

JASPER REPORTS

java web reporting engine



tutorialspoint
SIMPLY EASY LEARNING

www.tutorialspoint.com

 <https://www.facebook.com/tutorialspointindia>

 <https://twitter.com/tutorialspoint>

About the Tutorial

JasperReports is an open source java reporting engine. JasperReports is a Java class library, and it is meant for those Java developers who need to add reporting capabilities to their applications. This simple and user-friendly tutorial covers almost all the basics of JasperReports that a beginner should know.

Audience

This tutorial is designed for Software Professionals as well as for all those beginners who would like to learn the concepts of JasperReports.

Prerequisites

Before proceeding with this tutorial, it is expected that you have a basic understanding of Java programming language. A basic understanding of Java and other associated programming will be an additional advantage to understand the topic.

Copyright & Disclaimer

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of the contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness, or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience	i
Prerequisites	i
Copyright & Disclaimer	i
Table of Contents	ii
1. JASPERREPORTS – GETTING STARTED	1
What is a Report?	1
Report Template	1
JasperReports	2
2. JASPERREPORTS – ENVIRONMENT SETUP	3
JasperReport Environment	3
Jasper Managers Classes	6
Setting up Apache ANT	7
3. JASPERREPORTS – LIFE CYCLE	8
4. JASPERREPORTS – DESIGNS	10
Creating a JRXML Report Template	10
Previewing the XML Report Template	13
5. JASPERREPORTS - COMPILING REPORT DESIGN	15
Programmatic Compilation of JRXML	15
Template Compilation	18
Preview Compiled Report Template	19
Compilation through ANT Task	20
6. JASPERREPORTS – FILLING REPORTS	23
Filling Report Templates	23

Generating Reports	28
7. JASPERREPORTS – VIEW & PRINT REPORTS	31
Viewing Reports	31
Printing Reports	37
8. JASPERREPORTS – EXPORTING REPORTS	41
Exporting to Other Formats	41
9. REPORTS PARAMETERS	49
Parameter Declaration	49
Example	50
Report Generation	57
10. REPORT DATA SOURCES	60
Datasource Implementations	60
Rewindable Data Sources	62
Data Source Providers	62
11. REPORT FIELDS	63
Field Declaration	63
Sorted Report Example	64
12. REPORT EXPRESSION	74
Expression Declaration	74
Calculator	76
Conditional Expressions	76
Example of conditional Expression in Report	77
13. REPORT VARIABLES	86
Variable Declaration	86
Built-In Report Variables	88

Example	89
Report Generation	95
14. REPORT SECTIONS	99
Main Sections	99
Section Elements	102
Section Attributes	104
Example	104
15. REPORT GROUPS	112
Group Attributes	112
Example	113
Report Generation	120
16. REPORT FONTS	124
Report Fonts	124
Default Fonts and Inheritance	126
Example	127
17. UNICODE SUPPORT	137
Character Encoding	137
Unicode	137
Example	138
18. REPORT STYLES	146
Style Properties	146
Conditional Styles	147
Applying Styles to Report Elements	147
Style Templates	147
Example	148

Report Generation	155
19. REPORT SCRIPTLETS	158
Scriptlet Declaration	158
Global Scriptlets	159
Report Governors	159
Example	160
Report Generation	169
20. CREATE SUBREPORTS	173
<subreport> Element	173
Example	174
Report Generation	183
21. CREATING CHARTS	187
Chart Properties	187
Chart Datasets	189
Dataset Properties	189
Dataset Types	190
Chart Plots	193
Types of Charts	194
Example	195
Report Generation	201
22. JASPERREPORTS - CROSSTABS	206
Crosstab Properties	206
Example	209
Report Generation	215

23. JASPERREPORTS – INTERNATIONALIZATION ----- 219

 Example -----220

1. JASPERREPORTS – GETTING STARTED

What is a Report?

A report is a meaningful, well-defined, and summarized presentation of information. Usually, the routine activities are automated and data summarized into a decision-supporting "Reports". Reports represent usual messy data into charts, graphs, and other forms of graphical representations.

Report Template

Generally, the following layout is adopted to generate reports by most of the commercial report generating tools.

TITLE
PAGEHEADER
COLUMNHEADER
DETAIL
COLUMNFOOTER
PAGEFOOTER
SUMMARY

Following are the descriptions of each element mentioned in the diagram:

Element	Description
title	Title contains the 'Title' of the report. It appears only once at the very beginning of the report, for example, "Tutorials Point Report."
pageHeader	PageHeader may contain date and time information and/or organization name. This appears at the top of each page.
columnHeader	ColumnHeader lists the names of those specific fields, which you want to display in the report, for example, "Author Name," "Starting Hour," "Finishing Hour," "Hours Worked," "Date," etc.
detail	Detail is the part where entries of the specific fields (listed in columnHeader) are shown, for example "Manisha", "9:00", "18:00", "9", "10.02.2013."

columnFooter	ColumnFooter may display summation of any of the field, for example, "Total Hours Worked: "180."
pageFooter	PageFooter may contain page count information. It appears at the bottom of each page, for example, "1/23."
summary	Summary contains information inferred from "detail" part, for example, after listing the number of hours, worked by each author, total hours worked by each author can be put in visual chart like pie chart, graph, etc. for better comparison.

JasperReports

Following are the common troubles faced during the report development:

- **Core changes:** Usually, to reflect the business changes or enhancements it is required to change the core logic of the report.
- **Results exporting:** There are a wide range of formats, which your report can be exported to, such as: HTML, Text, PDF, MS Excel, RTF, ODT, Comma-separated values, XML, or image.
- **Complicated reports:** sub-reports and cross-tabs reports are good example.
- **Charts reports:** Visual charts, for example, Graph, Pie, XY Line, Bar, Meter, and Time series.

To remove the overhead of the above mentioned points and to facilitate the reporting process, a lot of frameworks, tools, libraries, and 3rd parties applications were introduced. **JasperReports** is one of them.

JasperReports is an open source java reporting engine. It is Java based and doesn't have its own expression syntax. JasperReports has the ability to deliver rich content onto the screen, to the printer, or into PDF, HTML, XLS, RTF, ODT, CSV, TXT, and XML files. As it is not a standalone tool, it cannot be installed on its own. Instead, it is embedded into Java applications by including its library in the application's CLASSPATH.

