com.vertica.Driver)
Hadoop Hive (org.apache.hadoop.hive.jdbc.HiveDriver)
Vertica (com.vertica.jdbc.Driver)

Figure 10-3 JDBC Drivers List

If a driver is displayed in red, the JDBC driver class for that driver is not present in the classpath and it is not possible to use it. See 10.3, "Working with Your JDBC Connection," on page 102 for how to install a JDBC driver.

Thanks to the JDBC URL Wizard, it is possible to automatically construct the JDBC URL to use the connection to the database by inserting the server name and the database name in the correct text fields. Click the **Wizard** button to create the URL.

Enter a username and password to access the database. By means of a check box option, you can save the password for the connection.

Jaspersoft Studio saves passwords in Eclipse secure storage.

If the password is empty, it is better if you specify that it be saved.

After you have inserted all the data, it is possible to verify the connection by clicking the **Test** button. If everything is okay, the dialog box shown in **Figure 10-4** will appear.

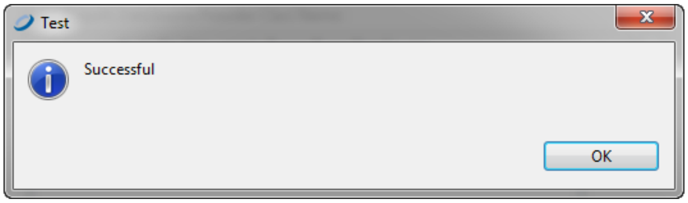


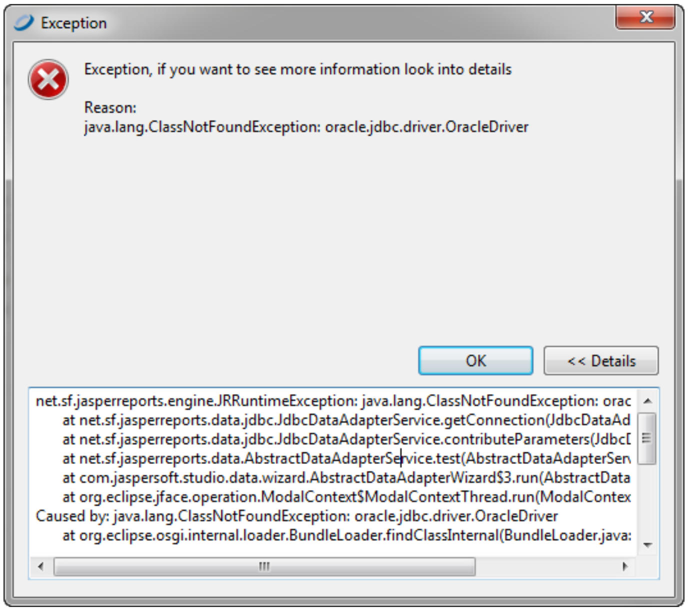
Figure 10-4 Test Confirmation Dialog

In general, the test can fail for a number reasons, the most frequent of which are:

- A `ClassNotFoundError` was thrown.
- The URL is not correct.
- Parameters are not correct for the connection (database is not found, the username or password is wrong, etc.).

10.2.1 ClassNotFoundError

The `ClassNotFoundError` exception occurs when the required JDBC driver is not present in the classpath. For example, suppose you wish to create a connection to an Oracle database. Jaspersoft Studio has no driver for this database, but you could be deceived by the presence of the `oracle.jdbc.driver.OracleDriver` driver in the JDBC drivers list shown in the window for creating new connections. If you were to select this driver, when you test the connection, the program will throw the `ClassNotFoundException`, as shown in **Figure 10-5**.



Exception, if you want to see more information look into details

Reason:
java.lang.ClassNotFoundException: oracle.jdbc.driver.OracleDriver

```
net.sf.jasperreports.engine.JRRuntimeException: java.lang.ClassNotFoundException: orac
    at net.sf.jasperreports.data.jdbc.JdbcDataAdapterService.getConnection(JdbcDataAd
    at net.sf.jasperreports.data.jdbc.JdbcDataAdapterService.contributeParameters(JdbcI
    at net.sf.jasperreports.data.AbstractDataAdapterService.test(AbstractDataAdapterSen
    at com.jaspersoft.studio.data.wizard.AbstractDataAdapterWizard$3.run(AbstractData
    at org.eclipse.jface.operation.ModalContext$ModalContextThread.run(ModalContex
Caused by: java.lang.ClassNotFoundException: oracle.jdbc.driver.OracleDriver
    at org.eclipse.osgi.internal.loader.BundleLoader.findClassInternal(BundleLoader.java:
```

Figure 10-5 `ClassNotFoundError` **exception**

What you have to do is to add the JDBC driver for Oracle, which is a file named ojdbc14.jar (or classes12.zip or classes11.zip for older versions) to the classpath (which is where the JVM searches for classes). As Jaspersoft Studio uses its own class loader, it will be enough add the ojdbc14.jar file to the Jaspersoft Studio classpath in the Options window (**Tools→Options**); the same can be done for directories containing classes and other resources.

10.2.2 URL Not Correct

If a wrong URL is specified (for example, due to a typing error), you'll get an arbitrary exception when you click the **Test** button. The exact cause of the error can be deduced by the stack trace available in the exception dialog box.

In this case, if possible, it is better to use the JDBC URL Wizard to build the JDBC URL and try again.

10.2.3 Parameters Not Correct for the Connection

The less-problematic error scenario is one in which you try to establish a connection to a database with the wrong parameters (invalid username or password, nonexistent of not running database, etc.). In this case, the same database will return a message that will be more or less explicit about the failure of the connection.

10.3 Working with Your JDBC Connection

When the report is created by using a JDBC connection, you specify a SQL query to extract the records to print from the database. This connection can also be used by a subreport or, for example, by a personalized lookup function for the decoding of particular data. For this reason, JasperReports puts at your disposal a special parameter named `REPORT_CONNECTION` of the `java.sql.Connection` type; it can be used in whatever expression you like, with a parameters syntax as follows:

```
$P{REPORT_CONNECTION}
```

This parameter contains the `java.sql.Connection` class passed to JasperReports from the calling program.

The use of JDBC or SQL connections is the simplest and easiest way to fill a report. The details for how to create a SQL query are explained in **Chapter 6**.

10.3.1 Fields Registration

In order to use SQL query fields in a report, you need to register them. It is not necessary to register all the selected fields—only those effectively used in the report are enough. For each field, you must specify its name and type. **Table 10-1** shows the mapping of the SQL types to the corresponding Java types.

Table 10-1 Conversion of SQL and JAVA types

SQL Type	Java Object	SQL Type	Java Object
CHAR	`String`	REAL	`Float`
VARCHAR	`String`	FLOAT	`Double`
LONGVARCHAR	`String`	DOUBLE	`Double`
NUMERIC	`java.math.BigDecimal`	BINARY	`byte[]`
DECIMAL	`java.math.BigDecimal`	VARBINARY	`byte[]`
BIT	`Boolean`	LONGVARBINARY	`byte[]`
TINYINT	`Integer`	DATE	`java.sql.Date`
SMALLINT	`Integer`	TIME	`java.sql.Time`
INTEGER	`Integer`	TIMESTAMP	`java.sql.Timestamp`
BIGINT	`Long`		

The table does not include the BLOB and CLOB types and other special types, such as ARRAY, STRUCT, and REF, because these types cannot be managed automatically by JasperReports. However, it is possible to use them by declaring them generically as `Object` and managing them by writing supporting static methods. The BINARY, VARBINARY, and LONGBINARY types should be dealt with in a similar way. With many databases, BLOB and CLOB can be declared as `java.io.InputStream`.

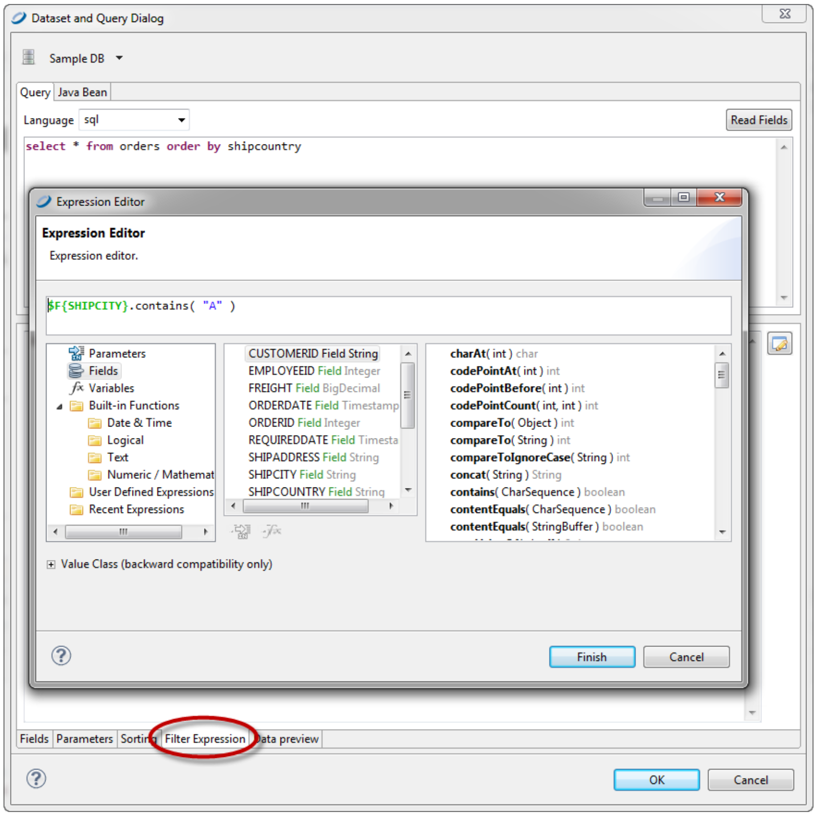
Whether a SQL type is converted to a Java object depends on the JDBC driver used.

For the automatic registration of SQL query fields, Jaspersoft Studio relies on the type proposed for each field by the driver itself.

10.3.2 Filtering Records

The records retrieved from a data source (or from the execution of a query through a connection) can be ordered and filtered.

Sort and filter options may be set from the Report query dialog box by clicking the **Dataset and Query** button ▤.



```
select * from orders order by shipcountry
```

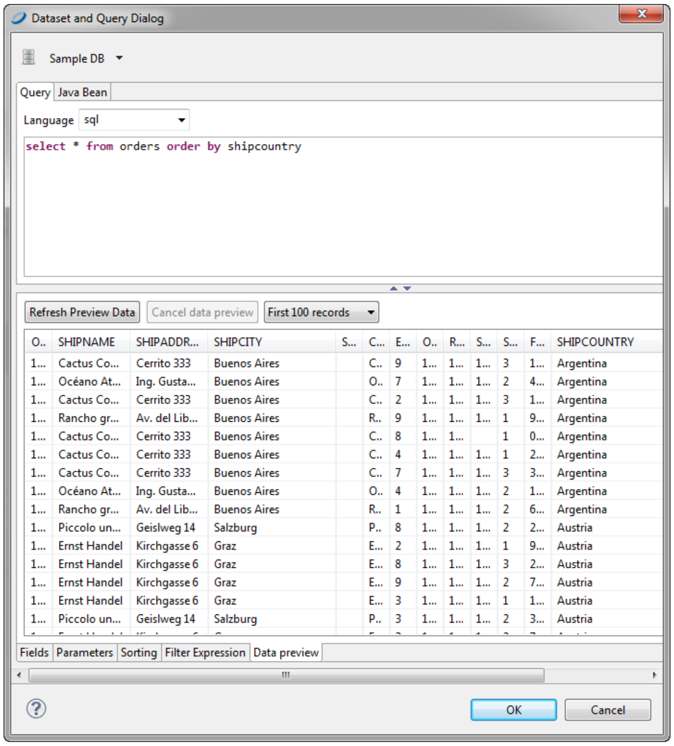
```
$F{SHIPCITY}.contains( "A" )
```

CUSTOMERID Field String
EMPLOYEEID Field Integer
FREIGHT Field BigDecimal
ORDERDATE Field Timestamp
ORDERID Field Integer
REQUIREDDATE Field Timesta
SHIPADDRESS Field String
SHIPCITY Field String
SHIPCOUNTRY Field String

charAt(int) char
codePointAt(int) int
codePointBefore(int) int
codePointCount(int, int) int
compareTo(Object) int
compareTo(String) int
compareToIgnoreCase(String) int
concat(String) String
contains(CharSequence) boolean
contentEquals(CharSequence) boolean
contentEquals(StringBuffer) boolean

Figure 10-6 Filter Expression Tab and Expression Editor

Clicking the **Data Preview** tab shows your filtered data.The filter expression must return a Boolean object: true if a particular record can be kept, false otherwise.



```
select * from orders order by shipcountry
```

O..	SHIPNAME	SHIPADDR...	SHIPCITY	S...	C...	E...	O..	R...	S...	S...	F...	SHIPCOUNTRY
1...	Cactus Co...	Cerrito 333	Buenos Aires		C..	9	1...	1...	1...	3	1...	Argentina
1...	Océano At...	Ing. Gusta...	Buenos Aires		O..	7	1...	1...	1...	2	4...	Argentina
1...	Cactus Co...	Cerrito 333	Buenos Aires		C..	2	1...	1...	1...	3	1...	Argentina
1...	Rancho gr...	Av. del Lib...	Buenos Aires		R..	9	1...	1...	1...	1	9...	Argentina
1...	Cactus Co...	Cerrito 333	Buenos Aires		C..	8	1...	1...		1	0...	Argentina
1...	Cactus Co...	Cerrito 333	Buenos Aires		C..	4	1...	1...	1...	1	2...	Argentina
1...	Cactus Co...	Cerrito 333	Buenos Aires		C..	7	1...	1...	1...	3	3...	Argentina
1...	Océano At...	Ing. Gusta...	Buenos Aires		O..	4	1...	1...	1...	2	1...	Argentina
1...	Rancho gr...	Av. del Lib...	Buenos Aires		R..	1	1...	1...	1...	2	6...	Argentina
1...	Piccolo un...	Geislweg 14	Salzburg		P..	8	1...	1...	1...	2	2...	Austria
1...	Ernst Handel	Kirchgasse 6	Graz		E...	2	1...	1...	1...	1	9...	Austria
1...	Ernst Handel	Kirchgasse 6	Graz		E...	8	1...	1...	1...	3	2...	Austria
1...	Ernst Handel	Kirchgasse 6	Graz		E...	9	1...	1...	1...	2	7...	Austria
1...	Ernst Handel	Kirchgasse 6	Graz		E...	3	1...	1...	1...	1	1...	Austria
1...	Piccolo un...	Geislweg 14	Salzburg		P..	3	1...	1...	1...	2	3...	Austria

Figure 10-7 Data Preview

If no fields can be selected with the **Add field** button, check to see if the report contains fields. If it does not, close the query dialog and register the fields and resume the sorting.

10.4 Understanding the JRDataSource Interface

Before proceeding with exploration of the different data sources Jaspersoft Studio provides at your disposal, it is necessary to understand how the `JRDataSource` interface works. Every JRDataSource must implement both of these methods:

```
public boolean next()
public Object getFieldValue(JRField jrField)
```

The first method, `public boolean next()`, is useful for moving a virtual cursor to the next record. In fact, data supplied by a JRDataSource is ideally organized into records as in a table. The second method, `public Object getFieldValue(JRField jrField)`, returns true if the cursor is positioned correctly in the subsequent record, false if there are no more available records.

Every time that JasperReports executes the `public boolean next()` method, all the fields declared in the report are filled and all the expressions (starting from those associated with the variables) are calculated again; subsequently, it will be decided whether to print the header of a new group, to go to a new page, and so on. When `next` returns false, the report is ended by printing all final bands (Group Footer, Column Footer, Last Page Footer, and Summary). The method can be called as many times as there are records present (or represented) from the data source instance.

The method `public Object getFieldValue(JRField jrField)` is called by JasperReports after a call to `next` results in a true value. In particular, it is executed for every single field declared in the report (see **Chapter 6** for the details on how to declare a report field). In the call, a `JRField` object is passed as a parameter; it is used to specify the name, the description and the type of the field from which you want to obtain the value (all this information, depending by the specific data source implementation, can be combined to extract the field value).

The type of the value returned by the `public Object getFieldValue(JRField jrField)` method has to be adequate to that declared in the `JRField` parameter, except when a `null` is returned. If the type of the field was declared as `java.lang.Object`, the method can return an arbitrary type. In this case, if required, a cast can be used in the expressions. A cast is a way to dynamically indicate the type on an object, the syntax of a cast is:

```
(type)object
```

in example:

```
(com.jaspersoft.ireport.examples.beans.PersonBean)$F{my_person}
```

Usually a cast is required when you need to call a method on the object that belongs to a particular class.

10.5 Data Source Types

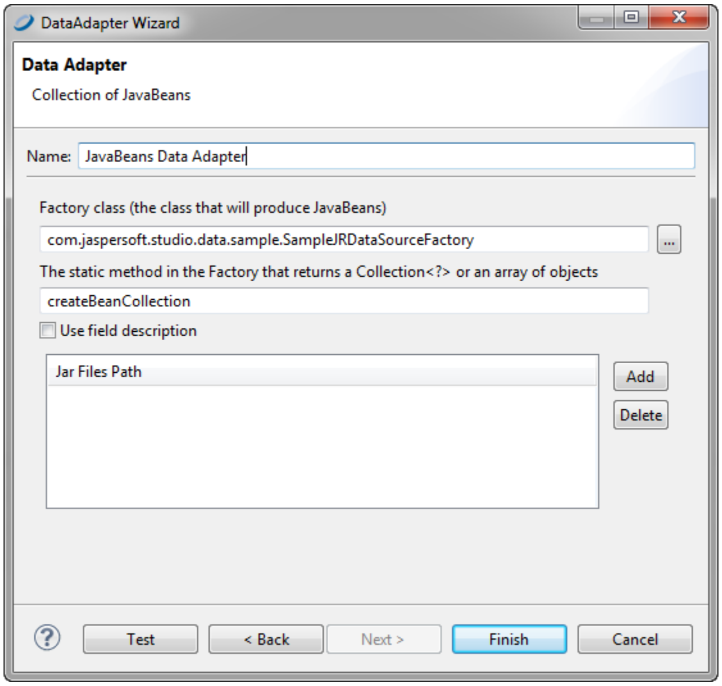
10.5.1 Using Collection of JavaBeans Data Sources

A JavaBeans set data source allows you to use JavaBeans as data to fill a report. In this context, a JavaBean is a Java class that exposes its attributes with a series of getter methods, with the following syntax:

```
public <returnType> getXXX()
```

where *`<returnType>`* (the return value) is a generic Java class or a primitive type (such as `int`, `double`, and so on).

In order to create a connection to handle JavaBeans, select **Collection of JavaBeans** in the list of data source types to bring up the dialog box shown in **Figure 10-8**.



DataAdapter Wizard

Data Adapter

Collection of JavaBeans

Name: JavaBeans Data Adapter

Factory class (the class that will produce JavaBeans)

com.jaspersoft.studio.data.sample.SampleJRDataSourceFactory

The static method in the Factory that returns a Collection<?> or an array of objects

createBeanCollection

□ Use field description

Jar Files Path

Figure 10-8 JavaBeans set data source

Once again, the first thing to do is to specify the name of the new data source.

10.5.2 Fields of a JavaBean Set Data Source

One peculiarity of a JavaBeans set data source is that the fields are exposed through `get` methods. This means that if the JavaBean has a `getXyz()` method, `xyz` is the name of a record field (the JavaBean represent the record).

In this example, the `PersonBean` object shows two fields: name and age; register them in the fields list as `String` and `Integer`, respectively.

Create a new empty report and add the two fields by right-clicking the **Fields** node in the outline view and selecting **Add field**. The name and the type of the fields are: name (`java.lang.String`) and age (`java.lang.Integer`).

Drag the fields into the **Detail** band and run the report (being sure the active connection is the Test Factory). To refer to an attribute of an attribute, you can use a special notation in which attributes are separated by periods. For example, to access the `street` attribute of a hypothetical `Address` class contained in the PersonBean, you can use the syntax `address.street`. The real call would be `<someBean>.getAddress().getStreet()`.

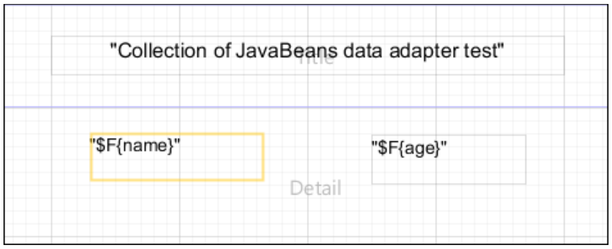


Figure 10-9 Layout of a JavaBeans-Based Report

If the flag `Use field description` is set when you are specifying the properties of your JavaBeans set data source, the mapping between JavaBean attribute and field value is done using the field description instead of the field name. The data source will consider only the description to look up the field value, and the field can have any name.

Jaspersoft Studio provides a visual tool to map JavaBean attributes to report fields. To use it, open the query window, go to the tab **JavaBean Data Source**, insert the full class name of the bean you want to explore, and click **Read attributes**. The tab will be populated with the attributes of the specified bean class.

- For attributes that are also Java objects, you can double-click the objects to display the objects' other attributes.
- To map a field, select an attribute name and click the **Add Selected Field(s)** button.

10.5.3 Using XML Data Adapters

JasperReports provides the ability to use XML as a data adapter in three different ways: XML documents, remote XML documents, and XMLA servers.

An XML document is typically organized as a tree, and its structure hardly matches the table-like form required by JasperReports. For this reason, you have to use an XPath expression to define a node set. The specifications of the XPath language are available at http://www.w3.org/TR/xpath; it is used to identify values or nodes in an XML document. Some examples will be useful to help you to know how to define the nodes.

The XML file in **Table 10-1** is a hypothetical address book in which different people appear, grouped in categories. At the end of the categories list, a second list, of favorites objects, appears. In this case, it is possible to define different node set types. The choice is determined by how you want to organize the data in your report.

Code Example 10-1 Example XML file

```
<addressbook>
  <category name="home">
    <person id="1">
      <lastname>Davolio</lastname>
      <firstname>Nancy</firstname>
    </person>
    <person id="2">
      <lastname>Fuller</lastname>
      <firstname>Andrew</firstname>
    </person>
    <person id="3">
      <lastname>Leverling</lastname>
    </person>
  </category>

  <category name="work">
    <person id="4">
      <lastname>Peacock</lastname>
      <firstname>Margaret</firstname>
    </person>
  </category>
  <favorites>
    <person id="1"/>
    <person id="3"/>
  </favorites>
</addressbook>
```

To select only the people contained in the categories (that is, all the people in the address book), use the following expression:

```
/addressbook/category/person
```

Four nodes will be returned. These are shown in **Table 10-1**.

Code Example 10-2 Node set with expression `/addressbook/category/person`

```
<person id="1">
  <lastname>Davolio</lastname>
  <firstname>Nancy</firstname>
</person>
<person id="2">
  <lastname>Fuller</lastname>
  <firstname>Andrew</firstname>
</person>
<person id="3">
  <lastname>Leverling</lastname>
</person>
<person id="4">
  <lastname>Peacock</lastname>
  <firstname>Margaret</firstname>
</person>
```

If you want to select the people appearing in the `favorites` node, the expression to use is

```
/addressbook/favorites/person
```

Two nodes will be returned nodes.

```
<person id="1"/>
<person id="3"/>
```

Here is another expression. It is a bit more complex, but it shows all the power of the Xpath language. The idea is to select the person nodes belonging to the work category. The expression to use is the following:

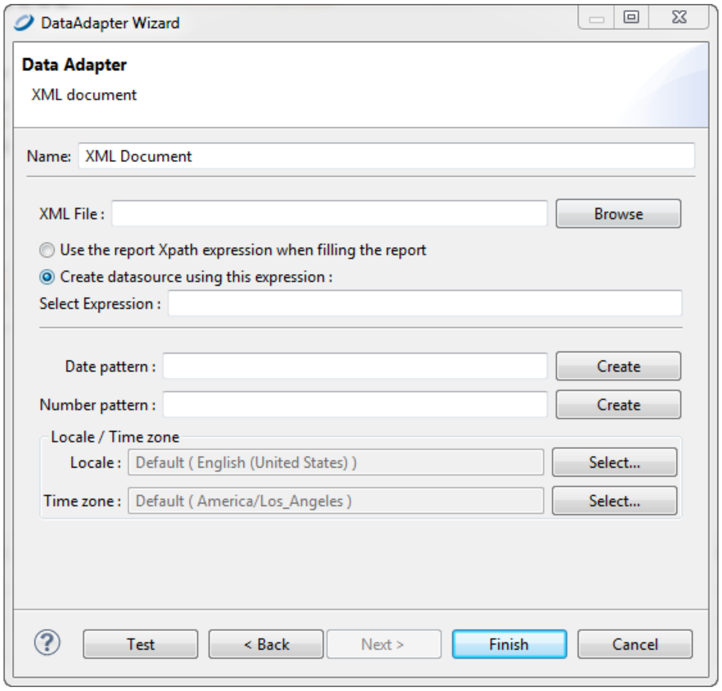
```
/addressbook/category[@name = "work"]/person
```

The expression will return only one node, that with an ID equal to 4, as shown here.

```
<person id="4">
  <lastname>Peacock</lastname>
  <firstname>Margaret</firstname>
</person>
```

After you have created an expression for the selection of a node set, you can proceed to the creation of an XML data source.

Open the window for creating a new data source and select **XML File** data source from the list of connection types to bring up the dialog box shown in Figure 10-10.



Name: XML Document

○ Use the report Xpath expression when filling the report

● Create datasource using this expression :

Locale : Default (English (United States))

Time zone : Default (America/Los_Angeles)

Figure 10-10 Configuring an XML Data Adapter

The only mandatory information to specify is the XML file name. Optionally, you can provide a set of nodes, using a pre-defined static XPath expression. Alternatively, the XPath expression can be set directly inside the report.

I always suggest that you use a report-defined XPath expression. The advantage of this solution is the ability to use parameters inside the XPath expression, which acts like a real query on the supplied XML data. Optionally, you can specify Java patterns to convert dates and numbers from plain strings to more appropriate Java objects (like **Date** and **Double**). For the same purpose, you can define a specific locale and time zone to use when parsing the XML stream.

10.5.4 Registration of the Fields for an XML Data Source

In the case of an XML data source, the definition of a field in the report needs a particular expression inserted as a field description in addition to the type and the name. As the data source aims always to be one node of the selected node set, the expressions are relative to the current node.

