JasperReports - Quick Guide

Previous Page Next Page

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 −

S.NO	Element and Description

1	**title** Title contains the 'Title' of the report. It appears only once at the very beginning of the report, for example, "Tutorials Point Report."
2	**pageHeader** PageHeader may contain date and time information and/or organization name. This appears at the top of each page.
3	**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.
4	**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."
5	**columnFooter** ColumnFooter may display summation of any of the field, for example, "Total Hours Worked: "180."
6	**pageFooter** PageFooter may contain page count information. It appears at the bottom of each page, for example, "1/23."
7	**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, 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.

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.

JasperReports - Environment Setup

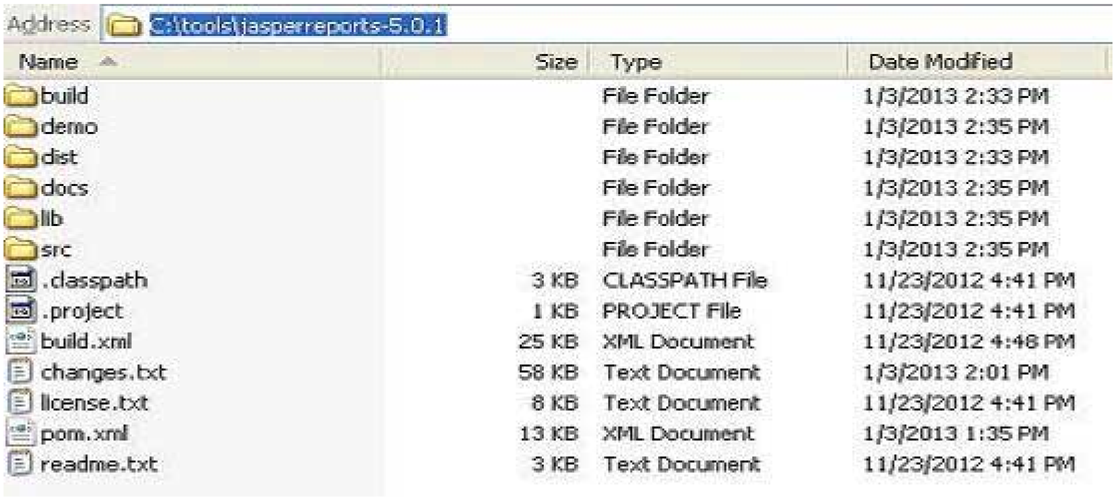
JasperReports is a pure Java library and not a standalone application. It cannot run on its own, hence it needs to be embedded into another client or server-side Java application. As it is Java based, it can be run on any platform that supports Java (JDK 1.3 and above). All the JasperReport's functionalities 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 −



Address C:\tools\jasperreports-5.0.1

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>

   <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

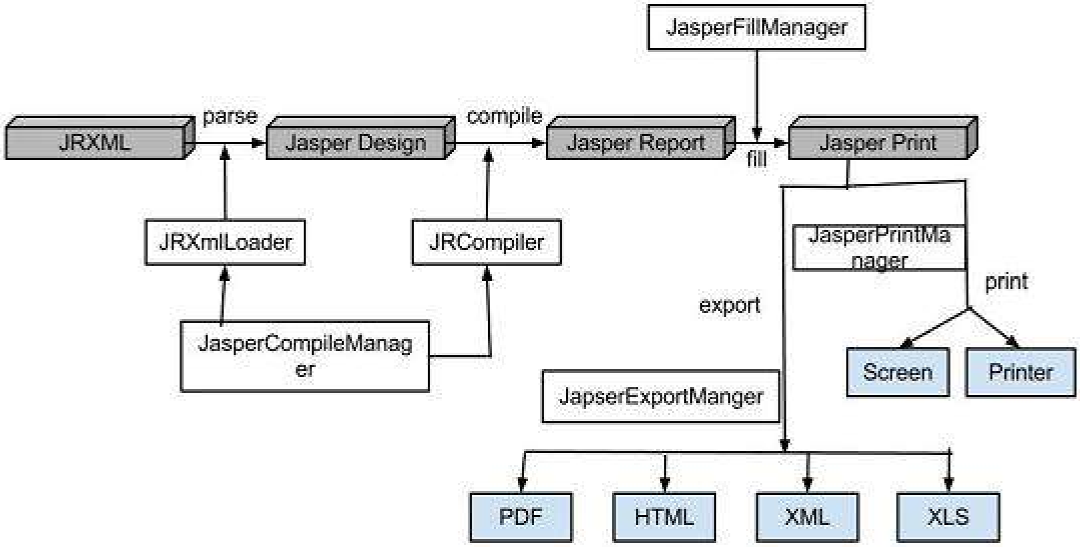
src.dir − is C:\tools\jasperreports-5.0.1\test\src *classes.dir* − is C:\tools\jasperreports-5.0.1\test\classes

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.

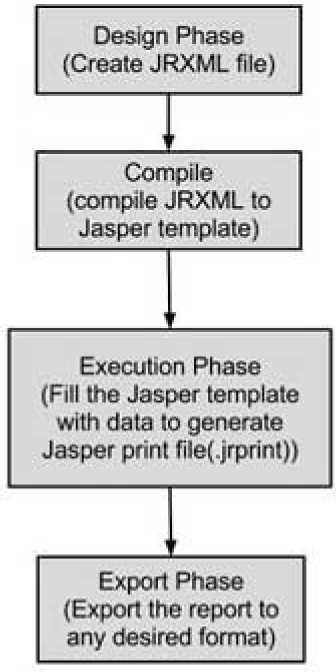
net.sf.jasperreports.view.JasperDesignViewer − Used at design time to preview the report templates.

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.

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 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.

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" 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[$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 datasources, variables, parameters, or report expressions.

<textFieldExpression> − This defines the appearance of the result field.

$F{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.

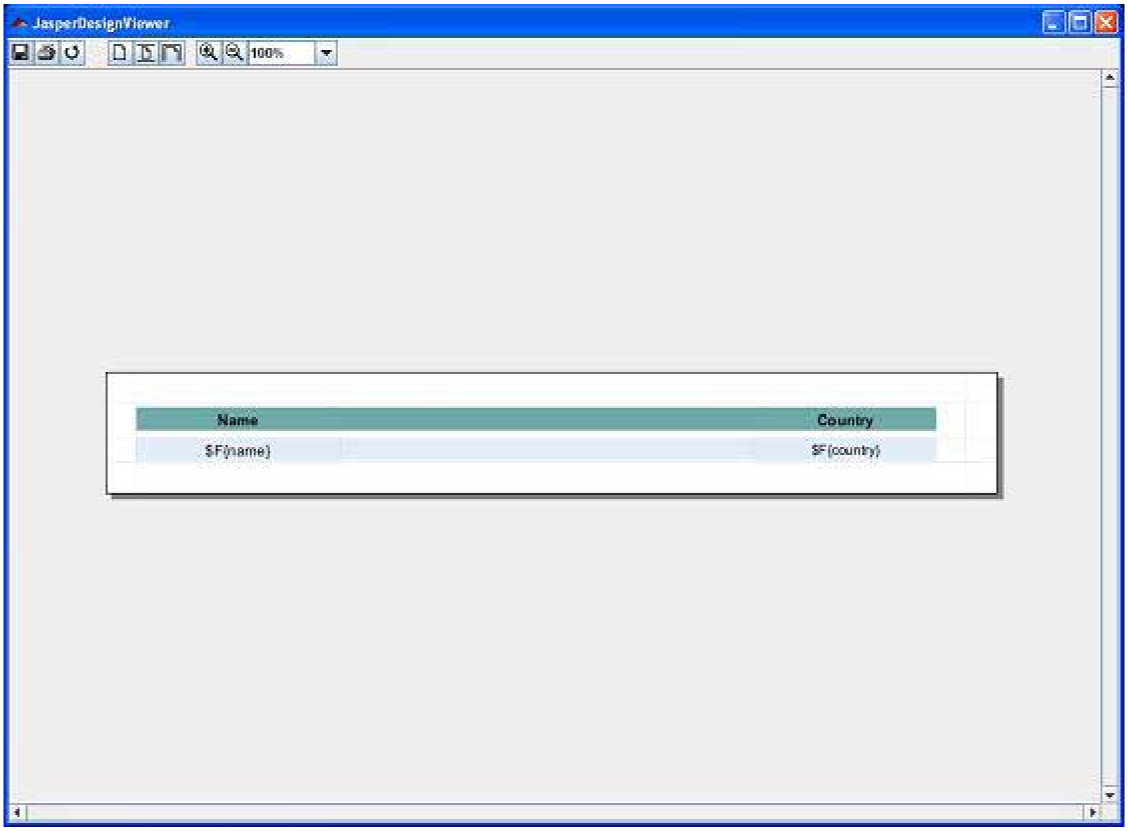
Let's write an ANT target **viewDesignXML** to view the JRXML. So, let's create and save **build.xml** under C:\tools\jasperreports-5.0.1\test directory (should be placed in the same directory where JRXML is placed). Here is the build.xml file –

```
<?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 above execution, a window labeled "JasperDesignViewer" opens, displaying our report template preview.

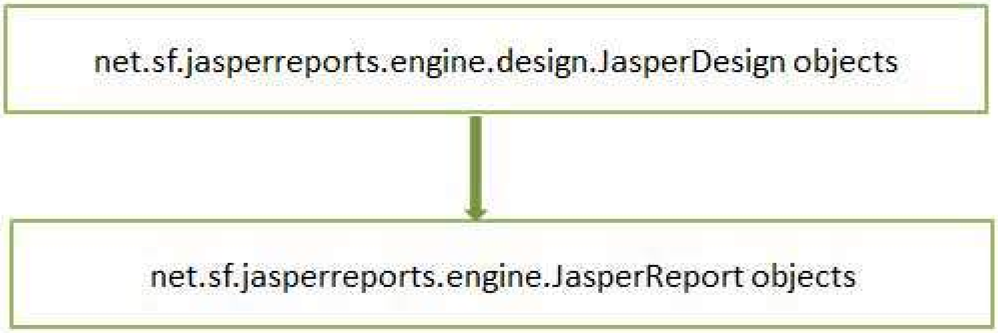


Name	Country
$F{name}	$F{country}

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.

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 −


net.sf.jasperreports.engine.design.JasperDesign objects
net.sf.jasperreports.engine.JasperReport objects

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[$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>
```

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 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 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

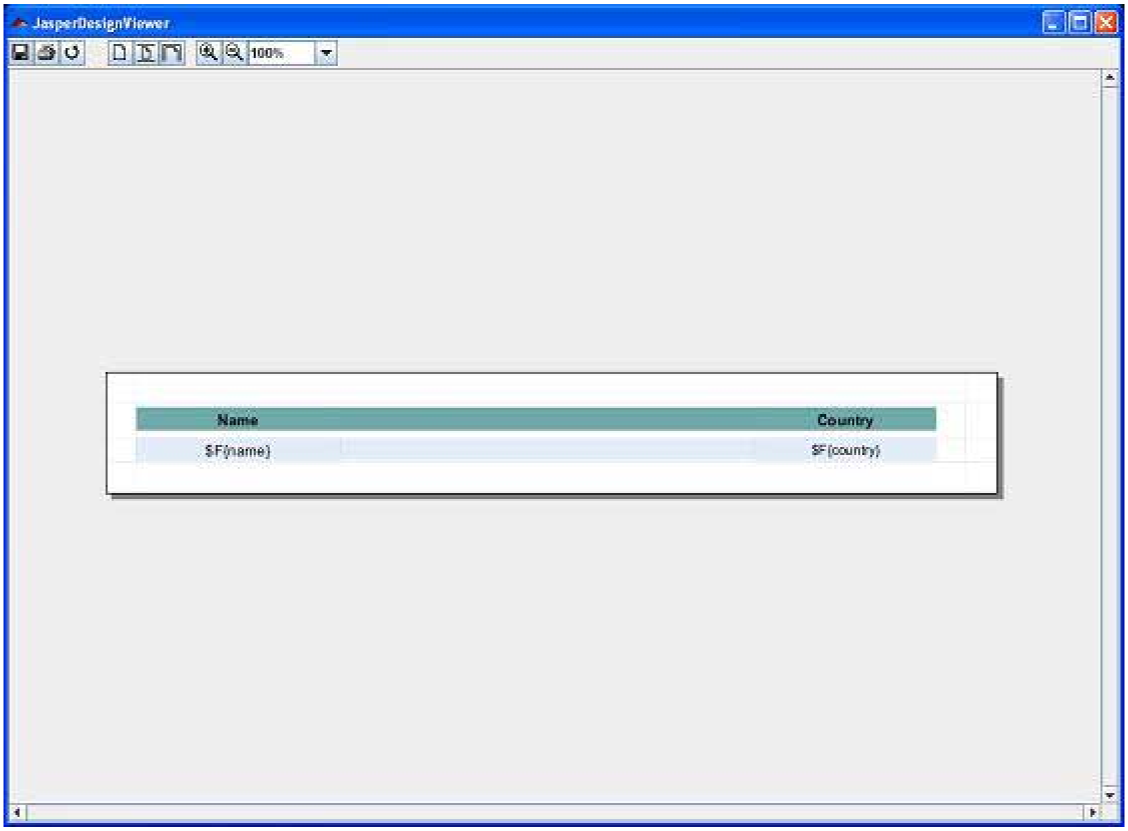
The *net.sf.jasperreports.view.JasperDesignViewer* can be used to preview compiled report templates and JRXML templates.

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.">

      <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**.

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 datasources 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 −

- 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.
- 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" 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[$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>
```

Next, let's pass a collection of Java data objects (Java beans), to the JasperReport 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
```

As a result of above execution, a file *jasper_report_template.jrprint* is generated in the same directory as the *.jasper* file (In this case, it is generated at C:\tools\jasperreports-5.0.1\test).

Jasper Report - View & Print Reports

The output of the report filling process *JasperPrint objects* can be viewed using a built-in viewer component, or printed, or exported to more popular document formats like PDF, HTML, RTF, XLS, ODT, CSV, or XML. Viewing and printing of the Jasper documents will be discussed in this chapter and exporting will be discussed in the next chapter i.e. 'Export Reports.'

Viewing Reports

JasperReport provides a built-in viewer for viewing the generated reports in its original format. It is a swing based component and other Java applications can integrate this component without having to export the documents to other formats in order to be viewed or printed. The *net.sf.jasperreports.view.JRViewer* class represents this visual component. This class can also be customized as per the application needs, by sub classing it.

JasperReports also has a Swing application, which uses the visual component for viewing the reports. This application helps to view reports in the same format as *.jrprint is produced. This Swing application is implemented in the class *net.sf.jasperreports.view.JasperViewer*. To view reports using this class, we need to wrap it into an ANT target.

Viewing the generated Report

The following example demonstrates – how to view a report using the JasperViewer class –

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 given 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" 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[$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>
```

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**.

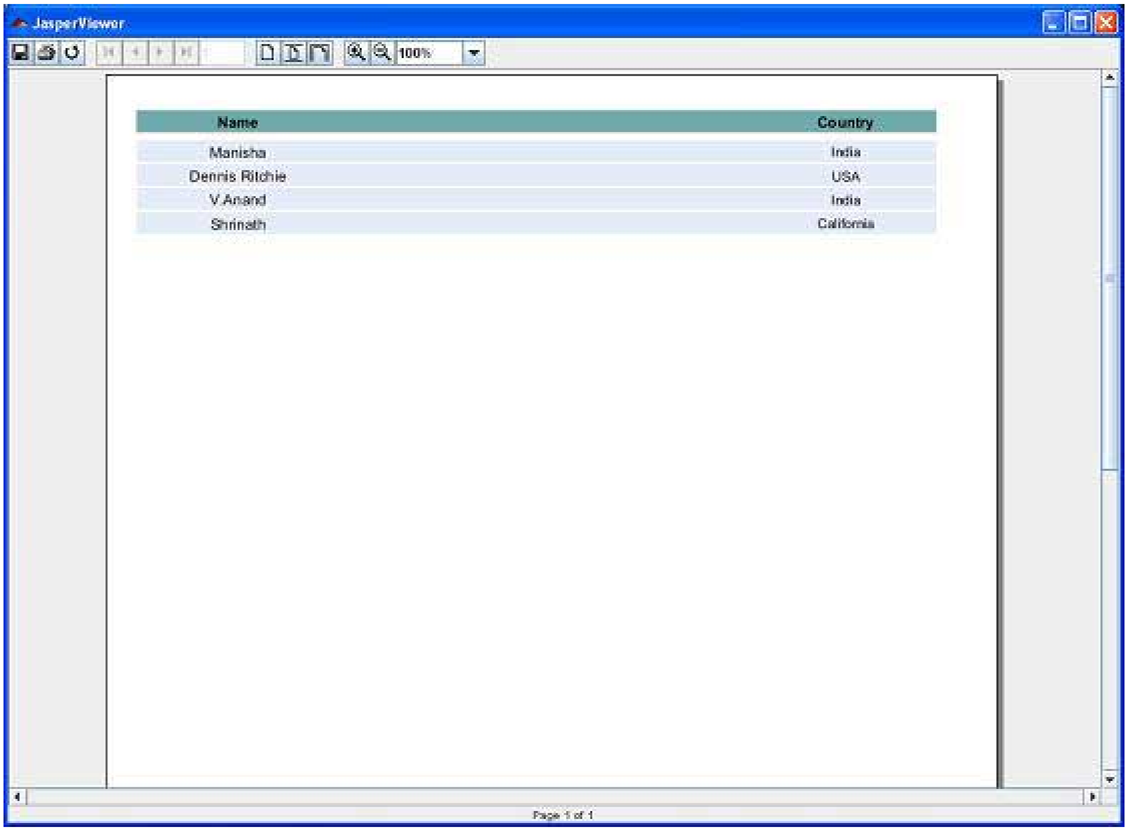
```
package com.tutorialspoint;

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();
      }
   }
}
```

Let's write a target **viewFillReport** to the build.xml file. The build.xml file is as follows −

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 = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml"/>

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer
      to preview the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFillReport is the default target). As a result, we see a JasperViewer window as shown in the screen given below –



Name	Country
Manisha	India
Dennis Ritchie	USA
V.Anand	India
Shrinath	California

Printing Reports

We can print the documents generated by the JasperReports library (in their proprietary format i.e. *JasperPrint* objects) using the *net.sf.jasperreports.engine.JasperPrintManager* class. This is a facade class that relies on the Java 2 Printing API. We can also print the documents once the JasperReport documents are exported to other formats such as HTML or PDF.

Printing the Generated Report

The following code demonstrates the printing of a report. Let's update our existing class JasperReportFill. We will use *JasperPrintManager.printReport()* method. This method takes source file name (here we pass the *.jrprint* file, which we generate in the previous step using the method JasperFillManager.fillReportToFile()) as first parameter. The second parameter is the boolean for displaying the standard print dialog (we have set it to **true** here).

```
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.JasperPrintManager;
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";
      String printFileName = null;
      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList();

      JRBeanCollectionDataSource beanColDataSource = new
         JRBeanCollectionDataSource(dataList);

      Map parameters = new HashMap();
      try {
           printFileName = JasperFillManager.fillReportToFile(
            sourceFileName, parameters, beanColDataSource);
         if(printFileName != null){
            JasperPrintManager.printReport( printFileName, true);
         }
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

Now, let's save this file to the directory **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint**. We will compile and execute this file using ANT. The contents of build.xml are as given below –

```
<?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 prompt and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportPrint**. As a result, a print dialog box appears. Click ok to print the document.

JasperReports - Exporting Reports

We have seen in the previous chapter, how to print and view a JasperReport generated document. Here, we shall see how to transform or export these reports into other formats such as PDF, HTML, and XLS. Facade class *net.sf.jasperreports.engine.JasperExportManager* is provided to achieve this functionality. Exporting means transforming the *JasperPrint* object (.jrprint file) into different format.

The following code (JasperReportExport.java) demonstrates the exporting process of the JasperReport document. The JasperExportManager provides methods to export a report into PDF, HTML, and XML only. To export to the XLS format, we have used the class *net.sf.jasperreports.engine.export.JRXlsExporter*. This code generates following three files –

sample_report.pdf

sample_report.html

sample_report.xls

Exporting to Other Formats

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" 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[$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>
```

Next, contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as given below –

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as given below −

```
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.JRExporterParameter;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
import net.sf.jasperreports.engine.export.JRXlsExporter;

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

      Map parameters = new HashMap();
      try {
         printFileName = JasperFillManager.fillReportToFile(sourceFileName,
            parameters, beanColDataSource);
         if (printFileName != null) {
            /**
             * 1- export to PDF
             */
            JasperExportManager.exportReportToPdfFile(printFileName,
               "C://sample_report.pdf");

            /**
             * 2- export to HTML
             */
            JasperExportManager.exportReportToHtmlFile(printFileName,
               "C://sample_report.html");

            /**
             * 3- export to Excel sheet
             */
            JRXlsExporter exporter = new JRXlsExporter();

            exporter.setParameter(JRExporterParameter.INPUT_FILE_NAME,
               printFileName);
            exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME,
               "C://sample_report.xls");

            exporter.exportReport();
         }
      } catch (JRException e) {
         e.printStackTrace();
      }
   }
}
```

Here, we have included the logic to export the jasper print file to pdf, html and xls format.

Generating Reports

Let's compile and execute above files using our regular ANT build process. The build.xml file is as given below −

```
<?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>
```

Go to the command prompt and then go to the directory C:\tools\jasperreports-5.0.1\test, where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill**. The output is 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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting t
   [javac] Compiling 4 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.

executereport:
   [echo] Im here

BUILD SUCCESSFUL
Total time: 32 seconds
```

As the result of above execution, you will find three files sample_report.pdf, sample_report.html, sample_report.xls generated in the C:\ directory.

Reports Parameters

The main input for filling a report are − report template, parameters, and data sources. This chapter will describe the parameters and in the next chapter we will discuss the data sources.

Parameters are the object references, those are passed during report-filling operations to the report engine. The data which cannot be passed through the data source, can be passed by using parameters. Data like author name, title of the report, etc. can be passed through parameters. A JasperReports template or JRXML template can have zero or more parameter elements.

Parameter Declaration

Parameter declaration as follows −

```
<parameter name = "exampleParameter" class = "java.lang.String" />
```

The Name Attribute

The *name* attribute of the <parameter> element is mandatory. It references the parameter in report expressions by name. Parameter name should be a single word. It should not contain any special characters like dot or comma.

The Class Attribute

The *class* attribute is also mandatory and it specifies the class name for the parameter values. Its default value is *java.lang.String*. This can be changed to any class available at runtime. Irrespective of the type of a report parameter, the engine takes care of casting in the report expressions in which the $P{} token is used, hence making the manual casts is unnecessary.

The report parameter values are always packed in a java.util.Map object, which has the parameter name as its key. Report parameters can be used in the query string of the report, so as to further customize the data set, retrieved from the database. These act like dynamic filters in the query that supplies data for the report.

Built-in Parameters

Following are the pre-defined report parameters, ready to use in the expressions −

S.NO	Parameter Name and Description
1	**REPORT_PARAMETERS_MAP** Contains a map with all user defined and built-in parameters.
2	**REPORT_CONNECTION** This points to the user supplied class java.sql.Connection, used for JDBC datasources.
3	**REPORT_DATA_SOURCE** This is a user supplied instance of JRDataSource representing either one of the built-in data source types or a user-defined one.
4	**REPORT_MAX_COUNT** This is a *java.lang.Integer* value, allowing the users to limit the records from datasource.
5	**REPORT_SCRIPTLET** This points to *net.sf.jasperreports.engine.JRAbstractScriptlet* and contains an instance of the report scriptlet provided by the user.
6	**REPORT_LOCALE** This a *java.util.Locale* instance, containing the resource bundle desired locale.
7	**REPORT_RESOURCE_BUNDLE**

	This points to *java.util.ResourceBundle* object and contains localized messages.
8	**REPORT_TIME_ZONE** This is a *java.util.TimeZone* instance, used for the date formatting.
9	**REPORT_VIRTUALIZER** This is an instance of *net.sf.jasperreports.engine.JRVirtualizer* object, and used for the page virtualization (optimize memory consumption).
10	**REPORT_CLASS_LOADER** This is a *java.lang.ClassLoader* instance to be used during the report filling process to load resources such as images, fonts, and subreport templates
11	**IS_IGNORE_PAGINATION** If set to *java.lang.Boolean.TRUE* the report will be generated on one long page and page break will not occur.

Example

Let us pass *ReportTitle* and *Author* to the report (generated by JasperReportFill.java). Revised file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** is as follows −

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as below −

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as given below −

```
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;
   }
}
```

Let's add parameters the <**ReportTitle**> and <**Author**> to our existing report template (Chapter Report Designs). The Report Title and Author will be displayed at the beginning of the report. The revised report template (jasper_report_template.jrxml) is as follows. Save it to C:\tools\jasperreports-5.0.1\test directory −

```
<?xml version = "1.0"?>
<!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" pageWidth = "595"
   pageHeight = "842" columnWidth = "515"
   leftMargin = "40" rightMargin = "40" topMargin = "50" bottomMargin = "50">

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

   <title>
      <band height = "70">

         <line>
            <reportElement x = "0" y = "0" width = "515" height = "1"/>
         </line>

         <textField isBlankWhenNull = "true" bookmarkLevel = "1">
            <reportElement x = "0" y = "10" width = "515" height = "30"/>
            <textElement textAlignment = "Center">
               <font size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>

            <anchorNameExpression>
               <![CDATA["Title"]]>
            </anchorNameExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "40" width = "515" height = "20"/>

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>
         </textField>

      </band>
   </title>

   <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[$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>
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as below.