JasperReports is a Java class library, and is not meant for the end users, but rather is targeted towards Java developers who need to add reporting capabilities to their applications.

Features of JasperReports

Some of the significant features of JasperReports are:

- It has a flexible report layout.
- It can present data either textually or graphically.
- Developers can supply data in multiple ways.
- It can accept data from the multiple data sources.

JasperReports

- It can generate watermarks (A watermark is like a secondary image that is laid over the primary image).
- It can generate sub-reports.
- It is capable of exporting reports in a variety of formats.

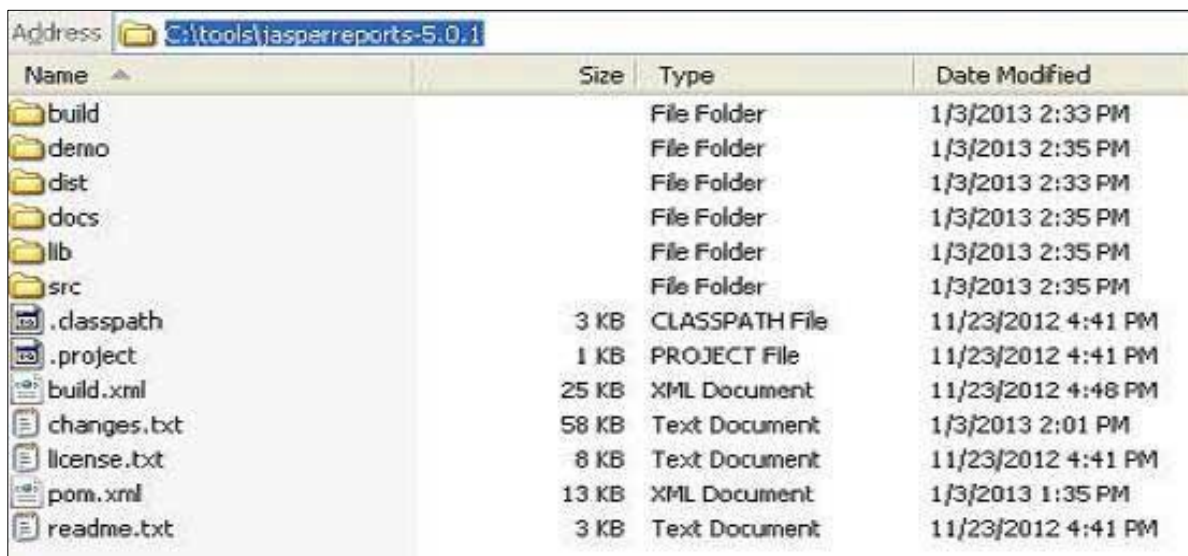
2. JASPERREPORTS – ENVIRONMENT SETUP

JasperReports is a pure Java library and not a standalone application. It cannot run on its own, hence it needs to be embedded into another client or server-side Java application. As it is Java based, it can be run on any platform that supports Java (JDK 1.3 and above). All the JasperReport's functionalities are gathered in a single JAR file, `jasperreports-x.x.x.jar`. This JAR along with the required and optional libraries (.ZIP file) can be downloaded from the site: **JasperReport Library Link**. Download the latest version from this link.

The ZIP file includes the JasperReports JAR file along with the JasperReports source code, dependent JARs, and a lot of examples demonstrating JasperReport's functionalities.

JasperReport Environment

To start creating the reports, we need to set up the environment ready. Extract the downloaded JasperReport.ZIP file to any location (in our case, we have extracted it to `C:\tools\jasperreports-5.0.1`). The directory structure of the extracted file is same as shown below:



Name	Size	Type	Date Modified
build		File Folder	1/3/2013 2:33 PM
demo		File Folder	1/3/2013 2:35 PM
dist		File Folder	1/3/2013 2:33 PM
docs		File Folder	1/3/2013 2:35 PM
lib		File Folder	1/3/2013 2:35 PM
src		File Folder	1/3/2013 2:35 PM
.classpath	3 KB	CLASSPATH File	11/23/2012 4:41 PM
.project	1 KB	PROJECT File	11/23/2012 4:41 PM
build.xml	25 KB	XML Document	11/23/2012 4:48 PM
changes.txt	58 KB	Text Document	1/3/2013 2:01 PM
license.txt	8 KB	Text Document	11/23/2012 4:41 PM
pom.xml	13 KB	XML Document	1/3/2013 1:35 PM
readme.txt	3 KB	Text Document	11/23/2012 4:41 PM

Here is the detail of all the directories:

- *build*: Contains the compiled JasperReport class files.
- *demo*: Contains various examples, demonstrating several aspects of JasperReports functionality.
- *dist*: Contains `jasperreports-x.x.x.jar` file. We shall add this JAR file to our CLASSPATH to take advantage of JasperReports.

- *docs*: Contains a local copy of the JasperReports documentation.
- *lib*: Contains all JARs needed, both to build JasperReports and to use it in our applications.
- *src*: Contains the JasperReports source code.
- *build.xml*: An ANT build file to build the JasperReports source code. If we don't intend to modify JasperReports, we don't need to use this file since JasperReports is distributed in the compiled form.
- *changes.txt*: A text document, explaining the differences between the current and previous versions of the JasperReports class library.
- *license.txt*: A text document that contains the full text of the LGPL (Lesser General Public License) license.
- *readme.txt*: A text document, containing instructions on how to build and execute the supplied examples.

Basically, we only use the `jasperreports-x.x.x.jar` under the *dist* and JARs under the *lib* directory for generating reports. As JasperReports being an open source tool, if any defect or bug is recognized during execution in the `jasperreports-x.x.x.jar`, we can fix it and build the JAR again using the `build.xml` file.

Set the CLASSPATH

To use JasperReport, we need to set the following files to our CLASSPATH:

- `jasperreports-x.x.x.jar`, where `x.x.x` is the JasperReports version. This found under directory `C:\tools\jasperreports-x.x.x\dist`).
- All the JAR files under the *lib* subdirectory (`C:\tools\jasperreports-x.x.x\lib`).

At the time of installation, we used JasperReport version 5.0.1. Right-click on 'My Computer' and select 'Properties', click on the 'Environment variables' button under the 'Advanced' tab. Now update the 'Path' variable with this **`C:\tools\jasperreports-5.0.1\dist\jasperreports-5.0.1.jar;C:\tools\jasperreports-5.0.1\lib`**. Now you are ready to create your reports.