To select the value of an attribute of the current node, use the following syntax:

```
@<name attribute>
```

For example, to define a field that must point to the `id` attribute of a person (attribute id of the node person), it is sufficient to create a new field, name it as you want, and set the description to

```
@id
```

Similarly, it is possible to get to the child nodes of the current node. For example, if you want to refer to the `lastname` node, child of person, use the following syntax:

```
lastname
```

To move to the parent value of the current node (for example, to determine the category to which a person belongs), use a slightly different syntax:

```
ancestor::category/@name
```

The ancestor keyword indicates that you are referring to a parent node of the current node; in particular, you are referring to the first parent of category type, of which you want to know the value of the name attribute.

Now, let’s see everything in action. Prepare a simple report with the registered fields shown here:

Field name	Description	Type
`id`	@id	`Integer`
`lastname`	lastname	`String`
`firstname`	firstname	`String`
`name of category`	ancestor::category/@name	`String`

Jaspersoft Studio provides a visual tool to map XML nodes to report fields; to use it, open the query window and select **XPath** as the query language. If the active connection is a valid XML data source, the associated XML document will be shown in a tree view. To register the fields, set the record node by right-clicking a Person node and selecting the menu item **Set record node**. The record nodes will become bold.

Then one by one, select the nodes or attributes and select the pop-up menu item **Add node as field** to map them to report fields. Jaspersoft Studio will determine the correct XPath expression to use and will create the fields for you. You can modify the generated field name and set a more suitable field type after the registration of the field in the report (which happens when you close the query dialog).

Insert the different fields into the Detail band. The XML file used to fill the report is that shown:

The XPath expression for the node set selection specified in the query dialog is:

```
/addressbook/category/person
```

10.5.5 XML Data Source and Subreports

A node set allows you to identify a series of nodes that represent, from a `JRDataSource` point of view, some records. However, due to the tree-like nature of an XML document, it may be necessary to see other node sets that are subordinated to the main nodes.

Consider the XML in **Table 10-1**. This is a slightly modified version of the document presented in **Code Example 10-1**. For each person node, a hobbies node is added which contains a series of hobby nodes and one or more e-mail addresses.

Code Example 10-3 Complex XML example

```
<addressbook>
  <category name="home">
    <person id="1">
      <lastname>Davolio</lastname>
      <firstname>Nancy</firstname>
      <email>davolio1@sf.net</email>
      <email>davolio2@sf.net</email>
      <hobbies>
              <hobby>Music</hobby>
              <hobby>Sport</hobby>
      </hobbies>
    </person>
    <person id="2">
      <lastname>Fuller</lastname>
      <firstname>Andrew</firstname>
      <email>af@test.net</email>
      <email>afullera@fuller.org</email>
      <hobbies>
              <hobby>Cinema</hobby>
              <hobby>Sport</hobby>
      </hobbies>
    </person>
  </category>

  <category name="work">
    <person id="3">
      <lastname>Leverling</lastname>
      <email>leverling@xyz.it</email>
    </person>
    <person id="4">
      <lastname>Peacock</lastname>
      <firstname>Margaret</firstname>
      <email>margaret@foo.org</email>
      <hobbies>
              <hobby>Food</hobby>
              <hobby>Books</hobby>
      </hobbies>
    </person>
  </category>
  <favorites>
    <person id="1"/>
    <person id="3"/>
  </favorites>
</addressbook>
```

What we want to produce is a document that is more elaborate than those you have seen until now—for each person, we want to present their e-mail addresses, hobbies, and favorite people.

To obtain such a document, it is necessary to use subreports; in particular, you will need a subreport for the e-mail addresses list, one for hobbies, and one for favorite people (that is a set of nodes out of the scope of the XPath query we used). To generate these subreports, you need to understand how to produce new data sources to feed them. In this case, you use the `JRXmlDataSource`, which exposes two extremely useful methods:

```
public JRXmlDataSource dataSource(String selectExpression)
public JRXmlDataSource subDataSource(String selectExpression)
```

The difference between the two is that the first method processes the expression by applying it to the whole document, starting from the actual root, while the second assumes the current node is the root.

Both methods can be used in the data source expression of a subreport element to produce dynamically the data source to pass to the element. The most important thing to note is that this mechanism allows you to make both the data source production and the expression of node selection dynamic.

The expression to create the data source that will feed the subreport of the e-mail addresses will be

```
((net.sf.jasperreports.engine.data.JRXmlDataSource)
  $P{REPORT_DATA_SOURCE}).subDataSource("/person/email")
```

This code returns all the e-mail nodes that are direct descendants of the present node (person).

The expression for the hobbies subreport will be similar, except for the node selection:

```
((net.sf.jasperreports.engine.data.JRXmlDataSource)
   $P{REPORT_DATA_SOURCE}).subDataSource("/person/hobbies/hobby")
```

Next, declare the master report’s fields. In the subreport, you have to refer to the current node value, so the field expression will be simply a dot (.),

Proceed with building your three reports: `xml_addressbook.jasper`, `xml_addresses.jasper`, and `xml_hobbies.jasper`.

In the master report, `xml_addressbook.jrxml`, insert a group named “Name of category,” in which you associate the expression for the category field (`$F{name of category}`). In the header band for Name of category, insert a field in which you will view the category name. By doing this, the names of the different people will be grouped by category (as in the XML file).

In the Detail band, position the `id`, `lastname`, and `firstname` fields. Underneath these fields, add the two Subreport elements, the first for the e-mail addresses, the second for the hobbies.

The e-mail and hobby subreports are identical except for the name of the field in each one. The two reports should be as large as the Subreport elements in the master report, so remove the margins and set the report width accordingly.

Preview both the subreports just to compile them and generate the relative `.jasper` files. You will get an error during the fill process, but it’s okay. We have not set an Xpath query, so JasperReports is not able to get any data. You can resolve the problem by setting a simple Xpath query (it will not be used in the final report), or you can preview the subreport using an empty data source (you will have to select it from the combo box in the tool bar).

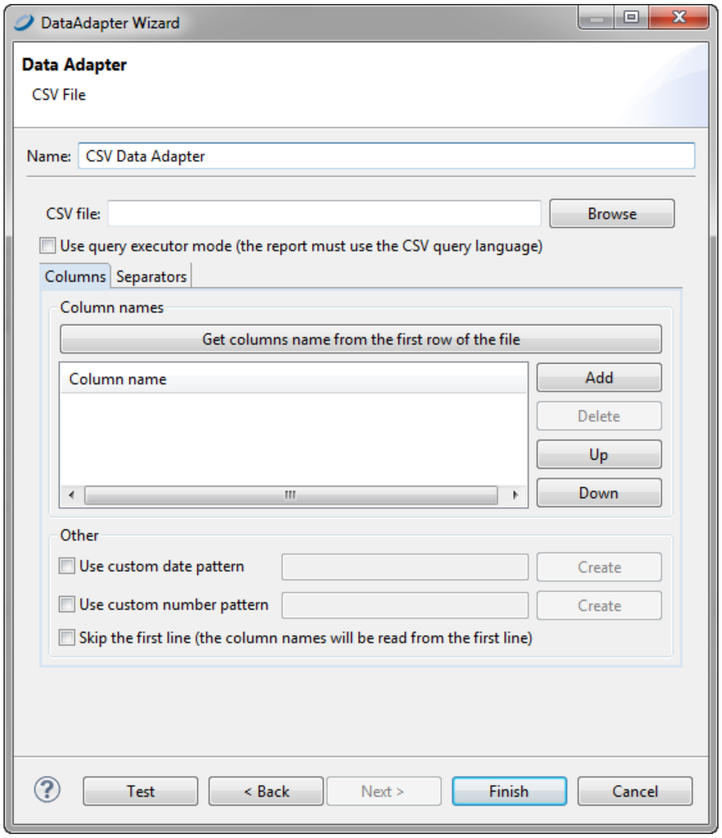
When the subreports are done, execute the master report. If everything is okay, you will see a display of people grouped by home and work categories and the subreports associated with each person.

As this example demonstrates, the real power of the XML data source is the versatility of XPath, which allows navigating the node selection in a refined manner.

10.5.6 Using CSV Data Sources

Initially, the data source for CSV documents was a very simple data source proof-of-concept that showed how to implement a custom data source. The CSV data source interface was improved when JasperReports added a native implementation to fill a report using a CSV file.

To create a connection based on a CSV file, click the **New** button in the Connections/Datasources dialog box and select **File CSV data source** from the data source types list to bring up the dialog box shown in **Figure 10-11**.



DataAdapter Wizard

Data Adapter

CSV File

Name: CSV Data Adapter

CSV file:

□ Use query executor mode (the report must use the CSV query language)

Column names

Column name

Other

□ Use custom date pattern

□ Use custom number pattern

□ Skip the first line (the column names will be read from the first line)

Figure 10-11 CSV Data Adapter

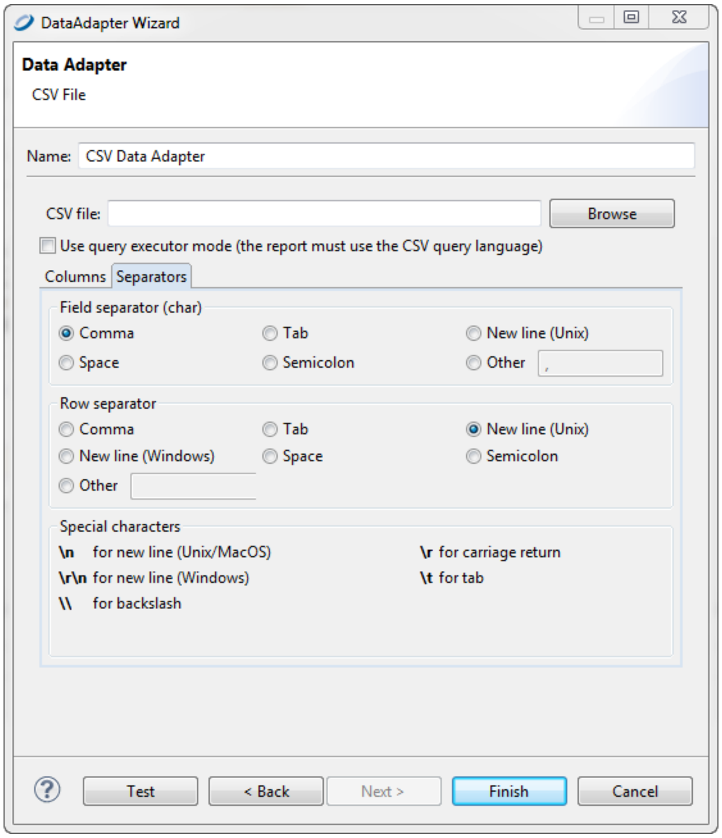
Set a name for the connection and choose a CSV file. Then declare the fields in the data source.

- If the first line in your file contains the names of the columns, click the **Get column names from the first row of the file** button and select the **Skip the first line** check box option. This forces JasperReports to skip the first line (the one containing your column labels). In any case, the column names that are read from the file are used instead of the declared ones, so avoid modifying the names found with the **Get column names** button.
- If the first line of your CSV file *doesn't* contain the column names, set a name for each column using the syntax `COLUMN_0, COLUMN_1`, and so on.

If you define more columns than the ones available, you'll get an exception at report filling time.

JasperReports assumes that, for each row, all the columns have a value (even if they are empty).

If your CSV file uses nonstandard characters to separate fields and rows, you can adjust the default setting for separators using the Separators tab.



DataAdapter Wizard

Data Adapter

CSV File

Name: CSV Data Adapter

CSV file: [Browse]

□ Use query executor mode (the report must use the CSV query language)

Columns | Separators

Field separator (char)

● Comma ○ Tab ○ New line (Unix)

○ Space ○ Semicolon ○ Other ,

Row separator

○ Comma ○ Tab ● New line (Unix)

○ New line (Windows) ○ Space ○ Semicolon

○ Other

Special characters

\n for new line (Unix/MacOS) **\r** for carriage return

\r\n for new line (Windows) **\t** for tab

**** for backslash

Figure 10-12 Separators Tab

10.5.7 Registration of the Fields for a CSV Data Source

When you create a CSV data source, you must define a set of column names that will be used as fields for your report. To add them to the fields list, set your CSV data source as the active connection and open the Report query dialog box. Open the **Dataset and Query** Dialog and click the **Read Fields** button.

By default, Jaspersoft Studio sets the class type of all fields to `java.lang.String`. If you are sure that the text of a particular column can be easily converted to a number, a date, or a Boolean value, set the correct field type yourself after the fields are added to your report.

The pattern used to recognize a timestamp (or date) object can be configured at the data source level by selecting the **Use custom date format** check box option.

10.5.8 Using JREmptyDataSource

JasperReports provides a special data source named `JREmptyDataSource`.

This source returns true to the `next` method for the record number (by default only one), and always returns `null` to every call of the `getFieldValue` method. It is like having records without fields, that is, an empty data source.

The two constructors of this class are:

```
public JREmptyDataSource(int count)
public JREmptyDataSource()
```

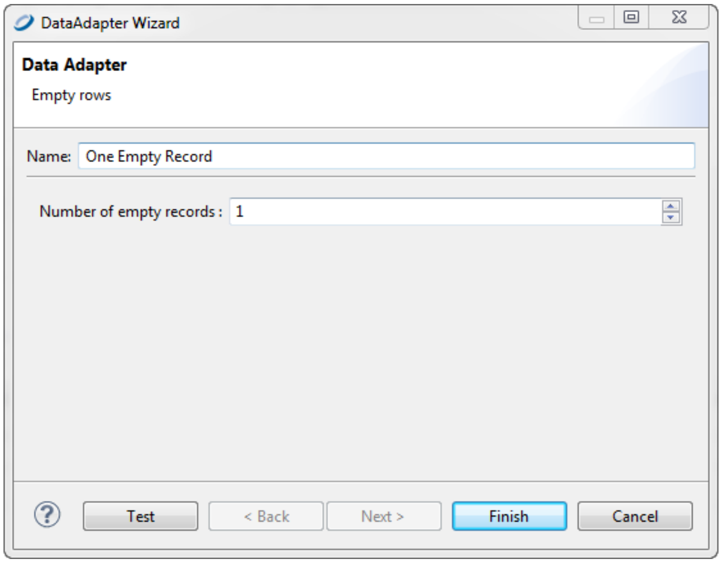
The first constructor indicates how many records to return, and the second sets the number of records to one.

By default, Jaspersoft Studio provides a pre-configured empty data source that returns a single record.

To create a new empty data source with more records:

1. Double-click **One Empty Record** from the Repository View.

The **Data Adapter Wizard** appears, with Empty rows.



Name: One Empty Record

Number of empty records : 1

Figure 10-13 Data Adapter Wizard > Empty Record

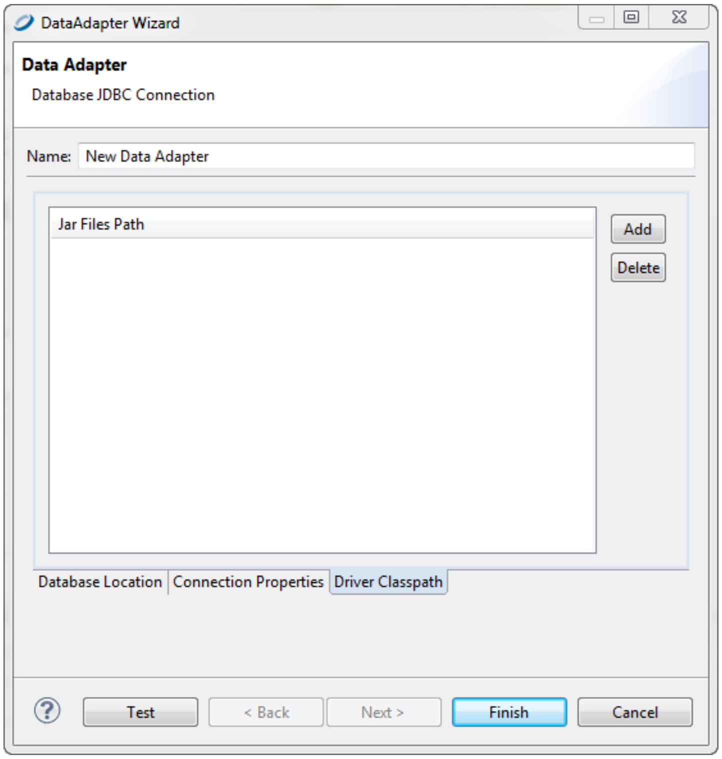
2. Set the number or empty records that you need. Remember, whatever field you will add to the report, its value will be set to `null`. Since this data source doesn't care about field names or types, this is a perfect way to test any report (keeping in mind that the fields will be always set to `null`).
3. Click **Finish**.

10.5.9 Using HQL and Hibernate Connections

JasperReports provides a way to use HQL directly in your report. To do so, first set up a Hibernate connection. Expand your classpath to include all classes, JARs, and configuration files used by your Hibernate mapping. In other words, Jaspersoft Studio must be able to access all the *.hbm.xml files you plan to use, the JavaBeans declared in those files, the `hibernate.cfg.xml` file, and any other JARs used (for example, JARs that access the database under Hibernate).

To add these objects to the classpath:

1. Double-click **New Data Adapter in** the Repository View.
2. Click the **Driver Classpath** tab.



DataAdapter Wizard

Data Adapter

Database JDBC Connection

Name: New Data Adapter

Figure 10-14 Data Adapter Wizard > Add JAR Files

3. Click the **Add** button.
4. Navigate to your JAR file.
5. Click **Open**.
6. Repeat for any additional files.
7. Give your data adapter a unique name.
8. Click the **Test** button to check the path resolution to be certain that `hibernate.cfg.xml` is in the classpath.

   Currently Jaspersoft Studio works only with the first hibernate.cfg.xml file found in the classpath.
9. Click **Finish**.

If you use the Spring framework, you can use a Spring configuration file to define your connection. In this case, you'll need to set the configuration file name and the Session Factory Bean ID.

Now that a Hibernate connection is available, use an HQL query to select the data to print. You can use HQL in the same way that you use SQL: open the Report query dialog box and choose HQL as the query language from the combo box at the top of the window.

When you enter an HQL query, Jaspersoft Studio tries to retrieve the available fields. According to the JasperReports documentation, the field mappings are resolved as follows:

- If the query returns one object per row, a field mapping can be one of the following:
  - If the object's type is a Hibernate entity or component type, the field mappings are resolved as the property names of the entity/component. If a select alias is present, it can be used to map a field to the whole entity/component object.
  - Otherwise, the object type is considered scalar, and only one field can be mapped to its value.
- If the query returns a tuple (object array) per row, a field mapping can be one of the following:
  - A select alias. The field will be mapped to the value corresponding to the alias.
  - A property name prefixed by a select alias and a ".". The field will be mapped to the value of the property for the object corresponding to the alias. The type corresponding to the select alias has to be an entity or component.

If you don't understand this field mapping information, simply accept the fields listed by Jaspersoft Studio when the query is parsed.

Jaspersoft Studio provides a mapping tool to map objects and attributes to report fields. The objects (or JavaBeans) available in each record are listed in the combo box on top of the object tree.

To add a field from the tree, select the corresponding node and click the **Add selected field(s)** button.

10.5.10 Using a Hadoop Hive Connection

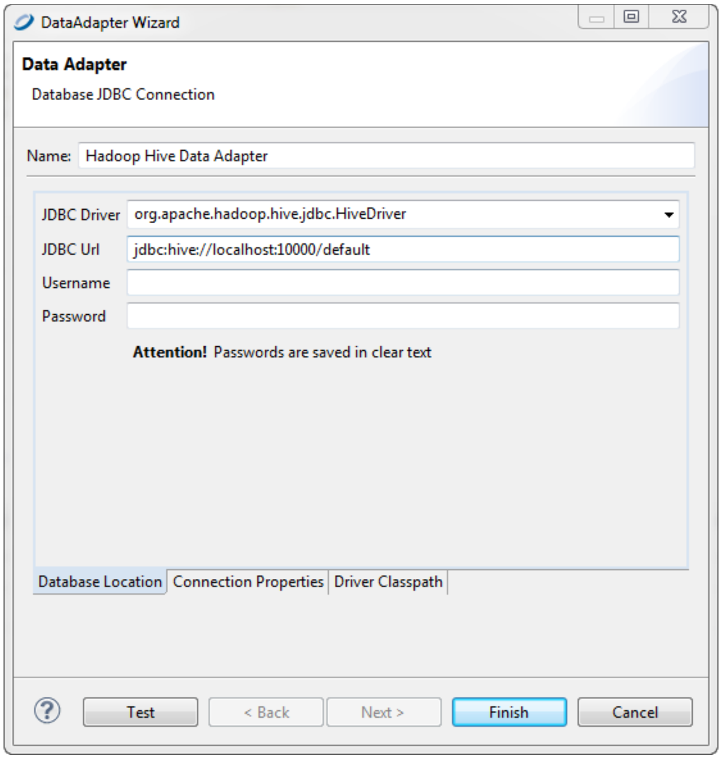
JasperReports provides a way to use Hive in your reports. Unlike traditional databases, Hadoop systems support huge amounts of data, generally called big data. However, this capability has a cost: high latency with access times between 30 seconds and 2 minutes.

Because of the latency, reports based on Hadoop-Hive data sources are best suited to be run in the background or to be scheduled. For example, the report could be run at 6 a.m. and the HTML or PDF could be exported and stored for anyone wanting to access the report during the day.

To use Hadoop Hive with Jaspersoft Studio:

1. Double-click **New Data Adapter** in the Repository view.

   The Data Adapter Wizard opens.



DataAdapter Wizard

Data Adapter

Database JDBC Connection

Name: Hadoop Hive Data Adapter

JDBC Driver: org.apache.hadoop.hive.jdbc.HiveDriver

JDBC Url: jdbc:hive://localhost:10000/default

Username:

Password:

Attention! Passwords are saved in clear text

Figure 10-15 Hadoop Hive Connection

2. Choose Hadoop Hive connection as your JDBC driver.
3. Enter the path to your HIve JDBC.

   Usually it will look something like `jdbc:hive://localhost:10000/default`.
4. Click the **Test** button to check the path resolution.
5. Click **Finish**.

Now that a Hive connection is available, use a HiveQL query to retrieve data in your report. You can use HiveQL the same way that you use SQL. To use a HiveQL query, open the Report Query dialog box and choose HiveQL as the query language from the combo box at the top of the window.

When you enter a HiveQL query, Jaspersoft Studio retrieves all available fields. In the next two screens, select the fields you wish to display in your report, and how you want to group them. After defining your HiveQL query and choosing your fields, your report is configured to receive data from your Hadoop Hive data source.

The Hive JDBC driver does not yet fully implement the JDBC specifications, and it does not yet work correctly with the JasperReports Server metadata layer (Data Domains). If reporting against a Hive data source and you are using JasperReports Server, use Topics rather than Domains.

10.5.11 Implementing a New JRDataSource

Sometimes the `JRDataSource` supplied with JasperReports cannot satisfy your needs. In these cases, it is possible to write a new `JRDataSource`. This operation is not complex; in fact, all you have to do is create a class that implements the `JRDataSource` interface that exposes two simple methods: `next` and `getFieldValue`:

Code Example 10-4 The JRDataSource interface

```
package net.sf.jasperreports.engine;
public interface JRDataSource
{
      public boolean next() throws JRException;
      public Object getFieldValue(JRField jrField) throws JRException;
}
```

The `next` method is used to set the current record into the data source. It has to return true if a new record to elaborate exists; otherwise it returns false.

If the `next` method has been called positively, the `getFieldValue` method has to return the value of the requested field or `null`. In particular, the requested field name is contained in the `JRField` object passed as a parameter. Also, `JRField` is an interface through which it is possible to get the information associated with a field—the name, description, and Java type that represents it (as mentioned previously in **10.4, “Understanding the JRDataSource Interface,” on page 104**).

Now try writing your personalized data source. The idea is a little original—you have to write a data source that explores the directory of a file system and returns the found objects (files or directories). The fields you will make to manage your data source will be the file name, which you will name FILENAME; a flag that indicates whether the object is a file or a directory, which you will name IS_DIRECTORY; and the file size, if available, which you will name SIZE.

There will be two constructors for your data source: the first will receive as a parameter the directory to scan, the second will have no parameters and will use the current directory to scan.

Once instantiated, the data source will look for the files and the directories present in the way you indicate and fill the array files.

The `next` method will increase the index variable that you use to keep track of the position reached in the array files, and it will return true until you reach the end of the array.

Code Example 10-5   Sample personalized data source

```
import net.sf.jasperreports.engine.*;
import java.io.*;

public class JRFileSystemDataSource implements JRDataSource
{
File[] files = null;
int    index = -1;

public JRFileSystemDataSource(String path)
{
File dir = new File(path);
if (dir.exists() && dir.isDirectory())
{
files = dir.listFiles();
}
}

public JRFileSystemDataSource()
{
this(".");
}

public boolean next() throws JRException
{
index++;
if (files != null && index < files.length)
{
return true;
}
return false;
}

        public Object  getFieldValue(JRField jrField) throws JRException
        {
        File f = files[index];
        if (f == null) return null;
        if (jrField.getName().equals("FILENAME"))
        {
        return f.getName();
        }
        else if (jrField.getName().equals("IS_DIRECTORY"))
        {
        return new Boolean(f.isDirectory());
        }
```

Code Example 10-5 Sample personalized data source, continued

```
        else if (jrField.getName().equals("SIZE"))
        {
        return new Long(f.length());
        }
        // Field not found...
        return null;
        }
}
```

The `getFieldValue` method will return the requested file information. Your implementation does not use the information regarding the return type expected by the caller of the method, but it assumes that the name has to be returned as a string, the flag `IS_DIRECTORY` as a Boolean object, and the file size as a `Long` object.

In the next section, you will learn how to use your personalized data source in Jaspersoft Studio and test it.

10.5.12 Using a Custom JasperReports Data Source with Jaspersoft Studio

Jaspersoft Studio provides support for almost all the data sources provided by JasperReports, such as `JRXmlDataSource`, `JRBeanArrayDataSource`, and `JRBeanCollectionDataSource`.