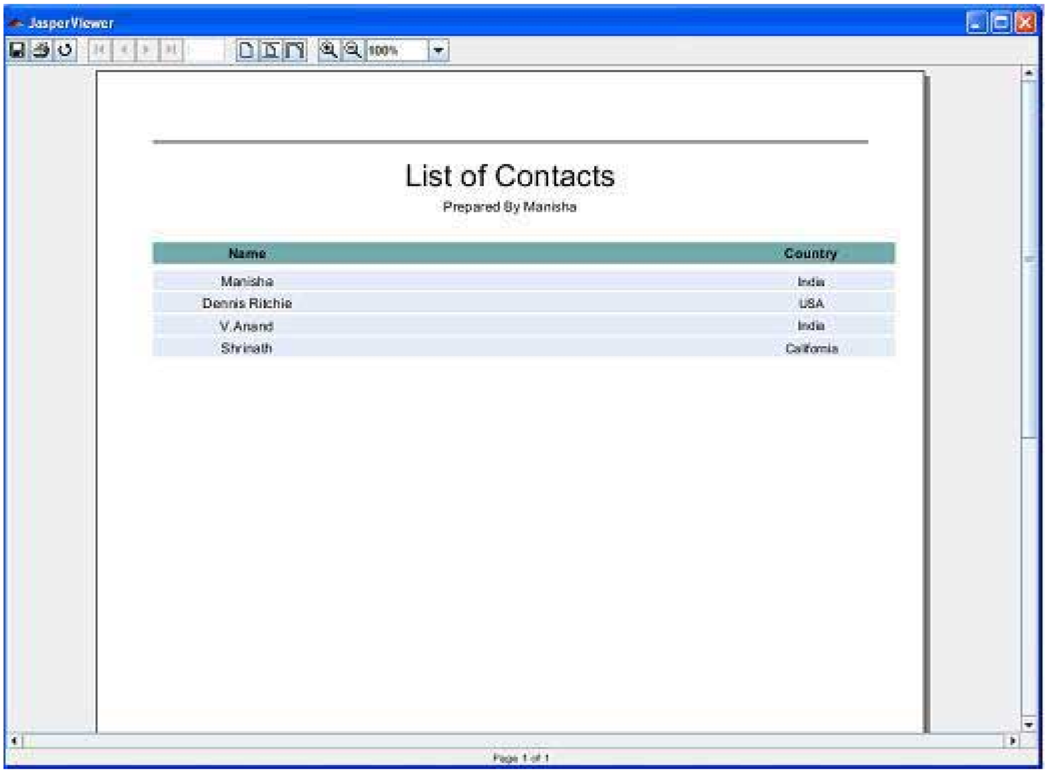
The import file - baseBuild.xml is picked from the chapter Environment Setup and should be placed in the same directory as the build.xml.

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

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">


      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport 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


clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint


compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28: warning:
   'includeantruntime' was not set, defaulting to build.sysclasspath=last;
   set to false for repeatable builds
   [javac] Compiling 7 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.

viewFillReport:
   [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: 18 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the following screen –



List of Contacts

Prepared By Manisha

Name	Country
Manisha	India
Dennis Ritchie	USA
V.Anand	India
Shrinath	California

Here, we see that, the ReportTitle "List Of Contacts" and Author "Prepared By Manisha" are displayed at the beginning of the report.

Report Data Sources

Datasources are structured data container. While generating the report, JasperReports engine obtains data from the datasources. Data can be obtained from the databases, XML files, arrays of objects, and collection of objects. We saw in the chapter Filling Reports , the fillReportXXX () method expects to receive a data source of the report, which has to fill, in the form of **net.sf.jasperreports.engine.JRDataSource** object or a **java.sql.Connection** (when the report data is found in a relational database).

The JRDataSource interface has only two methods, which should be implemented −

public boolean next() throws JRException;

> At the report filling time, this method is called on the data source object by the reporting engine when iterating through the data.

public Object getFieldValue(JRField jrField) throws JRException;

> This method provides the value for each report field in the current data source record.

The only way to retrieve data from the data source is by using the report fields. There are several default implementations of the JRDataSource interface, depending on the way, the records in the data source are acquired.

Datasource Implementations

The table given below summarizes the datasources and their implementation classes −

Datasource	Implementation Class
JDBC	net.sf.jasperreports.engine.JRResultSetDataSource
JavaBean	net.sf.jasperreports.engine.data.JRBeanCollectionDataSource, net.sf.jasperreports.engine.data.JRBeanArrayDataSource
Map-based	net.sf.jasperreports.engine.data.JRMapArrayDataSource, net.sf.jasperreports.engine.data.JRMapCollectionDataSource
TableModel	net.sf.jasperreports.engine.data.JRTableModelDataSource
XML	net.sf.jasperreports.engine.data.JRXmlDataSource
CSV	net.sf.jasperreports.engine.data.JRCsvDataSource
XLS	net.sf.jasperreports.engine.data.JRXlsDataSource
Empty	net.sf.jasperreports.engine.JREmptyDataSource

JDBC Data Sources

Class **JRResultSetDataSource** craps a *java.sql.ResultSet* object. This is the most commonly used data source implementations when report data are extracted from a relational database. If a *java.sql.Connection* is passed to the engine instead, it executes first the related query and stores the returned *java.sql.ResultSet* object in a JRResultSetDataSource instance.

JavaBean Data Sources

Classes **JRBeanArrayDataSource** and **JRBeanCollectionDataSource** represent implementations that can wrap arrays and collections of JavaBean objects. Each object inside the array or the collection will be seen as one record in this type of data source. The mapping between a particular JavaBean property and the corresponding report field is made by naming conventions. The name of the report field must be the same as the name of the JavaBean property as specified by the JavaBeans specifications.

In all the examples of this tutorial, we have used JRBeanCollectionDataSource.

Map-based Data Sources

The implementation classes **JRMapArrayDataSource** and **JRMapCollectionDataSource** are useful if the parent application already stores the reporting data available in-memory as *java.util.Map objects*. Each Map object in the wrapped array or collection is considered a virtual record in the data source, and the value of each report field is extracted from the map using the report field named as the key.

TableModel Data Sources

In many client-side applications, data is displayed in tabular format. A common requirement in many applications is to allow the user to print this tabular format as a report. Implementation class **JRTableModelDataSource** makes the task of generating reports from tabular format trivial for Swing applications. This class wraps a javax.swing.table.TableModel object. Columns in the wrapped TableModel object can be accessed either by their names or by their 0-based indexes.

XML Data Sources

Class **JRXmlDataSource** is a data source implementation based on DOM, which uses XPath expressions to select data from the XML document. Records in the XML data source are represented by node elements selected through the XPath expression. Field values are retrieved from each record using the XPath expression provided by the field description (<fieldDescription> element in JRXML).

XPath is a language used to navigate through an XML document's attributes and elements. More information about XPath can be found at http://www.w3.org/TR/xpath.

CSV Data Sources

JRCsvDataSource represents an implementation for data sources, which retrieve their data from structured text files; usually CSVs. Field values are retrieved using their column index.

XLS Data Sources

JRXlsDataSource represents an implementation for data sources, which retrieve their data from Excel documents. Report-field mapping for this data source implementation is also based on the field column index.

Empty Data Sources

The class **JREmptyDataSource**, simulates a data source with a given number of virtual empty records inside. It is used by the UI tools to offer basic report preview functionality, or in special report templates, or for testing and debugging purposes.

Rewindable Data Sources

The **net.sf.jasperreports.engine.JRRewindableDataSource** extends the basic *JRDataSource* interface. It adds only one method, called moveFirst (), to the interface. This method is intended to move the cursor to the first element in the datasource.

Rewindable data sources are useful when working with sub-reports placed inside a band that is not allowed to split due to the isSplitAllowed="false" setting and there is not enough space on the current page for the sub report to be rendered.

All the above data source implementations are rewindable except for the **JRResultSetDataSource**, as it does not support moving the record pointer back. This poses a problem only if this data source is used manually to wrap a java.sql.ResultSet before passing it to the sub-report. There is no problem, if the SQL query resides in the sub-report template, as the engine will execute it again when restarting the sub-report on the next page.

Data Source Providers

The JasperReports library has an interface **net.sf.jasperreports.engine.JRDataSourceProvider**. This helps in creating and disposing of data source objects. When creating a report template using GUI tools, a special tool for customizing the report's data source is needed. JRDataSourceProvider is the standard way to plug custom data sources into a design tool. A custom implementation of this interface should implement the following methods that allow creating and disposing of data source objects and also methods for listing the available report fields inside the data source if possible −

```
public boolean supportsGetFieldsOperation();

public JRField[] getFields(JasperReport report)
   throws JRException, UnsupportedOperationException;

public JRDataSource create(JasperReport report) throws JRException;

public void dispose(JRDataSource dataSource) throws JRException;
```

Reports Fields

Report fields are elements, which represent mapping of data between datasource and report template. Fields can be combined in the report expressions to obtain the desired output. A report template can contain zero or more <field> elements. When declaring report fields, the data source should supply data corresponding to all the fields defined in the report template.

Field Declaration

Field declaration is done as shown below −

```
<field name = "FieldName" class = "java.lang.String"/>
```

The Name Attribute

The *name* attribute of the <field> element is mandatory. It references the field in report expressions by name.

The Class Attribute

The *class* attribute specifies the class name for the field values. Its default value is *java.lang.String*. This can be changed to any class available at runtime. Irrespective of the type of a report field, the engine takes care of casting in the report expressions in which the $F{} token is used, hence making manual casts unnecessary.

Field Description

The <fieldDesciption> element is an optional element. This is very useful when implementing a custom data source. For example, we can store a key or some information, by which we can retrieve the value of field from the custom data source at runtime. By using the <fieldDesciption> element instead of the field name, you can easily overcome restrictions of field-naming conventions when retrieving the field values from the data source.

Following is a piece of code from our existing JRXML file (Chapter Report Designs). Here, we can see usage of ***name***, ***class***, and ***fieldDescription*** elements.

```
<field name = "country" class = "java.lang.String">
   <fieldDescription><![CDATA[country]]></fieldDescription>
</field>

<field name = "name" class = "java.lang.String">
   <fieldDescription><![CDATA[name]]></fieldDescription>
</field>
```

Sort Fields

At the times when data sorting is required and the data source implementation doesn't support it (for e.g. CSV datasource), JasperReports supports in-memory field-based data source sorting. The sorting can be done using one or more <sortField> elements in the report template.

If at least one sort field is specified, during report filling process, the data source is passed to a *JRSortableDataSource* instance. This in turn, fetches all the records from data source, performs in memory sort according to the specified fields, and replaces the original data source.

The sort field name should be identical to the report field name. Fields used for sorting should have types that implement java.util.Comparable. Natural order sorting is performed for all fields except those of type java.lang.String (for String type, collator corresponding to the report fill locale is used). When several sort Fields are specified, the sorting will be performed using the fields as sort keys in the order in which they appear in the report template. Following example demonstrates the sorting feature.

Sorted Report Example

Let's add the <**sortField**> element to our existing report template (Chapter Report designs). Let's sort field *country* in descending order. The revised report template (jasper_report_template.jrxml) is as follows. Save it to C:\tools\jasperreports-5.0.1\test directory −

```
<?xml version = "1.0"?>
<!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" pageWidth = "595" pageHeight = "842"
   columnWidth = "515" leftMargin = "40" rightMargin = "40"
   topMargin = "50" bottomMargin = "50">

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

   <sortField name = "country" order = "Descending"/>
   <sortField name = "name"/>

   <title>
      <band height = "70">

         <line>
            <reportElement x = "0" y = "0" width = "515" height = "1"/>
         </line>

         <textField isBlankWhenNull = "true" bookmarkLevel = "1">
            <reportElement x = "0" y = "10" width = "515" height = "30"/>

            <textElement textAlignment = "Center">
               <font size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>

            <anchorNameExpression>
               <![CDATA["Title"]]>
            </anchorNameExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "40" width = "515" height = "20"/>

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>

         </textField>

      </band>
   </title>

   <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[$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>
```

The java codes for report filling remains unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below –

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as given below –

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as given below –

```
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;
   }
}
```

Report generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are 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.

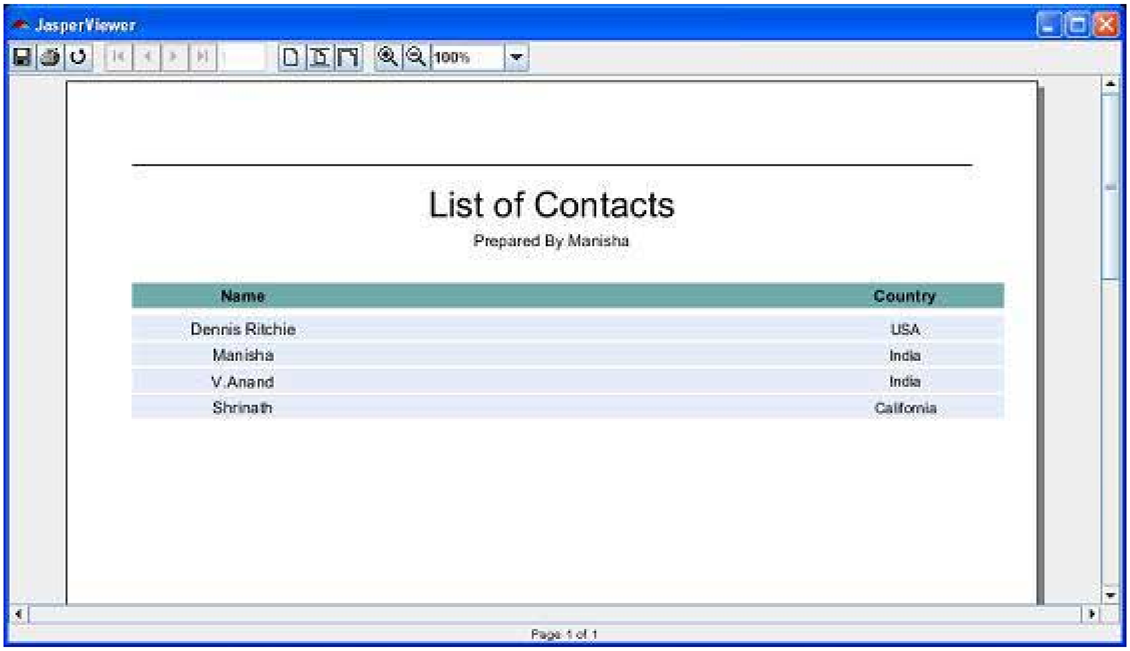
```
<?xml version = "1.0" encoding = "UTF-8"?>
<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport 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


clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint


compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28: warning:
   'includeantruntime' was not set, defaulting to build.sysclasspath=last;
   set to false for repeatable builds
   [javac] Compiling 7 source files to C:\tools\jasperreports-5.0.1\test\classes


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.


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.


viewFillReport:
   [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: 18 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



List of Contacts

Prepared By Manisha

Name	Country
Dennis Ritchie	USA
Manisha	India
V.Anand	India
Shrinath	California

Here, we can see that the country names are arranged in descending order alphabetically.

Report Expression

Report expressions are the powerful features of JasperReports, which allow us to display calculated data on a report. Calculated data is the data that is not a static data and is not specifically passed as a report parameter or datasource field. Report expressions are built from combining report parameters, fields, and static data. The Java language is used for writing report expressions by default. Other scripting languages for report expressions like Groovy scripting language, JavaScript, or BeanShell script are supported by JasperReports compilers.

This chapter will explain you − how do report expressions work, assuming that they have been written using the Java language only. In a JRXML report template, there are several elements that define expressions as −

<variableExpression>

<initialValueExpression>

<groupExpression>

<printWhenExpression>

<imageExpression>

<textFieldExpression>

Expression Declaration

Basically, all report expressions are Java expressions, which can reference the report fields, report variables, and report parameters.

Field Reference in Expression

To use a report field reference in an expression, the name of the field must be put between **$F{**and**}** character sequences, as shown below −

```
<textfieldexpression>
   $F{Name}
</textfieldexpression>
```

Following is a piece of code from our existing JRXML file (chapter Report Designs) −

```
<textFieldExpression class = "java.lang.String">
   <![CDATA[$F{country}]]>
</textFieldExpression>
```

Variable Reference in Expression

To reference a variable in an expression, we must put the name of the variable between **$V{**and**}** as shown in the example given below −

```
<textfieldexpression>
   "Total height : " + $V{SumOfHeight} + " ft."
</textfieldexpression>
```

Parameter Reference in Expression

To reference a parameter in an expression, the name of the parameter should be put between **$P{**and**}** as shown in the example given below −

```
<textfieldexpression>
   "ReportTitle : " + $P{Title}
</textfieldexpression>
```

Following is a piece of code from our existing JRXML file, which demonstrates the referencing of parameter in an expression. (JRXML from chapter Report Designs) −

```
<textField isBlankWhenNull = "true" bookmarkLevel = "1">
   <reportElement x = "0" y = "10" width = "515" height = "30"/>
   
   <textElement textAlignment = "Center">
      <font size = "22"/>
   </textElement>
   <textFieldExpression class = "java.lang.String">
      <![CDATA[$P{ReportTitle}]]>
   </textFieldExpression>

   <anchorNameExpression>
      <![CDATA["Title"]]>
   </anchorNameExpression>
</textField>

<textField isBlankWhenNull = "true">
   <reportElement  x = "0" y = "40" width = "515" height = "20"/>

   <textElement textAlignment = "Center">
      <font size = "10"/>
   </textElement>

   <textFieldExpression class = "java.lang.String">
      <![CDATA[$P{Author}]]>
   </textFieldExpression>
</textField>
```

As you have seen above, the parameter, field, and variable references are in fact real Java objects. Knowing their class from the parameter, field, or variable declaration made in the report template, we can even call methods on those object references in the expressions.

The following example shows − how to extract and display the first letter from java.lang.String report field "Name" −

```
<textFieldExpression>
   $F{Name}.substring(0, 1)
</textFieldExpression>
```

Resource Bundle Reference in Expression

To reference a resource in an expression, the *key* should be put between **$R{**and**}** as shown in the example given below −

```
<textfieldexpression>
   $R{report.title}
</textfieldexpression>
```

Based on the runtime-supplied locale and the *report.title* key, the resource bundle associated with the report template is loaded. Hence, the title of report is displayed by extracting the String value from the resource bundle. More on internationalization can be found in the chapter Internationalization .

Calculator

Calculator is an entity in JasperReports, which evaluates expressions and increments variables or datasets at report-filling time. During compiling process, the information is produced and stored in the compile report by the compiler. This information is used during the report-filling time to build an instance of the net.sf.jasperreports.engine.fill.JRCalculator class.

Java source file is generated and compiled by Java-based report compilers on the fly. This generated class is a subclass of the JRCalculator, and the bytecode produced by compiling it is stored inside the JasperReport object. This bytcode is loaded at the report filling time and the resulting class is instantiated to obtain the calculator object needed for expression evaluation.

Conditional Expressions

JasperReports doesn't support if-else statements when defining variable expressions. Instead, you can use the ternary operators **{cond} ? {statement 1} : {statement 2}**. This operator can be nested inside a Java expression to obtain the desired output based on multiple conditions.

Example of conditional Expression in Report

Let's modify existing report template (Chapter Report Designs) and add a conditional expression for the field country. The revised report template (jasper_report_template.jrxml) is as follows. Save it to C:\tools\jasperreports-5.0.1\test directory −

```
<?xml version = "1.0"?>
<!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" pageWidth = "595" pageHeight = "842"
   columnWidth = "515" leftMargin = "40" rightMargin = "40"
   topMargin = "50" bottomMargin = "50">

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

   <sortField name = "country" order = "Descending"/>
   <sortField name = "name"/>

   <title>
      <band height = "70">

         <line>
            <reportElement x = "0" y = "0" width = "515" height = "1"/>
         </line>

         <textField isBlankWhenNull = "true" bookmarkLevel = "1">
            <reportElement x = "0" y = "10" width = "515" height = "30"/>

            <textElement textAlignment = "Center">
               <font size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>

            <anchorNameExpression>
               <![CDATA["Title"]]>
            </anchorNameExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "40" width = "515" height = "20"/>

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>
         </textField>

      </band>
   </title>

   <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[$F{country}.isEmpty() ? "NO COUNTRY" : $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>
```

The java codes for report filling are as follows. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as –

```java
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as –

```java
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;
   }
}
```

We will add a new record with country field as empty in our Java bean List. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as −

```
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"));
      dataBeanList.add(produce("Tanmay", ""));

      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;
   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are given below.

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

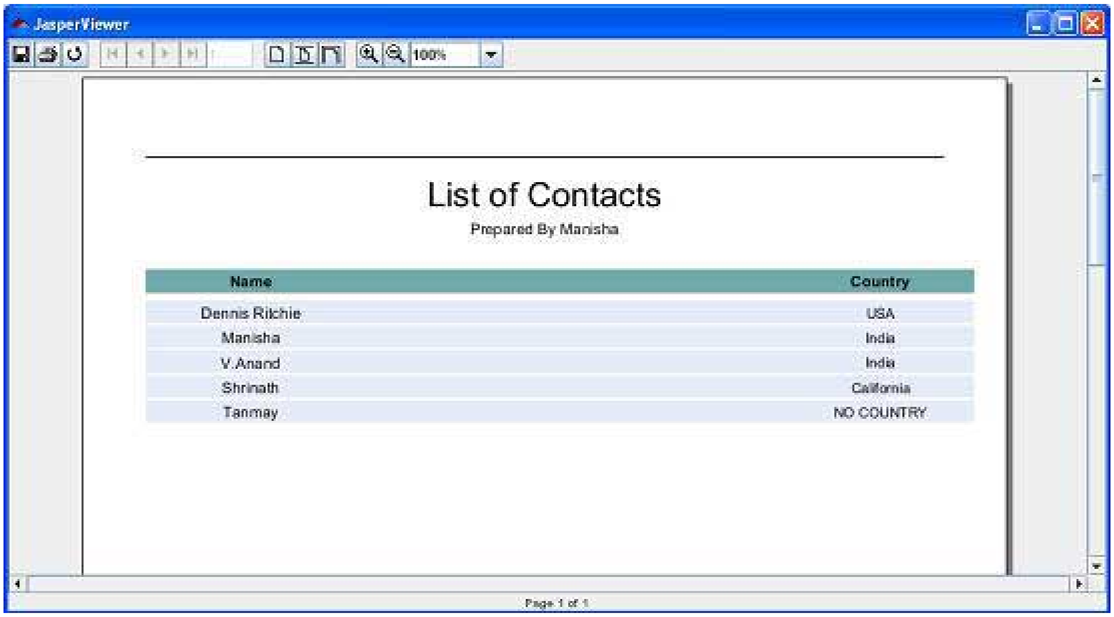
```
<?xml version = "1.0" encoding = "UTF-8"?>
<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class = com.tutorialspoint.JasperReportFill** (viewFullReport is the default target) as –

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting to build.sysclasspath=last;
   set to false for repeatable builds
   [javac] Compiling 3 source files to C:\tools\jasperreports-5.0.1\test\classes
```

```
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.

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.

viewFillReport:
    [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: 5 minutes 5 seconds

C:\tools\jasperreports-5.0.1\test>
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below –



List of Contacts

Prepared By Manisha

Name	Country
Dennis Ritchie	USA
Manisha	India
V.Anand	India
Shrinath	California
Tanmay	NO COUNTRY

Here, we can see, for the last record, we had not passed any data for the field country, "NO COUNTRY" is being printed.

Report Variables

Report variables are special objects built on top of the report expression.