In all the examples in this tutorial, we have used ANT tasks to generate reports. The **build** file takes care of importing all the required JARs for generating reports. Hence, setting CLASSPATH as mentioned above will only help those who wish to generate reports without using ANT.

Build Setup

All the examples in this tutorial:

- have been written using simple Text Editor.

- have been saved under the directory C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint.
- have been compiled and executed from command prompt, using Apache ANT. We will use a **baseBuild.xml** file, which we shall import in ANT **build.xml** file in the subsequent chapters. Save this file to C:\tools\jasperreports-5.0.1\test. Following is the content of baseBuild.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportExample" basedir=".">
  <description>Previews our JasperReport XML Design</description>
  <property name="file.name" value="jasper_report_template" />
  <!-- Directory where the JasperReports project file was extracted
  needs to be changed to match the local environment -->
  <property name="jasper.dir" value=".." />
  <property name="dist.dir" value="${jasper.dir}/dist" />
  <property name="lib.dir" value="${jasper.dir}/lib" />
  <property name="src.dir" value="src" />
  <property name="classes.dir" value="classes" />
  <property name="main-class" value="com.tutorialspoint.HelpMe" />

  <path id="classpath">
    <pathelement location="." />
    <pathelement location="${classes.dir}" />
    <fileset dir="${lib.dir}">
      <include name="**/*.jar" />
    </fileset>
    <fileset dir="${dist.dir}">
      <include name="**/*.jar" />
    </fileset>
  </path>

  <target name="compile" depends="clean-sample">
    <mkdir dir="${classes.dir}"/>
    <javac srcdir="${src.dir}" destdir="${classes.dir}"
      classpathref="classpath" />
  </target>
</project>
```

```

</target>

<target name="run" depends="compile">
  <echo message="Running class : ${main-class}"/>
  <java fork="true" classname="${main-class}">
    <classpath>
      <path refid="classpath" />
    </classpath>
  </java>
</target>
<target name="clean-sample">
  <delete dir="${classes.dir}" />
  <delete file="./${file.name}.jasper" />
  <delete file="./${file.name}.jrprint" />
</target>
</project>

```

This file has all the required targets, like cleaning the directories, compiling the java files, and executing the class files.

Following are the details, mentioned by various directories in baseBuild.xml. Assuming current directory is C:\tools\jasperreports-5.0.1\test):

- *jasper.dir*: is C:\tools\jasperreports-5.0.1 directory
- *lib.dir*: is C:\tools\jasperreports-5.0.1\lib directory
- *src.dir*: is C:\tools\jasperreports-5.0.1\test\src
- *classes.dir*: is C:\tools\jasperreports-5.0.1\test\classes
- *main-class*: com.tutorialspoint.HelpMe. This class is a simple class executed, when no class file name is passed from the command line. Save this file to C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint.

```

package com.tutorialspoint;

public class HelpMe {
    public static void main(String[] args) {
        System.out.println("This is the default class executed."
            + "Please pass the fully qualified class"

```

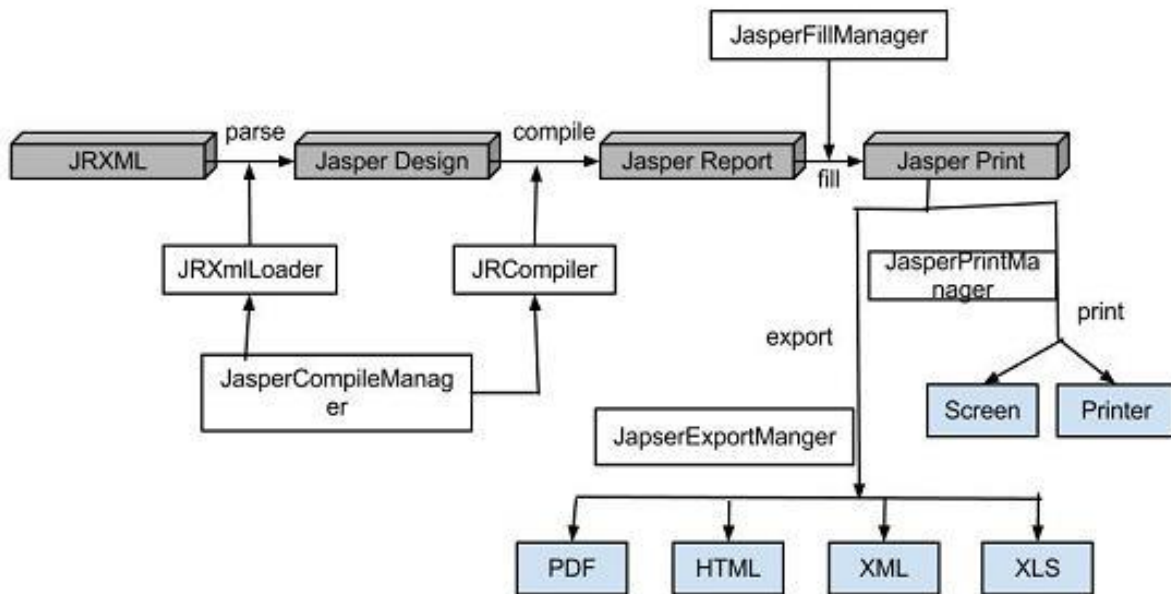
```