To use your personalized data sources, a special connection is provided. It is useful for employing whatever `JRDataSource` you want to use through some kind of factory class that provides an instance of that `JRDataSource` implementation. The factory is just a simple Java class useful to test your data source and to fill a report in Jaspersoft Studio. The idea is the same as what you have seen for the JavaBeans set data source—it is necessary to write a Java class that creates the data source through a static method and returns it. For example, if you want to test the `JRFileSystemDataSource` in the previous section, you need to create a simple class like that shown in this code sample:

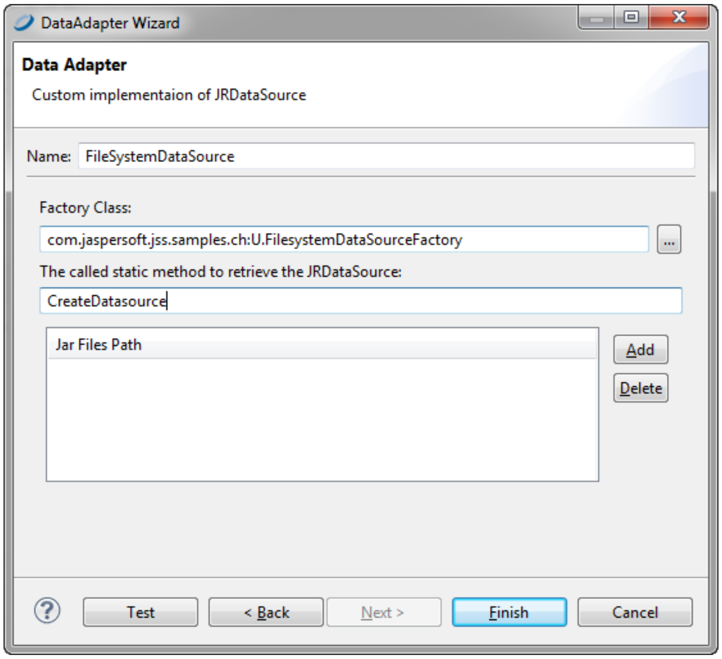
Code Example 10-6 Class for testing a custom data source

```
import net.sf.jasperreports.engine.*;
public class FileSystemDataSourceFactory {
  public static JRDataSource createDatasource()
{
return new JRFileSystemDataSource("/");
}
}
```

This class, and in particular the static method that will be called, will execute all the necessary code for instancing the data source correctly. In this case, you create a new `JRFileSystemDataSource` object by specifying a way to scan the directory root (`"/"`).

Now that you have defined the way to obtain the `JRDataSource` you prepared and the data source is ready to be used, you can create the connection through which it will be used.

Create a new connection as you normally would (see **10.2, "Creating and Using Database JDBC Connections," on page 99**), then select **Custom implementation of JRDataSource** from the list and specify a data source name such as `TestFileSystemDataSource` (or whatever name you wish), as shown in **Figure 10-16**.



DataAdapter Wizard

Data Adapter

Custom implementaion of JRDataSource

Name: FileSystemDataSource

Factory Class:

com.jaspersoft.jss.samples.ch:U.FilesystemDataSourceFactory

The called static method to retrieve the JRDataSource:

CreateDatasource

Figure 10-16 Configuring a Custom Data Adapter

Next, specify the class and method to use to obtain an instance of your `JRFileSystemDataSource`, that is, `TestFileSystemDataSource` and `test`.

There is no automatic method to find the fields managed by a custom data source.

In this case, you know that the `JRFileSystemDataSource` provides three fields: `FILENAME` (String), `IS_DIRECTORY` (Boolean), and `SIZE` (Long). After you have created these fields, insert them in the report's Detail band.

Divide the report into two columns, and in the Column Header band, insert Filename and Size tags. Then add two images, one representing a document and the other an open folder. In the `Print when` expression setting of the Image element that is placed in the foreground, insert the expression `$F{IS_DIRECTORY}`, or use as your image expression a condition like the following:

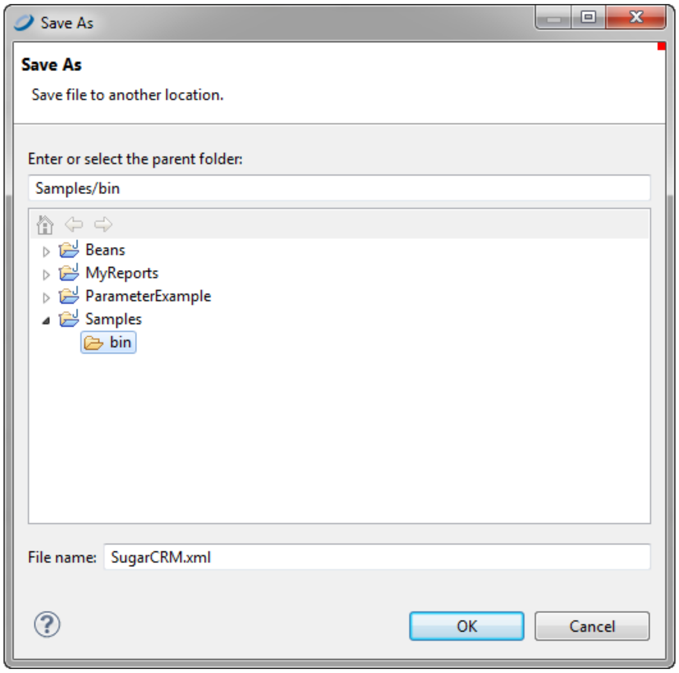
```
    ($F{IS_DIRECTORY}) ? "folder.png" : "file.png"
```

In this example, the class that instantiated the `JRFileSystemDataSource` was very simple. However, you can use more complex classes, such as one that obtains the data source by calling an Enterprise JavaBean or by calling a web service.

10.6 Importing and Exporting Data Sources

To simplify the process of sharing data source configurations, Jaspersoft Studio provides a mechanism to import and export data source definitions.

To export one or more data sources, right-click the data adapter and choose **Export to File**. Jaspersoft Studio will ask you to name the file and indicate the destination for the exported information. The created file is a simple XML file.



Enter or select the parent folder: Samples/bin

File name: SugarCRM.xml

Figure 10-17 Export to File Dialog

A file exported with Jaspersoft Studio can be imported. Right-click in the Repository view, and choose **Import from Workspace**. Since an exported file can contain more than one data source or connection definition, the import process will add all the data sources found in the specified file to the current list.

If a duplicated data source name is found during the import, Jaspersoft Studio will append a number to the imported data source name.

Chapter 11 Using Tables

The Table component displays data coming from a secondary dataset. It is an extremely powerful component, which is able in many situations to replace the use of subreports.

The Table wizard allows you to create a complex table with a few clicks. Each table cell can be simple as a text element or it can contain an arbitrary set of report elements including nested tables, creating very sophisticated layouts.

This chapter contains the following sections:

- **Creating a Table**
- **Editing a Table**
- **Table Structure**
- **Working with Columns**

11.1 Creating a Table

To create a table in a report, drag the Table element from the Elements palette inside any band of the report. The Table Wizard opens with a choice of creating a table from a new or existing dataset.

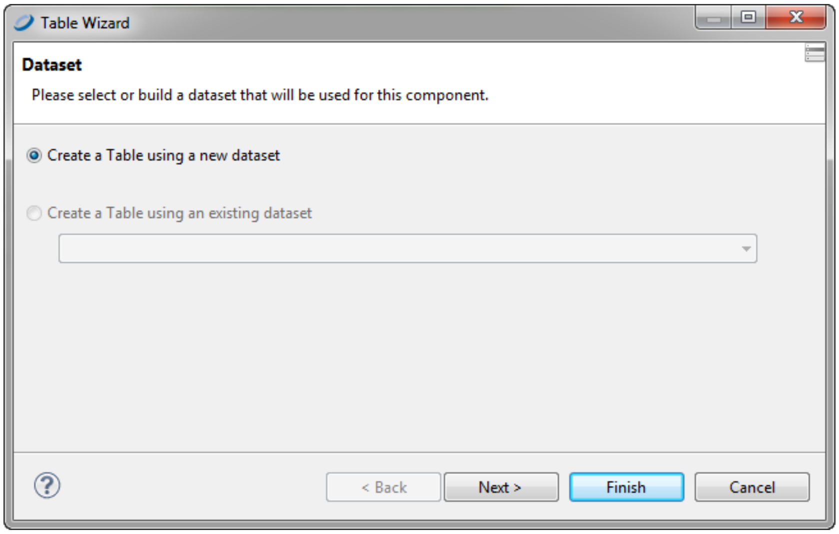


Table Wizard

Dataset

Please select or build a dataset that will be used for this component.

● Create a Table using a new dataset

○ Create a Table using an existing dataset

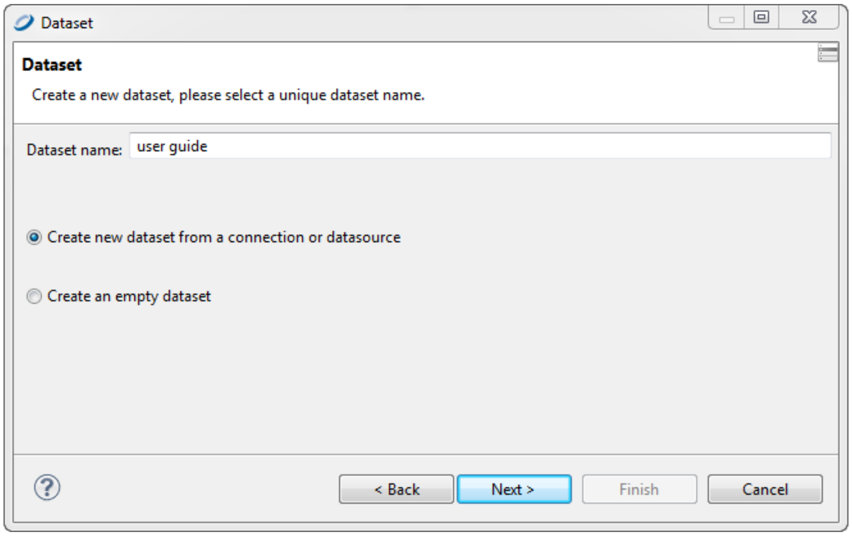
Figure 11-1 Table Wizard - New Table

If you choose to create an empty table, a new dataset is created and bound to the table. If you do not have an existing dataset, that choice will be greyed out and **Create a Table using a new dataset** will be selected.

To create a table from a new dataset:

1. From **Table Wizard - New Table** click **Next**.

   The Dataset window appears.



Dataset

Dataset

Create a new dataset, please select a unique dataset name.

Dataset name: user guide

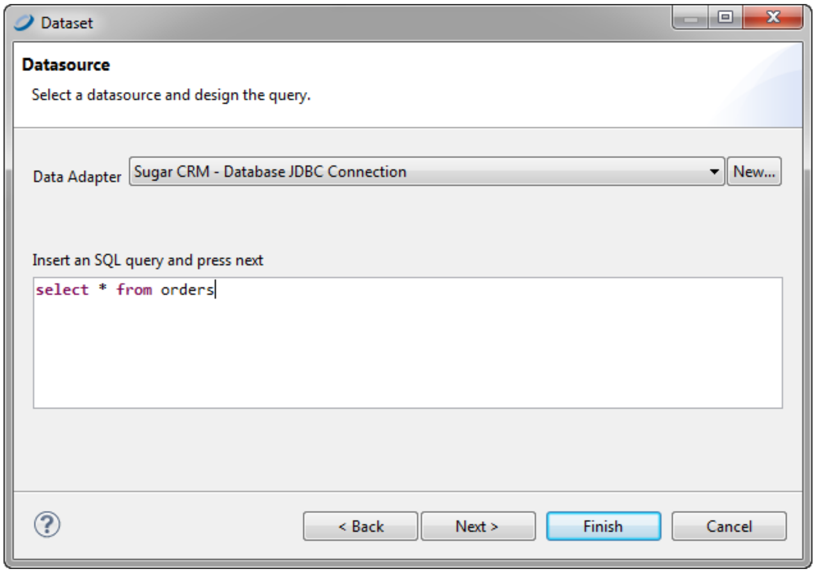
● Create new dataset from a connection or datasource

○ Create an empty dataset

Figure 11-2 Table Wizard - Dataset

2. Name your dataset and select whether to create the dataset from a connection or data source or to create an empty dataset. For this example, select **Create new dataset from a connection or datasource**. Click **Next**.

The **Dataset > Datasource** window appears.



Dataset

Datasource

Select a datasource and design the query.

Data Adapter: Sugar CRM - Database JDBC Connection

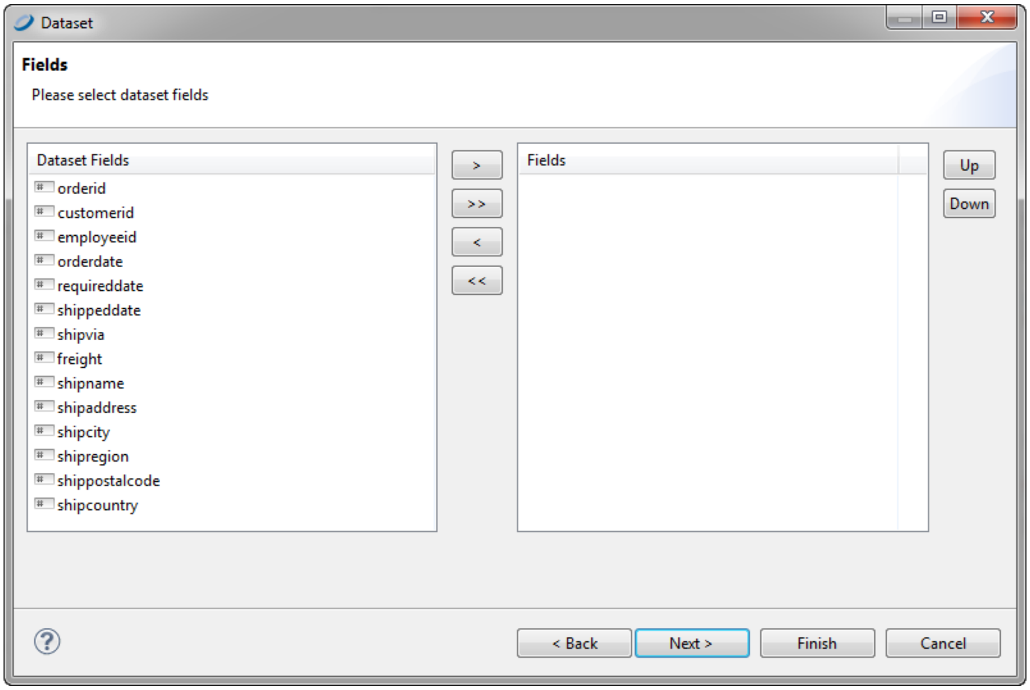
Insert an SQL query and press next

```
select * from orders
```

Figure 11-3 Table Wizard Dataset > Datasource

3. Select a data source and enter an SQL query such as: `select * from orders`. Click **Next**.

   The **Dataset > Fields** window appears.



Dataset

Fields

Please select dataset fields

Dataset Fields
- orderid
- customerid
- employeeid
- orderdate
- requireddate
- shippeddate
- shipvia
- freight
- shipname
- shipaddress
- shipcity
- shipregion
- shippostalcode
- shipcountry

Fields

Figure 11-4 Table Wizard Dataset > Fields

4. Using the arrow buttons, select the fields you want in your table. Click **Next**.

The **Dataset > Group By** window appears.

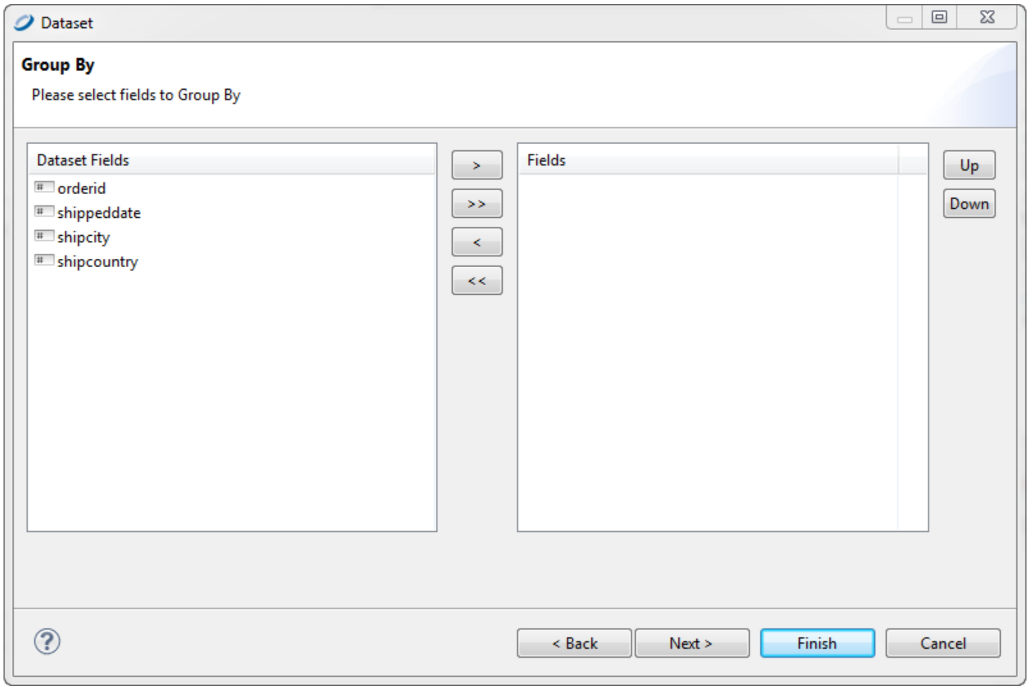


Figure 11-5 Table Wizard Dataset > Group By

5. Select a field or fields to group by. Click **Next**.

   The **Table Wizard > Connection** window appears.

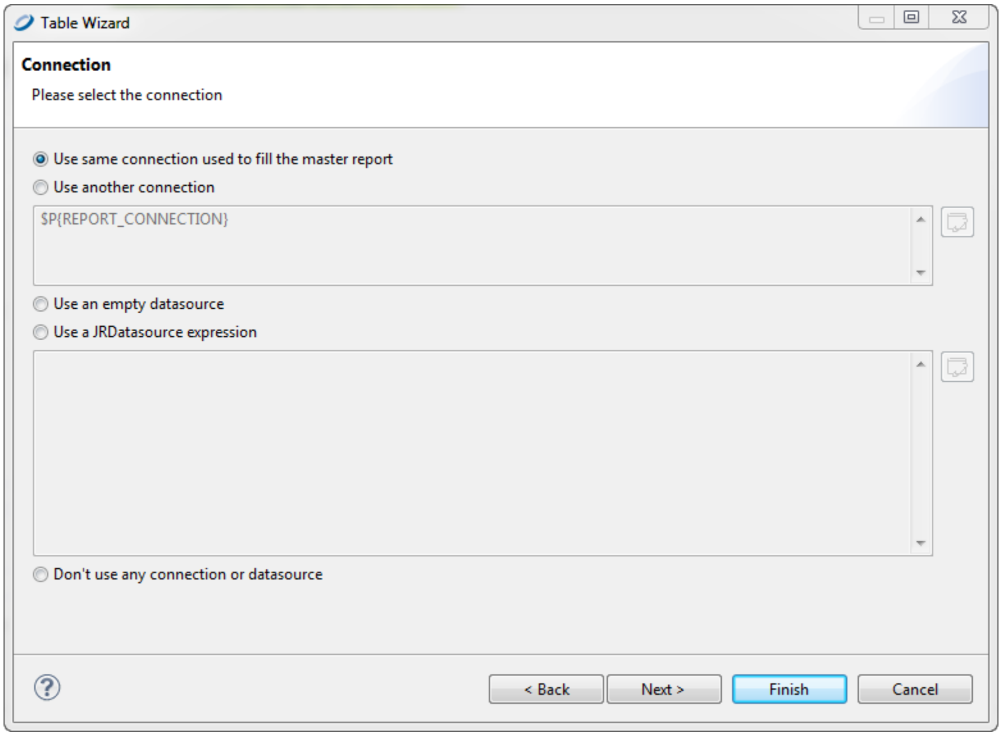


Table Wizard
Connection
Please select the connection
Use same connection used to fill the master report
Use another connection
$P{REPORT_CONNECTION}
Use an empty datasource
Use a JRDatasource expression
Don't use any connection or datasource
< Back
Next >
Finish
Cancel

Figure 11-6 Table Wizard > Connection

6. In this window you have the choice to use the same connection as the master report, another connection, an empty data source, a JRDatasrouce expression or no connection or data source.

   In most cases, choose **Use the same connection used to fill the master report**.

   Choose empty data source if you want to create a table without relying on external data.

   Choose **Don't use any connection or datasource** in the unusual case that the dataset uses a special query executer that does not need a source to produce data.

   Click **Next**. The **Table Wizard > Table Columns** window appears.

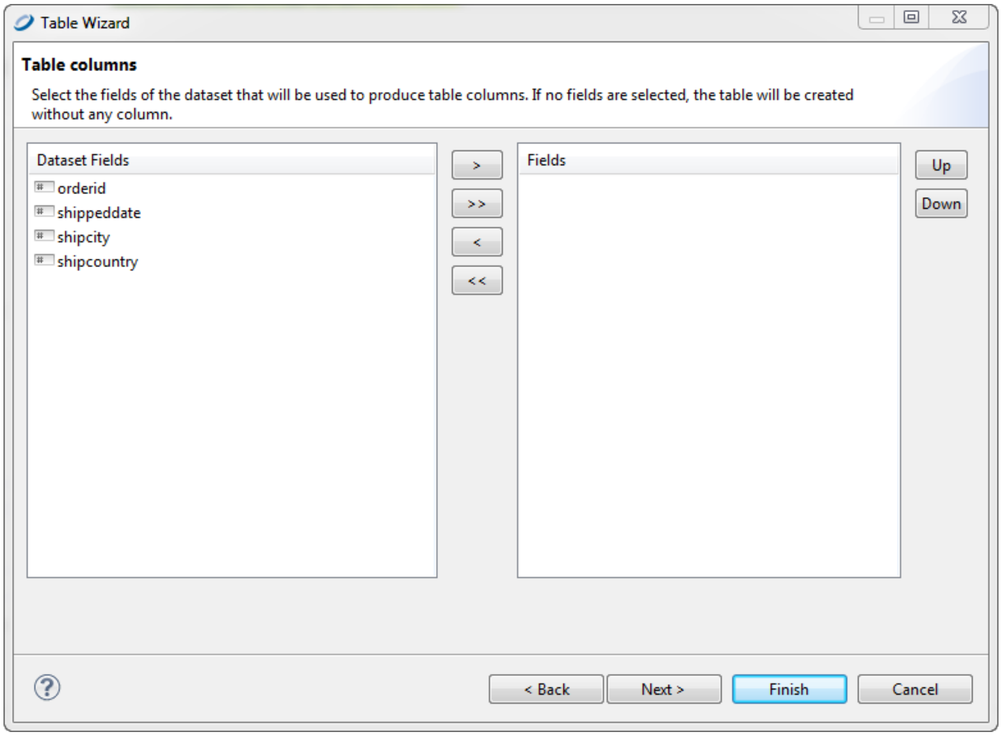


Table Wizard

Table columns

Select the fields of the dataset that will be used to produce table columns. If no fields are selected, the table will be created without any column.

Dataset Fields

- orderid
- shippeddate
- shipcity
- shipcountry

Fields

Figure 11-7 Table Wizard > Table Columns

7. Using the arrow buttons, select the fields you want to use as table columns. Click **Next**.

The **Table Wizard > Layout** window appears.

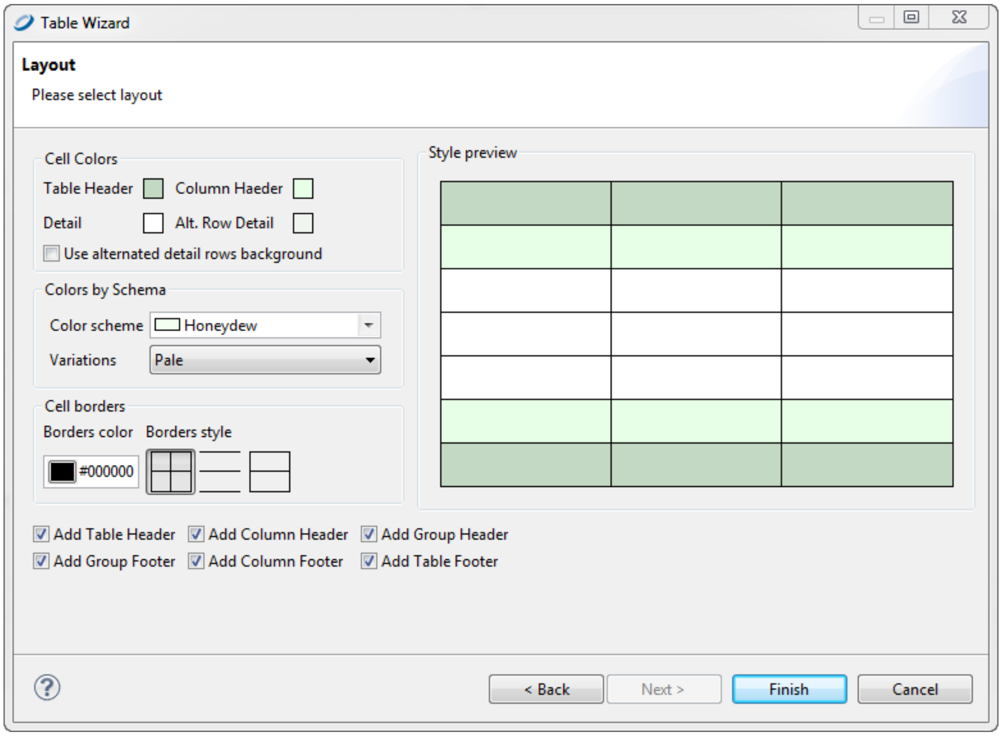


Table Wizard

Layout

Please select layout

Cell Colors

Table Header 🟩 Column Haeder □

Detail □ Alt. Row Detail □

□ Use alternated detail rows background

Colors by Schema

Color scheme: Honeydew

Variations: Pale

Cell borders

Borders color: #000000

Borders style

☑ Add Table Header ☑ Add Column Header ☑ Add Group Header

☑ Add Group Footer ☑ Add Column Footer ☑ Add Table Footer

Figure 11-8 Table Wizard > Layout

8. Select the layout for your table, and click **Finish**.

   The table appears in your report, in the spot where you dragged the element.