Report variables simplify the following tasks –

> Report expressions, which are heavily used throughout the report template. These expressions can be declared only once by using the report variables.
>
> Report variables can perform various calculations based on the corresponding expressions values such as count, sum, average, lowest, highest, variance, etc.

If variables are defined in a report design, then these can be referenced by new variables in the expressions. Hence, the order in which the variables are declared in a report design is important.

Variable Declaration

A variable declaration is as follows –

```
<variable name = "CityNumber" class = "java.lang.Integer" incrementType = "Group"
   incrementGroup = "CityGroup" calculation = "Count">
   <variableExpression>
      <![CDATA[Boolean.TRUE]]>
   </variableExpression>
</variable>
```

As seen above, <variable> element contains number of attributes. These attributes are summarized below –

The Name Attribute

Similar to *parameters* and *fields*, the *name* attribute of </variable> element is mandatory. It allows referencing the variable by its declared name in the report expressions.

The Class Attribute

The *class* attribute is also mandatory that specifies the class name for the variable values. Its default value is *java.lang.String*. This can be changed to any class available in the classpath, both at the report-compilation time and the report filling time. The engine takes care of type-casting in report expressions which the $V{} token is used, hence manual type-casting is not required.

Calculation

This attribute determines – what calculation to perform on the variable when filling the report. The following subsections describe all the possible values for the calculation attribute of the <variable> element.

Average − The variable value is the average of every non-null value of the variable expression. Valid for numeric variables only.

Count − The variable value is the count of non-null instances of the variable expression.

First − The variable value is the value of the first instance of the variable expression. Subsequent values are ignored.

Highest − The variable value is the highest value for the variable expression.

Lowest − The variable value is the lowest value for the variable expression in the report.

Nothing − No calculations are performed on the variable.

StandardDeviation − The variable value is the standard deviation of all non-null values matching the report expression. Valid for numeric variables only.

Sum − The variable value is the sum of all non-null values returned by the report expression.

System − The variable value is a custom calculation (calculating the value for that variable yourself, using the scriptlets functionality of JasperReports).

Variance − The variable value is the variance of all non-null values returned by evaluation of the report variable's expression.

Incrementer FactoryClass

This attribute determines the class used to calculate the value of the variable when filling the current record on the report. Default value would be any class implementing **net.sf.jasperreports.engine.fill.JRIncrementerFactory**. The factory class will be used by the engine to instantiate incrementer objects at runtime depending on the *calculation* attribute set for the variable.

IncrementType

This determines when to recalculate the value of the variable. This attribute uses values, as below −

Column − The variable value is recalculated at the end of each column.

Group − The variable value is recalculated when the group specified by incrementGroup changes.

None − The variable value is recalculated with every record.

Page − The variable value is recalculated at the end of every page.

Report − The variable value is recalculated once, at the end of the report.

IncrementGroup

This determines the name of the group at which the variable value is recalculated, when *incrementType* is *Group*. This takes name of any group declared in the JRXML report template.

ResetType

This determines when the value of a variable is reset. This attribute uses values, as below −

Column − The variable value is reset at the beginning of each column.

Group − The variable value is reset when the group specified by incrementGroup changes.

None − The variable value is never reset.

Page − The variable value is reset at the beginning of every page.

Report − The variable value is reset only once, at the beginning of the report.

ResetGroup

This determines the name of the group at which the variable value is reset, when *resetType* is *Group*. The values for this attribute would be the name of any group declared in the JRXML report template.

Built-In Report Variables

There are some built-in system variables, ready to use in expressions, as follows −

S.NO	Variable Name and Description
1	**PAGE_NUMBER** This variable's value is its current page number. It can be used to display both the current page number and the total number of pages using a special feature of JasperReports text field elements, the *evaluationTime* attribute.
2	**COLUMN_NUMBER** This variable contains the current column number.
3	**REPORT_COUNT** This report variable contains the total number of records processed.
4	**PAGE_COUNT**

	This variable contains the number of records that were processed when generating the current page.
5	**COLUMN_COUNT** This variable contains the number of records that were processed when generating the current column.
6	**GroupName_COUNT** The name of this variable is derived from the name of the group it corresponds to, suffixed with the _COUNT sequence. This variable contains the number of records in the current group.

Example

Let's add a variable (**countNumber**) to our existing report template (Chapter Report Designs). We will prefix the count to each record. The revised report template (jasper_report_template.jrxml) is as follows. Save it to C:\tools\jasperreports-5.0.1\test directory −

```
<?xml version = "1.0"?>
<!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" pageWidth = "595"
   pageHeight = "842" columnWidth = "515"
   leftMargin = "40" rightMargin = "40" topMargin = "50" bottomMargin = "50">

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

<variable name = "countNumber" class = "java.lang.Integer" calculation = "Count">
   <variableExpression>
      <![CDATA[Boolean.TRUE]]>
   </variableExpression>
</variable>

<title>
   <band height = "70">

      <line>
         <reportElement x = "0" y = "0" width = "515" height = "1"/>
      </line>

      <textField isBlankWhenNull = "true" bookmarkLevel = "1">
         <reportElement x = "0" y = "10" width = "515" height = "30"/>

         <textElement textAlignment = "Center">
            <font size = "22"/>
         </textElement>

         <textFieldExpression class = "java.lang.String">
            <![CDATA[$P{ReportTitle}]]>
         </textFieldExpression>

         <anchorNameExpression>
            <![CDATA["Title"]]>
         </anchorNameExpression>
      </textField>

      <textField isBlankWhenNull = "true">
         <reportElement  x = "0" y = "40" width = "515" height = "20"/>

         <textElement textAlignment = "Center">
            <font size = "10"/>
         </textElement>

         <textFieldExpression class = "java.lang.String">
            <![CDATA[$P{Author}]]>
         </textFieldExpression>
      </textField>

   </band>
</title>

<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[$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["  " + String.valueOf($V{countNumber}) +"."+$F{name}]]>
            </textFieldExpression>
         </textField>

      </band>
   </detail>

</jasperReport>
```

The java codes for report filling remains unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below –

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as given below –

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as given below –

```
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;
   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

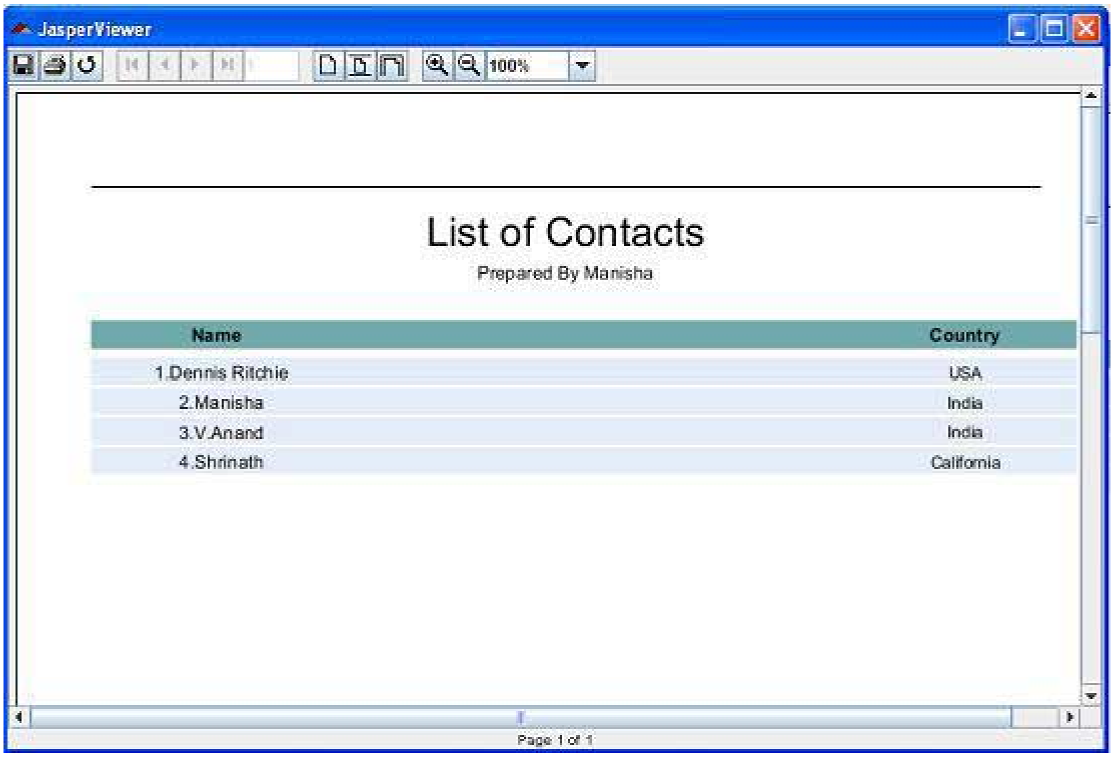
The import file - baseBuild.xml is picked from the chapter Environment Setup and should be placed in the same directory as the build.xml.

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

   <import file = "baseBuild.xml" />
   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport is the default target) as −

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28: warning:
   'includeantruntime' was not set, defaulting to build.sysclasspath=last;
   set to false for repeatable builds
   [javac] Compiling 7 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.

viewFillReport:
   [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: 18 seconds
```

As a result of above compilation, a JasperViewer window opens up as in the screen below –



List of Contacts

Prepared By Manisha

Name	Country
1.Dennis Ritchie	USA
2.Manisha	India
3.V.Anand	India
4.Shrinath	California

Here, we see that the count is prefixed for each record.

Report Sections

We discussed the structure of a simple report template in the chapter Getting Started . On similar lines, JasperReports structures the report template into multiple sections. Sections are portions of the report that have a specified height and can contain report objects like lines, rectangles, images, or text fields.

The report engine iterates through the virtual records of the supplied report data source, at report filling time. Depending on each section's defined behavior, the engine then renders each report section when appropriate. For instance, the detail section is rendered for each record in the data source. When page breaks occur, the page header and page footer sections are rendered as needed.

In JasperReports, terminology and report sections are also called as **report bands**. Sections are made up of one or more bands. These sections are filled repeatedly at report-generating time and prepare the final document.

Main Sections

A report template in JasperReports has the following main sections −

```
<title></title>

<pageheader></pageheader>

<columnheader></columnheader>

<groupheader></groupheader>
```

```
<detail></detail>

<groupfooter></groupfooter>

<columnfooter></columnfooter>

<pagefooter></pagefooter>

<lastpagefooter></lastpagefooter>

<summary></summary>

<nodata></nodata>

<background></background>
```

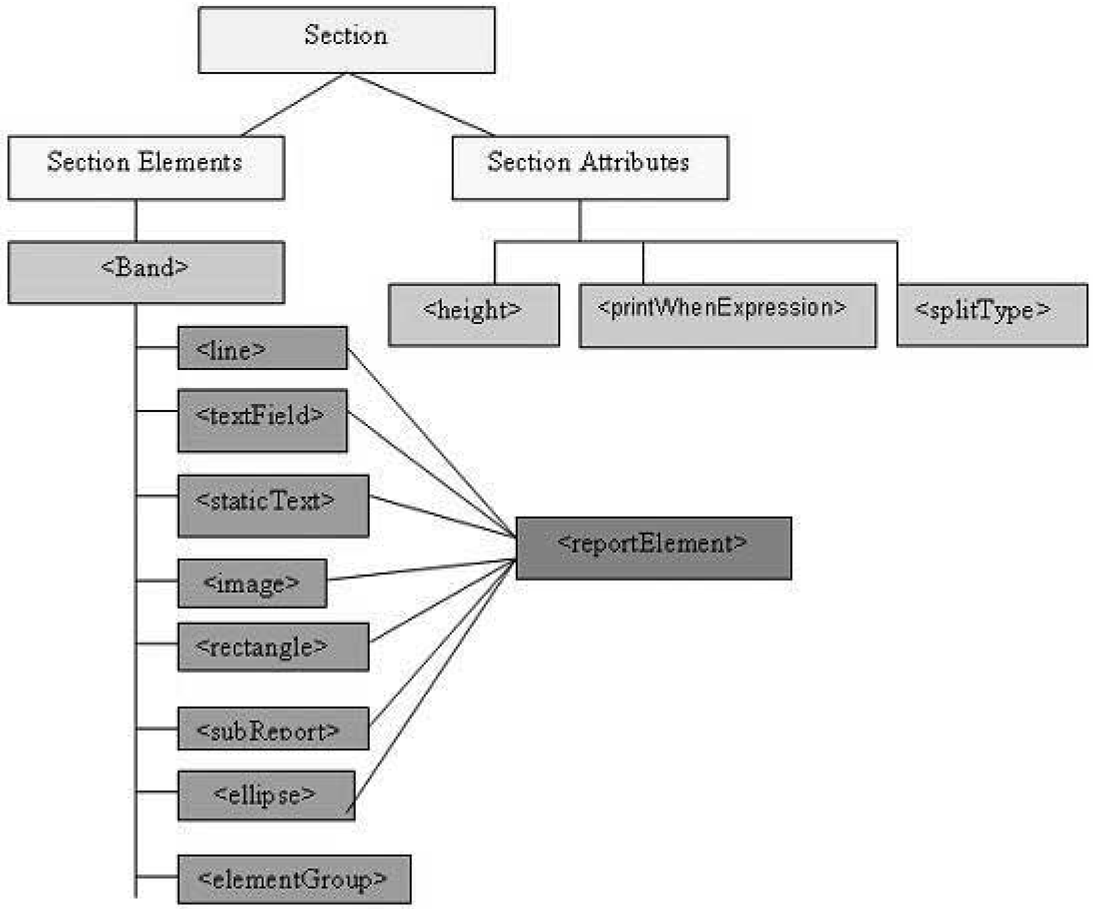
The following table summarizes each of the sections –

S.NO	Section and Description
1	**Title** This section appears only once at the beginning of the report.
2	**Page Header** This section appears at the beginning of each page in the generated document.
3	**Column Header** This section appears at the beginning of each column in the generated document. If the report has only one column defined, then column header and footer sections are ignored.
4	**Group Header** This section is introduced by a report group (chapter **Groups**). Each time the grouping expression changes its value, the group header section is printed above the detail section. In case, if more than one group is defined, the group header is printed in the order of group definition.
5	**Detail** This section is repeated for each line of data supplied by the report's data source. The detail section can be made of multiple bands.
6	**Group Footer**

	This section is introduced by a report group (chapter **Groups**). The group footer section is printed below the detail section before the value of the grouping expression changes. The group footer is always printed for the last line of data in data source. In case, if more than one group is defined, the group footer is printed in the reverse order of group definition.
7	**Column Footer** This section appears at the bottom of each column. If the report's column count is 1, then column header and footer sections are ignored.
8	**Page Footer** This section appears at the bottom of each page.
9	**Last Page Footer** This section replaces the regular page footer on the last page of the report. In case, the summary section is also present, then this might not be the very last page of the document. This section is sometimes useful when summary information has to be displayed at the bottom of the last page.
10	**Summary** This section appears only once at the end of the report.
11	**No Data** This section is printed when the *When No Data Print report* property is set to *No Data* Section. If the <noData> section is defined in the report template, and if the data source is empty, then the <noData> section will be the only one taken into account at fill time, and its content will produce the report output.
12	**Background** The background section is displayed on every page and cannot overflow to the next page. Elements placed on this section are evaluated at page initialization time and are displayed in the background. All other page objects are displayed on top of the background objects. This section is useful for creating page watermarks.

Section, Elements and Attribute Relation

The following diagram shows the elements and attributes relationship in a section of a report.



Section Elements

All the above mentioned report sections are optional. But any report template will have at least one such section. Each of these sections contains a single <**band**> element as its only sub-element. A <**band**> can contain zero or more following sub-elements −

<line>, <rectangle>, <ellipse>, <image>, <staticText>, <textField>, <subReport>, or <elementGroup>

Each of these elements must contain a single <**reportElement**> as its first element (except elementGroup). A <**reportElement**> determines how data is laid out for that particular element. Unlike variables and parameters, report elements are not required to have a name, because normally you do not need to obtain any individual element inside a report template.

The table below summarizes the attributes of <**reportElement**> −

Attribute	Description	Valid Values
x	Specifies the x	An integer value indicating the x

	coordinate of the band element.	coordinate of the element in pixels. This attribute is required.
y	Specifies the y coordinate of the band element.	An integer value indicating the y coordinate of the element in pixels. This attribute is required.
width	Specifies the width of the band element.	An integer value indicating the element width in pixels. This attribute is required.
height	Specifies the height of the band element.	An integer value indicating the element height in pixels. This attribute is required.
key	Unique identifier of band element.	A unique string value.
stretchType	Specifies how does the element stretch when the containing band stretches	**NoStretch (default)** − The element will not stretch. **RelativeToTallestObject** − The element will stretch to accommodate the tallest object in its group. **RelativeToBand** − The element will stretch to fit the band's height.
positionType	Specifies the element's position when the band stretches.	**Float** − The element will move depending on the size of the surrounding elements. **FixRelativeToTop (default)** − The element will maintain a fixed position relative to the band's top. **FixRelativeToBottom** − The element will maintain a fixed position relative to the band's bottom.
isPrintRepeatedValues	Specifies if repeated values are printed.	**true (default)** − Repeated values will be printed. **false** − Repeated values will not be printed.

mode	Specifies the background mode of the element	Opaque, Transparent
isRemoveLineWhenBlank	Specifies if the element should be removed when it is blank and there are no other elements in the same horizontal space.	true, false
isPrintInFirstWholeBand	Specifies if the element must be printed in a whole band, that is, a band that is not divided between report pages or columns.	true, false
isPrintWhenDetailOverFlows	Specifies if the element will be printed when the band overflows to a new page or column.	true, false
printWhenGroupChanges	Specifies that the element will be printed when the specified group changes.	A string value.
forecolor	Specifies the foreground color of the element.	Either a hexadecimal RGB value preceded by the # character, or one of the following predefined values: *black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, yellow, white.*
backcolor	Specifies the background color of the element.	Same as Valid Values for *forecolor*

Section Attributes

Following are the attributes of a report section −

Height

Height of the section specifies the height in pixels for that particular section and is very important in the overall report design.

Print When Expression

A Boolean expression that determines whether the section should be printed or not.

Split Allowed

A flag indicating whether the section is allowed to split when it does not fit on the current page. If true, the section will be transferred to the next page. Note that in case, the section does not fit on the next page, then the section will be split regardless of the flag's value. *splitType* can take following values –

> *splitType="Stretch:"* Splits stretched content. If the section stretches on the current page (if the available space is less than declared height), the region that is added to the original height is allowed to split onto the next page.
>
> *splitType="Prevent:"* Prevent split on first attempt. If the section does not fit on the next page, the split occurs normally, as band split prevention is effective only on the first split attempt.
>
> *splitType="Immediate:"* Split immediately. The band is allowed to split anywhere except above, its topmost element.

Example

To demonstrate each section, let's write report template (jasper_report_template.jrxml). Save this file to **C:\tools\jasperreports-5.0.1\test** directory. In this file, we would be displaying a text in each of the sections (we discussed above). The contents of the file are as given below –

```
<?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 = "jasper_report_template" pageWidth = "300" pageHeight = "300"
   columnWidth = "300" leftMargin = "0" rightMargin = "0"
   topMargin = "0" bottomMargin = "0" >

   <title>
      <band height = "50">

         <textField>
            <reportElement x = "100" y = "16" width = "100" height = "20"/>
            <textElement/>

            <textFieldExpression>
               <![CDATA["Title"]]>
            </textFieldExpression>

         </textField>
      </band>
   </title>

   <pageHeader>
      <band height = "40">

         <textField>
            <reportElement  mode = "Opaque" x = "100" y = "10"
               width = "90" height = "20"/>

            <textElement>
               <font isBold = "true"/>
            </textElement>

            <textFieldExpression>
               <![CDATA["Page Header"]]>
            </textFieldExpression>
         </textField>

      </band>
   </pageHeader>

   <columnHeader>
      <band height = "40">

         <textField>
            <reportElement  x = "100" y = "10" width = "90" height = "20"/>

            <textElement>
               <font isItalic = "true"/>
            </textElement>

            <textFieldExpression>
               <![CDATA["Column Header"]]>
            </textFieldExpression>
         </textField>

      </band>
   </columnHeader>

   <detail>
      <band height ="40">

         <textField>
            <reportElement mode = "Opaque" x = "100" y = "10"
               width = "90" height = "20" backcolor = "#99CCFF"/>
            <textElement/>

            <textFieldExpression>
               <![CDATA["Report Details"]]>
            </textFieldExpression>
         </textField>

      </band>
   </detail>

   <columnFooter>
      <band height = "40">

         <textField>
            <reportElement  x = "100" y = "10" width = "90" height = "20"/>
            <textElement/>

            <textFieldExpression>
               <![CDATA["Column Footer"]]>
            </textFieldExpression>
         </textField>

      </band>
   </columnFooter>

   <pageFooter>
      <band height = "40">

         <textField>
            <reportElement  x = "100" y = "10" width = "90" height = "20"/>
            <textElement/>

            <textFieldExpression>
               <![CDATA["Page Footer"]]>
            </textFieldExpression>
         </textField>

      </band>
   </pageFooter>

   <lastPageFooter>
      <band height = "40">

         <textField>
            <reportElement  x = "100" y = "10" width = "90" height = "20"/>
            <textElement/>

            <textFieldExpression>
               <![CDATA["Last Page Footer"]]>
            </textFieldExpression>
         </textField>

      </band>
   </lastPageFooter>

   <summary>
      <band height = "40">

         <textField>
            <reportElement  x = "100" y = "10" width = "90" height = "20"/>
            <textElement/>

            <textFieldExpression>
               <![CDATA["Summary"]]>
            </textFieldExpression>
         </textField>

      </band>
   </summary>

</jasperReport>
```

The java code to fill and generate the report is given below. Let's save this file **JasperReportFill.java** to C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint directory.

```
package com.tutorialspoint;

import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;

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

      try {
         JasperFillManager.fillReportToFile(sourceFileName, null,
            new JREmptyDataSource());
      } catch (JRException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }
}
```

Here, we use an instance of JREmptyDataSource when filling reports to simulate a data source with one record in it, but with all the fields in this single record being null.

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as below.

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

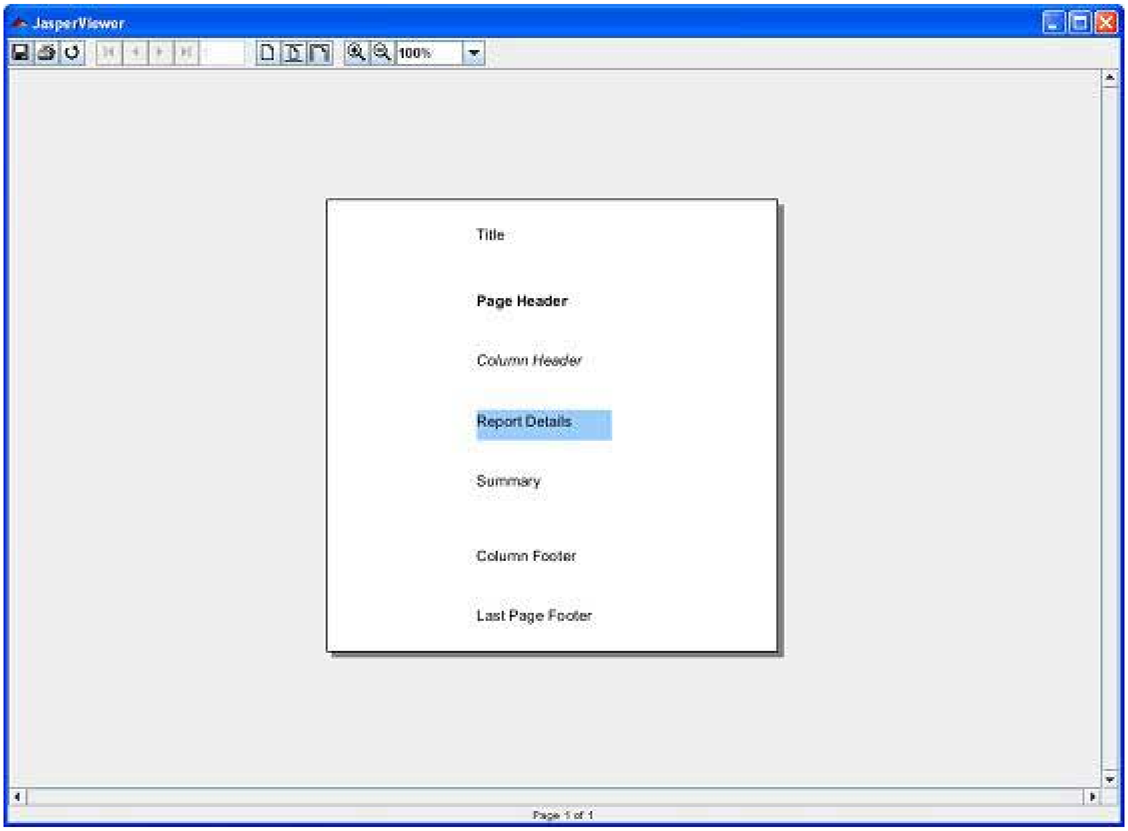
```
<?xml version = "1.0" encoding = "UTF-8"?>
<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">

   <import file = "baseBuild.xml" />
   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport 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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defau
   [javac] Compiling 1 source file to C:\tools\jasperreports-5.0.1\test\classes

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

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

```
viewFillReport:
   [java] log4j:WARN No appenders could be found for logger
   (net.sf.jasperreports.extensions.ExtensionsEnviro
   [java] log4j:WARN Please initialize the log4j system properly.