+ " name to be executed as command line"
+ " parameter, for example,"
+ " com.tutorialspoint.HelpMe ");
}
}

```

Jasper Managers Classes

There are number of classes, which will be used to compile a JRXML report design, to fill a report, to print a report, to export to PDF, HTML & XML files, view the generated reports, and report design.



The list of these classes is:

- net.sf.jasperreports.engine.JasperCompileManager: Used to compile a JRXML report template.
- net.sf.jasperreports.engine.JasperFillManager: Used to fill a report with data from the data source.
- net.sf.jasperreports.engine.JasperPrintManager: Used to print the documents generated by the JasperReports library.

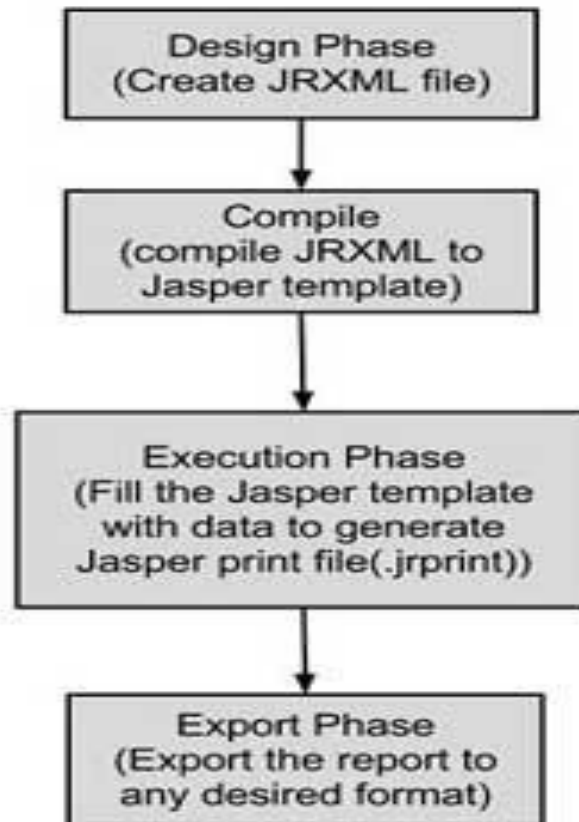
- `net.sf.jasperreports.engine.JasperExportManager`: Used to obtain PDF, HTML, or XML content for the documents produced by the report-filling process.
- `net.sf.jasperreports.view.JasperViewer`: It represents a simple Java Swing application, which can load and display reports.
- `net.sf.jasperreports.view.JasperDesignViewer`: Used at design time to preview the report templates.

Setting up Apache ANT

We are going to build all the examples using Apache ANT. So, kindly check **ANT - Environment Setup** chapter to setup Apache ANT on your system.

3. JASPERREPORTS – LIFE CYCLE

The main purpose of JasperReports is to create page oriented, ready to print documents in a simple and flexible manner. The following flow chart depicts a typical work flow while creating reports.



As shown in the image, the life cycle has the following distinct phases:

- **Designing the report:** In this step, we create the JRXML file, which is an XML document that contains the definition of the report layout. We can use any text editor or **iReportDesigner** to manually create it. If iReportDesigner is used, the layout is designed in a visual way, hence real structure of the JRXML can be ignored.
- **Compiling the report:** In this step, JRXML is compiled in a binary object called a Jasper file (*.jasper). This compilation is done for performance reasons. Jasper files are what you need to ship with your application in order to run the reports.

- **Executing the report (Filling data into the report):** In this step, data from the application is filled in the compiled report. The class `net.sf.jasperreports.engine.JasperFillManager` provides necessary functions to fill the data in the reports. A Jasper print file (*.jrprint) is created, which can be used either to print or export the report.
- **Exporting the report to desired format:** In this step, we can export the Jasper print file created in the previous step to any format using `JasperExportManager`. As Jasper provides various forms of exports, hence with the same input, we can create multiple representations of the data.

A detailed overview of each of the above steps will be given in the subsequent chapters.

4. JASPERREPORTS – DESIGNS

The JRXML templates (or JRXML files) in JasperReport are standard XML files, having an extension of .jrxml. All the JRXML files contain tag <jasperReport>, as root element. This in turn contains many sub-elements (all of these are optional). JasperReport framework can handle different kinds of data sources. In this tutorial, we shall show how to generate a basic report, just by passing a collection of Java data object (using Java beans), to the JasperReport Engine. The final report shall display a list of people with the categories including their names and countries.

The Following steps are covered in this chapter to describe — how to design a JasperReport:

- Creating a JRXML Report Template and
- Previewing the XML Report Template.

Creating a JRXML Report Template

Create the JRXML file, which is **jasper_report_template.jrxml** using a text editor and save this file in *C:\tools\jasperreports-5.0.1\test* as per our environment setup.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<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="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">
  <queryString>
    <![CDATA[]]>
  </queryString>
  <field name="country" class="java.lang.String">
    <fieldDescription><![CDATA[country]]></fieldDescription>
  </field>
  <field name="name" class="java.lang.String">
```

```

    <fieldDescription><![CDATA[name]]></fieldDescription>
</field>
<columnHeader>
  <band height="23">
    <staticText>
      <reportElement mode="Opaque" x="0" y="3" width="535"
        height="15" bgcolor="#70A9A9" />
      <box>
        <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
      </box>
      <textElement />
      <text><![CDATA[]]> </text>
    </staticText>
    <staticText>
      <reportElement x="414" y="3" width="121" height="15" />
      <textElement textAlignment="Center"
        verticalAlignment="Middle">
        <font isBold="true" />
      </textElement>
      <text><![CDATA[Country]]></text>
    </staticText>
    <staticText>
      <reportElement x="0" y="3" width="136" height="15" />
      <textElement textAlignment="Center"
        verticalAlignment="Middle">
        <font isBold="true" />
      </textElement>
      <text><![CDATA[Name]]></text>
    </staticText>
  </band>
</columnHeader>
<detail>
  <band height="16">

```

```

<staticText>
  <reportElement mode="Opaque" x="0" y="0" width="535"
    height="14" backcolor="#E5ECF9" />
  <box>
    <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
  </box>
  <textElement />
  <text><![CDATA[]]> </text>
</staticText>
<textField>
  <reportElement x="414" y="0" width="121" height="15" />
  <textElement textAlignment="Center"
    verticalAlignment="Middle">
    <font size="9" />
  </textElement>
  <textFieldExpression class="java.lang.String">
    <![CDATA[{$F{country}}]>
  </textFieldExpression>
</textField>
<textField>
  <reportElement x="0" y="0" width="136" height="15" />
  <textElement textAlignment="Center"
    verticalAlignment="Middle" />
  <textFieldExpression class="java.lang.String">
    <![CDATA[{$F{name}}]>
  </textFieldExpression>
</textField>
</band>
</detail>
</jasperReport>

```

Here are the details of main fields in the above report template:

- `<queryString>`: This is empty (as we are passing data through Java Beans). Usually contains the SQL statement, which retrieves the report result.