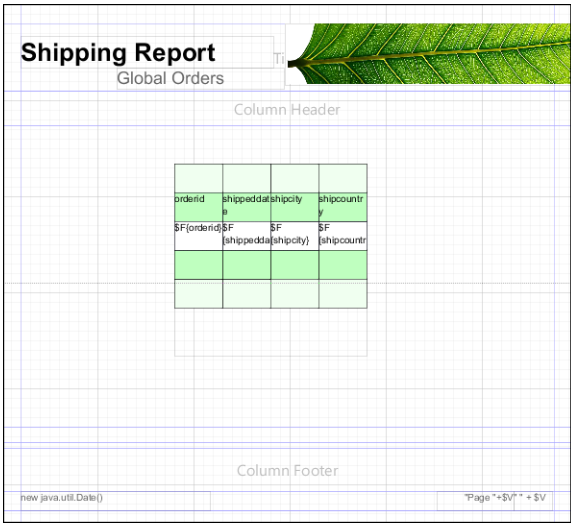
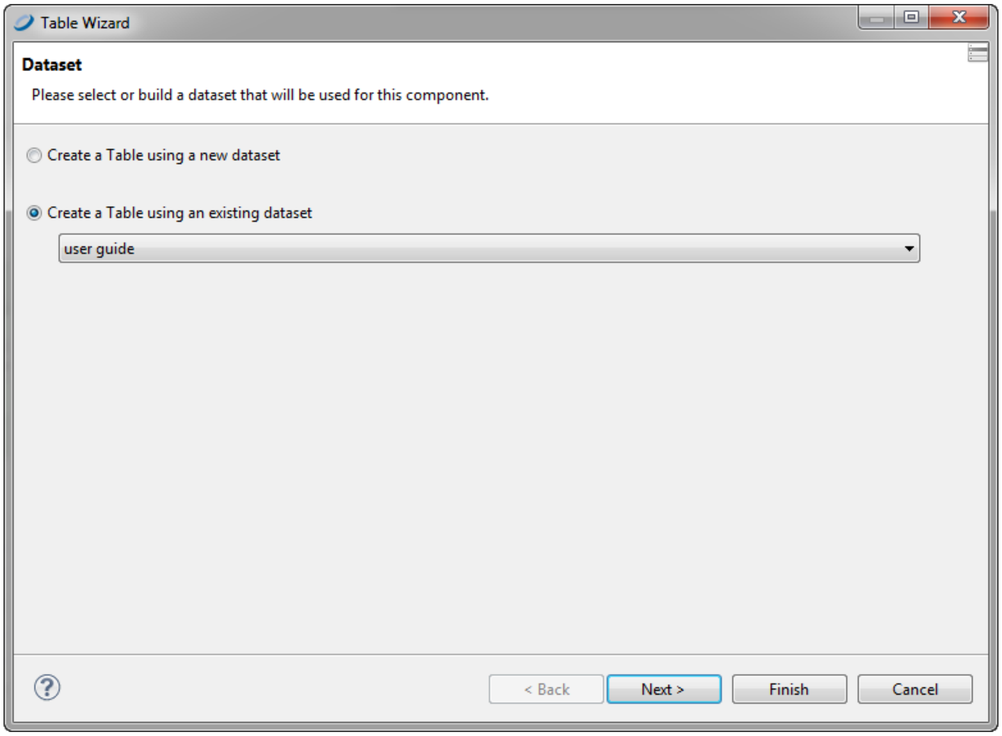


Figure 11-9 Report Containing a Table

To create a table using an existing dataset:

Creating a table using an existing dataset is largely the same as creating a table using a new dataset.

1. In the **Table Wizard > Dataset** window, select **Create a Table using an existing dataset**.
2. Select a dataset from the drop-down.



Dataset

Please select or build a dataset that will be used for this component.

○ Create a Table using a new dataset

● Create a Table using an existing dataset

user guide

Figure 11-10 Table Wizard > Dataset

3. Click **Next**. The **Table Wizard > Connection** window appears.

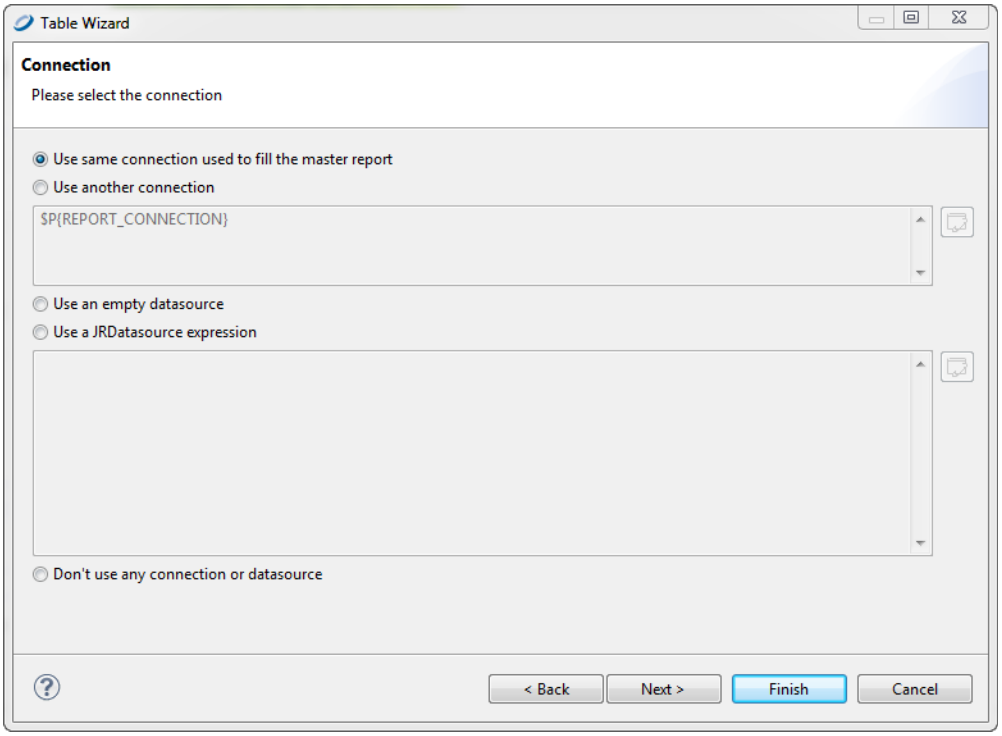


Table Wizard

Connection

Please select the connection

● Use same connection used to fill the master report

○ Use another connection

$P{REPORT_CONNECTION}

○ Use an empty datasource

○ Use a JRDatasource expression

○ Don't use any connection or datasource

Figure 11-11 Table Wizard > Connection

From this point the steps are the same as creating a table using a new dataset.

11.2 Editing a Table

You can edit tables from the **Design** tab or the **Source** tab.

In the source the tags are labeled:

- `Table`: External border of the table.
- `Table_TH`: Table header background color and cell borders.
- `Table_CH`: Table column background.
- `Table_TD`: Detailed cell style. `Table_TD` can be nested to display alternating background color for the detail rows.

11.2.1 Editing Table Styles

The look and feel of a table can be edited by choosing an element from the **Table Styles** tab. To create a style, click the **Create a New Style** button. The **Table Wizard > Layout** window opens (Figure 11-8). You can select an existing style, which adds it to your palette, or you can create a new style and save it to your palette. If you want the style to have a name, you must add a name in the **Layout** window, even for the default styles.

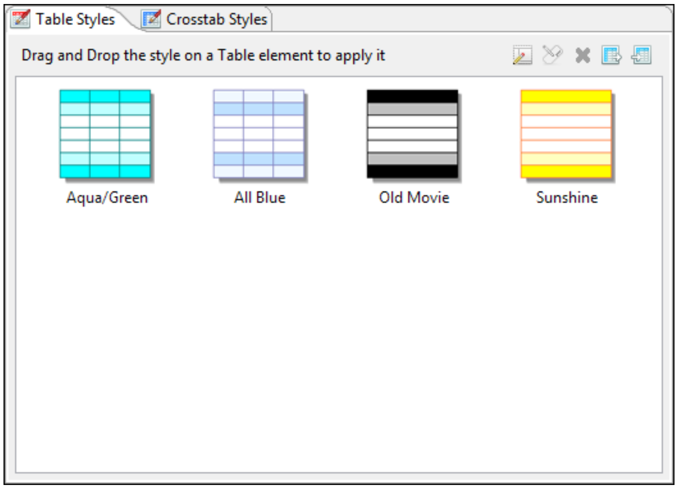


Figure 11-12 Table Styles Tab

To edit a style in the palette, double-click the style. The **Table Wizard > Layout** window opens, allowing you to edit the style and save it either as a new style or with the same name. To edit table layout in the **Design** view, right-click the table and choose **Change Table Style**.

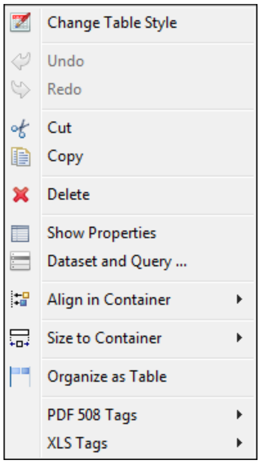


Figure 11-13 Change Table Style

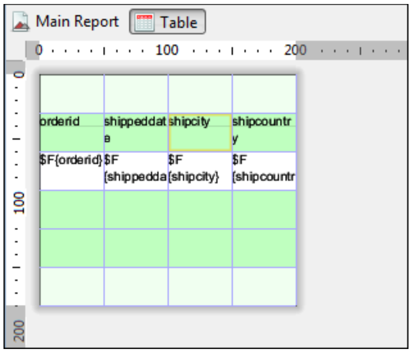
The **Table Wizard > Layout** window appears (**Figure 11-8**).

You can also decide which table sections to create. If the dataset for the table contains groups, it can be convenient to select the check boxes **Add Group Headers** and **Add Group Footers**, which are by default not selected. This option will create group header and footer sections in addition to the other sections.

To delete a style from the **Table Styles** tab, right click and choose **Delete Style**.

11.2.2 Editing Cell Contents

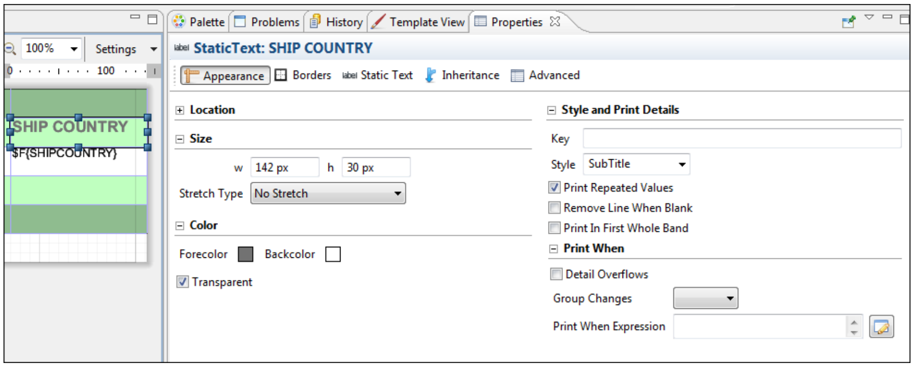
You can edit the content, style, and size of each cell or group of cells in your table. To edit table content, double-click your table. The table opens in a separate tab in **Design** view.



orderid	shippeddat e	shipcity	shipcountr y
$F{orderid}	$F {shippedda	$F {shipcity}	$F {shipcountr

Figure 11-14 Table Editing Tab

To edit the content and style of a cell, click the cell. Switch to the **Properties** tab. On this tab you can edit location, size, color, style, and print details for the cell.



StaticText: SHIP COUNTRY

Location

Size

w 142 px h 30 px

Stretch Type No Stretch

Color

Forecolor ■ Backcolor □

☑ Transparent

Style and Print Details

Key

Style SubTitle

☑ Print Repeated Values

□ Remove Line When Blank

□ Print In First Whole Band

Print When

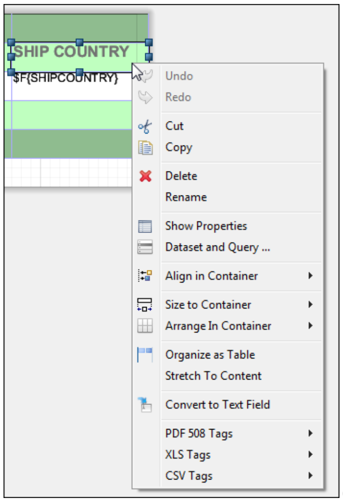
□ Detail Overflows

Group Changes

Print When Expression

Figure 11-15 Cell with Open Properties Tab

You can edit the size and position of a cell, as well as copy or delete it, by right-clicking on the cell.



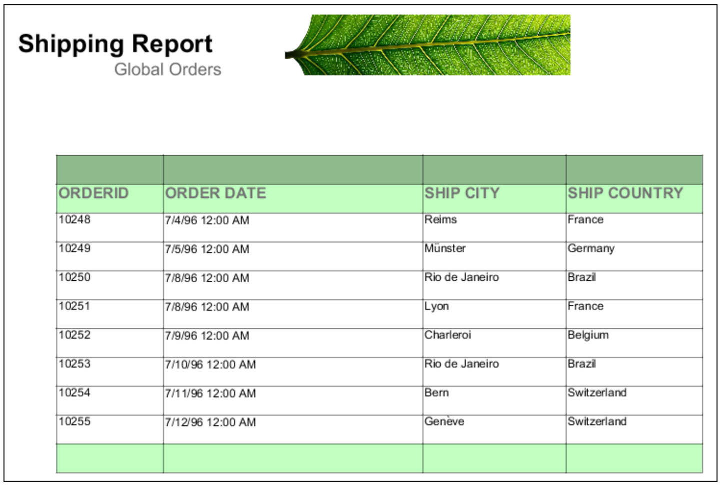
o

Figure 11-16 Cell Right-Click Menu

If you want multiple cells to have the same properties, shift-click the cells. Any changes you apply on the **Properties** tab will apply to every selected cell.

You cannot select an entire row with one click. You must shift-click all the cells in the row. You can select an entire column. See **“Working with Columns” on page 136** for more information about columns.

The following figure is the table created in **“Creating a Table” on page 124**, after formatting and editing it:



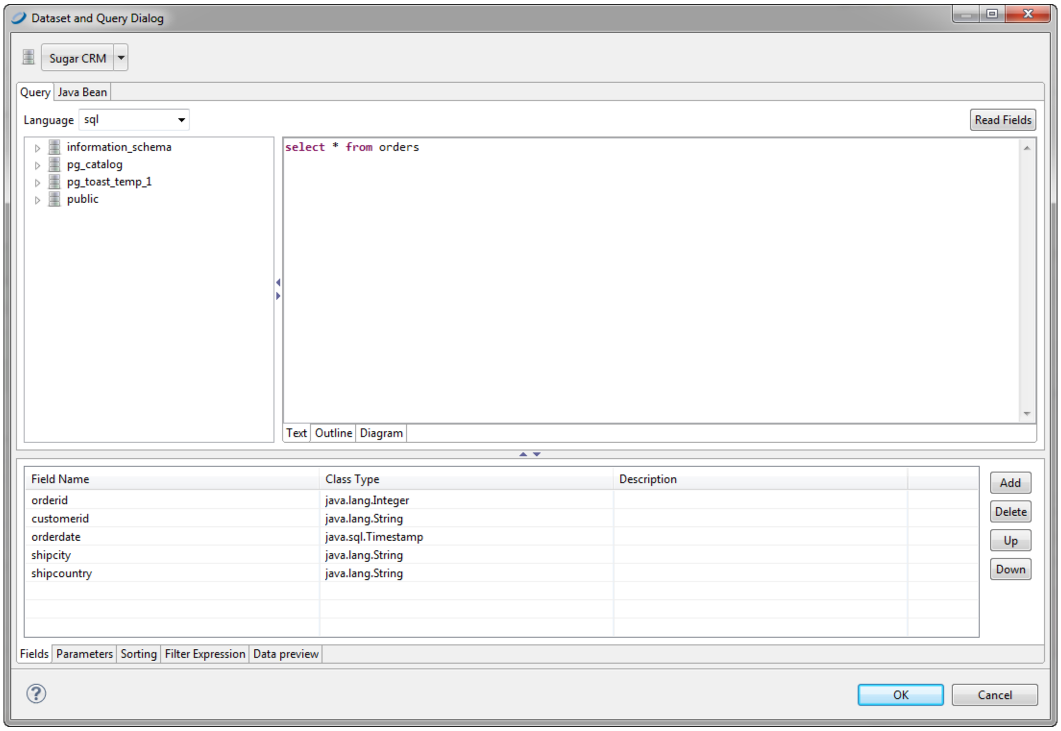
Shipping Report
Global Orders

ORDERID	ORDER DATE	SHIP CITY	SHIP COUNTRY
10248	7/4/96 12:00 AM	Reims	France
10249	7/5/96 12:00 AM	Münster	Germany
10250	7/8/96 12:00 AM	Rio de Janeiro	Brazil
10251	7/8/96 12:00 AM	Lyon	France
10252	7/9/96 12:00 AM	Charleroi	Belgium
10253	7/10/96 12:00 AM	Rio de Janeiro	Brazil
10254	7/11/96 12:00 AM	Bern	Switzerland
10255	7/12/96 12:00 AM	Genève	Switzerland

Figure 11-17 Formatted Table

11.2.3 Editing Table Data

The dataset run for the table can be edited by right-clicking the table and selecting **Dataset and Query**. The **Dataset and Query** dialog appears, filled in with the data and query from your current table.



```
select * from orders
```

Field Name	Class Type	Description
orderid	java.lang.Integer	
customerid	java.lang.String	
orderdate	java.sql.Timestamp	
shipcity	java.lang.String	
shipcountry	java.lang.String	

Figure 11-18 Dataset and Query Dialog

In this dialog, you can set the values of the dataset parameters. The use of parameters allows you to dynamically filter the data used to fill the table. Suppose, for instance, that you have a report that prints a set of orders. If we want to use a table to display the order details, a parameter allows us to specify the order ID to filter the order details in an SQL query.

Unlike charts and crosstabs, a table always requires a subdataset. Tables cannot use the main dataset.

11.3 Table Structure

In Jaspersoft Studio, tables consist of cells and columns. This section provides more information about working with each of these elements.

11.3.1 Table Elements

A table must have at least one column, but it can have more than one. A set of columns can be grouped into column groups; the groups can have headers that span several columns.

Each table is divided into sections similar the main document bands:

- **Table header and footer**, which are each printed only once.
- **Column header and footer**, which are repeated on each page that the table spans. If there are one or more column groups, the table can display a group header and footer section for each group, as well as for each column.
- **Detail** section that is repeated for each record of the table. Each column contains only one detail section, and the section cannot span multiple columns.

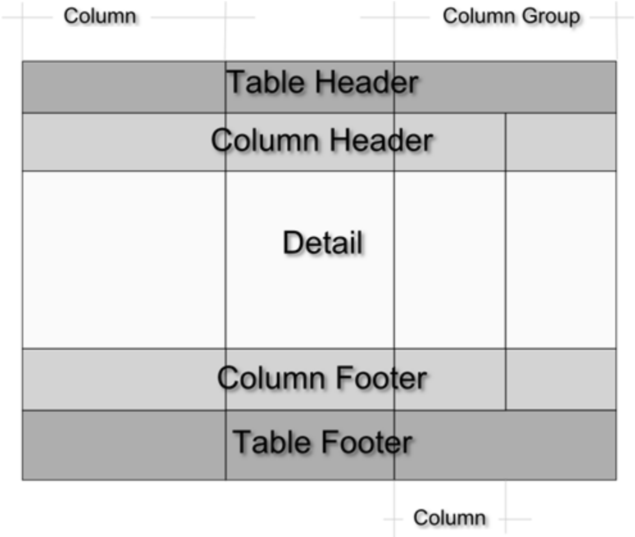


Figure 11-19 Table Structure

In the **Outline** view, table sections are shown as child nodes of the table element node.

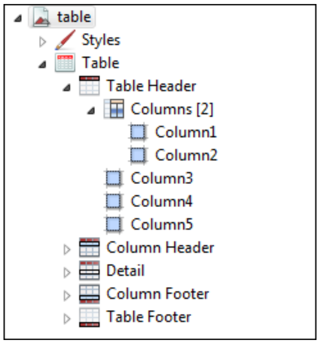


Figure 11-20 Table in Outline View

In the **Design** view, each column has a cell for each section (*i.e.*, one cell for the table header section, another for the table footer, etc.). A cell can be undefined. If all the cells of a section are undefined, the section is not printed. If the height of all the cells of a section is zero, the section will be printed but will not be visible in the **Design** view.

11.3.2 Table Cells

A cell can contain any element provided by JasperReports, but since usually a cell displays only text, when an element is dropped on a cell, Jaspersoft Studio automatically arranges the element to fit the cell size. The elements in the cell can be arranged differently by right-clicking the cell (or an element in the cell) and selecting **Arrange Elements Vertically** or **Arrange Elements Horizontally**. If this approach does not fit the user requirements, the best solution is to insert a frame element in the cell and add all the required elements in that frame.

You can delete a cell by right-clicking and choosing **Delete cell**. If the cell is the only defined cell in the column, the entire column will be removed. Similarly, if a cell is undefined, right-click it and select **Add cell** to create the cell. An undefined cell is automatically created when the user drags an element into it (see **11.4, "Working with Columns," on page 136**).

Edit cell properties from the **Properties** tab:

- The **Appearance** sub-tab allows you to set location, size, color, style and print details.
- You can set cell padding as well as borders from the **Properties > Borders** tab.
- Cell height defines the vertical dimension of a cell. When its value is changed, the new dimension is propagated to all the cells in the row.

11.4 Working with Columns

To add a column to a table, right-click the column and choose **Create Column After**, **Create Column Before**, **Create Column at the Beginning**, or **Create Column At The End**.

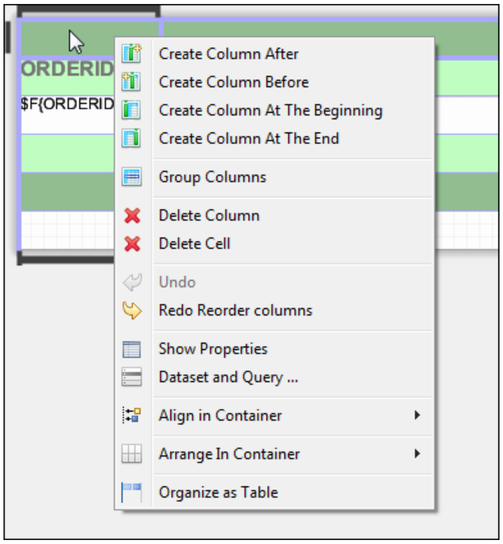


Figure 11-21 Column Right-Click Menu

By default, when Jaspersoft Studio adds a column to a table, the new column inherits the properties of the other columns in the table.

A column can be dragged to any position, inside or outside of a group. Columns can also be moved within the same section by dragging the nodes that represent the columns in the Report Inspector.

To remove a column, right-click the column and choose **Delete Column** from the menu. The remaining columns will automatically move to the left to fill in the empty space. Switch to the **Main Report Design** view to reposition the table in your report.

11.4.1 Column Groups

A column is composed of a set of cells, one for each section of the table. In order for a header cell to span several columns, we have to create a column group. The group must contain one or more columns and can contain other column groups. It also provides an additional header cell for each section of the table except for the detail section. These new header cells span all the columns of the group.

Figure 11-22 shows a simple column and a column group. On the left is a simple column. On the right is a column group with a header cell (the violet one) that spans the columns. This new group header cell does not replace the table header cell or the individual column header cells; those header cells remain.

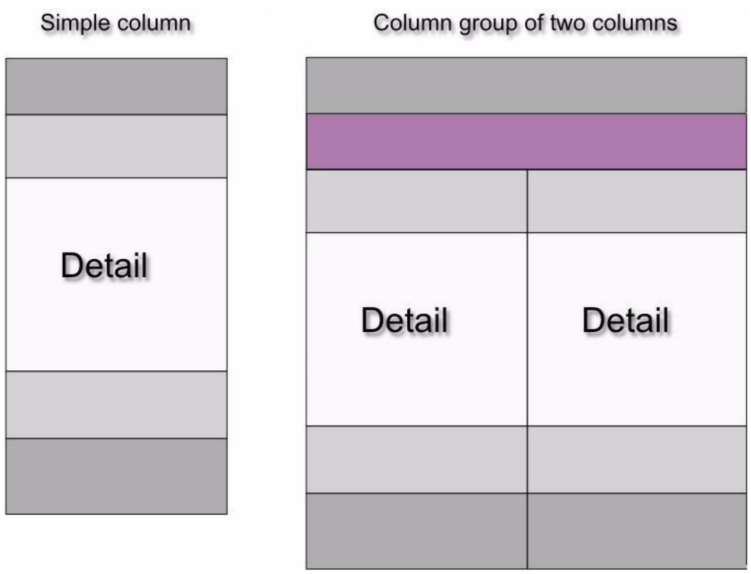


Figure 11-22 Simple Column vs. Column Group

If a column group is selected, the group will act as a single big column when it is dragged. If a column is the only column in a group and the column is dragged out of the group, the column becomes a simple column and the remaining group cells are deleted.

When you create a column group, every column section gets a group header, as shown in **Figure 11-23**, but you can remove unnecessary ones. On the left of the figure there are two columns (most of the sections in the columns have only one record; one section has two records). When the columns are grouped, each column section gets a group header, as shown in the center. However, most of the group headers are unnecessary, so their heights have been set to zero to hide them, as shown on the right.

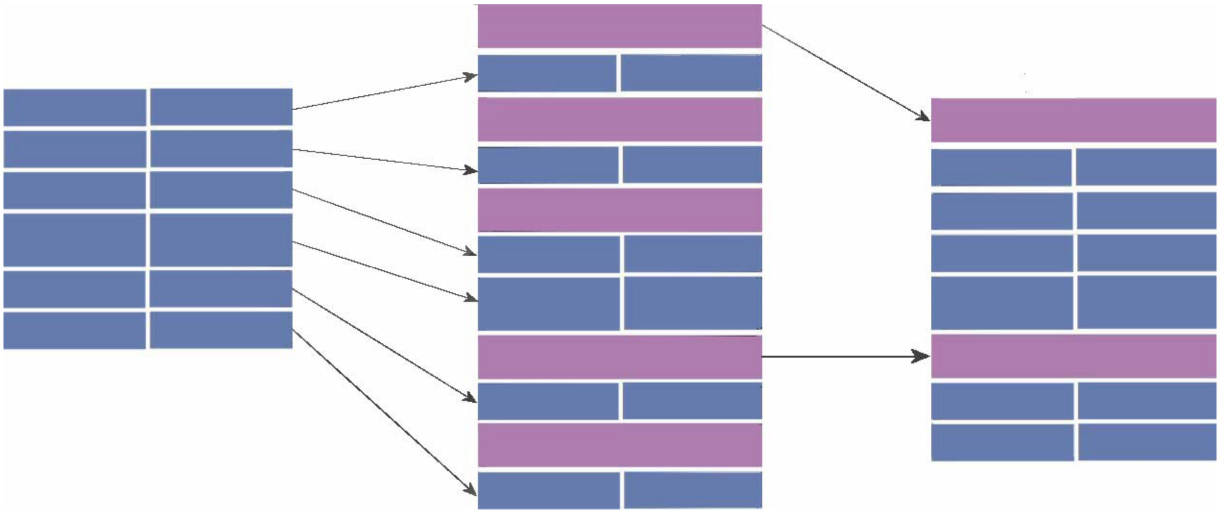


Figure 11-23 Group headers

CHAPTER 12 WORKING WITH CHARTS

Jaspersoft Studio provides the ability to render charts inside a report to different ways. In a chart, it is possible to print the data coming from the main dataset or using a subdataset. This allows you to include many different charts in one document without using subreports.