BUILD SUCCESSFUL
Total time: 18 minutes 22 seconds
```

As a result of above compilation, a JasperViewer window opens up as in the screen below –



Title

Page Header

Column Header

Report Details

Summary

Column Footer

Last Page Footer

Here, we can see in each of the sections a text is printed. It is to be noted that as JRXML contains a <lastPageFooter> element, it will be displayed in the last page of the report instead of the <pageFooter> element being displayed. The <columnHeader> and <columnFooter> elements will only be displayed on the report, if it has more than one column.

Report Groups

Groups in JasperReports help to organize data on report in a logical manner. A report group represents a sequence of consecutive records in the data source, which have something in common, such as the value of a certain report fields. A report group is defined by the <group> element. A report can have any number of groups. Once declared, groups can be referred throughout the report.

A report group has three elements –

Group expression − This indicates the data that must change to start a new data group.

Group header section − Helps place label at the beginning of the grouped data.

Group footer section − Helps place label at the end of the grouped data.

During the iteration through the data source at report-filling time if the value of the group expression changes, a group rupture occurs and the corresponding <groupFooter> and <groupHeader> sections are inserted in the resulting document.

Report group mechanism does not perform any sorting on the data supplied by the data source. Data grouping works as expected only when the records in the data source are already ordered according to the group expressions used in the report.

Group Attributes

The <group> element contains attributes that allow us to control how grouped data is laid out. The attributes are summarized in table below −

S.NO	Attribute and Description
1	**name** This is mandatory. It references the group in report expressions by name. It follows the same naming conventions that we mentioned for the report parameters, fields, and report variables. It can be used in other JRXML attributes when you want to refer a particular report group.
2	**isStartNewColumn** When set to *true*, each data group will begin on a new column. Default value is *false*.
3	**isStartNewPage** When set to *true*, each data group will begin on a new page. Default value is *false*.
4	**isResetPageNumber** When set to *true*, the report page number will be reset every time a new group starts. Default value is *false.*
5	**isReprintHeaderOnEachPage**

	When set to *true*, the group header will be reprinted on every page. Default value is *false*.
6	**minHeightToStartNewPage** Defines minimum amount of vertical space needed at the bottom of the column in order to place the group header on the current column. The amount is specified in report units.
7	**footerPosition** Renders position of the group footer on the page, as well as its behavior in relation to the report sections that follow it.Its value can be: *Normal*, *StackAtBottom*, *ForceAtBottom*, and *CollateAtBottom*. Default value is *Normal*.
8	**keepTogether** When set to *true*, prevents the group from splitting on its first break attempt.

Example

Let's add a group (**CountryGroup**) to existing report template (Chapter Report Designs). Occurrence of each country is counted and the count is displayed as the group footer. In the group header, the count of each record is prefixed. The revised report template (jasper_report_template.jrxml) is as follows. Save it to C:\tools\jasperreports-5.0.1\test directory −

```
<?xml version = "1.0"?>
<!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" pageWidth = "595"
   pageHeight = "842" columnWidth = "515"
   leftMargin = "40" rightMargin = "40" topMargin = "50" bottomMargin = "50">

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

<sortField name = "country" order = "Descending"/>
<sortField name = "name"/>

<variable name = "CountryNumber" class = "java.lang.Integer"
   incrementType = "Group" incrementGroup = "CountryGroup"
   calculation = "Count">
   <variableExpression><![CDATA[Boolean.TRUE]]></variableExpression>
</variable>

<group name = "CountryGroup" minHeightToStartNewPage = "60">
   <groupExpression><![CDATA[$F{country}]]></groupExpression>

   <groupHeader>
      <band height = "20">

         <textField evaluationTime = "Group" evaluationGroup = "CountryGroup"
            bookmarkLevel = "1">
            <reportElement mode = "Opaque" x = "0" y = "5" width = "515"
               height = "15" backcolor = "#C0C0C0"/>

            <box leftPadding = "10">
               <bottomPen lineWidth = "1.0"/>
            </box>
            <textElement/>

            <textFieldExpression class = "java.lang.String">
               <![CDATA["  " + String.valueOf($V{CountryNumber}) + ". "
               + String.valueOf($F{country})]]>
            </textFieldExpression>

            <anchorNameExpression>
               <![CDATA[String.valueOf($F{country})]]>
            </anchorNameExpression>
         </textField>

      </band>
   </groupHeader>

   <groupFooter>
      <band height = "20">

         <staticText>
            <reportElement x = "400" y = "1" width = "60" height = "15"/>
            <textElement textAlignment = "Right"/>
            <text><![CDATA[Count :]]></text>
         </staticText>

         <textField>
            <reportElement x = "460" y = "1" width = "30" height = "15"/>
            <textElement textAlignment = "Right"/>

            <textFieldExpression class = "java.lang.Integer">
               <![CDATA[$V{CountryGroup_COUNT}]]>
            </textFieldExpression>
         </textField>
      </band>
   </groupFooter>

</group>

<title>
   <band height = "70">

      <line>
         <reportElement x = "0" y = "0" width = "515" height = "1"/>
      </line>

      <textField isBlankWhenNull = "true" bookmarkLevel = "1">
         <reportElement x = "0" y = "10" width = "515" height = "30"/>

         <textElement textAlignment = "Center">
            <font size = "22"/>
         </textElement>

         <textFieldExpression class = "java.lang.String">
            <![CDATA[$P{ReportTitle}]]>
         </textFieldExpression>

         <anchorNameExpression>
            <![CDATA["Title"]]>
         </anchorNameExpression>
      </textField>

      <textField isBlankWhenNull = "true">
         <reportElement  x = "0" y = "40" width = "515" height = "20"/>

         <textElement textAlignment = "Center">
            <font size = "10"/>
         </textElement>

         <textFieldExpression class = "java.lang.String">
            <![CDATA[$P{Author}]]>
         </textFieldExpression>
      </textField>

   </band>
</title>

<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[$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>
```

The java codes for report filling remains unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below –

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as below –

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as given below –

```
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;
   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as below.

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

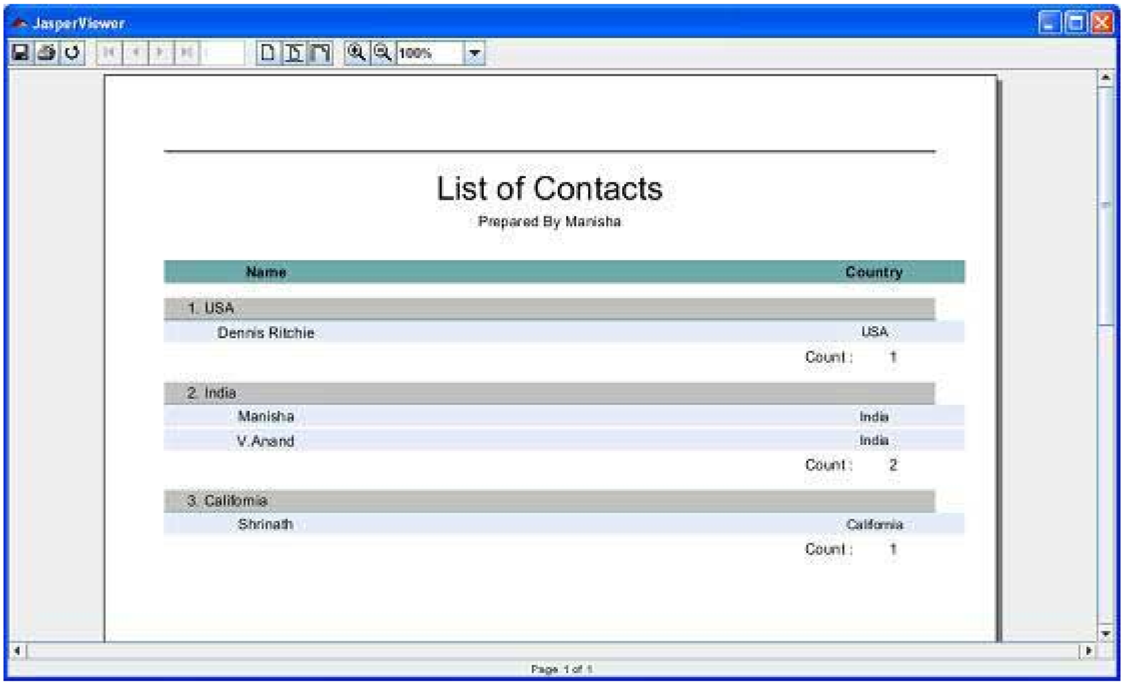
```
<?xml version = "1.0" encoding = "UTF-8"?>
<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport is the default target) as −

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28: warning:
   'includeantruntime' was not set, defaulting to build.sysclasspath=last;
   set to false for repeatable builds
   [javac] Compiling 7 source files to C:\tools\jasperreports-5.0.1\test\classes
```

```
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.

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.

viewFillReport:
   [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: 18 seconds
```

As a result of above compilation, a JasperViewer window opens up as in the screen below –



List of Contacts

Prepared By Manisha

Name	Country
1. USA	
Dennis Ritchie	USA
	Count : 1
2. India	
Manisha	India
V.Anand	India
	Count : 2
3. California	
Shrinath	California
	Count : 1

Here, we see that the each country is grouped and the count of occurrence of each country is displayed at the footer of each group.

Report Fonts

A report contains text elements and each of these can have its own font settings. These settings can be specified using the <**font**> tag available in the <textElement> tag. A report can define a number of fonts. Once defined, they can be used as default or base font settings for other font definitions throughout the entire report.

Report Fonts

A report font is a collection of font settings, declared at the report level. A report font can be reused throughout the entire report template when setting the font properties of text elements.

Report fonts are now deprecated. Do not use <reportFont/> elements declared within the document itself. Use the <style/> element instead.

Font Attributes

Table below summarizes the main attributes of the <**font**> element −

S.NO	Attribute and Description
1	**fontName** The font name, which can be the name of a physical font, a logical one, or the name of a font family from the registered JasperReports font extensions.
2	**size** The size of the font measured in points. It defaults to 10.
3	**isBold** The flag specifying if a bold font is required. It defaults to false.
4	**isItalic** The flag specifying if an italic font is required. It defaults to false.
5	**isUnderline** The flag specifying if the underline text decoration is required. It defaults to false.
6	**isStrikeThrough**

	The flag specifying if the strikethrough text decoration is required. It defaults to false.
7	**pdfFontName** The name of an equivalent PDF font required by the iText library when exporting documents to PDF format.
8	**pdfEncoding** The equivalent PDF character encoding, also required by the iText library.
9	**isPdfEmbedded** The flag that specifies whether the font should be embedded into the document itself. It defaults to false. If set to true, helps view the PDF document without any problem.

Font Types

In JasperReports fonts can be categorized as −

Logical Fonts − Five font types, which have been recognized by the Java platform since version 1.0, are called logical fonts. These are − **Serif, SansSerif, Monospaced, Dialog, and DialogInput**. These logical fonts are not actual font libraries that are installed anywhere on the system. They are merely font type names recognized by the Java runtime. These must be mapped to some physical font that is installed on the system.

Physical Fonts − These fonts are the actual font libraries consisting of, for example, TrueType or PostScript Type 1 fonts. The physical fonts may be Arial, Time, Helvetica, Courier, or any number of other fonts, including international fonts.

Font Extensions − The JasperReports library can make use of fonts registered on-the-fly at runtime, through its built-in support for font extensions. A list of font families can be made available to the JasperReports using font extension. These are made out of similarly looking font faces and supporting specific locales.

As described in the table above we need to specify in the attribute *fontName* the name of a physical font, the name of a logical font, or the name of a font family from the registered JasperReports font extensions.

PDF Font Name

JasperReports library uses the iText library, when exporting reports to PDF(Portable Document Format). PDF files can be viewed on various platforms and will always look the same. This is partially because in this format, there is a special way of dealing with fonts. *fontName* attribute is of no use when exporting to PDF. Attribute *pdfFontName* exist where we need to specify the font settings.

The iText library knows how to deal with built-in fonts and TTF files and recognizes the following built-in font names −

- Courier
- Courier-Bold
- Courier-BoldOblique
- Courier-Oblique
- Helvetica
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Oblique
- Symbol
- Times-Roman
- Times-Bold
- Times-BoldItalic
- Times-Italic
- ZapfDingbats

As per iText library pre-requisite, to work with fonts, we need to specify one of the following as the font name −

- A built-in font name from the above list.
- The name of a TTF (True Type Font) file, which it can locate on disk.
- The real name of the font, provided that the TTF file containing the font has been previously registered with iText or that an alias was defined when the font was registered.

Based on the above pre-requisites, the *pdfFontName* attribute can contain one of the following values −

- The name of a built-in PDF font from the above list.
- The name of a TTF file that can be located on disk at runtime when exporting to PDF.

The real name of a registered font.

The suffix of the key (the part after *net.sf.jasperreports.export.pdf.font*) for a font registered with iText as a font file.

Default Fonts and Inheritance

Each text element inherits font and style attributes from its parent element, which in turn inherits these attributes from its parent. If no styles and/or fonts are defined for elements, the default style (and/or font - but this is now deprecated) declared in the <jasperReport/> root element will be applied.

Defining default styles or fonts in JasperReports is not mandatory. If no font is defined for a given element, the engine looks either for the inherited font attributes, or, if no attributes are found on this way, it looks for the *net.sf.jasperreports.default.font.name* property in the */src/default.jasperreports.properties* file. Its value defines the name of the font family to be used when font properties are not explicitly defined for a text element or inherited from its parent.

The main default font properties and their values defined in the */src/default.jasperreports.properties* file are in the table below −

Property	**Description**
net.sf.jasperreports.default.font.name=SansSerif	The default font name.
net.sf.jasperreports.default.font.size=10	The default font size.
net.sf.jasperreports.default.pdf.font.name=Helvetica	The default PDF font.
net.sf.jasperreports.default.pdf.encoding=Cp1252	The default PDF character encoding.
net.sf.jasperreports.default.pdf.embedded=false	By default PDF fonts are not embedded.

Example

To demonstrate using fonts and font attributes in order to get a particular text appearance, let's write new report template (jasper_report_template.jrxml). The contents of the JRXML are as below. Save it to C:\tools\jasperreports-5.0.1\test directory. Here, we will display a text in the title of the report in various font formats.

```
<?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 = "jasper_report_template" pageWidth = "595" pageHeight = "842"
columnWidth = "555" leftMargin = "20" rightMargin = "20" topMargin = "30"
bottomMargin = "30">

<title>
   <band height = "682">

   <staticText>
      <reportElement x = "0" y = "50" width = "150" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[Welcome to TutorialsPoint!]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "50" width = "390" height = "40"/>
      <textElement/>

      <text>
        <![CDATA[<staticText>
        <reportElement x = "0" y = "50" width = "150" height = "40"/>
        <text>Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "0" y = "100" width = "150" height = "40"/>

      <textElement>
         <font size = "12"/>
      </textElement>

      <text><![CDATA[Welcome to TutorialsPoint!]]></text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "100" width = "390" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[<staticText>
         <reportElement x = "0" y = "100" width = "150" height = "40"/>

         <textElement>
            <font size = "14"/>
         </textElement>

         <text> Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "0" y = "150" width = "150" height = "40"/>

      <textElement>
         <font fontName = "DejaVu Serif" size = "12" isBold = "false"/>
      </textElement>

      <text><![CDATA[Welcome to TutorialsPoint!]]></text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "150" width = "390" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[<staticText>
         <reportElement x = "0" y = "250" width = "150" height = "40"/>

         <textElement>
            <font fontName = "DejaVu Serif" size = "12" isBold = "false"/>
         </textElement>

         <text>Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "0" y = "200" width = "150" height = "40"/>

      <textElement>
         <font fontName = "DejaVu Serif" size = "12" isBold = "true"/>
      </textElement>

      <text><![CDATA[Welcome to TutorialsPoint!]]></text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "200" width = "390" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[<staticText>
         <reportElement x = "0" y = "300" width = "150" height = "40"/>

         <textElement>
            <font fontName = "DejaVu Serif" size = "12" isBold = "true"/>
         </textElement>

         <text>Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "0" y = "250" width = "150" height = "40"/>

      <textElement>
         <font fontName = "Monospaced" size = "12" isItalic = "true"
            isUnderline = "true" pdfFontName = "Courier-Oblique"/>
      </textElement>

      <text><![CDATA[Welcome to TutorialsPoint!]]></text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "250" width = "390" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[<staticText>
         <reportElement x = "0" y = "350" width = "150" height = "40"/>

         <textElement>
            <font fontName = "Monospaced" size = "12" isItalic = "true"
               isUnderline = "true" pdfFontName = "Courier-Oblique"/>
         </textElement>

         <text>Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "0" y = "300" width = "150" height = "40"/>

      <textElement>
         <font fontName = "Monospaced" size = "12" isBold = "true"
            isStrikeThrough = "true" pdfFontName = "Courier-Bold"/>
      </textElement>
      <text><![CDATA[Welcome to TutorialsPoint!]]></text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "300" width = "390" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[<staticText>
         <reportElement x = "0" y = "400" width = "150" height = "40"/>

         <textElement>
            <font fontName = "Monospaced" size = "12" isBold = "true"
               isStrikeThrough = "true" pdfFontName = "Courier-Bold"/>
         </textElement>

         <text>Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
   </staticText>

   <staticText>
      <reportElement x = "0" y = "350" width = "150" height = "40"
         forecolor = "#FF0000"/>

      <textElement>
         <font size = "14"/>
      </textElement>

      <text><![CDATA[Welcome to TutorialsPoint!]]></text>
   </staticText>

   <staticText>
      <reportElement x = "160" y = "350" width = "390" height = "40"/>
      <textElement/>

      <text>
         <![CDATA[<staticText>
         <reportElement x = "0" y = "450" width = "150" height = "40"
            forecolor = "red"/>

         <textElement><font size = "14"/></textElement>
         <text>Welcome to TutorialsPoint!</text></staticText>]]>
      </text>
```

```
      </staticText>

      <staticText>
         <reportElement x = "0" y = "400" width = "150" height = "40" mode = "Opaque"
            forecolor = "#00FF00" backcolor = "#FFFF00"/>

         <textElement>
            <font fontName = "Serif" size = "12" isBold = "true"
               pdfFontName = "Times-Bold"/>
         </textElement>

         <text><![CDATA[Welcome to TutorialsPoint!]]></text>
      </staticText>

      <staticText>
         <reportElement x = "160" y = "400" width = "390" height = "40"/>
         <textElement/>

         <text>
            <![CDATA[<staticText>
            <reportElement x = "0" y = "500" width = "150" height = "40"
               forecolor = "green" backcolor = "#FFFF00" mode = "Opaque"/>

            <textElement>
               <font fontName = "Serif" size = "12" isBold = "true"
                  pdfFontName = "Times-Bold"/>
            </textElement>

            <text>Welcome to TutorialsPoint!</text></staticText>]]>
         </text>
      </staticText>

      <staticText>
         <reportElement x = "0" y = "450" width = "150" height = "40" mode = "Opaque"
            forecolor = "#0000FF" backcolor = "#FFDD99"/>

         <textElement textAlignment = "Center" verticalAlignment = "Middle">
            <font fontName = "SansSerif" size = "12" isBold = "false"
            isItalic = "true" pdfFontName = "Sans.Slanted" isPdfEmbedded = "true"/>
         </textElement>

         <text><![CDATA[Welcome to TutorialsPoint!]]></text>
      </staticText>

      <staticText>
         <reportElement x = "160" y = "450" width = "390" height = "40"/>
         <textElement/>

         <text>
            <![CDATA[<staticText>
            <reportElement x = "0" y = "550" width = "150" height = "90"
               forecolor = "blue" backcolor = "#FFDD99" mode = "Opaque"/>

            <textElement textAlignment = "Center" verticalAlignment = "Middle">
               <font fontName = "SansSerif" size = "12" isBold = "false"
                  pdfFontName = "Sans.Slanted" isPdfEmbedded = "true"/>
            </textElement>

            <text>Welcome to TutorialsPoint!</text></staticText>]]>
         </text>
      </staticText>
      <staticText>
         <reportElement mode = "Opaque" x = "0" y = "500" width = "150" height = "40"
            forecolor = "#FF0000" backcolor = "#99DDFF"/>

         <textElement textAlignment = "Right" verticalAlignment = "Bottom">
            <font fontName = "SansSerif" size = "12" isBold = "true"
               pdfFontName = "DejaVu Sans Bold" isPdfEmbedded = "true"/>
         </textElement>

         <text><![CDATA[Welcome to TutorialsPoint!]]></text>
      </staticText>

      <staticText>
         <reportElement x = "160" y = "500" width = "390" height = "40"/>
         <textElement/>

         <text>
            <![CDATA[<staticText>
            <reportElement x = "0" y = "650" width = "150" height = "90"    forecolor = "red"
               backcolor = "#99DDFF" mode = "Opaque"/>

            <textElement textAlignment = "Right" verticalAlignment = "Bottom">
               <font fontName = "SansSerif" size = "12" isBold = "true"
                  pdfFontName = "DejaVu Sans Bold" isPdfEmbedded = "true"/>
            </textElement>

            <text>Welcome to TutorialsPoint!</text></staticText>]]>
         </text>

      </staticText>

   </band>
</title>

</jasperReport>
```

The java code to fill and generate the report is as given below. Let's save this file **JasperFontsReportFill.java** to C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint directory.

```
package com.tutorialspoint;

import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;

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

      try {
         JasperFillManager.fillReportToFile(sourceFileName, null,
            new JREmptyDataSource());
      } catch (JRException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
   }
}
```

Here, we use an instance of *JREmptyDataSource* when filling reports to simulate a data source with one record in it, but with all the fields being *null*.

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

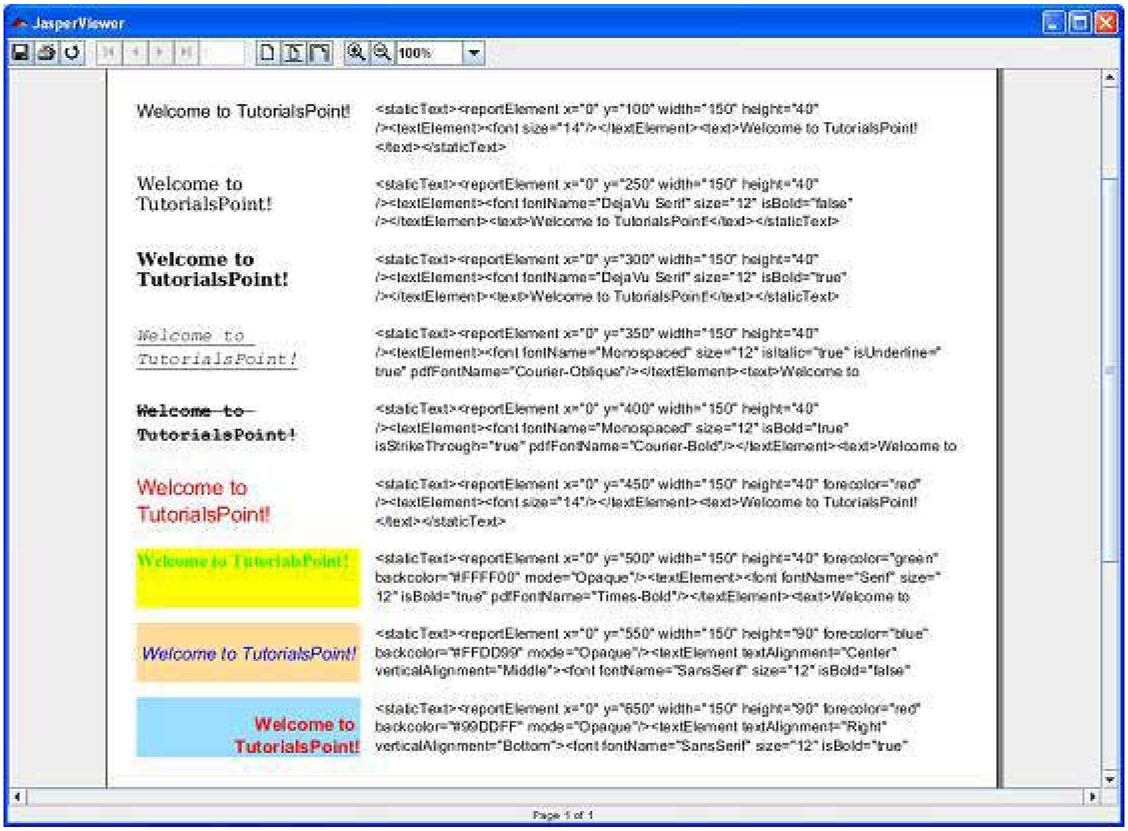
The import file - baseBuild.xml is picked from chapter Environment Setup and should be placed in the same directory as the build.xml.