- `<field name>`: This element is used to map data from data sources or queries, into report templates. **name** is re-used in the report body and is case-sensitive.
- `<fieldDescription>`: This element maps the field name with the appropriate element in the XML file.
- `<staticText>`: This defines the static text that does not depend on any data sources, variables, parameters, or report expressions.
- `<textFieldExpression>`: This defines the appearance of the result field.
- `#{country}`: This is a variable that contains the value of result, predefined field in the tag `<field name>`.
- `<band>`: Bands contain the data, which is displayed in the report.

Once the report design is ready, save it in C:\ directory.

Previewing the XML Report Template

There is a utility *net.sf.jasperreports.view.JasperDesignViewer* available in JasperReports JAR file, which helps in previewing the report design without having to compile or fill it. This utility is a standalone Java application, hence can be executed using ANT.

Let's write an ANT target **viewDesignXML** to view the JRXML. So, let's create and save **build.xml** under C:\tools\jasperreports-5.0.1\test directory (should be placed in the same directory where JRXML is placed). Here is the build.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewDesignXML" basedir=".">

  <import file="baseBuild.xml" />
  <target name="viewDesignXML" description="Design viewer is launched
    to preview the JXML report design.">
    <java classname="net.sf.jasperreports.view.JasperDesignViewer"
      fork="true">
      <arg value="-XML" />
      <arg value="-F${file.name}.jrxml" />
      <classpath refid="classpath" />
    </java>
  </target>

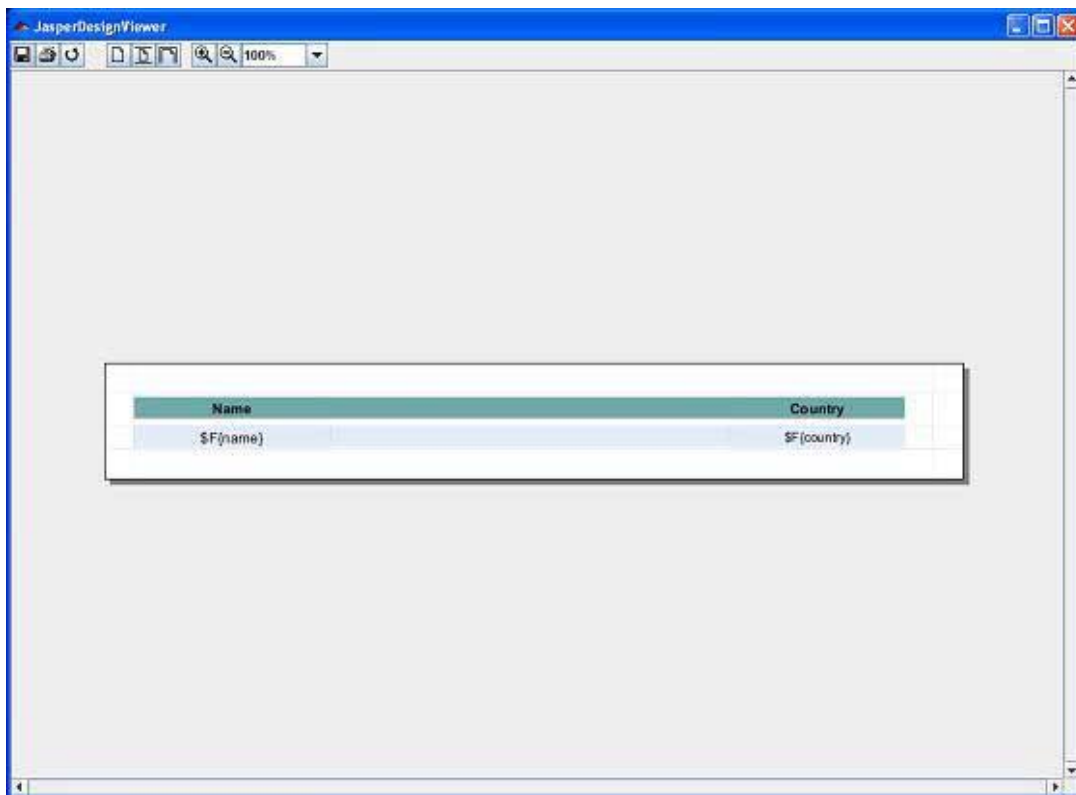
</project>
```

Next, let's open a command prompt and go to the directory where build.xml is placed. Execute the command **ant** (As the viewDesignXML is the default target). Output is follows:

```
C:\tools\jasperreports-5.0.1\test>ant
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml

viewDesignXML:
[java] log4j:WARN No appenders could be found for logger
(net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
[java] log4j:WARN Please initialize the log4j system properly.
```

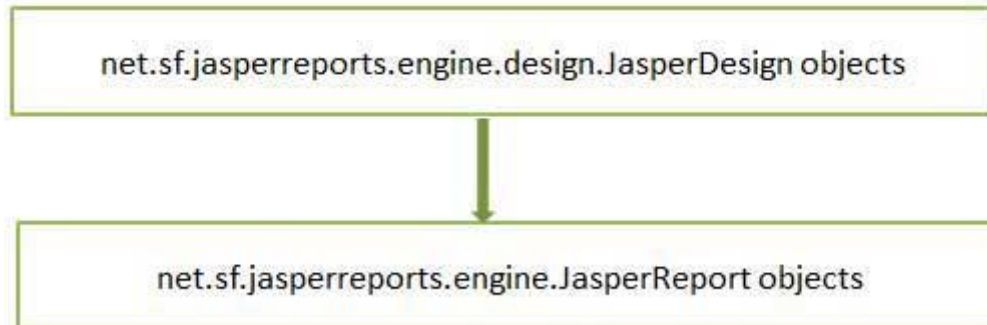
Log4j warning can be ignored, and as a result of the above execution, a window labeled "JasperDesignViewer" opens, displaying our report template preview.



As we see, only report expressions for obtaining the data are displayed, as JasperDesignViewer doesn't have access to the actual data source or report parameters. Terminate the JasperDesignViewer by closing the window or by hitting Ctrl-c in the command-line window.

5. JASPERREPORTS - COMPILING REPORT DESIGN

We have generated the JasperReport template (JRXML file) in the previous chapter. This file cannot be used directly to generate reports. It has to be compiled to JasperReport' native binary format, called **Jasper** file. On compiling, we transform JasperDesign object into JasperReport object:



Interface *net.sf.jasperreports.engine.design.JRCompiler* plays a central role during compilation. This interface has several implementations depending on the language used for report expressions, which can be written in Java, Groovy, JavaScript, or any other scripting language as long as compiler implementation can evaluate it at runtime.

We can compile JRXML file in the following two ways:

- Programmatic compilation.
- Compilation through ANT task.

Programmatic Compilation of JRXML

JasperReports API offers a facade class *net.sf.jasperreports.engine.JasperCompileManager* for compiling a JasperReport. This class consists of several public static methods for compiling report templates. The source of templates can be files, input streams and/or memory objects.

The contents of the JRXML file (*jasper_report_template.jrxml*) are as follows. It is saved at directory **C:\tools\jasperreports-5.0.1\test**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<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="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">

    <queryString>
        <![CDATA[]]>
    </queryString>
    <field name="country" class="java.lang.String">
        <fieldDescription><![CDATA[country]]></fieldDescription>
    </field>
    <field name="name" class="java.lang.String">
        <fieldDescription><![CDATA[name]]></fieldDescription>
    </field>
    <columnHeader>
        <band height="23">
            <staticText>
                <reportElement mode="Opaque" x="0" y="3" width="535"
                    height="15" backcolor="#70A9A9" />
                <box>
                    <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
                </box>
                <textElement />
                <text><![CDATA[]]> </text>
            </staticText>
            <staticText>
                <reportElement x="414" y="3" width="121" height="15" />
                <textElement textAlignment="Center"
                    verticalAlignment="Middle">
                    <font isBold="true" />
                </textElement>
                <text><![CDATA[Country]]></text>
            </staticText>