JasperReports supports a wide variety of chart types: Area, Bar, Bar 3D, Bubble, Line, Pie, Pie 3D, Scatter Plot, Stacked Bar, Stacked Bar 3D, Time Series, XY Area, Stacked Area, XY Bar, XY Line, Meter, Thermometer, Candlestick and High Low Open Close charts. A MultiAxis chart can be used to aggregate multiple charts into a single one.

This chapter has the following sections:

12.1 Datasets

The data represented by charts is collected when a report is generated and stored within the associated dataset. The dataset types are as follows:

- Pie
- Category
- Time period
- Time series
- XY
- XYZ
- High low
- Value

Think of a dataset as a table. Each dataset has different columns (fields). When a new record is inserted into the dataset, values are added to the fields.

Every report has a main dataset defined during its creation. But sometimes we need fields that are not returned by a query from the main dataset, or the fields needed may be in a different data source. The dataset element allows you to define many datasets inside a report, each with its own fields and data source. Every dataset is independent, so its fields are separated from the ones of the main dataset, and also from the ones of the other datasets

12.1.1 Creating a Dataset

1. Right-click on the report root node in the outline view and choose Create Dataset:

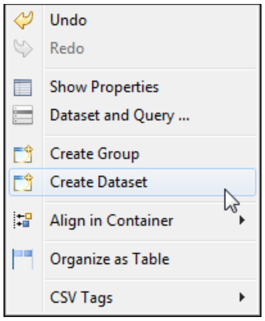
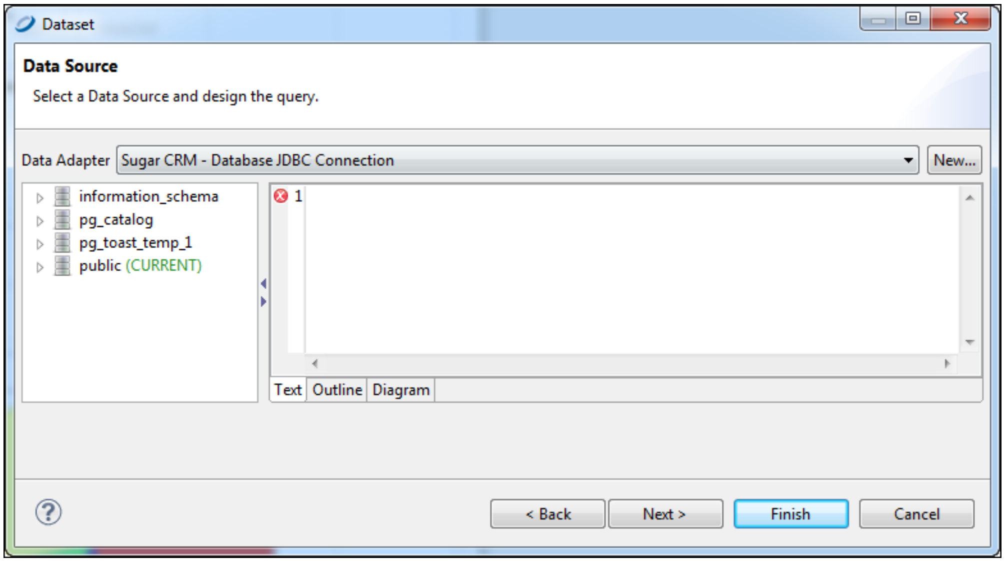


Figure 12-1 Create Dataset

The **Dataset** wizard opens.

2. Name your dataset, choose **Create new dataset from a connection** or Data Source and click **Next**.

   The **Dataset > Data Source** window opens.



Dataset

Data Source

Select a Data Source and design the query.

Data Adapter: Sugar CRM - Database JDBC Connection

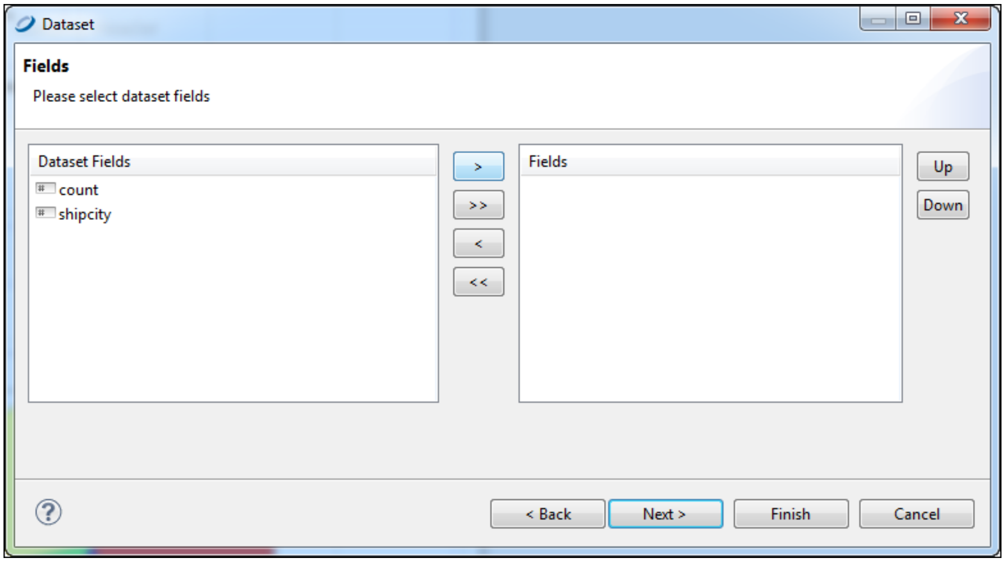
Figure 12-2 Dataset > Data Source Window

3. Select **Sugar CRM - Database JDBC Connection**.
4. Enter the following query:

```
select * from orders
```

5. Click **Next**.

   The **Dataset > Fields** window opens.



Dataset

Fields

Please select dataset fields

Figure 12-3 Dataset > Fields

6. Move the fields listed in the **Dataset Fields** box into the **Fields** box. Click **Next**.

   The **Dataset > Group By** window opens. In this window you can choose fields by which to group, as well as choose whether or not to use the group fields as sort fields.

7. Do not select any fields to group by. Click **Finish**.

   You will see the dataset you created in the **Outline** view.

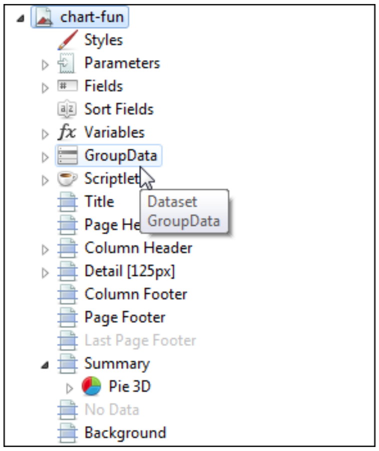


Figure 12-4 GroupData Data Set

12.2 Creating a Simple Chart

In this section, you will learn how to use the Chart tool to build a report containing a Pie 3D chart, then you will explore chart configuration.

To add a chart to a report:

1. Create a new report. Use the Sugar CRM data source
2. Use this query to display the count of orders in different countries:

   ```
   select COUNT(*) as orders, shipcountry from orders group by shipcountry
   ```
3. Place the fields in the **Detail** band by dragging them from the **Outline**. This will create a small table of the values to display in the chart.

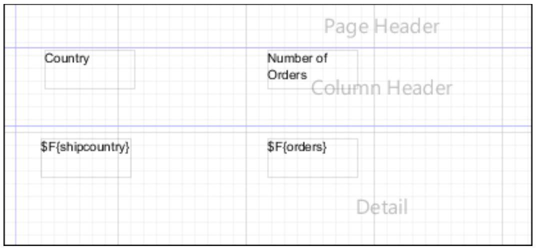
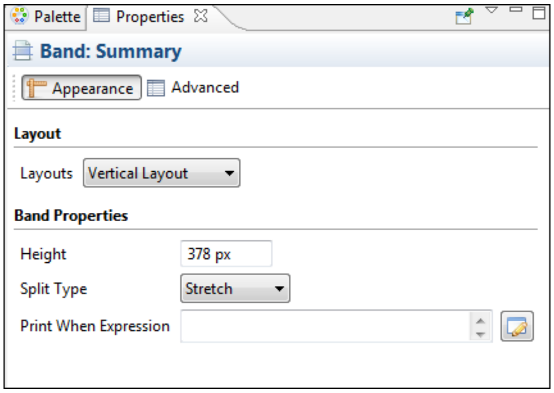


Figure 12-5 Initial Report Design

4. Expand the Summary band to 378 pixels.



Band: Summary

Layout

Layouts: Vertical Layout

Band Properties

Height: 378 px

Split Type: Stretch

Print When Expression:

Figure 12-6 Summary Band Properties

5. From the **Palette**, select the **Chart** tool and drag it into the **Summary** band. When you add a new chart element to a report, Jaspersoft Studio shows the **Chart Wizard** from which you can pick the chart type.

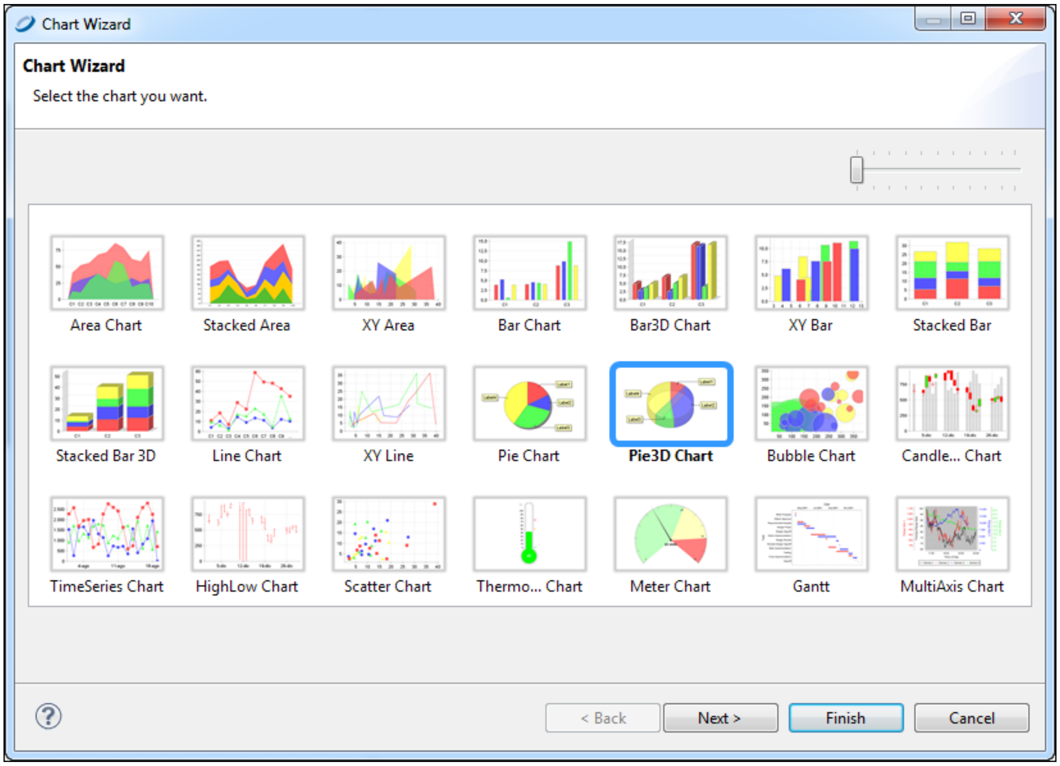


Chart Wizard

Chart Wizard

Select the chart you want.

Figure 12-7 Chart Wizard

6. Select the **Pie 3D Chart** icon and click **Next**.
7. Accept the default configuration and click **Finish**.
8. Expand the chart to fill the **Summary** band.

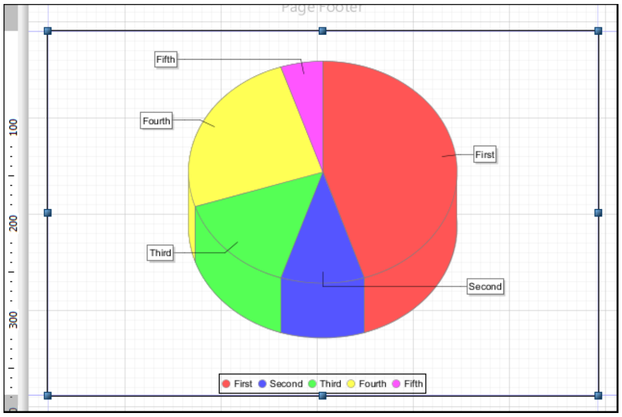


Figure 12-8 Chart in Summary Band

In Design view, the chart shown will not display information from your data.

To configure a chart:

1. Double click on the chart element. The **Chart Wizard > Chart Data Configuration** window opens.

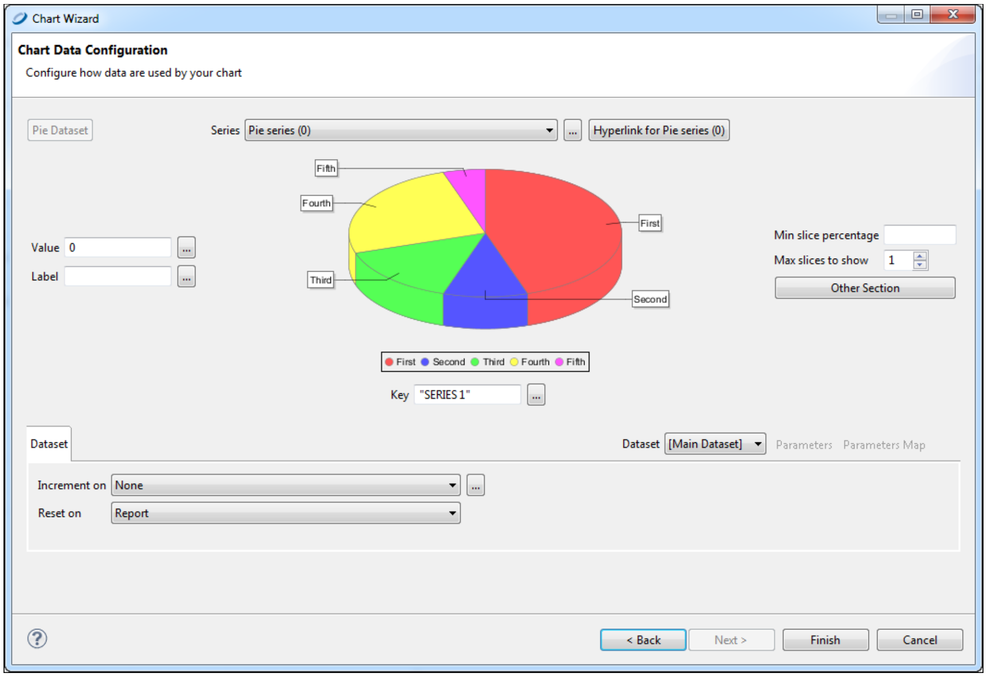


Chart Wizard

Chart Data Configuration

Configure how data are used by your chart

Series: Pie series (0)

Value: 0

Label:

Min slice percentage:

Max slices to show: 1

Key: "SERIES 1"

Dataset: [Main Dataset]

Increment on: None

Reset on: Report

Figure 12-9 Chart Wizard > Chart Data Configuration

In this window you select data to use in your chart.

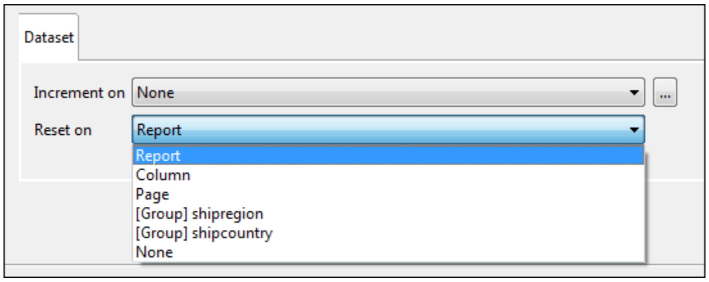
Depending on the dataset type that you have selected, the window's **Chart Data** tab shows the fields within the specified dataset. Detailed descriptions of the various field types and their functionality are available in the *JasperReports Ultimate Guide*.

2. The **Max slices to show** field is useful for charts with many slices. Your chart will only show the number of slices you specify, and any additional information will appear in a slice labeled **Other**.

   Set this field to `10`.

3. On the **Dataset** tab, you can define the dataset within the context of the report.

   The **Reset on** drop-downs allows you to periodically reset the dataset. This is useful, for example, when summarizing data relative to a special grouping. **Increment on** specifies the events that determine when new values must be added to the dataset. By default, each record of the dataset used to fill the chart corresponds to a value printed in the chart. This behavior can be changed, forcing the engine to collect the data for the chart at a specific time (for instance, every time the end of a group is reached).



Increment on: None

Reset on: Report

Figure 12-10 Dataset Tab

Set **Reset on** to **Report** since you don't want the data to be reset and leave **Increment Type** set to **None** so that each record will be appended to your dataset.

4. Also in the **Chart Data Configuration** dialog, enter an expression to associate with each value in the data source. For a Pie 3D chart, three expressions can be entered: `key`, `value`, and `label`.
   - **Key expression** must be a unique value to identify a slice of the pie chart. If a key value is repeated, the label and value values previously associated with that key are overwritten. A key can never be `null`.
   - **Value expression** specifies the numeric value associated with the key.
   - **Label expression** allows you to specify a label for each slice of the pie chart. This expression is optional, and the default value is the key value.

   Next to each field, click the ... button. Enter the following:

   Value: `$F{orders}`

   Label: `$F{shipcountry}`

   Key: `$F{shipcountry}`
5. Click **Finish**.
6. Save your report, and preview it to see the result:

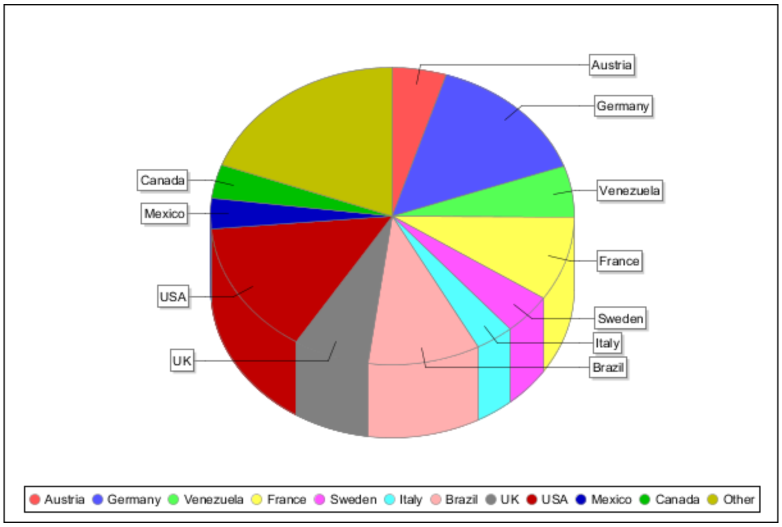


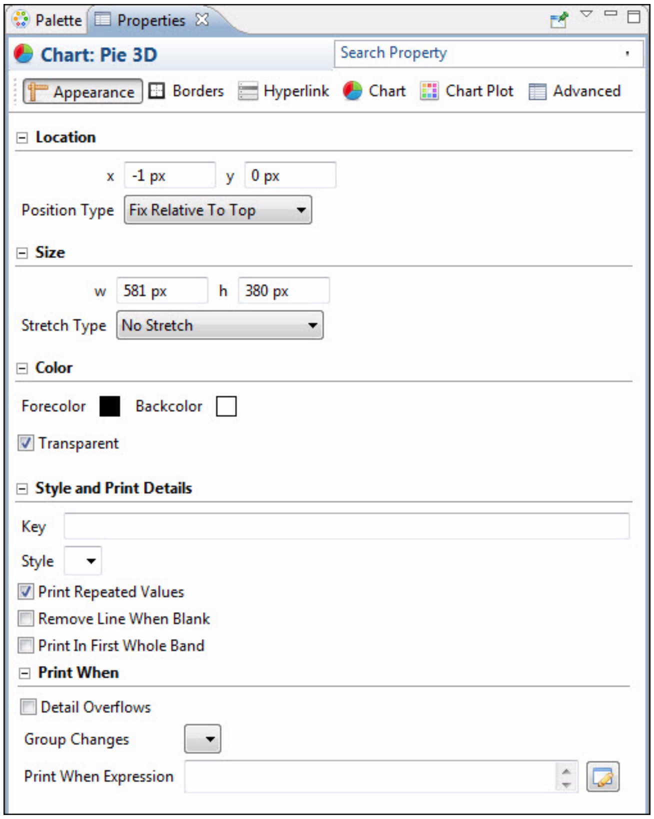
Figure 12-11 Final Chart

In this chart, each slice represents a country, and the value of the slice represents the shipping total for that country.

12.3 Setting Chart Properties

The appearance of a chart can be configured using the **Properties** view when selecting the chart in the **Design** view. You can edit properties common to all charts and graphs (such as the title and the visibility of the legend) as well as properties specific to the chart or graph that is being created.

The Properties view includes the following tabs to easily set properties: **Appearance**, **Borders**, **Hyperlink**, **Chart**, and **Chart Plot**. There is also an **Advanced** tab where you can manually set all the different types of properties.



⊟ Location

x -1 px y 0 px

Position Type Fix Relative To Top

⊟ Size

w 581 px h 380 px

Stretch Type No Stretch

⊟ Color

Forecolor ■ Backcolor □

☑ Transparent

⊟ Style and Print Details

Key

Style

☑ Print Repeated Values

□ Remove Line When Blank

□ Print In First Whole Band

⊟ Print When

□ Detail Overflows

Group Changes

Print When Expression

Figure 12-12 Properties View

For more information about setting hyperlinks, see **4.8, “Anchors, Bookmarks, and Hyperlinks,” on page 53**.

Currently, JasperReports takes advantage of only a small portion of the capabilities of the `JFreeChart` library. To customize a graph, a class must be written that implements the following interface:

```
net.sf.jasperreports.engine.JRChartCustomizer
```

The only method available from this interface is the following:

```
public void customize(JFreeChart chart, JRChart jasperChart);
```

It takes a `JFreeChart` object and a `JRChart` object as its arguments. The first object is used to actually produce the image, while the second contains all the features you specify during the design phase that are relevant to the customize method.

12.4 Spider Charts

Spider charts, also called radar charts, are a two-dimensional chart type designed to plot one or more series of values over multiple common quantitative variables by providing an axis for each variable, arranged as spokes around a central point. The values for adjacent variables in a single series are connected by lines. Frequently the shape created by these lines is filled in with color.

In Jaspersoft Studio, spider charts are in a separate element from the rest of the charts available in the Community Project. This is because they are a separate component in JasperReports Server. However, you use them the same way you use other JFree Charts.

To filter your data in a Spider Chart, you must filter it in the **Dataset and Query** dialog.

To create the report for the chart:

1. Open a new, blank report using a landscape template and the Sugar CRM database. Use the following query:

   ```
   select * from orders
   ```

   Click **Next**.
2. Move all the fields into the right-hand box. Click **Next**.
3. No need to group by any field. Click **Finish**.
4. Right-click on each band in the outline view, and choose **Delete** to delete all bands except for **Title** and **Summary**.
5. Drag a **Static Text** element into the title bar, and name your report something like “Employee Orders by Month and Country”.
6. Enlarge the Summary band to 350 pixels by changing the **Height** entry in the **Band Properties** view.

To create a spider chart:

1. Drag the **Spider Chart** element from the **Palette** into your report.

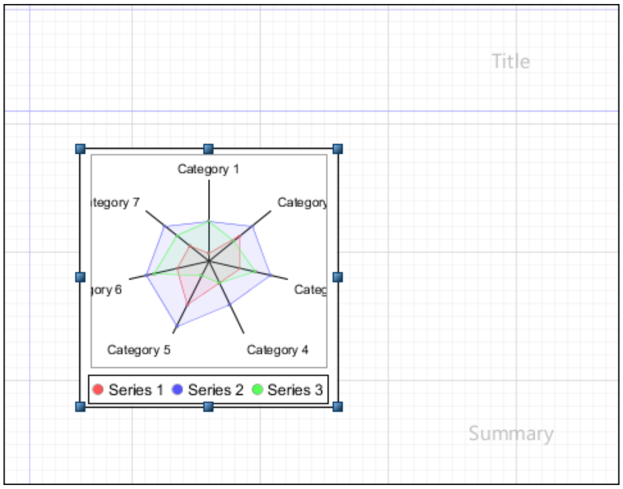
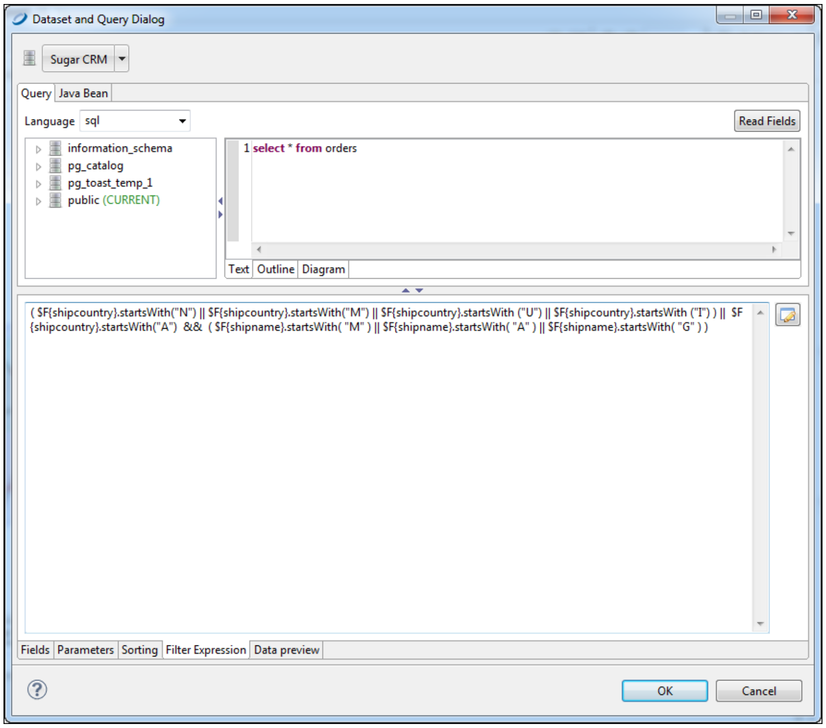


Figure 12-13 Spider Chart

2. Select the report in the Outline view, and in the Properties tab, click the **Edit query, filter, and sort options** button. The **Dataset and Query** dialog opens.



```
1 select * from orders
```

```
( $F{shipcountry}.startsWith("N") || $F{shipcountry}.startsWith("M") || $F{shipcountry}.startsWith ("U") || $F{shipcountry}.startsWith ("I") ) ||  $F
{shipcountry}.startsWith("A")  &&  ( $F{shipname}.startsWith( "M" ) || $F{shipname}.startsWith( "A" ) || $F{shipname}.startsWith( "G" ) )
```

Figure 12-14 Filter Expression for Spider Chart

3. To filter data so that the chart is more readable, click the **Filter Expression** tab and add the following expression:

   ```
   ( $F{shipcountry}.startsWith("N") || $F{shipcountry}.startsWith("M") ||
   $F{shipcountry}.startsWith ("U") || $F{shipcountry}.startsWith ("I") ) ||
   $F{shipcountry}.startsWith("A")  &&  ( $F{shipname}.startsWith( "M" ) ||
   $F{shipname}.startsWith( "A" ) || $F{shipname}.startsWith( "G" ) )
   ```

   Do not include any manual line breaks.