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

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the report
      stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperFontsReportFill** (viewFullReport is the default target) as −

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting to build.
   [javac] Compiling 5 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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

viewFillReport:
   [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: 45 minutes 3 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="100" width="150" height="40" /><textElement><font size="14"/></textElement><text>Welcome to TutorialsPoint! </text></staticText>`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="250" width="150" height="40" /><textElement><font fontName="DejaVu Serif" size="12" isBold="false" /></textElement><text>Welcome to TutorialsPoint!</text></staticText>`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="300" width="150" height="40" /><textElement><font fontName="DejaVu Serif" size="12" isBold="true" /></textElement><text>Welcome to TutorialsPoint!</text></staticText>`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="350" width="150" height="40" /><textElement><font fontName="Monospaced" size="12" isItalic="true" isUnderline="true" pdfFontName="Courier-Oblique"/></textElement><text>Welcome to`
~~**Welcome to**~~ **TutorialsPoint!**	`<staticText><reportElement x="0" y="400" width="150" height="40" /><textElement><font fontName="Monospaced" size="12" isBold="true" isStrikeThrough="true" pdfFontName="Courier-Bold"/></textElement><text>Welcome to`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="450" width="150" height="40" forecolor="red" /><textElement><font size="14"/></textElement><text>Welcome to TutorialsPoint! </text></staticText>`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="500" width="150" height="40" forecolor="green" backcolor="#FFFF00" mode="Opaque"/><textElement><font fontName="Serif" size="12" isBold="true" pdfFontName="Times-Bold"/></textElement><text>Welcome to`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="550" width="150" height="90" forecolor="blue" backcolor="#FFDD99" mode="Opaque"/><textElement textAlignment="Center" verticalAlignment="Middle"><font fontName="SansSerif" size="12" isBold="false"`
Welcome to TutorialsPoint!	`<staticText><reportElement x="0" y="650" width="150" height="90" forecolor="red" backcolor="#99DDFF" mode="Opaque"/><textElement textAlignment="Right" verticalAlignment="Bottom"><font fontName="SansSerif" size="12" isBold="true"`

Here, we can see that the text "Welcome to TutorialsPoint" is displayed in different font formats.

Unicode Support

In JasperReports, working with texts needs some dedicated tools to process both the character representations and the text formatting properties. Any text can be considered as a character sequence with a particular representation structure. The text appearance consists in both layout (and paragraph) and font settings. But while in most cases, the text layout remains invariant, font settings may change when running the report in different Locales.

We know that different languages need different character sets with respect to specific characters representation. Therefore, working with texts means working with fonts. However, a detailed discussion about how to use fonts in JasperReports is available in the chapter Report Fonts .

One of the main features concerning the text content in a given report is the possibility to internationalize it. It means, we can run the report in different localized environments, using different languages and other localization settings without any hardcoded modification. Character encoding is an important feature when a report is intended to be internationalized.

Character Encoding

A character is the smallest unit of writing conveying a meaningful information. It is an abstract concept, a character does not have a visual appearance. "Uppercase Latin A" is a different character from "lowercase Latin a" and from "uppercase Cyrillic A" and "uppercase Greek Alpha".

A visual representation of a character is known as a ***glyph***. A certain set of glyphs is called a ***font***. "Uppercase Latin A", "uppercase Cyrillic A" and "uppercase Greek Alpha" may have identical glyphs, but they are different characters. At the same time, the glyphs for "uppercase Latin A" can look very different in Times New Roman, Gill Sans and Poetica chancery italic, but they still represent the same character.

The set of available characters is called a *character repertoire*. The location (index) of a given character within a repertoire is known as its code position, or code point. The method of numerically representing a code point within a given repertoire is called the **character encoding**.

Encodings are normally expressed in terms of octets. An octet is a group of eight binary digits, i.e., eight ones and zeros. An octet can express a numeric range between 0 and 255, or between 0x00 and 0xFF, to use hexadecimal notation.

Unicode

A Unicode is a character repertoire that contains most of the characters used in the languages of the world. It can accommodate millions of characters, and already contains hundreds of thousands. Unicode is divided into "planes" of 64K characters. The only one used in most circumstances is the first plane, known as the basic multilingual plane, or BMP.

UTF-8 is the recommended encoding. It uses a variable number of octets to represent different characters.

In a JRXML file, the encoding attribute is specified in the header. It is used at report compilation time to decode the XML content. For instance, if the report contains French words only and characters such as ç, é, â, then the ISO-8859-1 (a.k.a Latin-1) encoding is sufficient −

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
```

As seen above, ideally we can choose the encoding fit to the minimal character set, which can correctly represent all the characters in the document. But in case of Multilanguage documents (i.e. documents containing words spelled in several languages), one should choose the encoding adapted to the minimal character set, able to correctly represent all the characters in the document, even if they belong to different languages. One of the character encodings able to handle multilingual documents is the **UTF-8**, used as default encoding value by JasperReports.

The texts are usually kept in resource bundle files rather than within the document during internationalization. So, there are cases where the JRXML itself looks completely ASCII-compatible, but generated reports at runtime do contain texts unreadable with ASCII. As a result, for a certain type of document export formats (such as CSV, HTML, XHTML, XML, and text) one has to know the encoding for the generated document too. Different languages are supported by different character encodings. So each time, we need to run a report in a localized environment. Further, we have to know, which is the most appropriate character encoding for the generated document language. In this case, the encoding property defined in the JRXML file itself might be no more useful.

To solve this kind of issues we can use an export customer property known as *net.sf.jasperreports.export.character.encoding*. This export custom property is default to UTF-8 and is present in JasperReports.

This default value is set in the *default.jasperreports.properties* file. For more specific options at export time, the CHARACTER_ENCODING export parameter is also available.

Example

To demonstrate using unicode support in Jasperreports, let's write new report template (jasper_report_template.jrxml). **Save it to C:\tools\jasperreports-5.0.1\test** directory. Here, we will display a text in different languages using the Unicode characters (\uXXXX). Any character encoded with UTF-8 can be represented using only its 4-digits hexadecimal code. For instance, the Greek letter Γ can be written as \u0393. When such a notation is encountered, the engine calls for the appropriate character representation in the character set, and only that particular character will be printed out. The contents of the JRXML are as below −

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

   <parameter name = "GreekText" class = "java.lang.String" isForPrompting = "false">
      <defaultValueExpression><![CDATA["\u0394\u03B5\u03BD "+
         "\u03BA\u03B1\u03C4\u03B1\u03BB\u03B1\u03B2\u03B1\u03AF"+
         "\u03BD\u03C9 \u0395\u03BB\u03BB\u03B7\u03BD\u03B9\u03BA\u03AC"]]>
      </defaultValueExpression>
   </parameter>

   <parameter name = "CyrillicText" class = "java.lang.String" isForPrompting = "false">
      <defaultValueExpression><![CDATA["\u042F \u043D\u0435 "+
         "\u043C\u043E\u0433\u0443 \u043F\u043E\u043D\u044F\u0442\u044C "+
         "\u0433\u0440\u0435\u0447\u0435\u0441\u043A\u0438\u0439"]]>
      </defaultValueExpression>
```

```
</parameter>

<parameter name = "ArabicText" class = "java.lang.String" isForPrompting = "false">
   <defaultValueExpression><![CDATA["\u0627\u0646\u0646\u0649 \u0644\u0627 "+
      "\u0627\u0641\u0647\u0645 \u0627\u0644\u0644\u063A\u0629 "+
      "\u0627\u0644\u0639\u0631\u0628\u064A\u0629"]]>
   </defaultValueExpression>
</parameter>

<parameter name = "HebrewText" class = "java.lang.String" isForPrompting = "false">
   <defaultValueExpression><![CDATA["\u05D0\u05E0\u05D9 \u05DC\u05D0 "+
      "\u05DE\u05D1\u05D9\u05DF \u05E2\u05D1\u05E8\u05D9\u05EA"]]>
   </defaultValueExpression>
</parameter>

<title>
   <band height = "782">

      <textField>
         <reportElement x = "0" y = "50" width = "200" height = "60"/>

         <textElement>
            <font fontName = "DejaVu Sans" size = "14"/>
         </textElement>

         <textFieldExpression class = "java.lang.String">
            <![CDATA[$P{GreekText} + "\n" + $P{CyrillicText}]]>
         </textFieldExpression>
      </textField>

      <staticText>
         <reportElement x = "210" y = "50" width = "340" height = "60"/>
         <textElement/>

         <text>
            <![CDATA["GreekText and CyrillicText"]]>
         </text>
      </staticText>

      <textField>
         <reportElement x = "0" y = "120" width = "200" height = "60"/>

         <textElement>
            <font fontName = "DejaVu Sans" size = "14" isBold = "true"/>
         </textElement>

         <textFieldExpression class = "java.lang.String">
            <![CDATA[$P{GreekText} + "\n" + $P{CyrillicText}]]>
         </textFieldExpression>

      </textField>

      <staticText>
         <reportElement x = "210" y = "120" width = "340" height = "60"/>
         <textElement/>
         <text><![CDATA["GreekText and CyrillicText"]]></text>
      </staticText>

      <textField>
         <reportElement x = "0" y = "190" width = "200" height = "60"/>
```

```
            <textElement>
               <font fontName = "DejaVu Sans" size = "14" isItalic = "true"
                  isUnderline = "true"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{GreekText} + "\n" + $P{CyrillicText}]]>
            </textFieldExpression>

         </textField>

         <staticText>
            <reportElement x = "210" y = "190" width = "340" height = "60"/>
            <textElement/>
            <text><![CDATA["GreekText and CyrillicText"]]></text>
         </staticText>

         <textField>
            <reportElement x = "0" y = "260" width = "200" height = "60"/>

            <textElement>
               <font fontName = "DejaVu Sans" size = "14" isBold = "true"
                  isItalic = "true" isUnderline = "true"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{GreekText} + "\n" + $P{CyrillicText}]]>
            </textFieldExpression>

         </textField>

         <staticText>
            <reportElement x = "210" y = "260" width = "340" height = "60"/>
            <textElement/>
            <text><![CDATA["GreekText and CyrillicText"]]></text>
         </staticText>

         <textField>
            <reportElement x = "0" y = "330" width = "200" height = "60"/>

            <textElement textAlignment = "Right">
               <font fontName="DejaVu Sans" size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{ArabicText}]]>
            </textFieldExpression>

         </textField>

         <textField>
            <reportElement x = "210" y = "330" width = "340" height = "60"/>

            <textElement textAlignment = "Right">
               <font fontName = "DejaVu Sans" size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{HebrewText}]]>
            </textFieldExpression>
         </textField>

      </band>
   </title>

</jasperReport>
```

In the above file, we can see the presence of the UTF-8 encoding. Also the localized Unicode pieces of text are stored in document parameters.

The java code to fill and generate the report is as below. Let's save this file **JasperUnicodeReportFill.java** to C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint directory.

```
package com.tutorialspoint;

import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;

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

      try {
         JasperFillManager.fillReportToFile(sourceFileName, null,
            new JREmptyDataSource());
      } catch (JRException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }
}
```

Here we use an instance of *JREmptyDataSource* when filling reports to simulate a data source with one record in it, but with all the fields in this single record being *null*.

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as below.

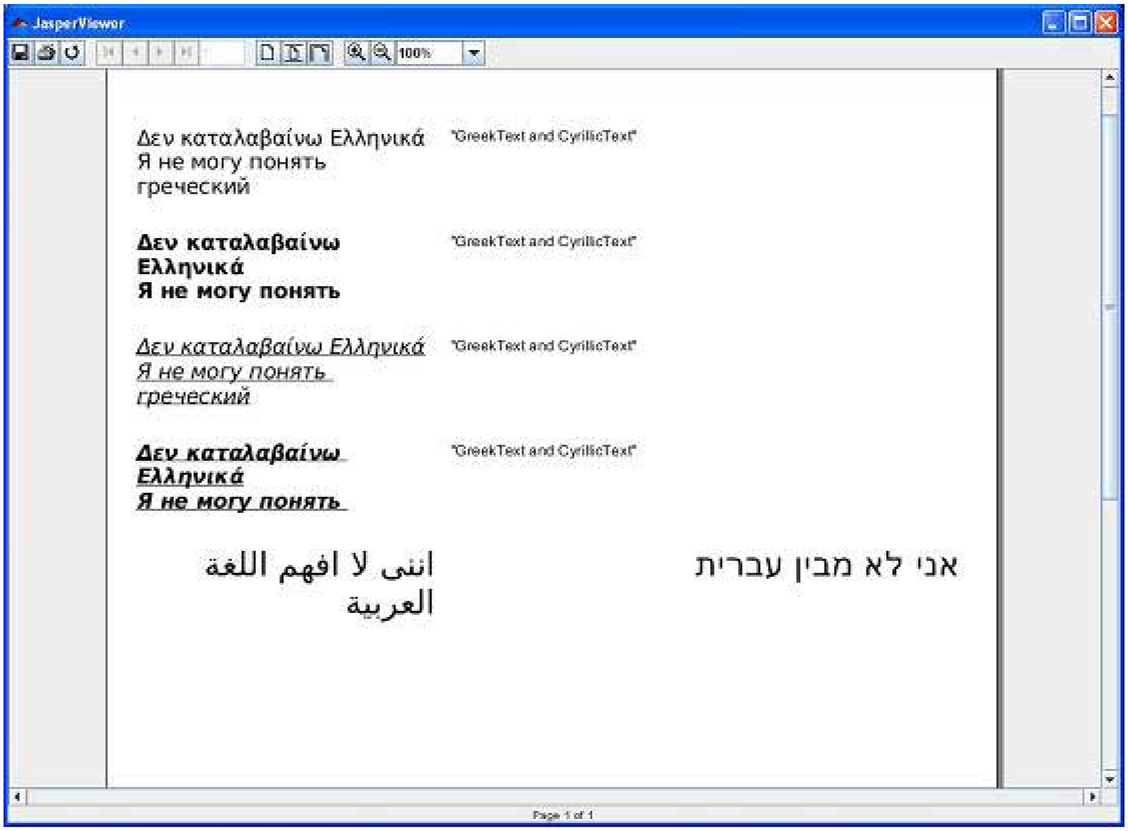
The import file - baseBuild.xml is picked from chapter Environment Setup and should be placed in the same directory as the build.xml.

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

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the report
      stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperUnicodeReportFill** (viewFullReport is the default target) as follows −

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting t
   [javac] Compiling 1 source file to C:\tools\jasperreports-5.0.1\test\classes

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.

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

viewFillReport:
   [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: 4 minutes 1 second
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



Δεν καταλαβαίνω Ελληνικά Я не могу понять греческий "GreekText and CyrillicText"

Δεν καταλαβαίνω Ελληνικά Я не могу понять "GreekText and CyrillicText"

Δεν καταλαβαίνω Ελληνικά Я не могу понять греческий "GreekText and CyrillicText"

Δεν καταλαβαίνω Ελληνικά Я не могу понять "GreekText and CyrillicText"

اننى لا افهم اللغة العربية

אני לא מבין עברית

Here, we can see that the text being displayed is in different languages. Also we see that the languages are grouped together on the same page and also mixed into the same text element.

Report Styles

JasperReports has a feature <style> which helps to control text properties in a report template. This element is a collection of style settings declared at the report level. Properties like foreground color, background color, whether the font is bold, italic, or normal, the font size, a border for the font, and many other attributes are controlled by <style> element. Styles can extend other styles, and add to, or override properties of the parent style as well.

Style Properties

A <style> element has many attributes. Some of the most commonly used are listed in the table given below −

S.NO	Attribute and Description
1	**name** Is mandatory. It must be unique because it references the corresponding report style throughout the report.
2	**isDefault** Indicates whether this style is the document's default style.
3	**style** Is a reference to the parent style.
4	**mode** Specifies the element's transparency. Possible values are *Opaque* and *Transparent*.
5	**forecolor** Is the foreground color of object.
6	**backcolor** Is the background color of object.
7	**fill** Determines the fill pattern used to fill the object. At the moment, the single value allowed is *Solid*.

6	**radius** Specifies the radius of the rectangle's corner arc.
7	**scaleImage** Specifies scale for the images only. Possible values: *Clip, FillFrame, RetainShape, RealHeight,* and *RealSize*.
8	**hAlign** Specifies the horizontal alignment. Possible values: *Left, Center, Right,* and *Justified*.
9	**vAlign** Specifies the vertical alignment. Possible values: *Top, Middle,* and *Bottom*.
10	**rotation** Specifies the element's rotation. Possible values: *None, Left, Right,* and *UpsideDown*.
11	**lineSpacing** Specifies the line spacing between lines of text. Possible values: *Single, 1_1_2, Double*.
12	**markup** Specifies the markup style for styled texts.
13	**fontName** Specifies the font name.
14	**fontSize** Specifies the font size.
15	**isBold** Indicates if the font style is bold.
16	**isItalic**

	Indicates if the font style is italic.
17	**isUnderline** Indicates if the font style is underline.
18	**isStrikeThrough** Indicates if the font style is strikethrough.
19	**pdfFontName** Specifies the related PDF font name.
20	**pdfEncoding** Specifies the character encoding for the PDF output format.
22	**isPdfEmbedded** Indicates if the PDF font is embedded.
23	**pattern** Specifies the format pattern for formatted texts.
24	**isBlankWhenNull** Indicates if an empty string (whitespace) should be shown if the expression evaluates to null.

Conditional Styles

In some situations, a style should be applied only when certain condition is met (for example, to alternate adjacent row colors in a report detail section). This can be achieved using conditional styles.

A conditional style has two elements −

- a Boolean condition expression
- a style

The style is used only if the condition evaluates to *true*.

Applying Styles to Report Elements

Any type of report element can reference a report style definition using the style attribute. Hence, all the style properties declared by the style definition that are applicable to the current element will be inherited. To override the inherited values, style properties specified at the report element level can be used.

Style Templates

We can make a set of reports with a common look by defining the style at a common place. This common style template can then be referenced by the report templates. A style template is an XML file that contains one or more style definitions. Style template files used by convention the ***.jrtx** extension, but this is not mandatory.

A style template contains following elements −

<jasperTemplate> − This is the root element of a style template file.

<template> − This element is used to include references to other template files. The contents of this element are interpreted as the location of the referred template file.

<style> − This element is identical to the element with the same name from report design templates (JRXML files), with the exception that a style in a style template cannot contain conditional styles. This limitation is caused by the fact that conditional styles involve report expressions, and expressions can only be interpreted in the context of a single report definition.

References to style templates are included in JRXML reports as <template> elements. The style templates are loaded at report fill time, and style name references are resolved once all the templates have been loaded. When loading style templates and resolving style names to styles, a tree/graph of style templates is created, the top of the tree being the set of styles defined in the report. On this tree, style name references are resolved to the last style that matches the name in a depth-first traversal.

Example

Let's try out the conditional styles and style templates. Let's add the <**style**> element **alternateStyle** to our existing report template (Chapter Report Designs). Based on the condition, font color changes to blue for even count. We have also included a style template **"styles.jrtx"**. The revised report template (jasper_report_template.jrxml) is as follows. Save it to C:\tools\jasperreports-5.0.1\test directory −

```
<?xml version = "1.0"?>
<!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" pageWidth = "595"
   pageHeight = "842" columnWidth = "515"
   leftMargin = "40" rightMargin = "40" topMargin = "50" bottomMargin = "50">

   <template>"styles.jrtx"</template>

   <style name = "alternateStyle" fontName = "Arial" forecolor = "red">
      <conditionalStyle>
         <conditionExpression>
            <![CDATA[new Boolean($V{countNumber}.intValue() % 2 == 0)]]>
         </conditionExpression>

         <style forecolor = "blue" isBold = "true"/>
      </conditionalStyle>
   </style>

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

   <variable name = "countNumber" class = "java.lang.Integer" calculation = "Count">
      <variableExpression><![CDATA[Boolean.TRUE]]></variableExpression>
   </variable>

   <title>
      <band height = "70">

         <line>
            <reportElement x = "0" y = "0" width = "515" height = "1"/>
         </line>

         <textField isBlankWhenNull = "true" bookmarkLevel = "1">
            <reportElement x = "0" y = "10" width = "515" height = "30"/>

            <textElement textAlignment = "Center">
               <font size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>

            <anchorNameExpression><![CDATA["Title"]]></anchorNameExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "40" width = "515" height = "20"/>

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>

         </textField>

      </band>
   </title>

   <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 style = "alternateStyle" 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"
               style = "Strong"/>
            <textElement textAlignment = "Center" verticalAlignment = "Middle" />

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$F{name}]]>
            </textFieldExpression>
         </textField>

      </band>
   </detail>

</jasperReport>
```

The contents of style template **styles.jrtx** are as follows. Save it to C:\tools\jasperreports-5.0.1\test directory.

```
<?xml version = "1.0"?>

<!DOCTYPE jasperTemplate PUBLIC "-//JasperReports//DTD Template//EN"
  "http://jasperreports.sourceforge.net/dtds/jaspertemplate.dtd">

<jasperTemplate>
   <style name = "Strong" isBold = "true" pdfFontName = "Helvetica-Bold"
      backcolor = "lightGray forecolor = "green"/>
</jasperTemplate>
```

The java codes for report filling remain unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below –