```

```

<staticText>
  <reportElement x="0" y="3" width="136" height="15" />
  <textElement textAlignment="Center"
    verticalAlignment="Middle">
    <font isBold="true" />
  </textElement>
  <text><![CDATA[Name]]></text>
</staticText>
</band>
</columnHeader>
<detail>
  <band height="16">
    <staticText>
      <reportElement mode="Opaque" x="0" y="0" width="535"
        height="14" backcolor="#E5ECF9" />
      <box>
        <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
      </box>
      <textElement />
      <text><![CDATA[]]> </text>
    </staticText>
    <textField>
      <reportElement x="414" y="0" width="121" height="15" />
      <textElement textAlignment="Center"
        verticalAlignment="Middle">
        <font size="9" />
      </textElement>
      <textFieldExpression class="java.lang.String">
        <![CDATA[#{country}]]>
      </textFieldExpression>
    </textField>
    <textField>
      <reportElement x="0" y="0" width="136" height="15" />

```

```

        <textElement textAlignment="Center"
            verticalAlignment="Middle" />
        <textFieldExpression class="java.lang.String">
            <![CDATA[ $F$ {name}]]>
        </textFieldExpression>
    </textField>
</band>
</detail>
</jasperReport>

```

The following code demonstrates compilation of the above *jasper_report_template.jrxml* file.

```

package com.tutorialspoint;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;

public class JasperReportCompile {

    public static void main(String[] args) {
        String sourceFileName = "C://tools/jasperreports-5.0.1/test" +
            "/jasper_report_template.jrxml";

        System.out.println("Compiling Report Design ...");
        try {
            /**
             * Compile the report to a file name same as
             * the JRXML file name
             */
            JasperCompileManager.compileReportToFile(sourceFileName);
        } catch (JRException e) {
            e.printStackTrace();
        }
        System.out.println("Done compiling!!! ...");
    }
}

```

```

    }
}

```

Template Compilation

As next step, let's save the above content in the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportCompile.java** and import the *baseBuild.xml* in the build.xml file as below. The baseBuild.xml already has the **compile** and **run** targets:

```

<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="run" basedir=".">

    <import file="baseBuild.xml"/>

</project>

```

Next, let's open command line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportCompile** as:

```

C:\tools\jasperreports-5.0.1\test>ant -Dmain-
class=com.tutorialspoint.JasperReportCompile
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml
compile:
[javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:27:
warning: 'includeantruntime' was not set, defaulting to
build.sysclasspath=last;set to false for repeatable builds
[javac] Compiling 1 source file to C:\tools\jasperreports-5.0.1\test\classes

run:
[echo] Runnin class : com.tutorialspoint.JasperReportCompile
[java] Compiling Report Design ...
[java] log4j:WARN No appenders could be found for logger
(net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
[java] log4j:WARN Please initialize the log4j system properly.
[java] Done compiling!!! ...

```

```
BUILD SUCCESSFUL
Total time: 8 seconds
```

As a result of the above compilation, you will see that template file *jasper_report_template.jasper* got generated in C:\tools\jasperreports-5.0.1\test directory.

Preview Compiled Report Template

The *net.sf.jasperreports.view.JasperDesignViewer* can be used to preview compiled report templates and JRXML templates.

To move further, let's add a new target **viewDesign** to the above build.xml file, which will allow us to preview the compiled report. Below is the revised build.xml:

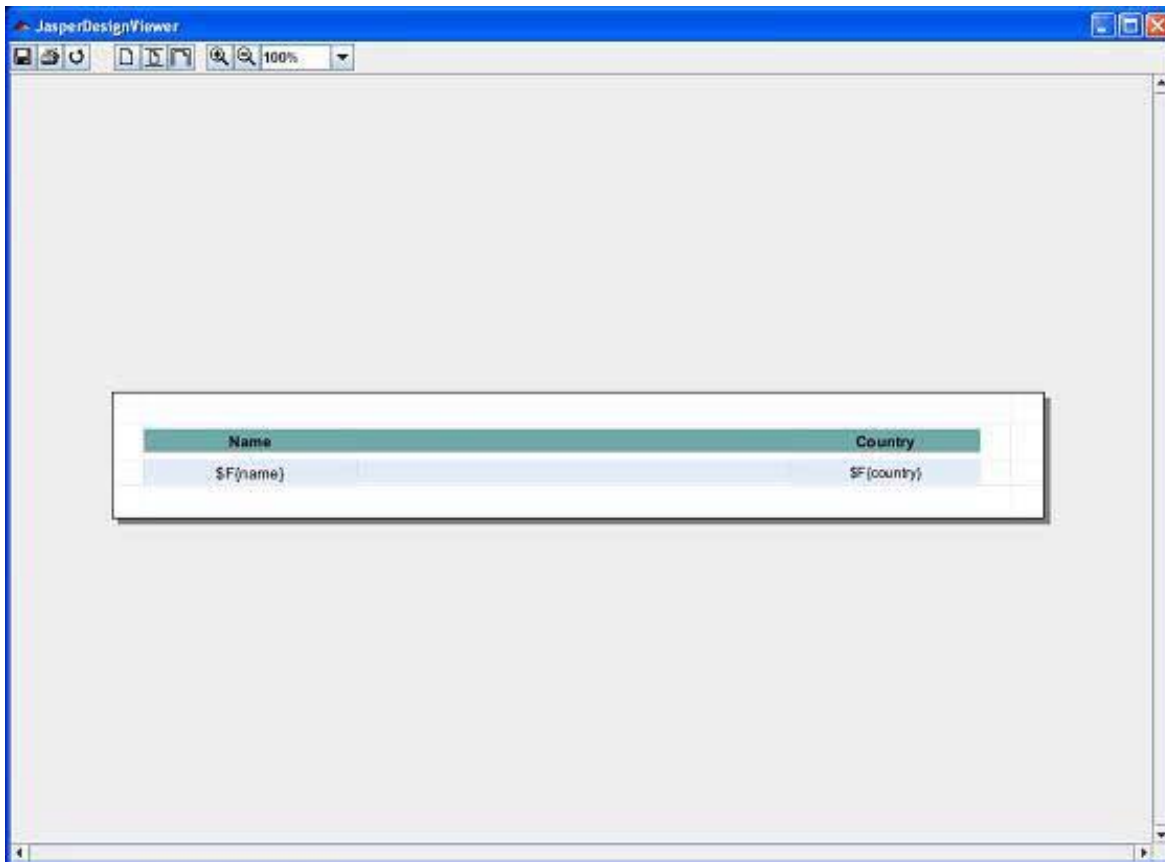
The import file - baseBuild.xml is picked from chapter **Environment Setup** and should be placed in the same directory as the build.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewDesign" basedir=".">

    <import file="baseBuild.xml" />
    <target name="viewDesign" description="Design viewer is launched
        to preview the compiled report design.">
        <java classname="net.sf.jasperreports.view.JasperDesignViewer"
            fork="true">
            <arg value="-F${file.name}.jasper" />
            <classpath refid="classpath" />
        </java>
    </target>

</project>
```

Let's execute the command: **ant** (viewDesign is the default target) at command prompt. JasperDesignViewer window opens up displaying the Jasper file as below:



Compilation through ANT Task

As report template compilation is more like a design time job than a runtime job, JasperReport library has a custom ANT task. For certain situations, when JRXML file is created at runtime, we can't use this ANT task. The custom ANT task is called JRC and is implemented by the class: *net.sf.jasperreports.ant.JRAntCompileTask*. Its syntax and behavior are very similar to the built-in **<javac>** ANT task.

Template Compilation

Let's add new target **compilereportdesing** to our existing build.xml. Here, the source folder is specified using a nested **<src>** tag with the filesets. The nested source tag allows compiling report templates that are scattered through many different locations and are not grouped under a single root report source folder. Below is the revised build.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="compilereportdesing" basedir=".">
  <import file="baseBuild.xml" />
  <target name="viewDesign" description="Design viewer is launched
    to preview the compiled report design.">
```

29

```

    <java classname="net.sf.jasperreports.view.JasperDesignViewer"
        fork="true">
    <arg value="-F${file.name}.jasper" />
    <classpath refid="classpath" />
    </java>
</target>

<target name="compilereportdesing" description="Compiles the JXML
    file and produces the .jasper file.">
    <taskdef name="jrc"
        classname="net.sf.jasperreports.ant.JRAntCompileTask">
        <classpath refid="classpath" />
    </taskdef>
    <jrc destdir=".">
        <src>
            <fileset dir=".">
                <include name="*.jrxml" />
            </fileset>
        </src>
        <classpath refid="classpath" />
    </jrc>
</target>

</project>

```

Next, let's open command prompt and go to the directory where build.xml is placed. Execute the command **ant** (compilereportdesing is the default target); Output is as follows:

```

C:\tools\jasperreports-5.0.1\test>ant
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml

compilereportdesing:
    [jrc] Compiling 1 report design files.
    [jrc] log4j:WARN No appenders could be found for logger
    (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).

```

```
[jrc] log4j:WARN Please initialize the log4j system properly.  
[jrc] log4j:WARN See  
http://logging.apache.org/log4j/1.2/faq.html#noconfig  
for more info.  
[jrc] File :  
C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrxml ... OK.
```

BUILD SUCCESSFUL

Total time: 5 seconds

File *jasper_report_template.jasper* is generated in the file system (in our case C:\tools\jasperreports-5.0.1\test directory). This file is identical to the file generated programmatically by calling the `net.sf.jasperreports.engine.JasperCompileManager.compileReportToFile()`. We can preview this jasper file, executing **ant viewDesign**.

6. JASPERREPORTS – FILLING REPORTS

The main purpose of any reporting tool is to produce high quality documents. Report filling process helps reporting tool to achieve this by manipulating sets of data.

The main inputs required for report-filling process are:

- **Report Template:** This is actual JasperReport file.
- **Report Parameters:** These are basically named values that are passed at the report filling time to the engine. We will discuss them in **Report Parameter** chapter.
- **Data Source:** We can fill a Jasper file from a range of data sources like an SQL query, an XML file, a csv file, an HQL (Hibernate Query Language) query, a collection of Java Beans, etc. This will be discussed in detail in **Report Data Sources** chapter.

The output generated by this process is a **.jrprint** document which is ready to be viewed, printed, or exported to other formats. The facade class *net.sf.jasperreports.engine.JasperFillManager* is usually used for filling a report template with data. This class has various *fillReportXXX()* methods that fill report templates (templates could be located on disk, picked from input streams, or are supplied directly as in-memory).

There are two categories of *fillReportXXX()* methods in this facade class:

1. The first type, receive a `java.sql.Connection` object as the third parameter. Most of the times, reports are filled with data from a relational database. This is achieved by:
 - Connect to the database through JDBC.
 - Include an SQL query inside the report template.
 - JasperReports engine uses the connection passed in and executes the SQL query.
 - A report data source is thus produced for filling the report.
2. The second type, receive a `net.sf.jasperreports.engine.JRDataSource` object, when the data that need to be filled is available in other forms.