   Click **OK**.

4. Double-click the chart. The **Chart Data Configuration** dialog opens.
5. Click the **Series** button and create the series `$F{employeeid}`.
6. Click the **Value** button and create the value `MONTH($F{orderdate})`.
7. Click the **Category** button and create the category `$F{shipcountry}`.
8. Click **Finish**.

To customize the look of your chart:

1. Single-click your chart, and click the **Properties** view.
2. In the **Legend** category, set the drop-down menu for **Show Legend** to True.
3. Use the **Position** drop-down, to move the legend to the left.
4. Drag a Static Text element just to the left and above the legend. Label the legend **Employee Number**.
5. Save your report. The Design view should look like the following:

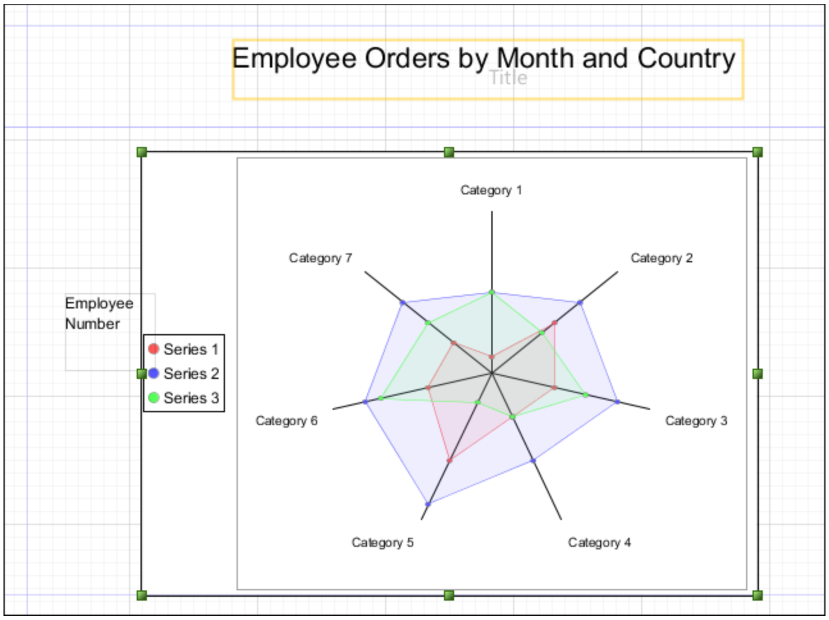


Figure 12-15 Spider Chart Design

6. Preview your report. It should look like the following:

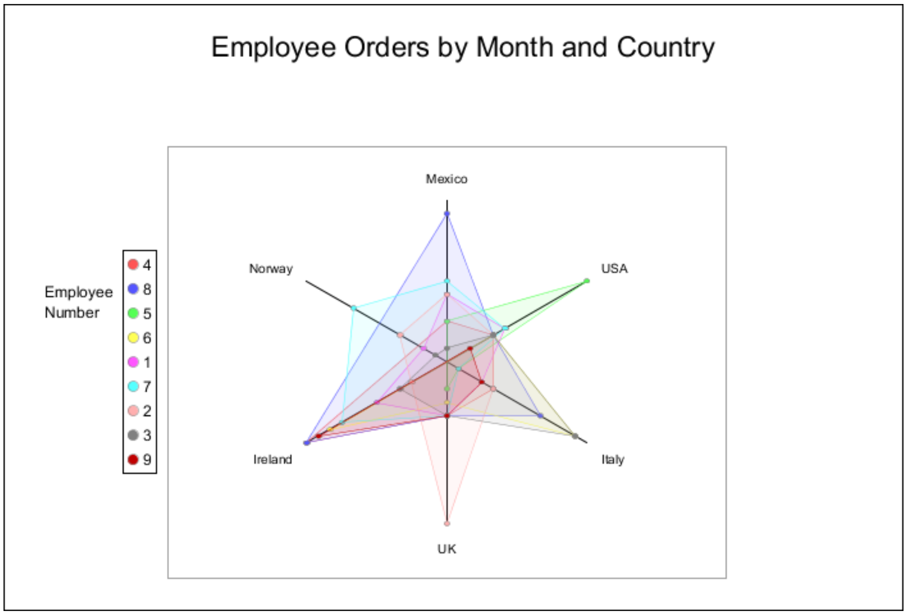


Figure 12-16 Spider Chart Preview

12.5 Chart Themes

Another way to customize graphs is by creating a chart theme, which gives you full control over the style of the chart. Chart themes allow you to customize the design of a chart. Chart themes are reusable: you can use the same chart theme in many reports to have a unified appearance. You can also update a chart theme to modify the appearance across all reports. As of the current release, Jaspersoft Studio is the supported tool for creating chart themes.

To create a JasperReports chart theme:

1. Select **File > New > Other.**

   The **Select a wizard** window opens.

2. Select the **Chart Themes** wizard. Click **Next**.

   The **New Chart Theme** wizard opens.

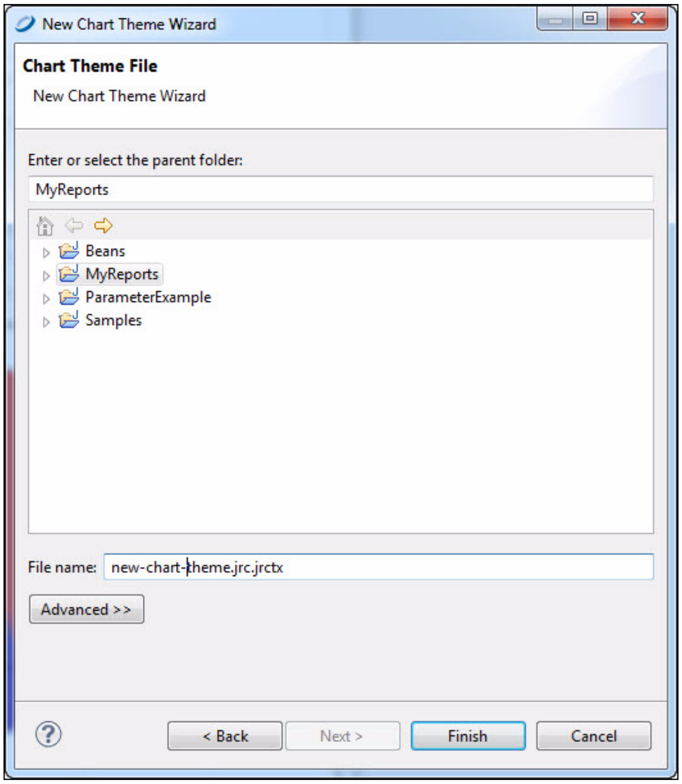


Figure 12-17

3. Select the file in which you want to store your theme, and name it. The file needs to have the extension .jrctx.

   Click **Finish**. The **Chart Theme Designer** opens.

12.5.1 Using the Chart Theme Designer

JasperReports organizes a chart theme into seven sub-sections, visible in the **Outline** view.

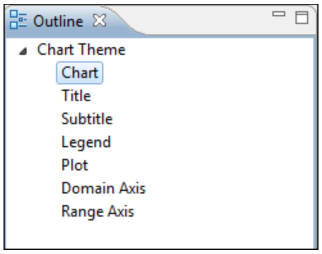


Figure 12-18 Chart Theme Designer Outline View

- **Chart**: Set properties for borders, color, background, and padding.
- **Title**: Set properties for position, color, alignment, padding, and font.
- **Subtitle**: Set properties for position, color, alignment, padding, and font. These can be set to `INHERITED` on the **Advanced** tab.
- **Legend**: Set whether or not to show a legend, and if so, set position, alignment, colors, frame, and font.
- **Plot**: Set properties for label rotation, foreground, orientation, colors, image alignment, padding, grid lines, chart outline, series stroke and colors, and display and tick label fonts.
- **Domain Axis**: Set properties, color, and visibility for line, tick count, tick mark, tick label, tick padding, and tick label font. Set properties, color, and visibility for label, label padding, and label font.
- **Range Axis**: Set properties, color, and visibility for line, tick count, tick mark, tick label, tick padding, and tick label font. Set properties, color, and visibility for label, label padding, and label font.

Any properties you set are applied to all chart types. If you want to see one of the chart types in close up, click on that chart. Click on the close up view of the chart to return to viewing all charts.

12.5.2 Editing Chart Theme XML

You can see and edit the chart theme XML from the **Source** tab. However, the XML appears as one long line. It is better to copy-and-paste it into the text editor of your choice and edit it there.

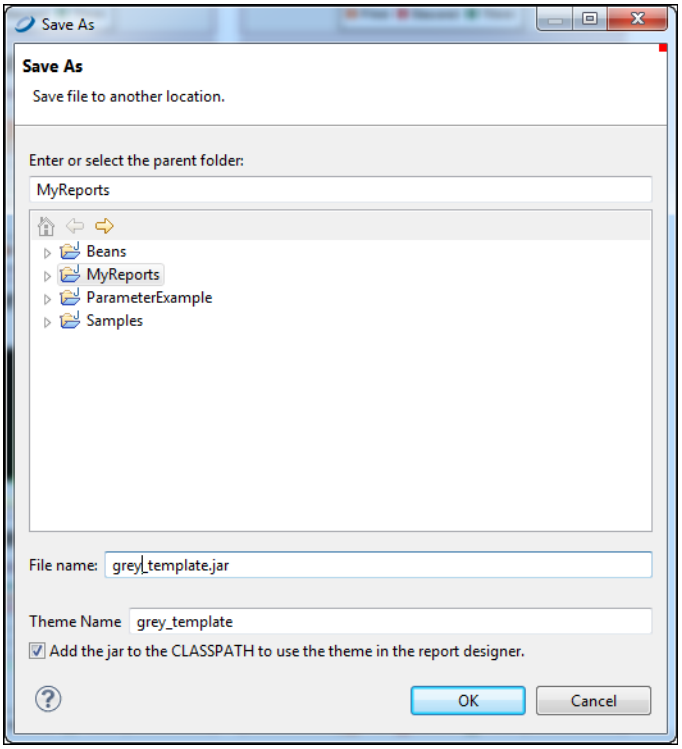
12.5.3 Creating a JasperReports Extension for a Chart Theme

The JRCTX file cannot be used in a report until it is wrapped into a JasperReports extension JAR file.

To export the theme as a JAR:

1. From the **Preview** tab, click the button.

   The Save As window opens.



Save As

Save As

Save file to another location.

Enter or select the parent folder:

MyReports

- Beans
- MyReports
- ParameterExample
- Samples

File name: grey_template.jar

Theme Name grey_template

☑ Add the jar to the CLASSPATH to use the theme in the report designer.

Figure 12-19 Save As JAR

2. Enter or select the parent folder, name the file, and name your theme. Click **OK**.

   A pop-up will tell you that the Chart Theme was generated.

12.5.4 Applying a Chart Theme

After you create a chart theme, you want to be able to see it in the list of themes. For that to happen, add the theme to your classpath.

Your theme will now be available in the list of themes.

CHAPTER 13 ADDITIONAL CHART OPTIONS IN COMMERCIAL EDITIONS

In commercial editions of Jaspersoft Studio, you have additional charting options.

HTML5 Charts are written in pure HTML5 and JavaScript,. They offer sophisticated, interactive charts that you can add to your reports. Jaspersoft Studio currently supports most of the charts available in the Highcharts library

This chapter has the following sections:

- **Overview of HTML5 Charts**
- **Simple HTML5 Charts**
- **Scatter Charts**
- **Dual-Axis, Multi-Axis, and Combination Charts**

13.1 Overview of HTML5 Charts

HTML5 charts can be used to create interactive reports. They are also more attractive than the basic charts available in Jaspersoft Studio. In this section, you will learn how to build a report containing a simple HTML5 chart and how to change chart types and edit charts.

HTML5 Charts in Jaspersoft Studio are designed to work similarly to Ad Hoc charts. Levels, for example, are the equivalent of what you would use the Ad Hoc Slider to see. Other explanations include:

- **Values**: Static properties.
- **Expressions**: Dynamic properties.
- **Categories**: Rows. In a pie chart, the categories are the slices.
- **Measures**: The same as in Ad Hoc. In a pie chart, these are the size of the slice.
- **Series Contributors**: In the Design view, these are defined at the measure level. In JRXML these are defined as Series.

Before you add a chart to your report, consider the best way to display your data. The following table describes the available chart types:

Table 13-1 HTML5 Chart Types

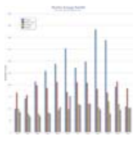
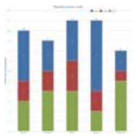
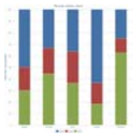
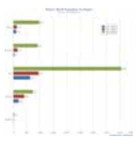
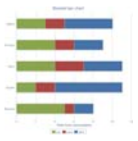
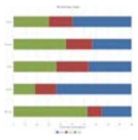
Icon	Description
Column charts - Compare values displayed as columns	
	**Column**. Multiple measures of a group are depicted as individual columns.
	**Stacked Column**. Multiple measures of a group are depicted as portions of a single column whose size reflects the aggregate value of the group.
	**Percent Column**. Multiple measures of a group are depicted as portions of a single column of fixed size representing 100% of the amounts for a category. Used when you have three or more data series and want to compare distributions within categories and at the same time display the differences between categories.
Bar charts - Compare values displayed as bars	
	**Bar**. Graphically summarize and display categories of data to let users easily compare amounts or values between different categories.
	**Stacked Bar**. Multiple measures of a group are depicted as portions of a single bar whose size reflects the aggregate value of the group.
	**Percent Bar**. Multiple measures of a group are depicted as portions of a single bar of fixed size representing 100% of the amounts for a category. Used when you have three or more data series and want to compare distributions within categories and at the same time display the differences between categories.
Line charts - Compare values displayed as points connected by lines	

Table 13-1 HTML5 Chart Types

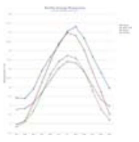
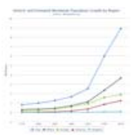
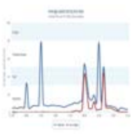
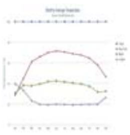
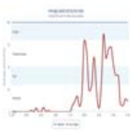
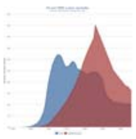
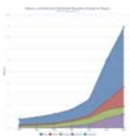
Icon	Description
	**Line**. Displays data points connected with straight lines, typically to show trends.
	Spline. Displays data points connected with a fitted curve. Allow you to take a limited set of known data points and approximate intervening values.
	**Stacked Line**. Displays series as a set of points connected by a line. Values are represented on the y-axis and categories are displayed on the x-axis. Lines do not overlap because they are cumulative at each point.
	**Stacked Spline**. Displays series as a set of points connected with a fitted curve. Values are represented on the y-axis and categories are displayed on the x-axis. Lines do not overlap because they are cumulative at each point
	**Percent Line**. A variation of al line chart in which each series adjoins but does not overlap the preceding series.
	**Percent Spline**. A variation of a spline chart in which each series adjoins but does not overlap the preceding series.
Area charts - Compare values displayed as shaded areas. Compared to line charts, area charts emphasize quantities rather than trends.	
	**Area**. Displays data points connected with a straight line and a color below the line; groups are displayed as transparent overlays.
	**Stacked Area**. Displays data points connected with a straight line and a solid color below the line; groups are displayed as solid areas arranged vertically, one on top of another.
	Percent Area. Displays data points connected with a straight line and a solid color below the line; groups are displayed as portions of an area of fixed sized, and arranged vertically one on top of the another.

Table 13-1 HTML5 Chart Types

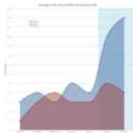
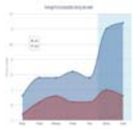
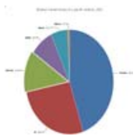
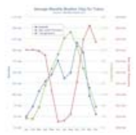
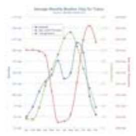
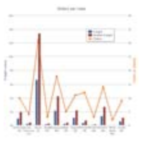
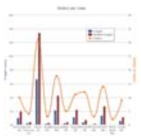
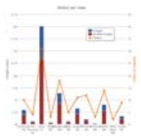
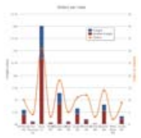
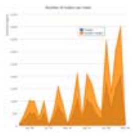
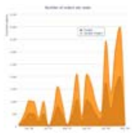
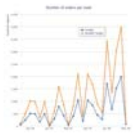
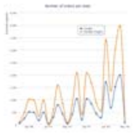
Icon	Description
	**Area Spline**. Displays data points connected with a fitted curve and a color below the line; groups are displayed as transparent overlays.
	**Stacked Area Spline**. Displays series as a set of points connected by a smooth line with the area below the line filled in. Values are represented on the y-axis and categories are displayed on the x-axis. Areas do not overlap because they are cumulative at each point.
	Percent Area Spline. A variation of area spline charts that present values as trends for percentages, totalling 100% for each category.
Pie charts - Compare values displayed as slices of a circular graph	
	**Pie**. Multiple measures of a group are displayed as sectors of a circle.
Scatter Charts - Show the extent of correlation, if any, between the values of observed quantities.	
	Scatter Chart. Displays a single point for each point in a data series without connecting the points.
Multi-Axis Charts - Compare trends in two or more data sets whose numeric range differ greatly.	
	Multi-Axis Column. A column chart with two series and two axis ranges.
	**Multi-Axis Line**. A line chart with two series and two axis ranges.
	**Multi-Axis Spline**. A spline chart with two series and two axis ranges.

Table 13-1 HTML5 Chart Types

Icon	Description
Combination Charts - Display multiple data series in a single chart, combining the features of a area, bar, column, or line charts.	
	**Column Line**. Combines the features of a column chart with a line chart.
	**Column Spline**. Combines the features of a column chart with a spline chart.
	**Stacked Column Line**. Combines the features of a stacked column chart with a line chart.
	**Stacked Column Spline**. Combines the features of a stacked column chart with a line chart.
Time Series Charts - Illustrate data points at successive time intervals. Also called Fever Chart.	
	**Time Series Area.** Displays data points over time connected with a straight line and a color below the line.
	**Time Series Area Spline**. Displays data points over time connected with a fitted curve and a color below the line.
	**Time Series Line**. Displays data points over time connected with straight lines.
	**Time Series Spline**. Displays data points over time connected with a fitted curve.

13.2 Simple HTML5 Charts

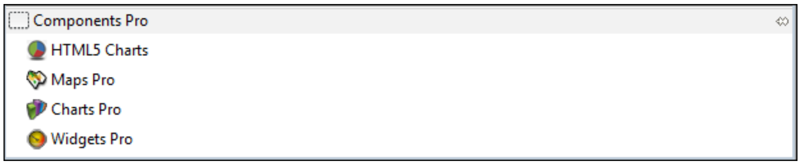
Before you add a chart, consider the best way to display your data. Table 13-1 can help you with the determination.

13.2.1 Creating an HTML5 chart

1. Create a new report using the Sugar CRM data source. Start with the query:

   ```
   select * from orders
   ```

2. Choose **HTML5 Charts** from the **Components Pro** section of the **Palette**, and drag it into your report.

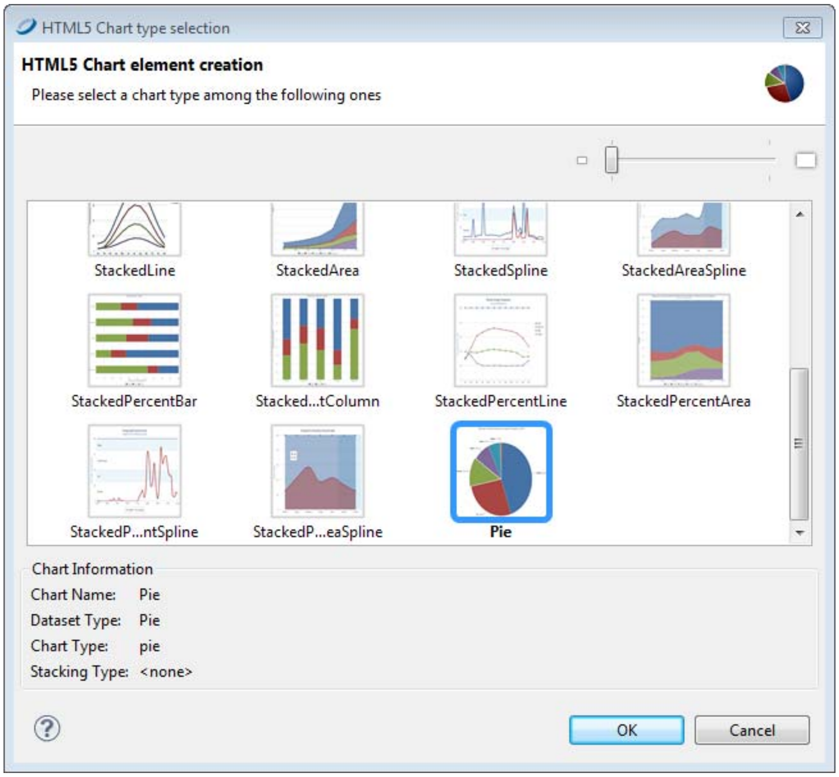


Components Pro
- HTML5 Charts
- Maps Pro
- Charts Pro
- Widgets Pro

Figure 13-1 Palette

3. Select the type of chart you wish to add. Table 13-1 can help you choose the most appropriate way to display your information.

   For this example, choose **Pie chart**.



HTML5 Chart type selection

HTML5 Chart element creation

Please select a chart type among the following ones

Chart Information

Chart Name: Pie
Dataset Type: Pie
Chart Type: pie
Stacking Type: <none>

Figure 13-2 Chart Types

Your report will now include a sample chart. Jaspersoft Studio does not display live data and charts in the **Design** view.

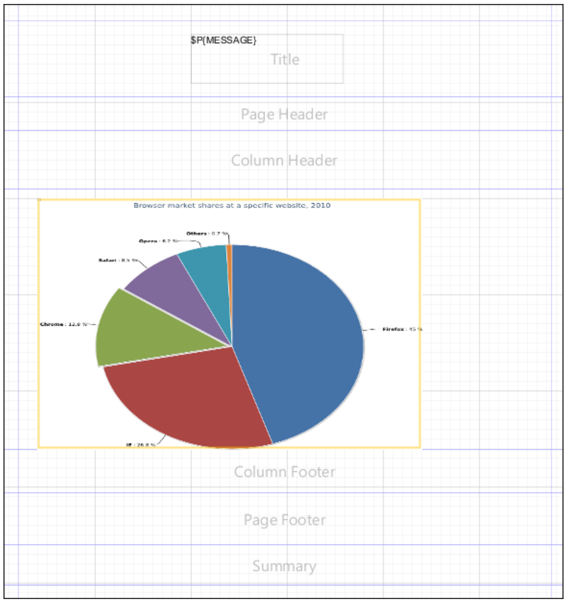


Figure 13-3 Pie Chart Example

4. Right-click the chart, and choose **Edit Chart Properties**.

   The **HTML5 Charts Properties** dialog appears with tabs for configuring chart properties, chart data, and optional hyperlinks.

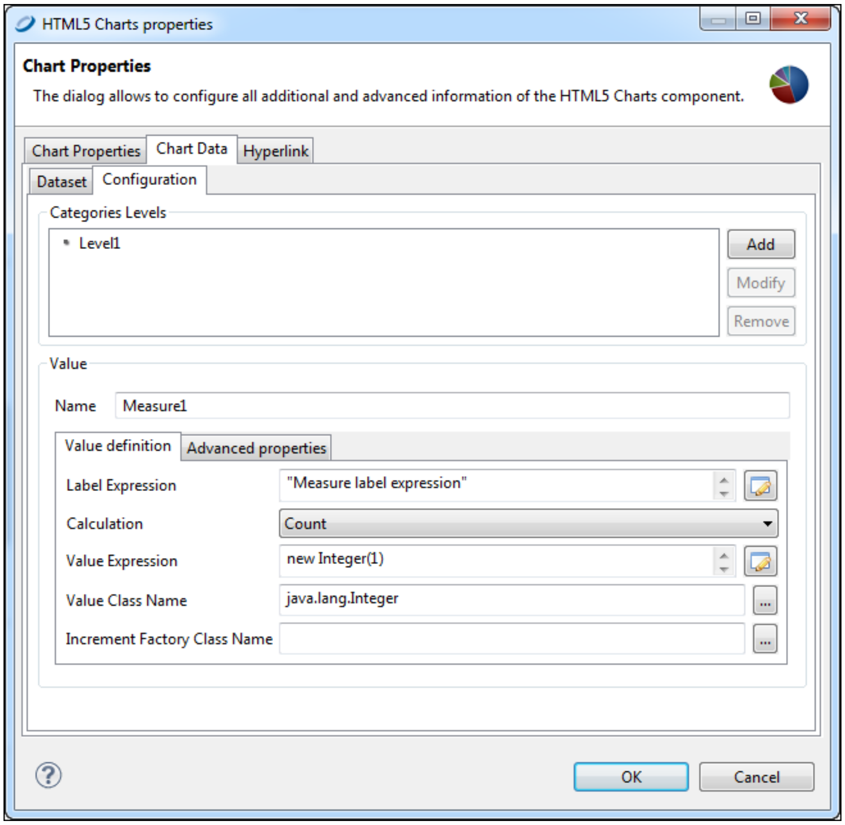


Chart Properties

The dialog allows to configure all additional and advanced information of the HTML5 Charts component.