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as below –

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as below −

```
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;
   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

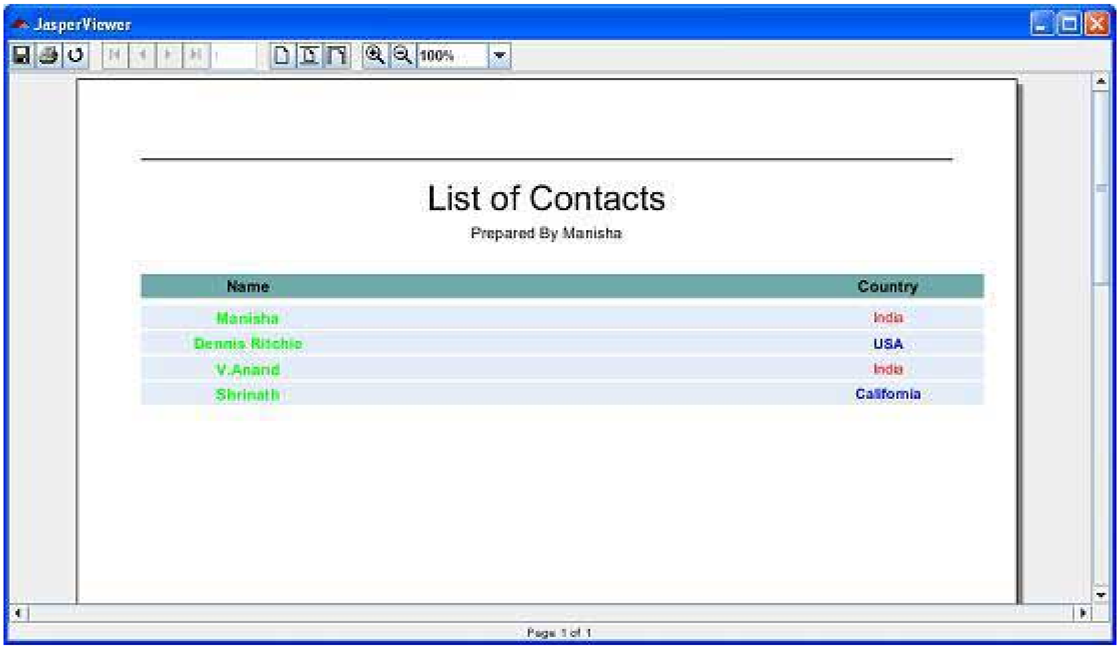
The import file - baseBuild.xml is picked up from the chapter Environment Setup and should be placed in the same directory as the build.xml.

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

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the
      report stored in the .JRprint file.">
      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport is the default target) as −

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28: warning:
   'includeantruntime' was not set, defaulting to build.sysclasspath=last;
   set to false for repeatable builds
   [javac] Compiling 3 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.

viewFillReport:
   [java] log4j:WARN No appenders could be found for logger
   (net.sf.jasperreports.extensions.ExtensionsEnvironment).
   [java] log4j:WARN Please initialize the log4j system properly.
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



List of Contacts

Prepared By Manisha

Name	Country
Manisha	India
Dennis Ritchie	USA
V.Anand	India
Shrinath	California

Here, we can see that the color of the font is changed to blue for even count (in column country). In the column name, the font color is changed to green (this style is referenced from the style template).

Report Scriptlets

We have seen in our previous chapters, data displayed on the report is usually fetched from report parameters and report fields. This data can be processed using the report variables and their expressions. There are situations when a complex functionality cannot be achieved easily using report expressions or variables. Examples of this may be complex String manipulations, building of Maps, or Lists of objects in memory or manipulations of dates using 3rd party Java APIs. For such situations, JasperReports provides us with a simple and powerful means of doing this with **Scriptlets**.

Scriptlets are sequences of Java code that are executed every time a report event occurs. Values of report variables can be affected through scriptlets.

Scriptlet Declaration

We can declare a scriptlet in two ways −

- Using <**scriptlet**> element. This element has *name* attribute and *class* attribute. The *class* attribute should specify the name of the class, which extends *JRAbstractScriptlet* class. The class must be available in the classpath at report filling time and must have an empty constructor, so that the engine can instantiate it on the fly.

- Using the attribute **scriptletClass** of the element <**jasperReport**>, in the report template (JRXML). By setting this attribute with fully qualified name of scriptlet (including the entire package name), we indicate that we want to use a scriptlet. The scriptlet instance, created with this attribute, acts like the first scriptlet in the list of scriptlets and has the predefined name REPORT.

Scriptlet class

A scriptlet is a java class, which must extend either of the following classes −

- **net.sf.jasperreports.engine.JRAbstractScriptlet** − This class contains a number of abstract methods that must be overridden in every implementation. These methods are called automatically by JasperReports at the appropriate moment. Developer must implement all the abstract methods.

- **net.sf.jasperreports.engine.JRDefaultScriptlet** − This class contains default empty implementations of every method in JRAbstractScriptlet. A developer is only required to implement those methods he/she needs for his/her project.

The following table lists the methods in the above class. These methods will be called by the report engine at the appropriate time, during report filling phase.

S.NO	Method and Description
1	**public void beforeReportInit()** Called before report initialization.
2	**public void afterReportInit()** Called after report initialization.

3	**public void beforePageInit()** Called before each page is initialized.
4	**public void afterPageInit()** Called after each page is initialized.
5	**public void beforeColumnInit()** Called before each column is initialized.
6	**public void afterColumnInit()** Called after each column is initialized.
7	**public void beforeGroupInit(String groupName)** Called before the group specified in the parameter is initialized.
8	**public void afterGroupInit(String groupName)** Called after the group specified in the parameter is initialized.
9	**public void beforeDetailEval()** Called before each record in the detail section of the report is evaluated.
10	**public void afterDetailEval()** Called after each record in the detail section of the report is evaluated.

Any number of scriptlets can be specified per report. If no scriptlet is specified for a report, the engine still creates a single JRDefaultScriptlet instance and registers it with the built-in REPORT_SCRIPTLET parameter.

We can add any additional methods that we need to our scriptlets. Reports can call these methods by using the built-in parameter REPORT_SCRIPTLET.

Global Scriptlets

We can associate scriptlets in another way to reports, which is by declaring the scriptlets globally. This makes the scriptlets apply to all reports being filled in the given JasperReports deployment. This is made easy by the fact that scriptlets can be added to JasperReports as extensions. The scriptlet extension point is represented by the *net.sf.jasperreports.engine.scriptlets.ScriptletFactory* interface. JasperReports will load all scriptlet factories available through extensions at runtime. Then, it will ask each one of them for the list of scriptlets instances that they want to apply to the current report that is being run. When asking for the list of scriptlet instances, the engine gives some context information that the factory could use in order to decide, which scriptlets actually apply to the current report.

Report Governors

Governors are just an extension of global scriptlets that enable us to tackle a problem of report engine entering infinite loop at runtime, while generating reports. Invalid report templates cannot be detected at design time, because most of the time, the conditions for entering the infinite loops depend on the actual data that is fed into the engine at runtime. Report Governors help in deciding whether a certain report has entered an infinite loop and they can stop it. This prevents resource exhaustion for the machine that runs the report.

JasperReports has two simple report governors that would stop a report execution based on a specified maximum number of pages or a specified timeout interval. They are −

- **net.sf.jasperreports.governors.MaxPagesGovernor** − This is a global scriptlet that is looking for two configuration properties to decide if it applies or not to the report currently being run. The configuration properties are −

    - net.sf.jasperreports.governor.max.pages.enabled=[true|false]

    - net.sf.jasperreports.governor.max.pages=[integer]

- **net.sf.jasperreports.governors.TimeoutGovernor**− This is also a global scriptlet that is looking for the following two configuration properties to decide if it applies or not.

    The configuration properties are −

    - net.sf.jasperreports.governor.timeout.enabled=[true|false]

    - net.sf.jasperreports.governor.timeout=[milliseconds]

The properties for both governors can be set globally, in the jasperreports.properties file, or at report level, as custom report properties. This is useful because different reports can have different estimated size or timeout limits and also because you might want turn on the governors for all reports, while turning it off for some, or vice-versa.

Example

Let's write a scriptlet class (**MyScriptlet**). The contents of file C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\MyScriptlet.java are as follows −

```
package com.tutorialspoint;

import net.sf.jasperreports.engine.JRDefaultScriptlet;
import net.sf.jasperreports.engine.JRScriptletException;


public class MyScriptlet extends JRDefaultScriptlet {

   public void afterReportInit() throws JRScriptletException{
      System.out.println("call afterReportInit()");
      // this.setVariableValue("AllCountries", sbuffer.toString());
      this.setVariableValue("someVar", new String("This variable value
         was modified by the scriptlet."));
   }

   public String hello() throws JRScriptletException {
      return "Hello! I'm the report's scriptlet object.";
   }

}
```

Details of the above scriptlet class are as follows –

In the *afterReportInit* method, we set a value to the variable **"someVar"** this.setVariableValue("someVar", new String("This variable value was modified by the scriptlet.")).

At the end of the class, an extra method called **'hello'** has been defined. This is an example of a method that can be added to the Scriptlet that actually returns a value, rather than setting a Variable.

Next, we will add the scriptlet class reference in our existing report template (Chapter Report Designs). The revised report template (jasper_report_template.jrxml) are as follows. Save it to C:\tools\jasperreports-5.0.1\test directory –

```
<?xml version = "1.0"?>
<!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" pageWidth = "595"
   pageHeight = "842" columnWidth = "515"
   leftMargin = "40" rightMargin = "40" topMargin = "50" bottomMargin = "50"
   scriptletClass = "com.tutorialspoint.MyScriptlet">

   <style name = "alternateStyle" fontName = "Arial" forecolor = "red">

      <conditionalStyle>
         <conditionExpression>
            <![CDATA[new Boolean($V{countNumber}.intValue() % 2 == 0)]]>
         </conditionExpression>
         <style forecolor = "blue" isBold = "true"/>
      </conditionalStyle>
   </style>

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <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>

   <variable name = "countNumber" class = "java.lang.Integer"
      calculation = "Count">
      <variableExpression><
         ![CDATA[Boolean.TRUE]]>
      </variableExpression>
   </variable>

   <variable name = "someVar" class = "java.lang.String">
      <initialValueExpression>
        <![CDATA["This is the initial variable value."]]>
      </initialValueExpression>
   </variable>

   <title>
      <band height = "100">

         <line>
            <reportElement x = "0" y = "0" width = "515" height = "1"/>
         </line>

         <textField isBlankWhenNull = "true" bookmarkLevel = "1">
            <reportElement x = "0" y = "10" width = "515" height = "30"/>

            <textElement textAlignment = "Center">
              <font size = "22"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
              <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>

            <anchorNameExpression>
               <![CDATA["Title"]]>
            </anchorNameExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "40" width = "515" height = "20"/>
            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "50" width = "515"
               height = "30" forecolor = "#993300"/>

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$V{someVar}]]>
            </textFieldExpression>

         </textField>

      </band>
   </title>

   <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 style = "alternateStyle" 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>

   <summary>
      <band height = "45">

         <textField isStretchWithOverflow = "true">
            <reportElement x = "0" y = "10" width = "515" height = "15" />
            <textElement textAlignment = "Center"/>
            <textFieldExpression class = "java.lang.String">
               <![CDATA["There are " + String.valueOf($V{REPORT_COUNT}) +
                  " records on this report."]]>
            </textFieldExpression>
         </textField>

         <textField isStretchWithOverflow = "true">
            <reportElement positionType = "Float" x = "0" y = "30" width = "515"
               height = "15" forecolor = "# 993300" />

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{REPORT_SCRIPTLET}.hello()]]>
            </textFieldExpression>

         </textField>

      </band>
   </summary>

</jasperReport>
```

The details of the revised report template is given below −

We have referenced the MyScriptlet class in the attribute *scriptletClass* of <jasperReport> element.

Scriptlets can only access, but not modify the report fields and parameters. However, scriptlets can modify report variable values. This can be accomplished by calling the setVariableValue() method. This method is defined in JRAbstractScriptlet class, which is always the parent class of any scriptlet. Here, we have defined a variable *someVar*, which will be modified by the MyScriptlet to have the value *This value was modified by the scriptlet*.

The above report template has a method call in the Summary band that illustrates how to write new methods (in scriptlets) and use them in the report template. (**$P{REPORT_SCRIPTLET}.hello()**)

The java codes for report filling remain unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below −

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as given below –

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as given below –

```
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;
   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

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

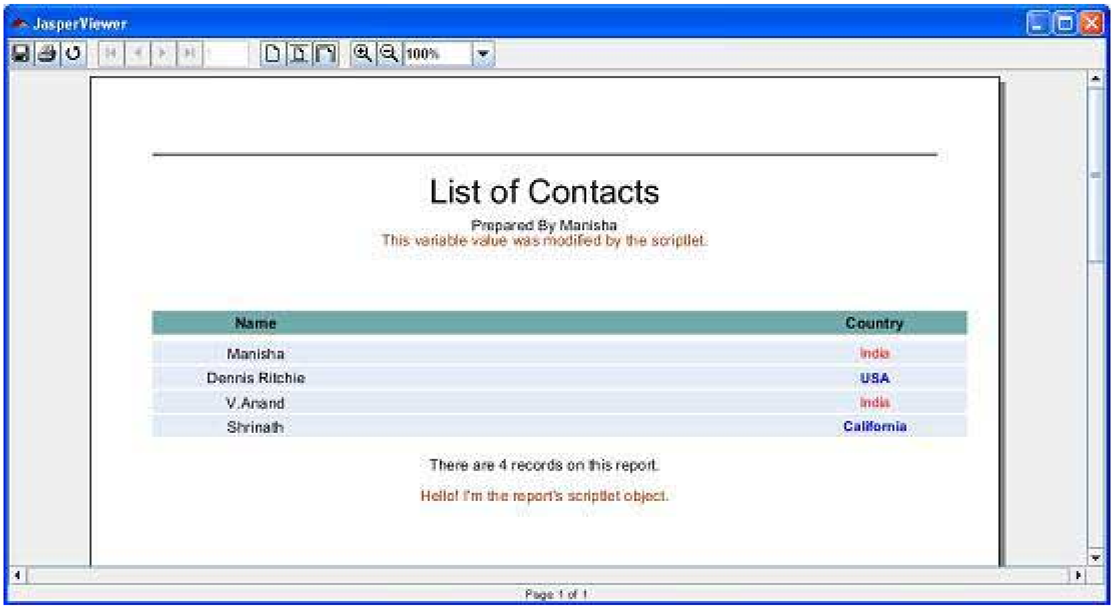
```
<?xml version = "1.0" encoding = "UTF-8"?>
<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview
      the report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport is the default target) as –

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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting to bu
   [javac] Compiling 4 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.
   [java] call afterReportInit()
   [java] call afterReportInit()
```

```
viewFillReport:
   [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: 18 minutes 49 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



List of Contacts

Prepared By Manisha
This variable value was modifed by the scriptlet.

Name	Country
Manisha	India
Dennis Ritchie	USA
V.Anand	India
Shrinath	California

There are 4 records on this report.

Hello! I'm the report's scriptlet object.

Here we see two messages are displayed from MyScriptlet class −

In title section − *This variable value was modified by the scriptlet*

At the bottom − *Hello! I'm the report's scriptlet object.*

Create SubReports

Subreports are one of the nice features of the JasperReports. This feature allows incorporating a report within another report, that is, one report can be a subreport of another. Subreports help us keep report designs simple, as we can create many simple reports and encapsulate them into a master report. Subreports are compiled and filled just like normal reports. Any report template can be used as a subreport when incorporated into another report template, without anything changed inside (of the report template).

Subreports are like normal report templates. They are in fact *net.sf.jasperreports.engine.JasperReport* objects, which are obtained after compiling a *net.sf.jasperreports.engine.design.JasperDesign object*.

<subreport> Element

A <subreport> element is used when introducing subreports into master reports. Here is the list of sub-elements in the <subreport> JRXML element.

<reportElement>

<parametersMapExpression> − This is used to pass a map containing report parameters to the subreport. The map is usually obtained from a parameter in the master report, or by using the built-in REPORTS_PARAMETERS_MAP parameter to pass the parent report's parameters to the subreport. This expression should always return a *java.util.Map* object in which the keys are the parameter names.

<subreportParameter> − This element is used to pass parameters to the subreport. It has an attribute *name*, which is mandatory.

<connectionExpression > − This is used to pass a *java.sql.Connection* to the subreport. It is used only when the subreport template needs a database connection during report filling phase.

<dataSourceExpression> − This is used to pass a datasource to the subreport. This datasource is usually obtained from a parameter in the master report or by using the built-in REPORT_DATA_SOURCE parameter to pass the parent report's datasource to the subreport.

The elements (*connectionExpression and dataSourceExpression*) cannot be present at the same time in a <subreport> element declaration. This is because we cannot supply both a data source and a connection to the subreport. We must decide on one of them and stick to it.

<returnValue> − This is used to assign the value of one of the subreport's variables to one of the master report's variables. This sub element has attributes as follows −

subreportVariable − This attribute specifies the name of the subreport variable whose value is to be returned.

toVariable − This attribute specifies the name of the parent report variable whose value is to be copied/incremented with the value from the subreport.

calculation − This attribute can take values : Nothing, Count, DistinctCount, Sum, Average, Lowest, Highest, StandardDeviation, Variance. Default value for attribute *calculation* is "Nothing".

incrementerFactoryClass − This attribute specifies the factory class for creating the incrementer instance.

<subreportExpression> − This indicates where to find the compiled report template for the subreport. This element has a **class** attribute. The *class* attribute

can take any of these values:java.lang.String, java.io.File, java.net.URL, java.io.InputStream, net.sf.jasperreports.engine.JasperReport. Default value is *java.lang.String*.

isUsingCache − This is an attribute of the <subreport> element. This is a Boolean, when set to *true*, the reporting engine will try to recognize previously loaded subreport template objects, using their specified source. This caching functionality is available only for subreport elements that have expressions returning java.lang.String objects as the subreport template source, representing file names, URLs, or classpath resources.

Example

Let take up a simple example to demonstrate creation of subreports using JRDataSource. Let's first write two new report templates, one being subreport and the other Master report. The contents of the subreport (address_report_template.jrxml) template is as given below. Save it to C:\tools\jasperreports-5.0.1\test directory.

```
<?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 = "address_report_template" pageWidth = "175" pageHeight = "842"
   columnWidth = "175" leftMargin = "0" rightMargin = "0"
   topMargin = "0" bottomMargin = "0">

   <field name = "city" class = "java.lang.String"/>
   <field name = "street" class = "java.lang.String"/>

   <background>
      <band splitType = "Stretch"/>
   </background>

   <title>
      <band height = "20" splitType = "Stretch">

         <staticText>
            <reportElement x = "0" y = "0" width = "100" height = "20"/>

            <textElement>
               <font size = "14" isBold = "true"/>
            </textElement>

            <text><![CDATA[Addresses]]></text>
         </staticText>

      </band>
   </title>

   <pageHeader>
      <band height = "12" splitType = "Stretch"/>
   </pageHeader>

   <columnHeader>
      <band height = "12" splitType = "Stretch"/>
   </columnHeader>

   <detail>
      <band height = "27" splitType = "Stretch">

         <textField>
            <reportElement x = "0" y = "0" width = "120" height = "20"/>

            <textElement>
               <font size = "12" isBold = "true"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$F{city}+" Address:"]]>
            </textFieldExpression>
         </textField>

         <textField isStretchWithOverflow = "true">
            <reportElement x = "120" y = "0" width = "435" height = "20"/>

            <textElement>
               <font size = "12"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$F{street}]]>
            </textFieldExpression>
         </textField>

      </band>
   </detail>

   <columnFooter>
      <band height = "8" splitType = "Stretch"/>
   </columnFooter>

   <pageFooter>
      <band height = "11" splitType = "Stretch"/>
   </pageFooter>

   <summary>
      <band height = "9" splitType = "Stretch"/>
   </summary>

</jasperReport>
```

As we use a data source, we need to write a corresponding POJO file **SubReportBean.java** as shown below. Save it to directory C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint –

```
package com.tutorialspoint;

public class SubReportBean {
   private String city;
   private String street;
   public String getCity() {
      return city;
   }

   public void setCity(String city) {
      this.city = city;
   }

   public String getStreet() {
      return street;
   }

   public void setStreet(String street) {
      this.street = street;
   }
}
```

Here, we have declared two fields *'city'* and *'street'* and respective getter and setter methods are defined.

Now, let's update our existing **DataBean** file. We will add a new field *subReportBeanList*, which is a java.util.List. This field will hold the list of SubReportBean objects. The contents of the file DataBean are as below. Save it to directory C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint.

```
package com.tutorialspoint;

import java.util.List;

public class DataBean {
   private String name;
   private String country;
   private List<SubReportBean> subReportBeanList;

   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;
   }

   public List<SubReportBean> getSubReportBeanList() {
      return subReportBeanList;
   }

   public void setSubReportBeanList(List<SubReportBean> subReportBeanList) {
      this.subReportBeanList = subReportBeanList;
   }
}
```

Let's now update the file C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java. The contents of this file are as −

```
package com.tutorialspoint;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class DataBeanList {
   public ArrayList<DataBean> getDataBeanList() {

      // Create sub report data
      SubReportBean subBean1 = new SubReportBean();
      subBean1.setCity("Mumbai");
      subBean1.setStreet("M.G.Road");
      SubReportBean subBean2 = new SubReportBean();
      subBean2.setCity("New York");
      subBean2.setStreet("Park Street");
      SubReportBean subBean3 = new SubReportBean();
      subBean3.setCity("San Fransisco");
      subBean3.setStreet("King Street");

      ArrayList<DataBean> dataBeanList = new ArrayList<DataBean>();

      // Create master report data
      dataBeanList.add(produce("Manisha", "India",
         Arrays.asList(subBean1)));
      dataBeanList.add(produce("Dennis Ritchie", "USA",
         Arrays.asList(subBean2)));
      dataBeanList.add(produce("V.Anand", "India",
         Arrays.asList(subBean1)));
      dataBeanList.add(produce("Shrinath", "California",
         Arrays.asList(subBean3)));

      return dataBeanList;
   }

   /*
    * This method returns a DataBean object,
    * with name, country and sub report
    * bean data set in it.
    */
   private DataBean produce(String name, String country,
      List<SubReportBean> subBean) {
      DataBean dataBean = new DataBean();

      dataBean.setName(name);
      dataBean.setCountry(country);
      dataBean.setSubReportBeanList(subBean);

      return dataBean;
   }
}
```

In the method produce() in the above file, we are setting the list of SubReportBean.

Now, let's write a new master report template (jasper_report_template.jrxml). Save this file to directory **C:\tools\jasperreports-5.0.1\test**. The contents for this file are as below −

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

   <parameter name = "SUBREPORT_DIR" class = "java.lang.String" isForPrompting = "false">
      <defaultValueExpression>
         <![CDATA["C:\\tools\\jasperreports-5.0.1\\test\\"]]>
      </defaultValueExpression>
   </parameter>

   <field name = "country" class = "java.lang.String"/>
   <field name = "name" class = "java.lang.String"/>
   <field name = "subReportBeanList" class = "java.util.List"/>

   <background>
      <band splitType = "Stretch"/>
   </background>

   <title>
      <band height = "35" splitType = "Stretch">

         <staticText>
            <reportElement x = "0" y = "0" width = "204" height = "34"/>

            <textElement>
               <font size = "26" isBold = "true"/>
            </textElement>

            <text><![CDATA[Contact Report]]></text>
         </staticText>

      </band>
   </title>

   <pageHeader>
      <band height = "17" splitType = "Stretch"/>
   </pageHeader>

   <columnHeader>
      <band height = "21" splitType = "Stretch"/>
   </columnHeader>

   <detail>
      <band height = "112" splitType = "Stretch">

         <staticText>
            <reportElement x = "0" y = "0" width = "100" height = "20"/>

            <textElement>
               <font size = "12" isBold = "true"/>
            </textElement>

            <text><![CDATA[Name:]]></text>
         </staticText>

         <staticText>
            <reportElement x = "0" y = "20" width = "100" height = "20"/>

            <textElement>
               <font size = "12" isBold = "true"/>
            </textElement>

            <text><![CDATA[Country:]]></text>
         </staticText>

         <textField>
            <reportElement x = "104" y = "0" width = "277" height = "20"/>

            <textElement>
               <font size = "12"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$F{name}]]>
            </textFieldExpression>
         </textField>

         <textField>
            <reportElement x = "104" y = "20" width = "277" height = "20"/>

            <textElement>
               <font size = "12"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$F{country}]]>
            </textFieldExpression>
         </textField>

         <subreport>
            <reportElement positionType = "Float" x = "335" y = "25" width = "175"
               height = "20" isRemoveLineWhenBlank = "true" backcolor = "#99ccff"/>

            <dataSourceExpression>
               new net.sf.jasperreports.engine.data.JRBeanCollectionDataSource
                  ($F{subReportBeanList})
            </dataSourceExpression>

            <subreportExpression class = "java.lang.String">
               <![CDATA[$P{SUBREPORT_DIR} + "address_report_template.jasper"]]>
            </subreportExpression>
         </subreport>

         <line>
            <reportElement x = "0" y = "50" width = "550" height = "1"/>
         </line>

      </band>
   </detail>

   <columnFooter>
```

```
      <band height = "19" splitType = "Stretch"/>
   </columnFooter>

   <pageFooter>
      <band height = "18" splitType = "Stretch"/>
   </pageFooter>

   <summary>
      <band height = "14" splitType = "Stretch"/>
   </summary>

</jasperReport>
```

In the above template, we have defined a new parameter "SUBREPORT_DIR," which defines the path of the subreport. We have defined a field *subReportBeanList* of type *java.util.List,* which corresponds to property in the file DataBean. The element <subreport> has sub-element <dataSourceExpression>. We have put the list *subReportBeanList* in an instance of JRBeanCollectionDataSource. In the sub-element <subreportExpression/>, we have given the subreport name (AddressReport.jasper).

Now, let's write a new class **CreateReport** to compile and execute our report template. The contents of file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\CreateReport.java** are as given below –

```
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.JasperCompileManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

public class CreateReport {

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

      DataBeanList DataBeanList = new DataBeanList();
      ArrayList<DataBean> dataList = DataBeanList.getDataBeanList();
      JRBeanCollectionDataSource beanColDataSource = new
         JRBeanCollectionDataSource(dataList);

      try {
         /* Compile the master and sub report */
         JasperReport jasperMasterReport = JasperCompileManager
            .compileReport(masterReportFileName);
         JasperReport jasperSubReport = JasperCompileManager
```

```
            .compileReport(subReportFileName);

         Map<String, Object> parameters = new HashMap<String, Object>();
         parameters.put("subreportParameter", jasperSubReport);
         JasperFillManager.fillReportToFile(jasperMasterReport,
            destFileName, parameters, beanColDataSource);

      } catch (JRException e) {

         e.printStackTrace();
      }
      System.out.println("Done filling!!! ...");
   }
}
```

Here, we see that we are compiling both the master and sub report templates and passing the master report (.jasper) file for the report filling.