Filling Report Templates

Let's write a report template. The contents of the JRXML file (C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrxml) are as below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
```

```

<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="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20">

  <queryString>
    <![CDATA[]]>
  </queryString>
  <field name="country" class="java.lang.String">
    <fieldDescription><![CDATA[country]]></fieldDescription>
  </field>
  <field name="name" class="java.lang.String">
    <fieldDescription><![CDATA[name]]></fieldDescription>
  </field>
  <columnHeader>
    <band height="23">
      <staticText>
        <reportElement mode="Opaque" x="0" y="3" width="535"
          height="15" bgcolor="#70A9A9" />
        <box>
          <bottomPen lineWidth="1.0" lineColor="#CCCCCC" />
        </box>
        <textElement />
        <text><![CDATA[]]> </text>
      </staticText>
      <staticText>
        <reportElement x="414" y="3" width="121" height="15" />
        <textElement textAlignment="Center"
          verticalAlignment="Middle">
          <font isBold="true" />

```

```

    </textElement>
    <text><![CDATA[Country]]></text>
</staticText>
<staticText>
    <reportElement x="0" y="3" width="136" height="15" />
    <textElement textAlignment="Center"
        verticalAlignment="Middle">
        <font isBold="true" />
    </textElement>
    <text><![CDATA[Name]]></text>
</staticText>
</band>
</columnHeader>
<detail>
    <band height="16">
    <staticText>
        <reportElement mode="Opaque" x="0" y="0" width="535"
            height="14" bgcolor="#E5ECF9" />
        <box>
            <bottomPen lineWidth="0.25" lineColor="#CCCCCC" />
        </box>
        <textElement />
        <text><![CDATA[]]> </text>
    </staticText>
    <textField>
        <reportElement x="414" y="0" width="121" height="15" />
        <textElement textAlignment="Center"
            verticalAlignment="Middle">
            <font size="9" />
        </textElement>
        <textFieldExpression class="java.lang.String">
            <![CDATA[#{country}]]>
        </textFieldExpression>
    </textField>
    </band>
</detail>
</table>

```

```

</textField>
<textField>
  <reportElement x="0" y="0" width="136" height="15" />
  <textElement textAlignment="Center"
    verticalAlignment="Middle" />
  <textFieldExpression class="java.lang.String">
    <![CDATA[ $\$F\{name\}$ ]]>
  </textFieldExpression>
</textField>
</band>
</detail>
</jasperReport>

```

Next, let's pass a collection of Java data objects (Java beans), to the JasperReports Engine, to fill this compiled report.

Write a POJO DataBean.java, which represents the data object (Java bean). This class defines two String objects i.e. 'name' and 'country.' Save it to the directory **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint**.

```

package com.tutorialspoint;

public class DataBean {
    private String name;
    private String country;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCountry() {
        return country;
    }
}

```

```

    }

    public void setCountry(String country) {
        this.country = country;
    }
}

```

Write a class `DataBeanList.java`, which has business logic to generate a collection of java bean objects. This is further passed to the JasperReports engine, to generate the report. Here we are adding 4 `DataBean` objects in the List. Save it to the directory **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint**.

```

package com.tutorialspoint;

import java.util.ArrayList;

public class DataBeanList {
    public ArrayList<DataBean> getDataBeanList() {
        ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

        dataBeanList.add(produce("Manisha", "India"));
        dataBeanList.add(produce("Dennis Ritchie", "USA"));
        dataBeanList.add(produce("V.Anand", "India"));
        dataBeanList.add(produce("Shrinath", "California"));

        return dataBeanList;
    }

    /**
     * This method returns a DataBean object,
     * with name and country set in it.
     */
    private DataBean produce(String name, String country) {
        DataBean dataBean = new DataBean();
        dataBean.setName(name);
    }
}

```

```

        dataBean.setCountry(country);
        return dataBean;
    }
}

```

Write a main class file **JasperReportFill.java**, which gets the java bean collection from the class (DataBeanList) and passes it to the JasperReports engine, to fill the report template. Save it to the directory **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint**.

```

package com.tutorialspoint;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class JasperReportFill {
    @SuppressWarnings("unchecked")
    public static void main(String[] args) {
        String sourceFileName =
            "c://tools/jasperreports-5.0.1/test/jasper_report_template.jasper";
        DataBeanList DataBeanList = new DataBeanList();
        ArrayList<DataBean> dataList = DataBeanList.getDataBeanList();

        JRBeanCollectionDataSource beanColDataSource =
            new JRBeanCollectionDataSource(dataList);

        Map parameters = new HashMap();
        try {

```

```

        JasperFillManager.fillReportToFile(
            sourceFileName,
            parameters,
            beanColDataSource);
    } catch (JRException e) {
        e.printStackTrace();
    }
}
}
}

```

Generating Reports

We will now compile and execute these files using our regular ANT build process. The build.xml file is as given below:

The import file - baseBuild.xml is picked from chapter **Environment Setup** and should be placed in the same directory as the build.xml.

```

<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="executereport" basedir=".">
    <import file="baseBuild.xml"/>

    <target name="executereport" depends="compile,compilereportdesing,run">
        <echo message="Im here"/>
    </target>

    <target name="compilereportdesing"
        description="Compiles the JXML file and
        produces the .jasper file.">
        <taskdef name="jrc"
            classname="net.sf.jasperreports.ant.JRAntCompileTask">
            <classpath refid="classpath" />
        </taskdef>
        <jrc destdir=".">
            <src>
                <fileset dir=".">

```

```

        <include name="*.jrxml" />
    </fileset>
</src>
    <classpath refid="classpath" />
</jrc>
</target>
</project>

```

Next, let's open command line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill (executereport** is the default target) as follows:

```

C:\tools\jasperreports-5.0.1\test>ant -Dmain-
class=com.tutorialspoint.JasperReportFill
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml

compile:
  [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:27:
  warning: 'includeantruntime' was not set, defaulting to
  build.sysclasspath=last; set to false for repeatable builds
  [javac] Compiling 1 source file to
  C:\tools\jasperreports-5.0.1\test\classes

run:
  [echo] Runnin class : com.tutorialspoint.JasperReportFill
  [java] log4j:WARN No appenders could be found for logger
  (net.sf.jasperreports.extensions.ExtensionsEnvironment).
  [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 8 seconds

```


End of ebook preview
If you liked what you saw...
Buy it from our store @ <https://store.tutorialspoint.com>