Categories Levels

- Level1

Value

Name: Measure1

Label Expression: "Measure label expression"

Calculation: Count

Value Expression: new Integer(1)

Value Class Name: java.lang.Integer

Increment Factory Class Name:

Figure 13-4 HTML5 Charts Properties > Chart Data > Configuration

5. Choose the **Chart Data** tab and the **Configuration** sub-tab.

   Highlight **Level1**, and click **Modify**.

6. Name the Expression `Country`. Open the Expression editor and change the default to the `shipcountry` field.

   Click **OK**.

7. In the **Values** field, create a measure called **Total Orders**.
   - The **Label Expression** should be `"Total Orders"`.
   - The **Calculation** type is `DistinctCount`.
   - In the **Value Expression** field, add the `orderid` field.
   - The **Value Class Name** should be `java.lang.Integer`.

8. Click **OK** to close the **Chart Properties** dialog.

9. Save and click the **Preview** tab to see your chart. To preview your chart in a web browser, choose XHTML Preview from the Preview drop-down.

   HTML5 charts require XHTML for web preview. The HTML preview option will be deprecated.

   In an HTML preview, the chart is interactive. Mousing over a slice of the pie shows the exact number of orders.

13.2.2 Editing HTML5 Charts

1. To add a title to the chart, in the **Design** tab, double-click the chart.

2. Click **Title** and in the **Title** box type Orders by Country.

   Here you can also customize alignment, color, and font if you want.

3. To see fewer countries in your pie chart, select **Edit Chart Properties > Chart Data > Dataset**.

4. You can filter your data set. To do so, add a Filter expression such as:

   ```
   $F{shipcountry}.startsWith("A") ||
   $F{shipcountry}.startsWith ("U") ||
   $F{shipcountry}.startsWith ("I") ||
   $F{shipcountry}.startsWith ("S")
   ```

   Close the dialog.

5. Save and click **Preview** again to view the edited chart.

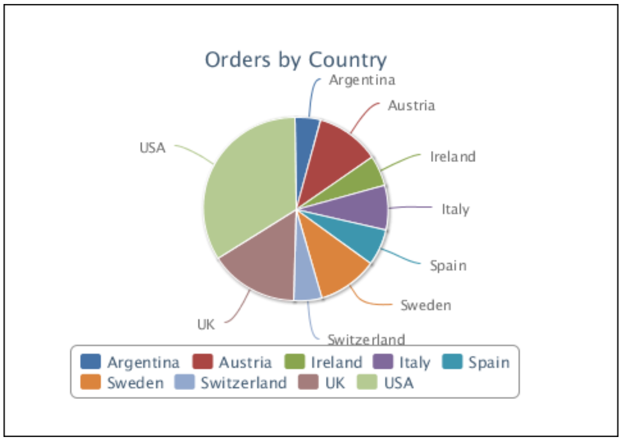


Figure 13-5 Filtered Pie Chart

To change chart type:

Before you change the chart used, ensure you have appropriate data for the type of chart.

1. From the **Design** tab, right-click your chart and choose **Edit Chart Properties**.
2. Click the **Change Chart Type** button.

   The **HTML5 Chart element creation** window opens. This window will give you very slight guidance as to what dataset type is used by each chart.
3. Choose an appropriate type of chart. For this example, choose Bar, and click **OK**.

   Jaspersoft Studio will discard data if not right for the new chart type.

   Click **Close**.
4. Save and preview your report. Because it was originally designed as a pie chart, there is no label for the Y-values.

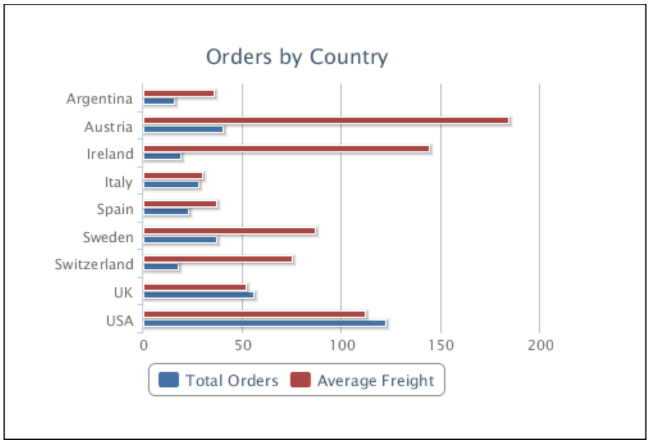


Figure 13-6 Chart Type Changed to Bar Chart

To edit a bar chart:

Editing all charts works similarly, depending on the type of data presented.

1. From the **Design** tab, right-click and choose **Edit Chart Properties**.
2. On the **Chart Data > Configuration** tab, there are now places to define Series and more than a single measure.
3. Add a second measure, Average Freight:
   - **Label Expression**: `"Average Freight"`
   - **Calculation**: `Average`
   - **Value Expression**: `$F{freight}`
   - **Value Class Name**: `java.lang.Integer`

   Click **OK** and Close.
4. Save and Preview the chart.

   In HTML the chart is interactive.
5. Create a series:
   - **Name**: `Year`
   - **Expression**: `new java.text.SimpleDateFormat("yyyy").format( $F{orderdate} )`
   - **Value Class Name**: `java.lang.Comparable`
   - **Order**: `Ascending`

   Click **OK** and Close.

6. Save and Preview your report. You see two measures per year, Total Orders and Average Freight, grouped by Country.

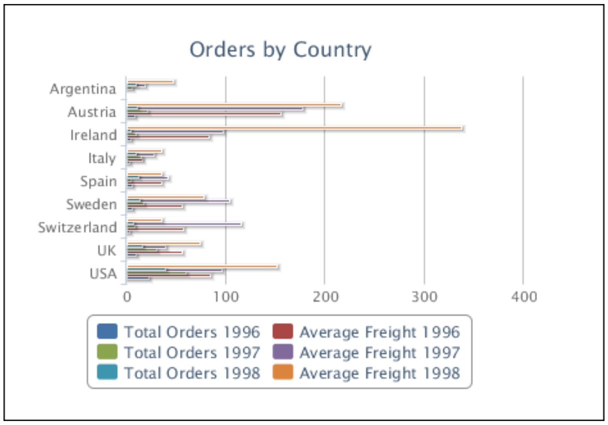


Figure 13-7 Bar Chart with Second Measure and Series

13.2.3 Creating Hyperlinks

With HTML5 charts (Highcharts) functionality added to JasperReports, using hyperlinks in charts is similar to using JFreeCharts (regular Charts) and Fusion charts (Charts Pro). You open the chart properties, select the **Hyperlink** tab, and add your link.

There is no way to add hyperlinks to HTML5 chart items (like bar sections or pie slices), at least not the way with other chart libraries.

1. Go to **Chart Properties > Chart Data > Data Axes**.
2. Double click Country.
3. Click the **Properties** tab. The **Add/modify property** dialog appears.
4. Create a property named `url`. Add the expression:

   `“http://www.ask.com/web?q=” + F{shipcountry}`

   Click OK.
5. Double-click the measure Total Orders.
6. Click the **Advanced Properties** tab.
7. Click **Add**. The **Add/modify property** dialog appears.
8. Click the down arrow and choose **Built-In Hyperlink Property**. Close that window.
9. Click the **Hyperlink** button. That provides a shortcut to the two most important properties needed to create a hyperlink.
10. Double-click **hyperlinkReference (SeriesItemHyperlink)**. The **Add/modify property** dialog opens with some information filled in. You customize this information to the values you need.
11. Click the radio button for **Use a bucket property value** and type `Country.url`. Click OK.
12. Save and preview your report. In the HTML preview, if you click the bar for any country, you will be taken to the Ask.com page for that country.

To create hyperlinks that use expressions:

Bucket-level properties are the only place where you can create expressions to be used with your link. If you are not using an expression, you must either use static values or reference a bucket level property.

1. Identify the series you want to use for your link.
2. Create a bucket property for that series. i.e. If you are using a pie chart, the category bucket may have a property called myUrl which contains your url built with the category value.

   This can be done in **Edit Chart Properties > Chart Data > Data Axes > Categories.**
3. Identify the measure to which you need to add a link.

   This can be done in **Edit Chart Properties > Chart Data > Data Axes > Measures.**
4. In the advanced properties of your measure click **Hyperlink**.

   This will create for you a couple of properties. Edit them by assigning the proper value (again, here you'll have to either use static values or reference a bucket level property).

To add a ReportExecution hyperlink:

If you want a ReportExecution hyperlink, you need to rename one of the measure's advanced properties (see step 4 in the previous procedure) `to _report` and enter the corresponding value, and if you need to pass parameter values you will also need to add them as measure properties.

For more information about setting hyperlinks, see **4.8, “Anchors, Bookmarks, and Hyperlinks,” on page 53**.

13.3 Scatter Charts

Scatter charts plot two or more data series as points. The charts are intended to show raw data distribution of the series. The data is represented as follows:

1. The first data series is represented as the x values.
2. The second data series is represented as correlated y values.
3. Any additional data series are plotted as y values correlated to the x values provided by the first series.

As a result, a scatter chart always displays one less plot than the number of data series included.

In the following example, we will create a scatter chart that shows maximum and average freight in each city and each year.

To create the report for the chart:

1. Open a new, blank report using the Sugar CRM database and the following query:

   ```
   select * from orders
   ```

   Click **Next**.
2. Move all the fields into the right-hand box. Click **Next**.
3. No need to group by any field. Click **Finish**.
4. Right-click on each band in the outline view, and choose **Delete** to delete all bands except for **Title** and **Summary**.
5. Enlarge the Summary band to 500 pixels by changing the **Height** entry in the **Band Properties** view.

To create the chart:

1. Give your report a title like “Maximum and Average Freight in Years for City”.
2. Drag the **HTML5 Charts** element into the summary band, and select **Scatter**.
3. In the Outline view, right click on the chart element, and choose **Edit chart properties**.
4. In **HTML5 Chart Properties > Chart Properties > Chart Data > Configuration** create a Category with the following:
    - **Name**: `ShipCountry`
    - **Expression**: `$F{shipcountry}`
    - **Value Class Name**: `java.lang.String`
    - **Order**: `Ascending`

    Click **OK**.
5. Create a second Category with the following:
    - **Name**: `ShipCity`
    - **Expression**: `$F{shipcity}`
    - **Value Class Name**: `java.lang.String`
    - **Order**: `Ascending`

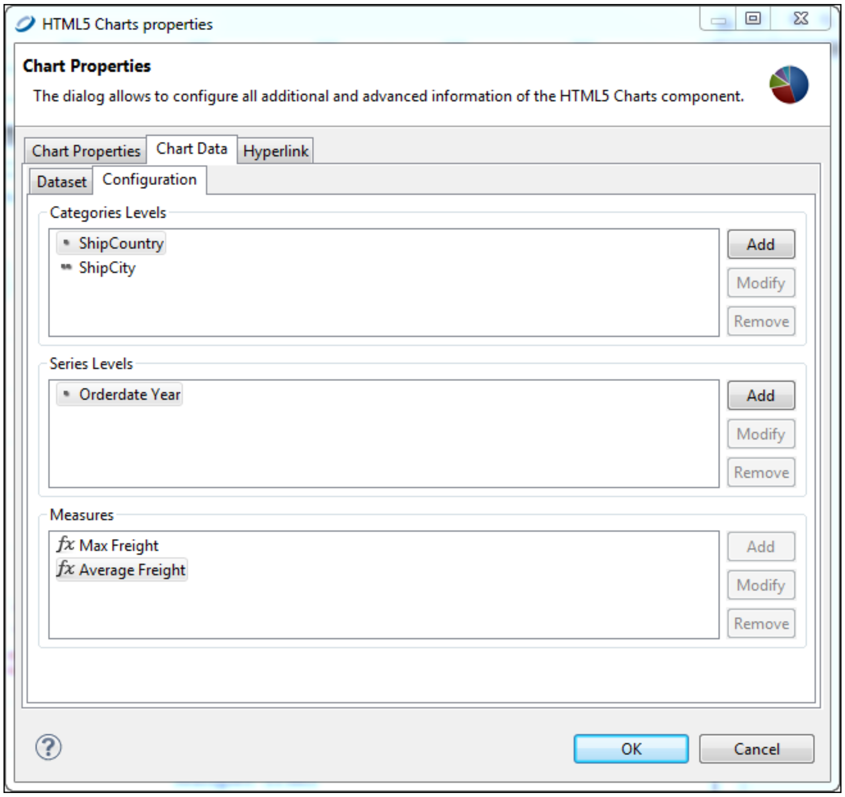
    Click **OK**.
6. Create a Series with the following:
    - **Name**: `Orderdate Year`
    - **Expression**: `YEAR($F{orderdate})`
    - **Value Class Name**: `java.lang.Integer`
    - **Order**: `Ascending`

    Click **OK**.
7. Add a Measure for maximum freight:
    - **Name**: `Max Freight`
    - **Label Expression**: `“Max Freight”`
    - **Calculation**: `Highest`
    - **Value Expression**: `${freight}`
    - **Value Class Name**: `java.math.BigDecimal`

    Click **OK**.
8. Add a Measure for average freight:
    - **Name**: `Average Freight`
    - **Label Expression**: `“Average Freight”`
    - **Calculation**: `Average`
    - **Value Expression**: `${freight}`
    - **Value Class Name**: `java.math.BigDecimal`

    Click **OK**.

Your **Chart Properties** dialog should look like the following:



HTML5 Charts properties

Chart Properties

The dialog allows to configure all additional and advanced information of the HTML5 Charts component.

Chart Properties | Chart Data | Hyperlink

Dataset | Configuration

Categories Levels

- ShipCountry
- ShipCity

Series Levels

- Orderdate Year

Measures

- *fx* Max Freight
- *fx* Average Freight

Figure 13-8 Scatter Chart Properties

9. To change the position or layout of the legend:

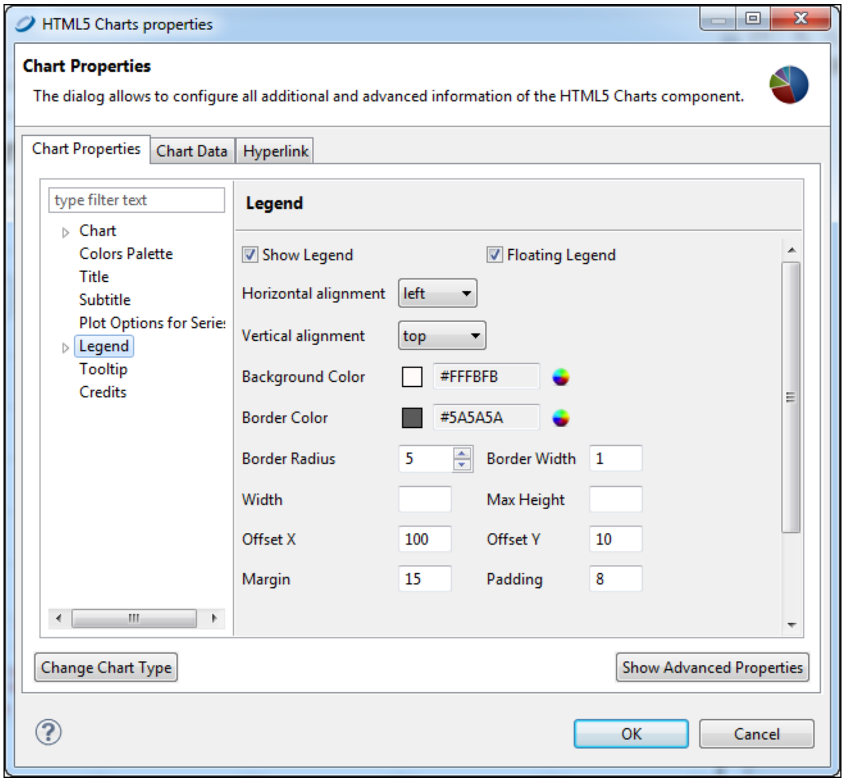


Chart Properties

The dialog allows to configure all additional and advanced information of the HTML5 Charts component.

Legend

☑ Show Legend ☑ Floating Legend

Horizontal alignment: left

Vertical alignment: top

Background Color: #FFFBFB

Border Color: #5A5A5A

Border Radius: 5 Border Width: 1

Width: Max Height:

Offset X: 100 Offset Y: 10

Margin: 15 Padding: 8

Figure 13-9 Legend Properties

a. Go to **Chart Properties > Chart Properties**.

b. Highlight **Legend**.

c. Check the boxes for **Show Legend** and **Floating Legend**.

d. Using the drop-down menus, arrange the legend to appear in the top left.

Click **OK**.

10. To add a border to the chart:
    a. Click the arrow next to **Chart** and highlight **Borders and Plot Area**.
    b. Create a border for your chart by using the arrows next to **Border Width** or by typing the number of pixels.
    c. Select a **Border Color**, **Border Radius**, and **Border Width**.
    d. Click **OK**.

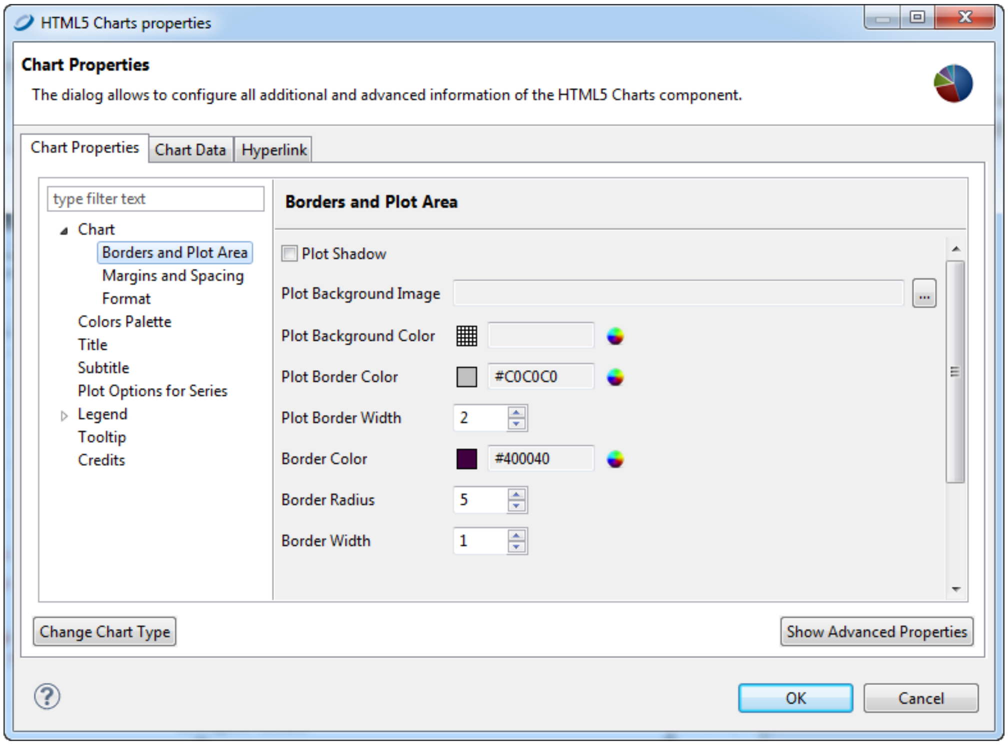


Chart Properties

The dialog allows to configure all additional and advanced information of the HTML5 Charts component.

Borders and Plot Area

☐ Plot Shadow

Plot Background Image

Plot Background Color

Plot Border Color #C0C0C0

Plot Border Width 2

Border Color #400040

Border Radius 5

Border Width 1

Figure 13-10 Border Properties

11. Save and preview your report. It should look like the following:

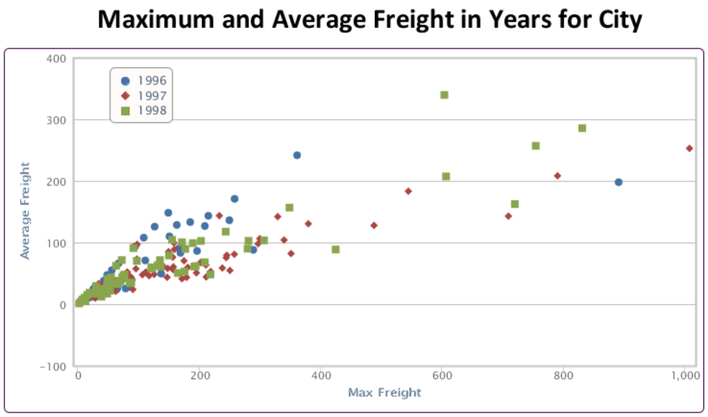


Figure 13-11 Scatter Chart

13.4 Dual-Axis, Multi-Axis, and Combination Charts

Dual- and multi-axis charts are useful when you want to plot multiple chart types on the same chart or use two different scales for each y-axis. This enables you to easily compare data items with very different scales.

Similarly, combination charts are used to display multiple data series in a single chart that combines the features of two different charts.

In this example, we will create a column spline chart that plots freight and orders per country, per year.

To create the report for the chart:

1. Open a new, blank report using the Sugar CRM database and the following query:

   `select * from orders`

   Click **Next**.
2. Move all the fields into the right-hand box. Click **Next**.
3. No need to group by any field. Click **Finish**.
4. Right-click on each band in the outline view, and choose **Delete** to delete all bands except for **Title** and **Summary**.
5. Enlarge the Summary band to 500 pixels by changing the **Height** entry in the **Band Properties** view.

To create the chart:

1. Drag the **HTML5 Charts** element into the summary band, and select **ColumnSpline**.
2. In **HTML5 Chart Properties > Chart Properties > Chart Data > Configuration** create a Category with the following:
   - **Name**: `Country`
   - **Expression**: `$F{shipcountry}`
   - **Value Class Name**: `java.lang.String`
   - **Order**: `None`

   Click **OK**.
3. Create a Series with the following:
   - **Name**: `Year`
   - **Expression**: new `java.text.SimpleDateFormat ("yyyy").format( $F{orderdate} )`
   - **Value Class Name**: `java.lang.Comparable`
   - **Order**: `Ascending`

   Click **OK**.
4. Add a Measure for freight:
   - **Name**: `Freight Values`
   - **Label Expression**: `“Freight”`
   - **Calculation**: `Count`
   - **Value Expression**: `${freight}`
   - **Value Class Name**: `java.lang.integer`

   Click **OK**.
5. Add a second Measure for orders:
   - **Name**: `Total Orders`
   - **Label Expression**: `“Total Orders”`
   - **Calculation**: `DistinctCount`
   - **Value Expression**: `$F{orderid}`
   - **Value Class Name**: `java.lang.integer`

   Click **OK**.

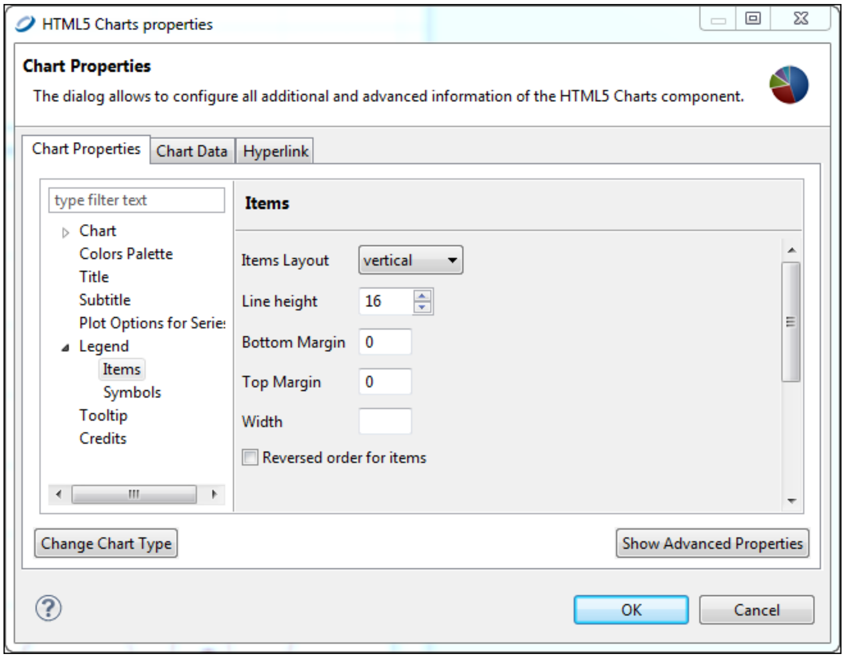
6. To see fewer countries in your chart, select **Edit Chart Properties > Chart Data > Dataset**.

7. Add a Filter expression such as:

```
$F{shipcountry}.startsWith("A") ||
$F{shipcountry}.startsWith ("U") ||
$F{shipcountry}.startsWith ("I")
```

   Click **OK**.

8. To change the position or layout of the legend:



HTML5 Charts properties

Chart Properties

The dialog allows to configure all additional and advanced information of the HTML5 Charts component.

Items

Items Layout: vertical

Line height: 16

Bottom Margin: 0

Top Margin: 0

Width:

□ Reversed order for items

Figure 13-12 Legend Properties

a. Go to **Chart Properties > Chart Properties**.

b. Click the down arrow next to **Legend** on the left.

c. Highlight **Items**.

d. Arrange the **Items Layout** to be Horizontal.

e. Click **OK**.

9. Save and preview your report. It should look like the following:

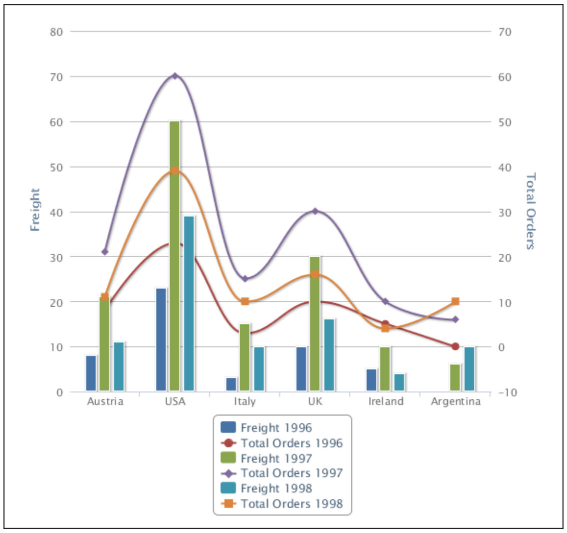


Figure 13-13 Column Spline Chart

CHAPTER 14 USING THE DATASET AND QUERY DIALOG AND THE QUERY BUILDER

Jaspersoft Studio provides tools to help you define report fields and create a proper query if a query language is used to acquire the data for the report. These tools are presented in the **Dataset and Query** dialog.