Report Generation

Now, all our files are ready, let's compile and execute them using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

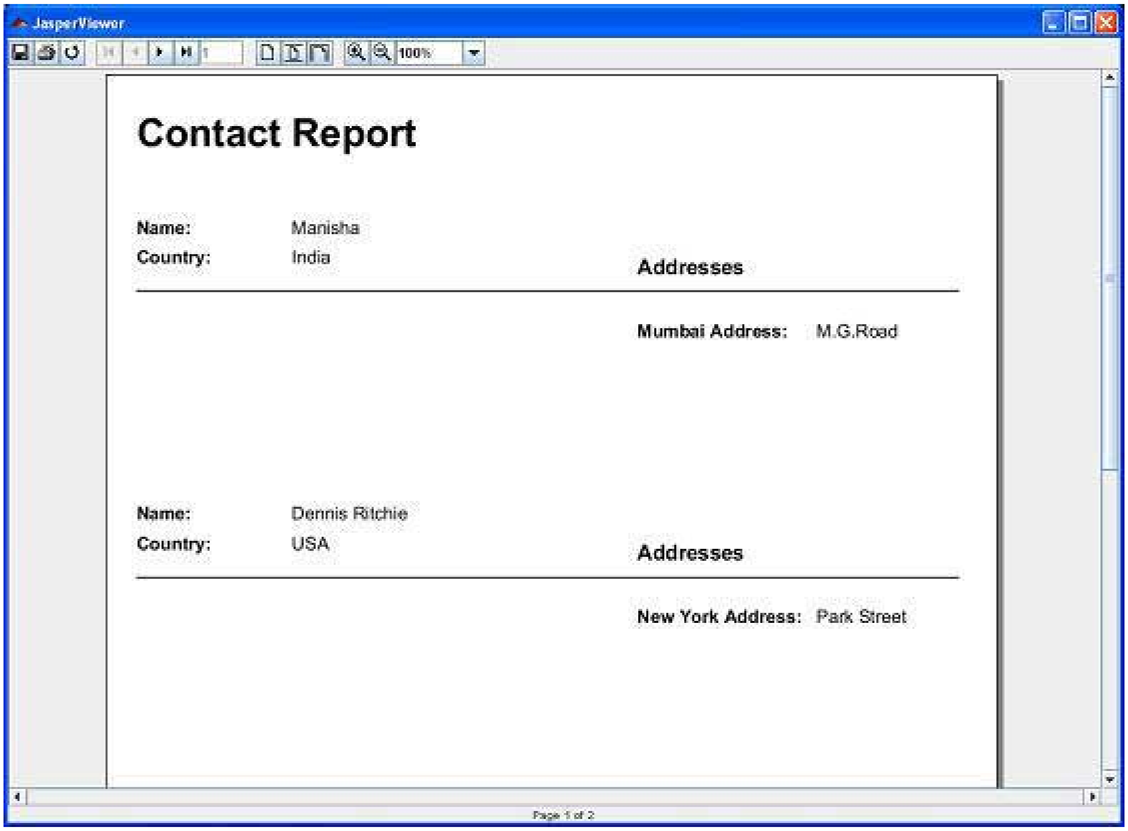
The import file - baseBuild.xml is picked up from the chapter Environment Setup and should be placed in the same directory as the build.xml.

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

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the
      report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.CreateReport** (viewFullReport is the default target) as follows –

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


clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes


compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
      warning: 'includeantruntime' was not set, defaulting to
   [javac] Compiling 7 source files to C:\tools\jasperreports-5.0.1\test\classes


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.


run:
   [echo] Runnin class : com.tutorialspoint.CreateReport
   [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 filling!!! ...


viewFillReport:
   [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: 72 minutes 13 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



Contact Report

Name: Manisha
Country: India

Addresses

Mumbai Address: M.G.Road

Name: Dennis Ritchie
Country: USA

Addresses

New York Address: Park Street

Here, we can see that the attributes Name, Country, and Address are displayed.

Creating Charts

Earlier people had to rely on scriptlets to gather the chart data and render the chart using an image element in the report template. JasperReports makes it simple now, as it has a built-in support for charts using the new chart component.

Using a new chart component, user needs to apply only the visual settings and define expressions that will help build the chart dataset. JasperReports uses JFreeChart as the underlying charting library. When configuring a new chart component, following three components are involved −

The overall chart component.

The chart dataset (which groups chart data-related settings).

The chart plot (which groups visual settings related to the way the chart items are rendered).

JasperReports currently supports the following types of charts: Pie, Pie 3D, Bar, Bar 3D, XY Bar, Stacked Bar, Stacked Bar 3D, Line, XY Line, Area, XY Area, Stacked Area, Scatter, Bubble, Time Series, High-Low-Open-Close, Candlestick, Multiple Axis, Meter, Thermometer, and Gantt.

Chart Properties

Charts are normal report elements, so they share some of their properties with all the other report elements. There is a JRXML element called <**chart**>, used to create all type of charts. This element groups special chart-specific settings that apply to all types of charts.

Chart Sub-Elements

The sub-elements of <chart> element are −

<reportElement> − These are displayable objects like static texts, text fields, images, lines, and rectangles that you put in your report template sections.

<Box> − This element is used to surround charts by a border that's customizable on each side.

<chartTitle> − This element is used to place the title of the chart. The *position* attribute decides the title position of the chart in the report. This element has attributes - **Position** (Values could be *Top*, *Bottom*, *Left*, *Right*. Deafult value is *Top*), **color**. <chartTitle> has *font* and *titleExpression* as subelements.

<chartSubtitle> − This element is used to place the subtitle of the chart. This element has attribute - **color**. <chartSubtitle> has *font* and *subtitleExpression* as subelements.

<chartLegend> − The element can control the font-related properties as well as the text color and the background color of the chart legend using this element. This element has attributes - **textColor** and **backgroundColor**.

<anchorNameExpression> − This element creates the target for the anchor.

<hyperlinkReferenceExpression> − This element contains a report expression indicating the name of the external resource (usually a URL).

<hyperlinkAnchorExpression> − Hyperlink points to an anchor in an external resource.

<hyperlinkPageExpression> − Hyperlink points to a page in the current report.

<hyperlinkTooltipExpression> − This element controls the ToolTip of hyperlink. The type of the expression should be *java.lang.String*.

<hyperlinkParameter> − This element when present generates a final hyperlink depending on the parameter values.

Chart attributes

Attributes in the <chart> element available for all chart types are −

isShowLegend – This attribute is used to determine, if a chart legend will be displayed on the report. Values could be *true*, or *false*. Default value is *true*.

evaluationTime – Determines when the chart's expression will be evaluated. Values could be *Now*, *Report*, *Page*, *Column*, *Group*, *Band*. Default value is *Now*.

evaluationGroup – This attribute determines the name of the group to be used to evaluate the chart's expressions. The value for this attribute must match the name of the group, we would like to use as the chart's evaluation group.

hyperlinkType – This attribute can hold any text value. Default value is *None*. This means, neither the text fields nor the images represent hyperlinks, even if the special hyperlink expressions are present.

hyperlinkTarget – This attribute helps to customize the behavior of the specified link when it is clicked in the viewer. Values could be *Self*, or *Blank*. Default value is *Self*.

bookmarkLevel – This attribute when set to a positive integer, generates bookmarks in the reports exported to PDF. Default value is *0*.

customizerClass – This is the name of a class (optional) that can be used to customize the chart. The value for this element must be a String containing the name of a customizer class.

Chart customization

As mentioned above, JasperReports uses *JFreeChart* as the underlying charting library. *JFreeChart* contains features that are directly not supported by JasperReports. We can take advantage of these features by supplying a customizer class via the *customizerClass* attribute in <chart> element. A customizer class is nothing, but an implementation of the *net.sf.jasperreports.engine.JRChartCustomizer* interface. The easiest way to implement this interface is by extending the *net.sf.jasperreports.engine.JRAbstractChartCustomizer* class and thus having access to parameters, fields, and variables for more flexible chart customization based on report data.

Chart Datasets

One of the common properties across all chart types is <**dataset**> element. Chart datasets help mapping report data and retrieving chart data at runtime. Each chart type contains different sub-elements to define chart's expressions. These expressions define the data used to generate the chart. All of these sub-elements contain a <dataset> element that defines when the chart's expressions are evaluated and reset.

Several types of chart datasets are available in JasperReports because each type of chart works with certain datasets: Pie, Category, XY, Time Series, Time Period, XYZ, and High-

Low. Each of these dataset types implements *net.sf.jasperreports.engine.JRChartDataset* interface that define chart datasets. All chart datasets initialize and increment in the same way; however, they differ only in the type of data or data series they map.

Dataset Properties

Table given below summarizes the attributes of the element <dataset> −

Attribute	Description	Values
resetType	This attribute determines when the value of the chart expression is to be reset.	None, Report, Page, Column, Group. Default value is **Report**.
resetGroup	This attribute determines the name of the group at which the chart expression value is reset.	The value for this attribute must match the name of any group declared in the JRXML report template.
incrementType	This attribute determines when to recalculate the value of the chart expression.	None, Report, Page, Column, Group. Default value is **"None"**.
incrementGroup	This attribute determines the name of the group at which the chart expression is recalculated.	The value for this attribute must match the name of a group declared in the JRXML report template.

The following table summarizes the sub-elements of the element <dataset> −

Sub element	Description
<incrementWhenExpression>	The way a chart dataset is incremented can be customized by filtering out unwanted data through the use of this sub element.
<datasetRun>	This contains information required to instantiate a report subdataset.

Dataset Types

Specific dataset types are explained below −

Pie Dataset

A pie dataset is characterized by the following expressions −

- <keyExpression> − represents the categories that will make up the slices in the pie chart. This expression can return any java.lang.Comparable object.

<valueExpression> − produces the values that correspond to each category/key in the dataset. Values are always java.lang.Number objects.

<labelExpression> − if this expression is missing, the chart will display default labels for each slice in the pie chart. Use this expression, which returns java.lang.String values, to customize the item labels for the pie chart.

<sectionHyperlink> − sets hyperlinks associated with the pie sections.

Category Dataset

A category dataset is characterized by the <categorySeries> element, which contains −

<seriesExpression> − indicates the name of the series. This expression can return any java.lang.Comparable object.

<categoryExpression> − returns the name of the category for each value inside the series specified by the series expression. Categories are java.lang.Comparable objects.

<valueExpression> − produces the values that correspond to each category in the dataset. Values are always java.lang.Number objects.

<labelExpression> − if this expression is missing, the chart will display default labels for each item in the chart. Use this expression, which returns java.lang.String values, to customize the item labels for the chart.

<itemHyperlink> − sets hyperlinks associated with chart items.

XY Dataset

An XY dataset is characterized by the <xySeries> element, which contains −

<seriesExpression> − indicates the name of the series. This expression can return any java.lang.Comparable object.

<xValueExpression> − returns the java.lang.Number value representing the X value from the (x, y) pair that will be added to the current data series.

<yValueExpression> − returns the java.lang.Number value representing the Y value from the (x, y) pair that will be added to the current data series.

<labelExpression> − if this expression is missing, the chart will display default labels for each item in the chart. Use this expression, which returns java.lang.String values, to customize the item labels for the chart.

<itemHyperlink> − sets hyperlinks associated with the chart items.

XYZ Dataset

An XYZ dataset is characterized by the <xyzSeries> element, which contains −

<seriesExpression> − indicates the name of the series. This expression can return any java.lang.Comparable object.

<xValueExpression> − returns the java.lang.Number value representing the X value from the (x, y, z) item that will be added to the current data series.

<yValueExpression> − returns the java.lang.Number value representing the Y value from the (x, y, z) item that will be added to the current data series.

<zValueExpression> − returns the java.lang.Number value representing the Z value from the (x, y, z) item that will be added to the current data series.

<labelExpression> − if this expression is missing, the chart will display default labels for each item in the chart. Use this expression, which returns java.lang.String values, to customize the item labels for the chart.

<itemHyperlink> − sets hyperlinks associated with the chart items.

Time Series Dataset

A time series dataset is characterized by the timePeriod attribute, and the <timeSeries> element. The timePeriod attribute specifies the type of the data series inside the dataset. Time series can contain numeric values associated with days, months, years, or other predefined time periods. Possible values are: *Year, Quarter, Month, Week, Day - this is the default value, Hour, Minute, Second, Millisecond.*

The <timeSeries> element contains −

<seriesExpression> − indicates the name of the series. This expression can return any java.lang.Comparable object.

<timePeriodExpression> − returns a java.util.Date value from which the engine will extract the corresponding time period depending on the value set for the timePeriod attribute mentioned above.

<valueExpression> − returns the java.lang.Number value to associate with the corresponding time period value when incrementing the current series of the dataset.

<labelExpression> − if this expression is missing, the chart will display default labels for each item in the chart. Use this expression, which returns java.lang.String values, to customize the item labels for the chart.

<itemHyperlink> − sets hyperlinks associated with the chart items.

Time Period Dataset

A time period dataset is characterized by the <timePeriodSeries> element, which contains −

<seriesExpression> − indicates the name of the series. This expression can return any java.lang.Comparable object.

<startDateExpression> − specifies the beginning of the date interval with which the numeric value will be associated when it is added to the time period series.

<endDateExpression> − specifies the end of the date interval with which the numeric value will be associated when it is added to the time period series.

<valueExpression> − returns the java.lang.Number value to associate with the current date interval specified by the start date and end date expressions.

<labelExpression> − if this expression is missing, the chart will display default labels for each item in the chart. Use this expression, which returns java.lang.String values, to customize the item labels for the chart.

<itemHyperlink> − sets hyperlinks associated with the chart items.

High Low Dataset

A high low dataset is characterized by the following expressions −

<seriesExpression> − currently, only one series is supported inside a High-Low or Candlestick chart. However, this single series must be identified by a java.lang.Comparable value returned by this expression, and it must also be used as the series name in the chart's legend.

<dateExpression> − returns the date to which the current (high, low, open, close, volume) item refers.

<highExpression> − returns a java.lang.Number value, which will be part of the data item added to the series when the dataset gets incremented.

<lowExpression> − returns a java.lang.Number value, which will be part of the data item added to the series when the dataset gets incremented.

<openExpression> − returns a java.lang.Number value, which will be part of the data item added to the series when the dataset gets incremented.

<closeExpression> − returns a java.lang.Number value, which will be part of the data item added to the series when the dataset gets incremented.

<volumeExpression> − a numeric expression that returns the volume value to use for the current data item. It is used only for Candlestick charts.

<itemHyperlink> − sets hyperlinks associated with the chart items.

Value Dataset

This is a special chart dataset implementation that contains a single value and is used for rendering Meter and Thermometer charts. The value is collected using the

<valueExpression> expression.

Chart Plots

Another common JRXML element through all chart types is the <**plot**> element. This allows us to define several of chart's characteristics like orientation and background color. Plots differ, based on the type of chart.

Plot Attribute

The table given below summarizes the attributes of <plot> element −

Attribute	**Description**	**Values**
backcolor	This attribute defines the chart's background color.	Any six digit hexadecimal value is a valid value for this attribute. The hexadecimal value must be preceded by a #.
orientation	This attribute defines the chart's orientation.	Horizontal,Vertical Default value is "Vertical"
backgroundAlpha	This attribute defines the transparency of the chart's background color.	The valid values for this attribute include any decimal number between 0 and 1, inclusive. The higher the number, the less transparent the background will be. Default value is "1."
foregroundAlpha	This attribute defines the transparency of the chart's foreground colors.	The valid values for this attribute include any decimal number between 0 and 1, inclusive. The higher the number, the less transparent the background will be. Default value is "1."
labelRotation	This attribute allows rotation of text labels on x-axis to rotate clockwise or anti-clockwise. This attribute applies only to charts for which the x axis is not numeric or does not display dates.	Default value is "0.0."

The <plot> element has a subelement <seriesColor> which attributes are: *seriesOrder* and *color*. This element customizes colors for series, and their position within in the color sequence.

Specific Settings for Chart Plots

piePlot − It has no specific settings

pie3DPlot − Contains the *depthFactor* attribute, a numeric value ranging from 0 to 1 that represents the depth of the pie as a percentage of the height of the plot area.

barPlot − One can show or hide tick labels, tick marks or item labels, and provides settings for both axis.

bar3DPlot − Provides the same settings as the barPlot, and generates a 3D effect using the xOffset and yOffset attributes.

linePlot − One can show or hide lines connecting item points, can show or hide shapes associated with item points, and provides settings for both axis.

scatterPlot − Similar to the linePlot, it can show or hide lines connecting item points, can show or hide shapes associated with item points, and provides settings for both axis.

areaPlot − Provides settings for both axis.

bubblePlot − One can set the bubble dimensions by setting the scaleType attribute, and provides settings for both axis.

timeSeriesPlot − One can show or hide lines connecting item points, can show or hide shapes associated with item points, and provides settings for both axis.

highLowPlot − One can show or hide open ticks, can show or hide close ticks, and provides settings for both axis.

candlestickPlot − One can show or hide the volume, and provides settings for both axis.

meterPlot − Contains specific settings for the dial shape, scale angle, measurement units, tick interval, dial color, needle color, tick color, value display font, color and format pattern, data range, and meter intervals.

thermometerPlot − Contains specific settings for the value location, mercury color, show/hide value lines, value display font, color and format pattern, data range, low range, medium range, and high range.

multiAxisChart − Contains specific settings for axis included in the plot.

Types of Charts

JasperReports offers built-in support for several chart types. They are listed as below −

pieChart − A combination of a Pie dataset and a Pie plot.

pie3DChart − Groups a Pie dataset and a Pie 3D plot.

barChart − A basic combination of a Category dataset and a Bar plot.

bar3DChart − Wraps a Category dataset and a Bar 3D plot.

xyBarChart − Supports Time Period datasets, Time Series datasets, and XY datasets, and uses a Bar plot to render the axis and the items.

stackedBarChart − Uses data from a Category dataset and renders its content using a Bar plot.

stackedBar3DChart − Uses data from a Category dataset and renders its content using a Bar 3D plot.

lineChart − Groups a Category dataset and a Line plot.

xyLineChart − Groups an XY dataset and a Line plot.

areaChart − Items from a Category dataset are rendered using an Area plot.

stackedAreaChart − Items from a Category dataset are rendered using an Area plot.

xyAreaChart − Uses data from an XY dataset and renders it through an Area plot.

scatterChart − Wraps an XY dataset with a Scatter plot.

bubbleChart − Combines an XYZ dataset with a Bubble plot.

timeSeriesChart − Groups a Time Series dataset and a Time Series plot.

highLowChart − A combination of a High-Low dataset and a High-Low plot.

candlestickChart − Uses data from a High-Low dataset but with a special Candlestick plot.

meterChart − Displays a single value from a Value dataset on a dial, using rendering options from a Meter plot.

thermometerChart − Displays the single value in a Value dataset using rendering options from a Thermometer plot.

multiAxisChart − Contains multiple range axes, all sharing a common domain axis.

Example

To demonstrate the charts, let's write a new report template (jasper_report_template.jrxml). Here, we will add the <**barChart**> element to the <pageHeader> section and <**pieChart**> to <summary> section. We would be displaying in charts the marks obtained for each subject. Save it to the directory **C:\tools\jasperreports-5.0.1\test**. The contents of the file are as given below −

```
<?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 = "jasper_report_template" pageWidth = "595" pageHeight = "860"
   columnWidth = "515" leftMargin = "40" rightMargin = "40"
   topMargin = "50" bottomMargin = "50">

   <field name = "subjectName" class = "java.lang.String"/>
   <field name = "marks" class = "java.lang.Integer"/>

   <variable name = "countNumber" class = "java.lang.Integer" calculation = "Count">
      <variableExpression>
         <![CDATA[Boolean.TRUE]]>
      </variableExpression>
   </variable>

   <background>
      <band splitType = "Stretch"/>
   </background>

   <title>
      <band height = "79" splitType = "Stretch"/>
   </title>

   <pageHeader>
      <band height = "200">

         <barChart>
            <chart evaluationTime = "Report">
               <reportElement x = "0" y = "0" width = "555" height = "200"/>

               <chartTitle>
                  <titleExpression>
                     <![CDATA["My First JR Bar Chart"]]>
                  </titleExpression>
               </chartTitle>
            </chart>

            <categoryDataset>
               <dataset incrementType = "None"/>

               <categorySeries>
                  <seriesExpression>
                     <![CDATA[$F{subjectName}]]>
                  </seriesExpression>

                  <categoryExpression>
                     <![CDATA[$F{subjectName}]]>
                  </categoryExpression>

                  <valueExpression>
                     <![CDATA[$F{marks}]]>
                  </valueExpression>

               </categorySeries>
            </categoryDataset>
            <barPlot isShowTickMarks = "false">
               <plot/>
            </barPlot>
         </barChart>

      </band>
   </pageHeader>

   <columnHeader>
      <band height = "20" splitType = "Stretch"/>
   </columnHeader>

   <detail>
      <band height = "20" splitType = "Stretch"/>
   </detail>

   <columnFooter>
      <band height = "20" splitType = "Stretch"/>
   </columnFooter>

   <pageFooter>
      <band height = "20" splitType = "Stretch"/>
   </pageFooter>

   <summary>
      <band height = "400" splitType = "Stretch">

         <pieChart>
            <chart evaluationTime = "Report">
               <reportElement x = "135" y = "0" width = "270" height = "300"/>

               <chartTitle>
                  <titleExpression>
                     <![CDATA["My First JR Pie Chart"]]>
                  </titleExpression>
               </chartTitle>
            </chart>

            <pieDataset>
               <dataset incrementType = "None"/>

               <keyExpression>
                  <![CDATA[$F{subjectName}]]>
               </keyExpression>

               <valueExpression>
                  <![CDATA[$F{marks}]]>
               </valueExpression>
            </pieDataset>

            <piePlot>
               <plot/>
               <itemLabel/>
            </piePlot>
         </pieChart>

      </band>
   </summary>

</jasperReport>
```

The details of the above file are as given below −

The JRXML element used to create a bar chart is </barChart> in the <pageHeader>. It contains a </chart> sub-element, which contains a <reportElement> sub-element defining the chart's dimensions and position.

The <dataset> element in a bar chart must be enclosed between <categoryDataset> and </categoryDataset> JRXML elements.

<categoryDataset> must contain a <categorySeries> element. This element defines what data element the bars will represent (subject names, in this example).

<categoryDataset> must also contain an element, which defines how the data will be separated into categories for comparison. Here, data is separated by subject names.

The <valueExpression> element defines that what expression is used to determine the value of each bar in the chart. Here, we are using "marks".

For the pie chart, we have used the element <pieChart> under the <summary> section. It contains a </chart> sub-element.

The sub-element contains a report expression indicating what to use as a key in the chart. Here, we have used subjectName.

The sub-element contains an expression used to calculate the value for the key. Here, we have used marks.