It also provides an easy way to build SQL queries by using a drag-and-drop query builder. This allows users who don't know SQL or just are not sure about the syntax to quickly join tables and produce complex data filters and where conditions. SQL Builder also provides a way for skilled users to explore the database and list the metadata such as schemas and available tables.

This chapter contains the following sections:

- Using the Dataset and Query Dialog
- Working with the Query Builder

14.1 Using the Dataset and Query Dialog

To open the **Dataset and Query** dialog, click the **Dataset and Query** icon ▤.

When working with a sub-dataset, open the dialog by right-clicking the dataset name inside the Outline view, and selecting **Dataset and Query...**.

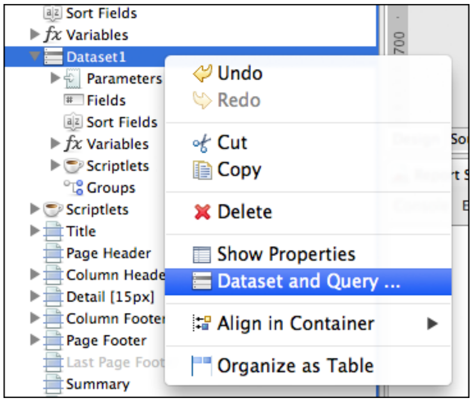
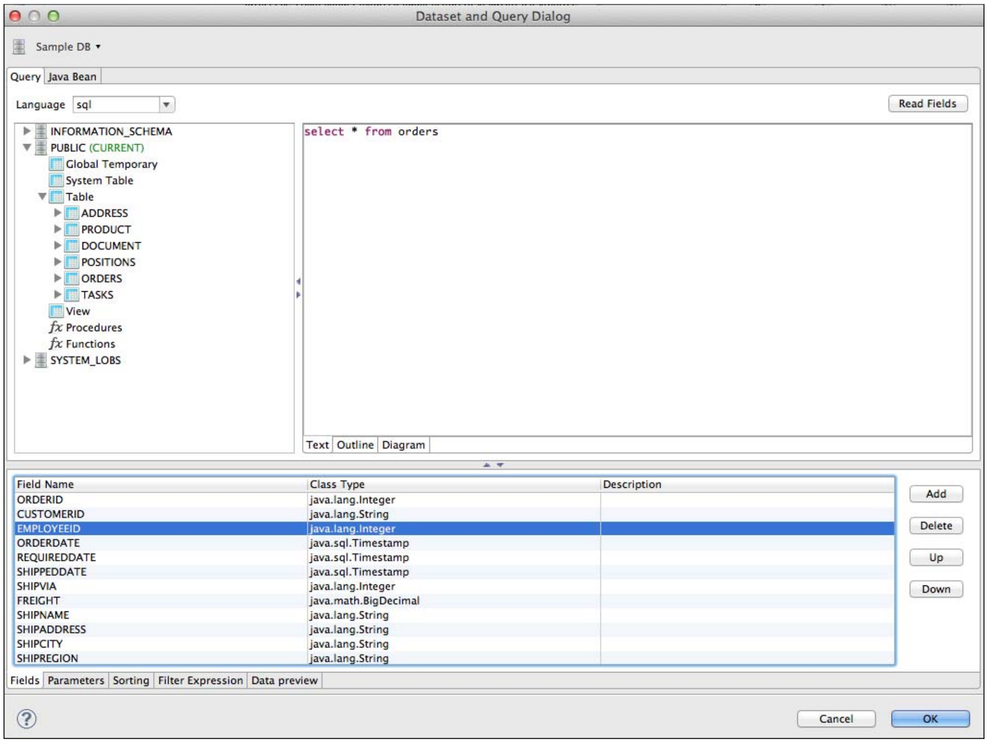


Figure 14-1 Right-Click Menu

The **Dataset and Query** dialog allows you to:

- Select a data adapter with which to configure the dataset. Usually a data adapter will be preselected, but it is possible to change it if necessary.
- Select a query language for the dataset being edited (which can be the main dataset or a sub-dataset used to populate a chart or a table).
- Specify the query by typing text or by using a tool, if available, designed for the selected language. A tool is available for several languages including SQL, XPath and JSON.
- Retrieve the fields from the selected Data Adapter: these can be provided directly by the Data Adapter or by executing the query and reading the response metadata.
- Add, edit, and remove fields and parameters.
- Edit sort options.
- Provide an expression to filter dataset records.
- Preview your data, if supported by the selected data adapter.



Language sql

```
select * from orders
```

Field Name	Class Type	Description
ORDERID	java.lang.Integer	
CUSTOMERID	java.lang.String	
EMPLOYEEID	java.lang.Integer	
ORDERDATE	java.sql.Timestamp	
REQUIREDDATE	java.sql.Timestamp	
SHIPPEDDATE	java.sql.Timestamp	
SHIPVIA	java.lang.Integer	
FREIGHT	java.math.BigDecimal	
SHIPNAME	java.lang.String	
SHIPADDRESS	java.lang.String	
SHIPCITY	java.lang.String	
SHIPREGION	java.lang.String	

Figure 14-2

Defining all the fields of a report by hand can be tedious, especially when there are a lot of fields. JasperReports requires all report fields to be named and configured with a proper class type. The main purpose of the Dataset and Query dialog is to simplify this process by automatically discovering the available fields provided by a data adapter without using a query or by the data which comes from the execution of a query. To execute a query it is important to use the proper data adapter: for example when using SQL as language, in order to execute a SQL query you must use a JDBC data adapter to connect to the database. The **Read Fields** button starts the discovery process: the fields found will be listed in the **Fields** tab and added to the report. Close this dialog by clicking the **OK** button.

Some query languages, like XPath, do not produce a result that resemble a table, but more complex structures which requires extra steps to correctly map result data to report fields. An example of this is the multidimensional result coming from MDX results (MDX is the query language used with OLAP and XML/A connections): in this case each field is mapped by specifying an expression stored in the field description. For some of languages, such as instance XPath and JSON query, the Query dialog displays a tool that simplifies the creation of both the query and the mapping.

14.2 Working with the Query Builder

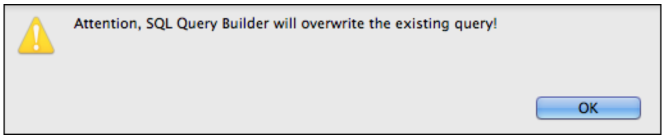
When your language is SQL, the Query Builder is called the SQL Builder. The tool requires a JDBC data adapter.

The builder is divided in two parts. On the left a tree-view shows all the available schemas and relative objects like tables, views, etc. found by using the JDBC connection provided by the data adapter. On the right side there are three tabs which present the query in different ways.

The first **Text** tab contains a text area in which you can write a query. Tables and other objects can be dragged from the metadata view into the text area, so you don't have to write the entire qualified name of that objects. Although the SQL builder does not support arbitrary complex queries (which may use database-specific syntax), this text area can be used for any supported query, included stored procedures if supported by the report query executer.

If a query has been already set for the report, this text area will show it when the query dialog is open.

The **Outline** and **Diagram** tabs are used to visually build the query. The current version of Jaspersoft Studio does not support back-parsing of SQL. For this reason, Outline and Diagram editing mode should be used only to create new queries, otherwise the new query will replace any existing query. If you do attempt to overwrite an existing query, a warning will appear.

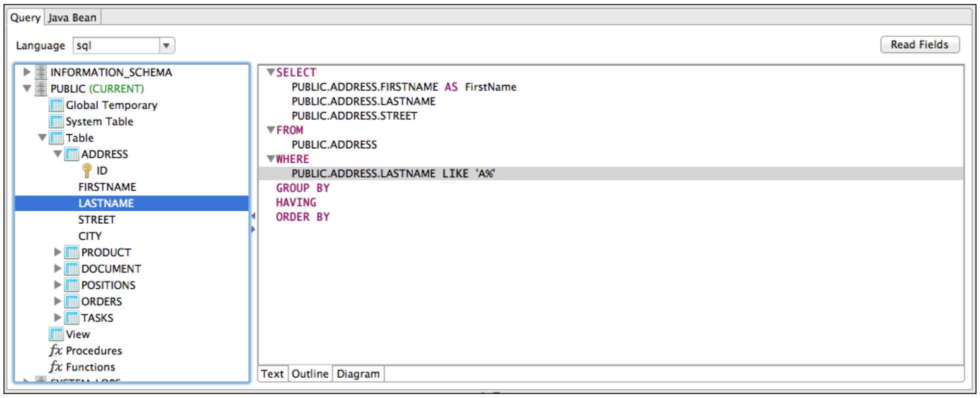


Attention, SQL Query Builder will overwrite the existing query!

Figure 14-3 Query Overwrite Warning

14.2.1 Query Outline View and Diagram View

The purpose of SQL is to select data from the tables of the database. SQL allows you to join tables, so that you can get data from more than one table at a time.



Language sql

```
SELECT
    PUBLIC.ADDRESS.FIRSTNAME AS FirstName
    PUBLIC.ADDRESS.LASTNAME
    PUBLIC.ADDRESS.STREET
FROM
    PUBLIC.ADDRESS
WHERE
    PUBLIC.ADDRESS.LASTNAME LIKE 'A%'
GROUP BY
HAVING
ORDER BY
```

Figure 14-4 Outline View

The **Outline** view is a good tool for people that have basic understanding of SQL. It works in conjunction with the **Diagram** view, which represents the most simple way to design a query. In the outline view the query is split in its main parts introduced by the relative keyword. Those parts include:

`SELECT` introduces the portion of query where are listed all the columns which will form a record of the final result set (which is basically a list of records).

`FROM` contains the list of tables involved in our queries and if specified the rules to join that tables.

`WHERE` is the portion of query that describes the filters and conditions that the data must satisfy in order to be part of the final result, conditions can be combined by using the `OR` and `AND` logical operators and can be aggregated by using parentheses.

GROUP BY indicates a set of fields used to aggregate data and is used when an aggregation function is present in the SELECT. For example, the following query counts the number of orders placed in each country.

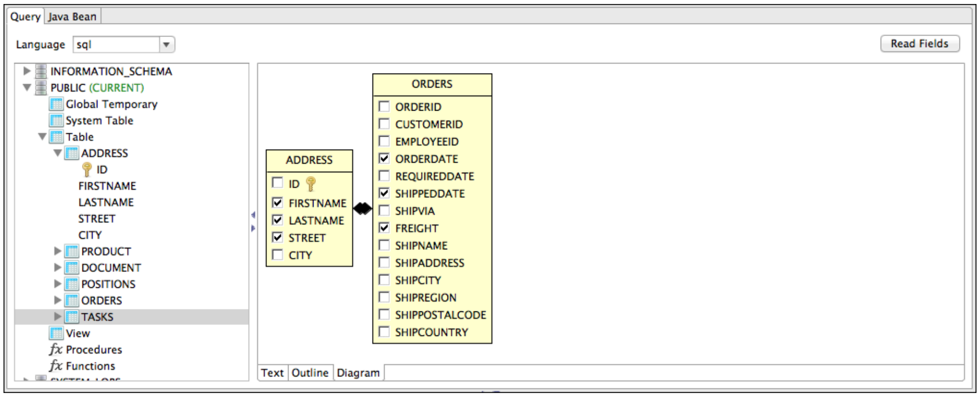
```
SELECT
  count(*) as number_of_orders,
  Orders.country
FROM
  Orders
GROUP BY
  Orders.country
```

HAVING works in a similarly to WHERE, but it is used with aggregate functions. For example the following query filters the records by showing only the countries with at least 40 orders:

ORDER BY specifies a set of columns to be used to sort the result set.

```
SELECT
  count(*) as number_of_orders,
  Orders.country
FROM
  Orders
GROUP BY
  Orders.country
HAVING
  count(*) > 40
```

The Diagram view shows the tables in the query with the relative join connections and the selected fields. This view is very useful especially to select the relevant fields and easily edit the table joins by double-clicking the connection arrows.



ADDRESS
- ☐ ID
- ☑ FIRSTNAME
- ☑ LASTNAME
- ☑ STREET
- ☐ CITY

ORDERS
- ☐ ORDERID
- ☐ CUSTOMERID
- ☐ EMPLOYEEID
- ☑ ORDERDATE
- ☐ REQUIREDDATE
- ☑ SHIPPEDDATE
- ☐ SHIPVIA
- ☑ FREIGHT
- ☐ SHIPNAME
- ☐ SHIPADDRESS
- ☐ SHIPCITY
- ☐ SHIPREGION
- ☐ SHIPPOSTALCODE
- ☐ SHIPCOUNTRY

Figure 14-5 Diagram View

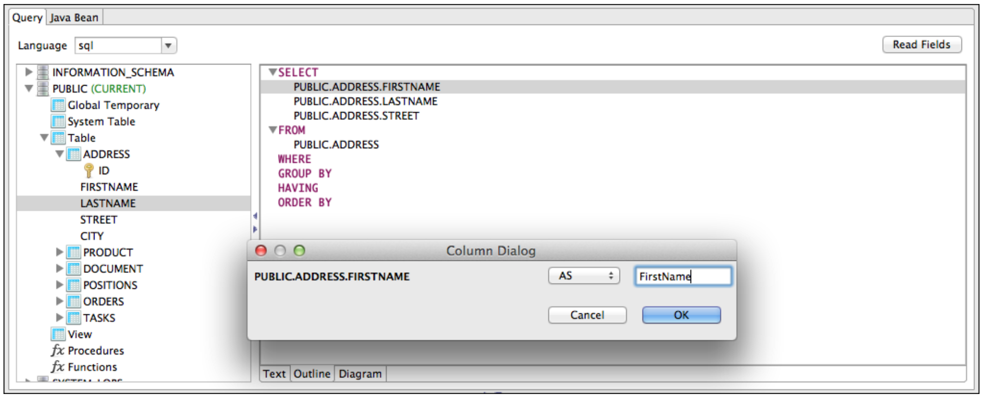
14.2.2 Selecting columns

Columns can be dragged from the database explorer to the outline view into the SELECT node or other nodes. From there they can be used like the ORDER BY node. When selecting a column, be sure the table it comes from is present in the FROM part of your query.

Fields can also be selected in the diagram view by selecting the relative checkbox.

Finally you can add a column by right-clicking the **SELECT** node in the outline view and selecting **Add Column**. A new dialog will allow you to pick the column from the ones available, which are the columns present in all tables mentioned in the FROM clause. In case the column to add is not a table column, but a more complex expression, for instance an aggregation function like COUNT(*), add it by right-clicking the SELECT node in the outline view and selecting **Add Expression**.

When a column is added to the query as part of the SELECT section, it is possible to set an alias for it by double-clicking it.



```
▼SELECT
    PUBLIC.ADDRESS.FIRSTNAME
    PUBLIC.ADDRESS.LASTNAME
    PUBLIC.ADDRESS.STREET
▼FROM
    PUBLIC.ADDRESS
 WHERE
 GROUP BY
 HAVING
 ORDER BY
```

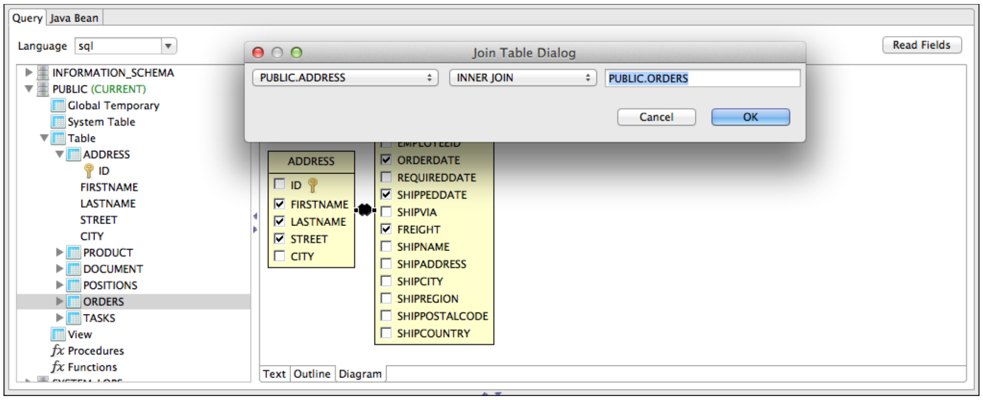
PUBLIC.ADDRESS.FIRSTNAME AS FirstName

Figure 14-6 Setting an Alias

Aliases are useful when you have several fields with the same name coming from two different tables.

14.2.3 Joining tables

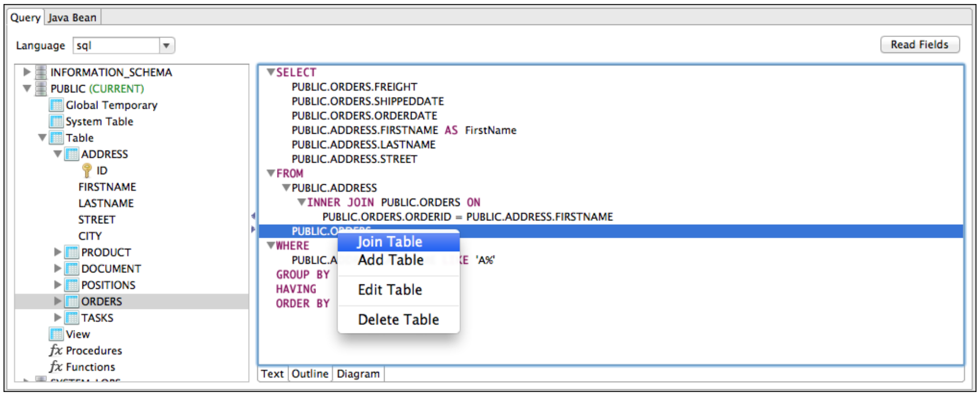
Tables added to the FROM clause or dragged inside the Diagram view can be joined by selecting shared fields. In the Diagram view the relationship can be created by dragging a column of the first table onto the column of the table to join. This type of join can be edited by double-clicking the join arrows. Currently a single join condition is supported.



PUBLIC.ADDRESS | INNER JOIN | PUBLIC.ORDERS

Figure 14-7 Column Dialog

Joins can also be edited by the outline view by right-clicking a table name and selecting **Add or Edit Table Join**.



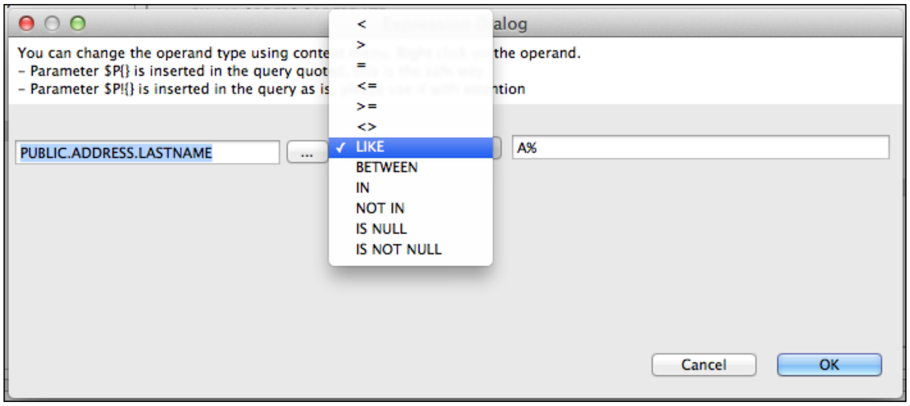
```
▼SELECT
    PUBLIC.ORDERS.FREIGHT
    PUBLIC.ORDERS.SHIPPEDDATE
    PUBLIC.ORDERS.ORDERDATE
    PUBLIC.ADDRESS.FIRSTNAME AS FirstName
    PUBLIC.ADDRESS.LASTNAME
    PUBLIC.ADDRESS.STREET
▼FROM
  ▼PUBLIC.ADDRESS
    ▼INNER JOIN PUBLIC.ORDERS ON
        PUBLIC.ORDERS.ORDERID = PUBLIC.ADDRESS.FIRSTNAME
    PUBLIC.O
▼WHERE
    PUBLIC.A                E 'A%'
 GROUP BY
 HAVING
 ORDER BY
```

Figure 14-8 Join Table

14.2.4 Data selection criteria (WHERE conditions)

A WHERE clause allows you to specify the criteria that must be satisfy by each record in order to be part of the query result. Right click the WHERE node in the outline view and select **Add Condition** to add a new condition.

If you know in advance which column should be involved in the condition, you can drag this column on the **WHERE** node to create the condition. In both cases the condition dialog appears.



You can change the operand type using context menu. Right click on the operand.
- Parameter $P{} is inserted in the query quoted, this is the safe way
- Parameter $P!{} is inserted in the query as is, please use it with attention

PUBLIC.ADDRESS.LASTNAME LIKE A%

Figure 14-9 Add Condition

Many conditions can be organized by creating condition groups and combined by using `OR` or `AND` operators. `OR` groups require that at least one condition is true, while `AND` requires that all the conditions of the group are verified. By double-clicking the group operator it is possible to change the value from `AND` to `OR` and vice versa. Groups may be formed by nested groups in a recursive way.

If the value of a condition is not fixed, it can be expressed by using a parameter with the expression: `$P{parameter_name}`. In very few cases it could also be useful to use the `$P!{}` syntax instead, but in general this is not the case when creating condition values, since by using `$P!{}` there is no escape of the parameters, making the query potentially unsecure.

When using parameters of type collection, the `$X{}`expressions allow you to use operators like `IN` and `NOT IN`, which test if a value is present or not in the provided list. `$X{}` syntax allows also the use of other operators like `BETWEEN` that can be used with numbers, Date Range and Time Range type of parameters.

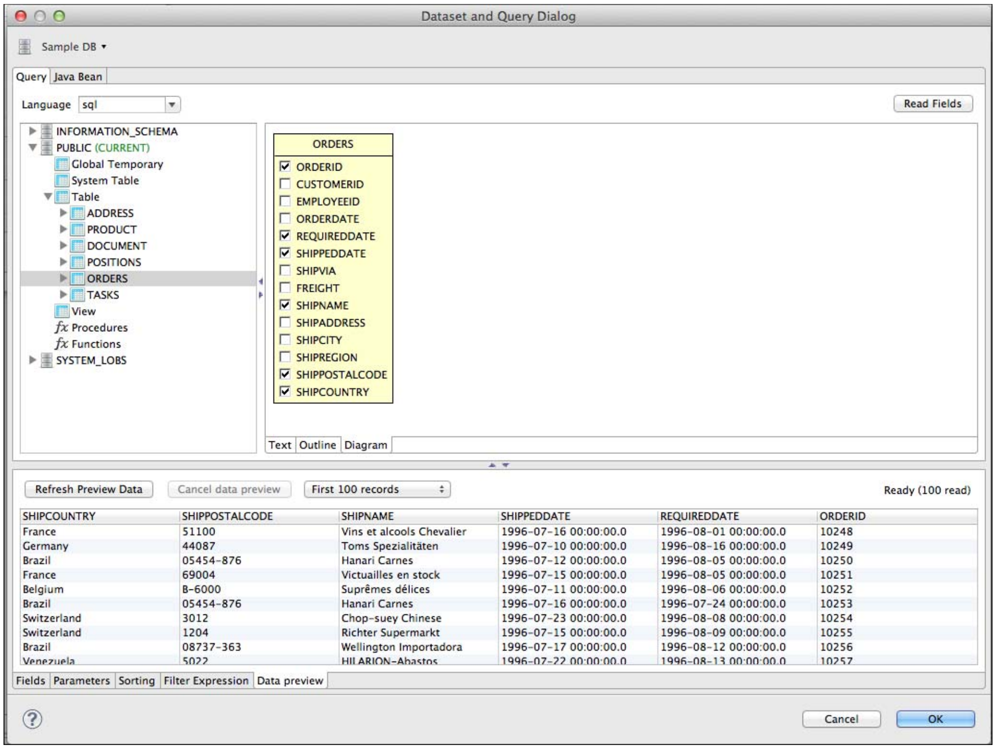
14.2.5 Acquiring fields

Once the query is ready, it's time map the columns of the result set to fields of the report. When using SQL, this is a pretty straight forward operation: by pressing the button Read Fields the query is executed, and the metadata is read; if the query is valid and no errors occurred, for each column a field is added to the fields list with the proper class type.

This is also a good way to test if a query contains syntactical errors. In general, if the query has been created by using the designer, it should always be valid.

14.2.6 Data Preview

The **Data Preview** tab allows you to generate a ghost report having the fields map to the fields tab and current query by using the selected Data Adapter. This tool works independently of the query language and is a good way to debug a query or check what records the dataset will return.



ORDERS

☑ ORDERID
☐ CUSTOMERID
☐ EMPLOYEEID
☐ ORDERDATE
☑ REQUIREDDATE
☑ SHIPPEDDATE
☐ SHIPVIA
☐ FREIGHT
☑ SHIPNAME
☐ SHIPADDRESS
☐ SHIPCITY
☐ SHIPREGION
☑ SHIPPOSTALCODE
☑ SHIPCOUNTRY

First 100 records

Ready (100 read)

SHIPCOUNTRY	SHIPPOSTALCODE	SHIPNAME	SHIPPEDDATE	REQUIREDDATE	ORDERID
France	51100	Vins et alcools Chevalier	1996-07-16 00:00:00.0	1996-08-01 00:00:00.0	10248
Germany	44087	Toms Spezialitäten	1996-07-10 00:00:00.0	1996-08-16 00:00:00.0	10249
Brazil	05454-876	Hanari Carnes	1996-07-12 00:00:00.0	1996-08-05 00:00:00.0	10250
France	69004	Victuailles en stock	1996-07-15 00:00:00.0	1996-08-05 00:00:00.0	10251
Belgium	B-6000	Suprêmes délices	1996-07-11 00:00:00.0	1996-08-06 00:00:00.0	10252
Brazil	05454-876	Hanari Carnes	1996-07-16 00:00:00.0	1996-07-24 00:00:00.0	10253
Switzerland	3012	Chop-suey Chinese	1996-07-23 00:00:00.0	1996-08-08 00:00:00.0	10254
Switzerland	1204	Richter Supermarkt	1996-07-15 00:00:00.0	1996-08-09 00:00:00.0	10255
Brazil	08737-363	Wellington Importadora	1996-07-17 00:00:00.0	1996-08-12 00:00:00.0	10256
Venezuela	5022	HILARION-Abastos	1996-07-22 00:00:00.0	1996-08-13 00:00:00.0	10257

Figure 14-10 Data Preview Tab