The java codes for report filling remains unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below −

```
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();
      }
   }
}
```

As we would be displaying the marks obtained for each subject, POJO needs to be changed. The file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** contents are as given below –

```
package com.tutorialspoint;

public class DataBean {
   private String subjectName;
   private Integer marks;

   public String getSubjectName() {
      return subjectName;
   }

   public void setSubjectName(String subjectName) {
      this.subjectName = subjectName;
   }

   public Integer getMarks() {
      return marks;
   }

   public void setMarks(Integer marks) {
      this.marks = marks;
   }

}
```

Even the contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** need to be updated as given below –

```
package com.tutorialspoint;

import java.util.ArrayList;

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

      dataBeanList.add(produce("English", 58));
      dataBeanList.add(produce("SocialStudies", 68));
      dataBeanList.add(produce("Maths", 38));
      dataBeanList.add(produce("Hindi", 88));
      dataBeanList.add(produce("Scince", 78));

      return dataBeanList;
   }

   /*
    * This method returns a DataBean object, with subjectName ,
    * and marks set in it.
    */
   private DataBean produce(String subjectName, Integer marks) {
      DataBean dataBean = new DataBean();

      dataBean.setSubjectName(subjectName);
      dataBean.setMarks(marks);

      return dataBean;
   }
}
```

Report Generation

Next, let's compile and execute the above files using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

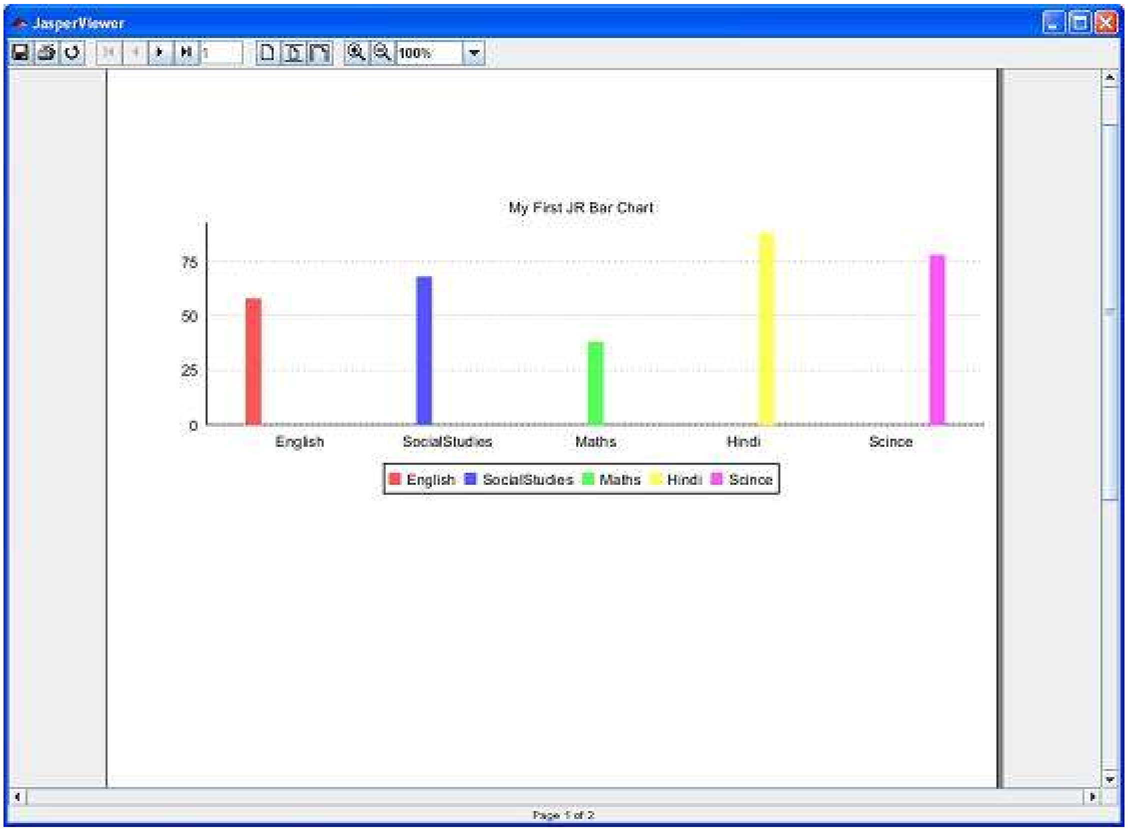
The import file - baseBuild.xml is picked up from the chapter Environment Setup and should be placed in the same directory as the build.xml.

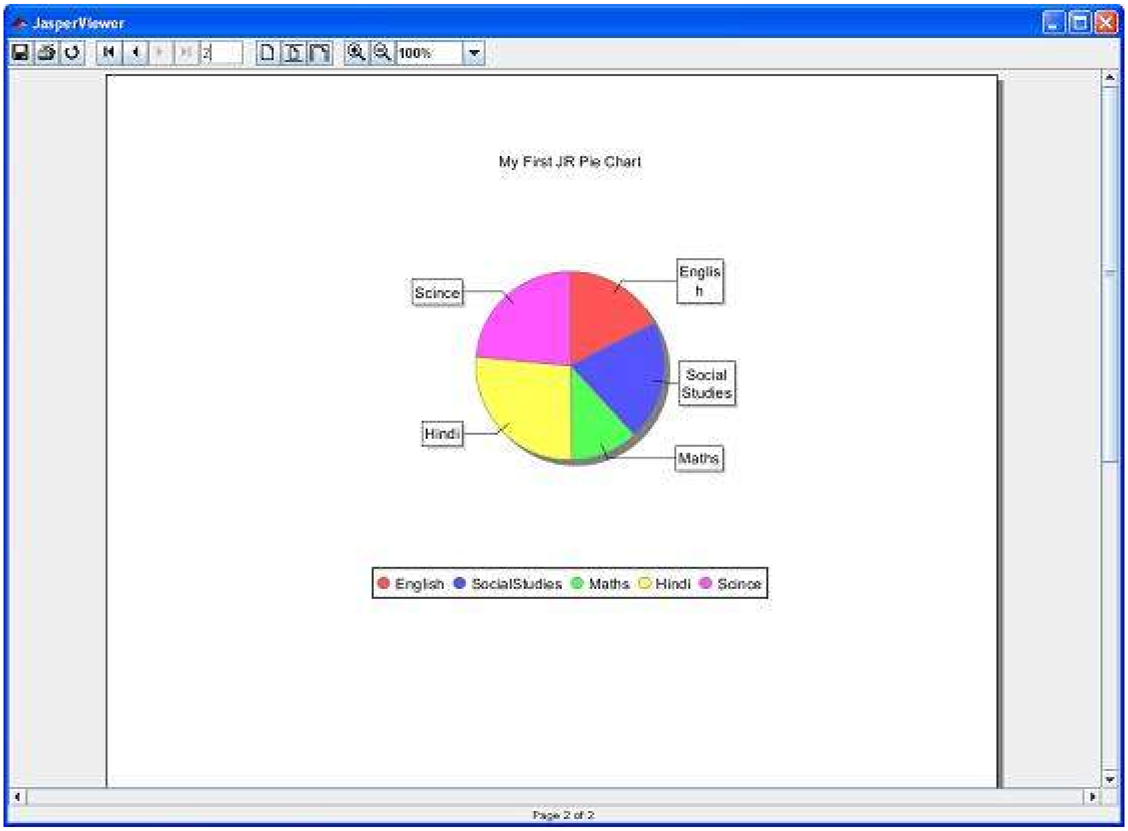
```
<?xml version = "1.0" encoding = "UTF-8"?>
<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the
      report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport 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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting to bu
   [javac] Compiling 3 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.

viewFillReport:
   [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: 19 minutes 45 seconds
```

As a result of above compilation, a JasperViewer window opens up as in the screen below –





Here, we see that the bar chart is created in the pageheader and the pie chart is created in the summary sections.

JasperReports - Crosstabs

Crosstab (cross-tabulation) reports are the reports containing tables that arrange data across rows and columns in a tabular form. Crosstab object is used for inserting a crosstab report within the main report. Crosstabs can be used with any level of data (nominal, ordinal, interval, or ratio), and usually display the summarized data, contained in the report variables, in the form of a dynamic table. Variables are used to display aggregate data such as sums, counts, average values.

Crosstab Properties

JRXML element <**crosstab**> is used to insert a crosstab into a report.

Attribute

Following is a list of attribute of a <**crosstab**> element −

isRepeatColumnHeaders − Indicates whether the column headers should be reprinted after a page break. The default value is *true*.

isRepeatRowHeaders − Indicates whether the row headers should be reprinted after a crosstab column break. The default value is *true*.

columnBreakOffset − When a column break occurs, indicates the amount of vertical space, measured in pixels, before the subsequent crosstab piece to be placed below the previous one on the same page. The default value is 10.

runDirection − Indicates whether the crosstab data should be filled from left to right (LTR) or from right to left (RTL). The default value is LTR.

ignoreWidth − Indicates whether the crosstab will stretch beyond the initial crosstab width limit and don't generate column breaks. Else it will stop rendering columns within the crosstab width limit and continue with the remaining columns only after all rows have started rendering. The default value is *false*.

Sub Elements

A <crosstab> element has following sub elements −

<reportElement> − This element defines the position, width, and height of the crosstab within its enclosing. Attributes for this element include all standard <reportElement> attributes.

<crosstabParameter> − This element is used to access report variables and parameters from within the crosstab. Attributes for this element include −

name − This defines the parameter name.

class − This indicates the parameter class.

<parametersMapExpression> − This element is used to pass a report variable or parameter containing an instance of *java.util.Map*, as a set of parameters for the crosstab. This element contains no attributes.

<crosstabDataset> − This element defines the dataset to use to populate the crosstab (see next section for a detailed explanation). Attributes for this element include −

> *isDataPreSorted* − This indicates whether the data in the dataset is pre-sorted. Default value is *false*.

<crosstabHeaderCell> − This element defines the content of the region found at the upper-left corner of the crosstab where column headers and row headers meet. The size of this cell is calculated automatically based on the defined row and column widths and heights.

<rowGroup> − This element defines a group used to split the data into rows. Attributes for this element include −

> *name* − This defines the name of the row group.
>
> *width* − This defines the width of the row group.
>
> *headerPosition* − This defines the position of the header contents (Top, Middle, Bottom, Stretch).
>
> *totalPosition* − This defines the position of the entire column (Start, End, None).

This element contains the following sub elements −

> *<bucket>*
>
> *<crosstabRowHeader>*
>
> *<crosstabTotalRowHeader>*

<columnGroup> − This element defines a group used to split the data into columns. Attributes for this element include −

> *name* − This defines the column group name.
>
> *height* − This defines the height of the column group header.
>
> *headerPosition* − This defines the position of the header contents (*Right, Left, Center, Stretch*).
>
> *totalPosition* − This defines the position of the entire column (*Start, End, None*).

This element contains the following sub elements −

<*bucket*>

<*crosstabColumnHeader*>

<*crosstabTotalColumnHeader*>

<measure> − This element defines the calculation to be performed across rows and columns. Attributes for this element include −

name − This defines the measure name.

class − This indicates the measure class.

calculation − This indicates the calculation to be performed between crosstab cell values. Its values could be any of these - *Nothing, Count, DistinctCount, Sum, Average, Lowest, Highest, StandardDeviation, Variance,* and *First*. Default value is **Nothing**.

<crosstabCell> − This element defines how data in non-header cells will be laid out. Attributes for this element include −

columnTotalGroup − This indicates the group to use to calculate the column total.

height − This defines the height of the cell.

rowTotalGroup − This indicates the group to use to calculate the row total.

width − This defines the width of the cell.

<whenNoDataCell> − This element defines what to display on an empty crosstab cell. This element contains no attributes.

Data Grouping in Crosstab

The crosstab calculation engine aggregates data by iterating through the associated dataset records. In order to aggregate data, one needs to group them first. In a crosstab, rows and columns are based on specific group items, called **buckets**. A bucket definition should contain −

bucketExpression − The expression to be evaluated in order to obtain data group items.

comparatorExpression − Needed in the case the natural ordering of the values is not the best choice.

orderByExpression − Indicates the value used to sort data.

Row and column groups (defined above) in a crosstab rely on **buckets**.

Built-In Crosstab Total Variables

Below is a list of current value of measure and totals of different levels corresponding to the cell can be accessed through variables named according to the following scheme −

The current value of a measure calculation is stored in a variable having the same name as the measure.

<Measure>_<Column Group>_ALL − This yields the total for all the entries in the column group from the same row.

<Measure>_<Row Group>_ALL − This yields the total for all the entries in the row group from the same column.

<Measure>_<Row Group>_<Column Group>_ALL − This yields the combined total corresponding to all the entries in both row and column groups.

Example

To demonstrate the crosstabs, let's write a new report template (jasper_report_template.jrxml). Here, we will add the crosstab to summary section. Save it to the directory **C:\tools\jasperreports-5.0.1\test**. The contents of the file are as given 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">

   <parameter name = "ReportTitle" class = "java.lang.String"/>
   <parameter name = "Author" class = "java.lang.String"/>

   <field name = "name" class = "java.lang.String"/>
   <field name = "country" class = "java.lang.String"/>

   <title>
      <band height = "70">

         <line>
            <reportElement x = "0" y = "0" width = "515" height = "1"/>
         </line>

         <textField isBlankWhenNull = "true" bookmarkLevel = "1">
            <reportElement x = "0" y = "10" width = "515" height = "30"/>

            <textElement textAlignment = "Center">
               <font size = "22"/>
            </textElement>
            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{ReportTitle}]]>
            </textFieldExpression>

            <anchorNameExpression>
               <![CDATA["Title"]]>
            </anchorNameExpression>
         </textField>

         <textField isBlankWhenNull = "true">
            <reportElement  x = "0" y = "40" width = "515" height = "20"/>

            <textElement textAlignment = "Center">
               <font size = "10"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{Author}]]>
            </textFieldExpression>
         </textField>

      </band>
   </title>

   <summary>
      <band height = "60">

      <crosstab>
         <reportElement width = "782" y = "0" x = "0" height = "60"/>

         <rowGroup name = "nameGroup" width = "100">

            <bucket>
               <bucketExpression class = "java.lang.String">
                  <![CDATA[$F{name}]]>
               </bucketExpression>
            </bucket>

            <crosstabRowHeader>

               <cellContents>
                  <box border = "Thin" borderColor = "black"/>

                  <textField>
                     <reportElement width = "100" y = "0" x = "0" height = "20"/>
                     <textElement textAlignment = "Right"
                        verticalAlignment = "Middle"/>

                     <textFieldExpression>
                        <![CDATA[$V{nameGroup}]]>
                     </textFieldExpression>
                  </textField>

               </cellContents>

            </crosstabRowHeader>

         </rowGroup>

         <columnGroup name = "countryGroup" height = "20">
```

```
            <bucket>

               <bucketExpression class = "java.lang.String">
                  $F{country}
               </bucketExpression>
            </bucket>

            <crosstabColumnHeader>
               <cellContents>
                  <box border = "Thin" borderColor = "black"/>
                  <textField isStretchWithOverflow = "true">
                     <reportElement width = "60" y = "0" x = "0" height = "20"/>
                     <textElement verticalAlignment = "Bottom"/>
                     <textFieldExpression>
                        <![CDATA[$V{countryGroup}]]>
                     </textFieldExpression>
                  </textField>
               </cellContents>
            </crosstabColumnHeader>

         </columnGroup>

         <measure name = "tailNumCount" class = "java.lang.Integer"
            calculation = "Count">
            <measureExpression>$F{country}</measureExpression>
         </measure>

         <crosstabCell height = "20" width = "60">
            <cellContents backcolor = "#FFFFFF">
               <box borderColor = "black" border = "Thin"/>
               <textField>
                  <reportElement x = "5" y = "0" width = "55" height = "20"/>
                  <textElement textAlignment = "Left"
                     verticalAlignment = "Bottom"/>
                  <textFieldExpression class = "java.lang.Integer">
                      $V{tailNumCount}
                  </textFieldExpression>
               </textField>
            </cellContents>
         </crosstabCell>

      </crosstab>

      </band>
   </summary>

</jasperReport>
```

The details of the above file are as follows −

- Crosstab is defined by the <crosstab> element.
- <rowGroup> element defines a group to split the data into rows. Here, each row will display data for a different name.
- The <bucket> and <bucketExpression> elements define what report expression to use as a group delimiter for <rowGroup>. Here, we used the name field as a delimiter, in order to split the rows by name.

The <crosstabRowHeader> element defines the expression to be used as a row header. It contains a single sub-element, namely <cellContents>, which acts like an inner band inside crosstab. Instead of defining variable name for the text field inside <crosstabRowHeader>, we have assigned the name to <rowGroup> (via its name attribute), hence it creates an implicit variable. The <crosstabRowHeader> element defines the contents of the header cell for the entire row. It takes a single <cellContents> element as its only sub-element.

The <columnGroup> element as well as its sub-elements is analogous to the <rowGroup> element, except that it influences columns instead of rows.

The <measure> element defines the calculation to be performed across rows and columns. The *calculation* attribute is set to *Count*.

The <crosstabCell> element defines how data in non-header cells will be laid out. This element also contains a single <crosstabCell> element as its only sub-element.

The java codes for report filling remains unchanged. The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\JasperReportFill.java** are as given below –

```
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();
      /**
       * Passing ReportTitle and Author as parameters
       */
      parameters.put("ReportTitle", "List of Contacts");
      parameters.put("Author", "Prepared By Manisha");

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

The contents of the POJO file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBean.java** are as follows −

```
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;
   }
}
```

The contents of the file **C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\DataBeanList.java** are as follows −

```
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;
   }
}
```

Report Generation

Next, let's compile and execute the above files using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

The import file - baseBuild.xml is picked up from the 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 = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the
      report stored in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFill** (viewFullReport 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

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting to
   [javac] Compiling 3 source files to C:\tools\jasperreports-5.0.1\test\classes

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.

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.

viewFillReport:
   [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: 20 minutes 53 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −

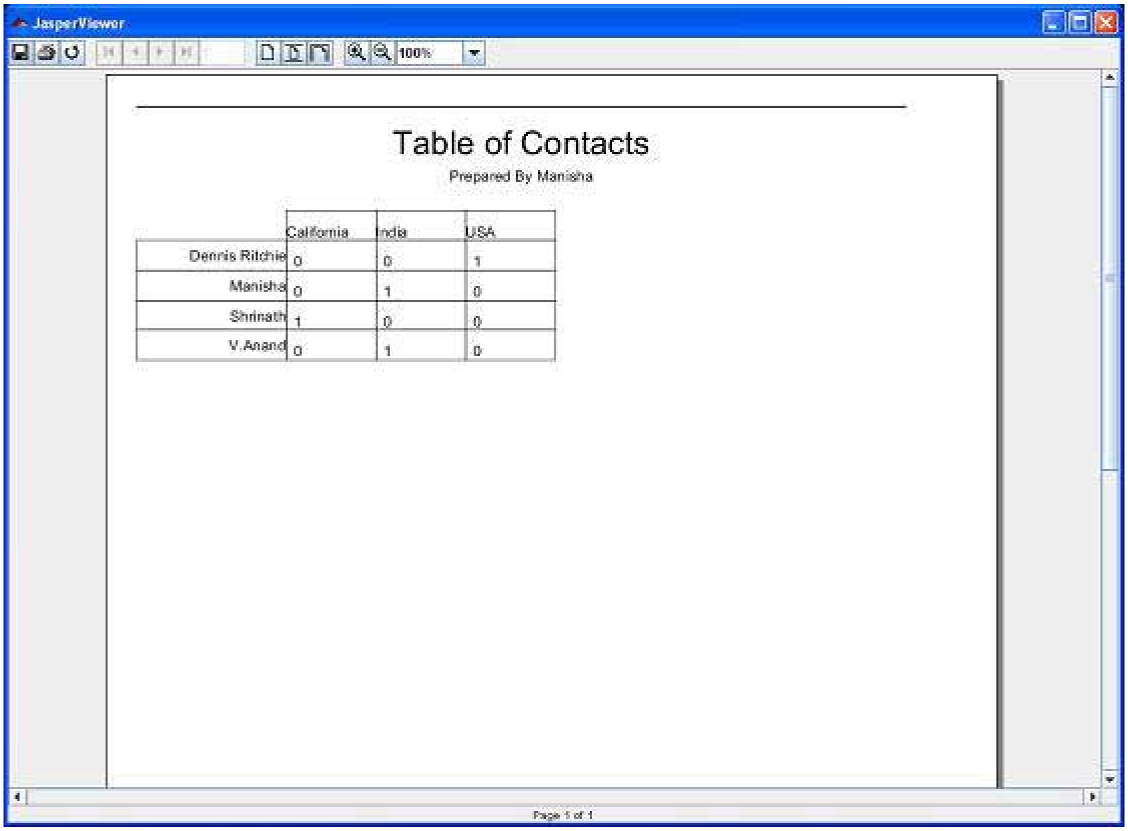


Table of Contents

Prepared By Manisha

	California	India	USA
Dennis Ritchie	0	0	1
Manisha	0	1	0
Shrinath	1	0	0
V.Anand	0	1	0

Here, we see that each country and name are tabulated.

JasperReports - Internationalization

Sometimes, we need reports in different languages. Writing the same report for each different language implies a lot of redundant work. Only pieces of text differing from language to language should be written separately, and loaded into text elements at runtime, depending on locale settings. This is the purpose of the report internationalization. Internationalized reports, once written can run everywhere.

In the following steps, we have listed how to generate a report in different languages and also some other features of report internationalization −

Associate a resource bundle *java.util.ResourceBundle* with the report template. There are two ways to associate the java.util.ResourceBundle object with the report template.

At design time, by setting the *resourceBundle* attribute of the report template object to the base name of the target resource bundle.

A dynamic/runtime association can be made by supplying a *java.util.ResourceBundle* object as the value for the REPORT_RESOURCE_BUNDLE parameter at report-filling time.

If the report needs to be generated in a locale that is different from the current one, the built-in REPORT_LOCALE parameter can be used to specify the runtime locale when filling the report.

To facilitate report internationalization, a special syntax **$R{}** is available inside report expressions to reference *java.lang.String* resources placed inside a *java.util.ResourceBundle* object associated with the report. The **$R{}** character syntax extracts the locale-specific resource from the resource bundle based on the key that must be put between the brackets –

```
<textFieldExpression>
   $R{report.title}
</textFieldExpression>
```

The above text field displays the title of the report by extracting the String value from the resource bundle associated with the report template based on the runtimesupplied locale and the *report.title* key.

Formatting messages in different languages based on the report locale, there's a built-in method inside the reports *net.sf.jasperreports.engine.fill.JRCalculator*. This method offers functionality similar to the *java.text.MessageFormat* class. This method, msg(), has three convenient signatures that allow you to use up to three message parameters in the messages.

A built-in *str()* method (the equivalent of the $R{} syntax inside the report expressions), which gives access to the resource bundle content based on the report locale.

For date and time formatting, the built-in REPORT_TIME_ZONE parameter can be used to ensure proper time transformations.

In the generated output, the library keeps information about the text run direction so that documents generated in languages that have right-to-left writing (like Arabic and Hebrew) can be rendered properly.

If an application relies on the built-in Swing viewer to display generated reports, then it needs to be internationalized by adapting the button ToolTips or other texts displayed. This is very easy to do since the viewer relies on a predefined resource bundle to extract locale-specific information. The base name for this resource bundle is *net.sf.jasperreports.view.viewer.*

Example

To demonstrate internationalization, let's write new report template (jasper_report_template.jrxml). The contents of the JRXML are as given below. Save it to C:\tools\jasperreports-5.0.1\test directory.

```
<?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" resourceBundle = "localizationdemo">

   <title>
      <band height = "552">

         <textField>
            <reportElement positionType = "Float" x = "150" y = "20"
               width = "400" height = "50"/>

            <textElement>
               <font size = "24"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$P{REPORT_LOCALE}.getDisplayName ($P{REPORT_LOCALE})]]>
            </textFieldExpression>
         </textField>

         <textField isStretchWithOverflow = "true" isBlankWhenNull = "true">
            <reportElement positionType = "Float" x = "20" y = "125"
               width = "530" height = "20"/>

            <textElement textAlignment = "Justified">
               <font size = "14"/>
            </textElement>

            <textFieldExpression class = "java.lang.String">
               <![CDATA[$R{localization.text1}]]>
            </textFieldExpression>

         </textField>

      </band>
   </title>

</jasperReport>
```

In the above file, the *resourceBundle* attribute of the <jasperReport> element tells JasperReports where to get the localized strings to use for the report. We need to create a property file with a root name matching the value of the attribute. This file must exist anywhere in the CLASSPATH when filling the report. In this example, the property file **localizationdemo.properties** is saved under the directory **C:\tools\jasperreports-5.0.1\test**. The contents of this file are as follows −

```
localization.text1 = This is English text.
```

To use a different locale, the name of the file must be localizationdemo[locale].properties. Here, we will write a file for spanish locale. Save this file as − **C:\tools\jasperreports-**

5.0.1\test\localizationdemo_es.properties. The contents of this file are as follow –

```
localization.text1 = Este texto es en Español.
```

The syntax to obtain the value for resourceBundle properties is $R{key}.

To let JasperReports know what locale we wish to use, we need to assign a value to a built-in parameter. This parameter's name is defined as a constant called REPORT_LOCALE, and this constant is defined in the *net.sf.jasperreports.engine.JRParameter* class. The constant's value must be an instance of *java.util.Locale*. This logic is incorporated in java code to fill and generate the report. Let's save this file **JasperReportFillI18.java** to C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint directory. The contents of the file are as follows –

```
package com.tutorialspoint;

import java.util.HashMap;
import java.util.Locale;

import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JRParameter;
import net.sf.jasperreports.engine.JasperFillManager;

public class JasperReportFillI18 {

   public static void main(String[] args) {
      String sourceFileName = "C://tools/jasperreports-5.0.1/test/"
         + "jasper_report_template.jasper";
      HashMap parameterMap = new HashMap();
      if (args.length > 0) {
         parameterMap.put(JRParameter.REPORT_LOCALE, new Locale(args[0]));
      }
      try {
         JasperFillManager.fillReportToFile(sourceFileName, null,
            new JREmptyDataSource());
      } catch (JRException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }
}
```

Report Generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as given below.

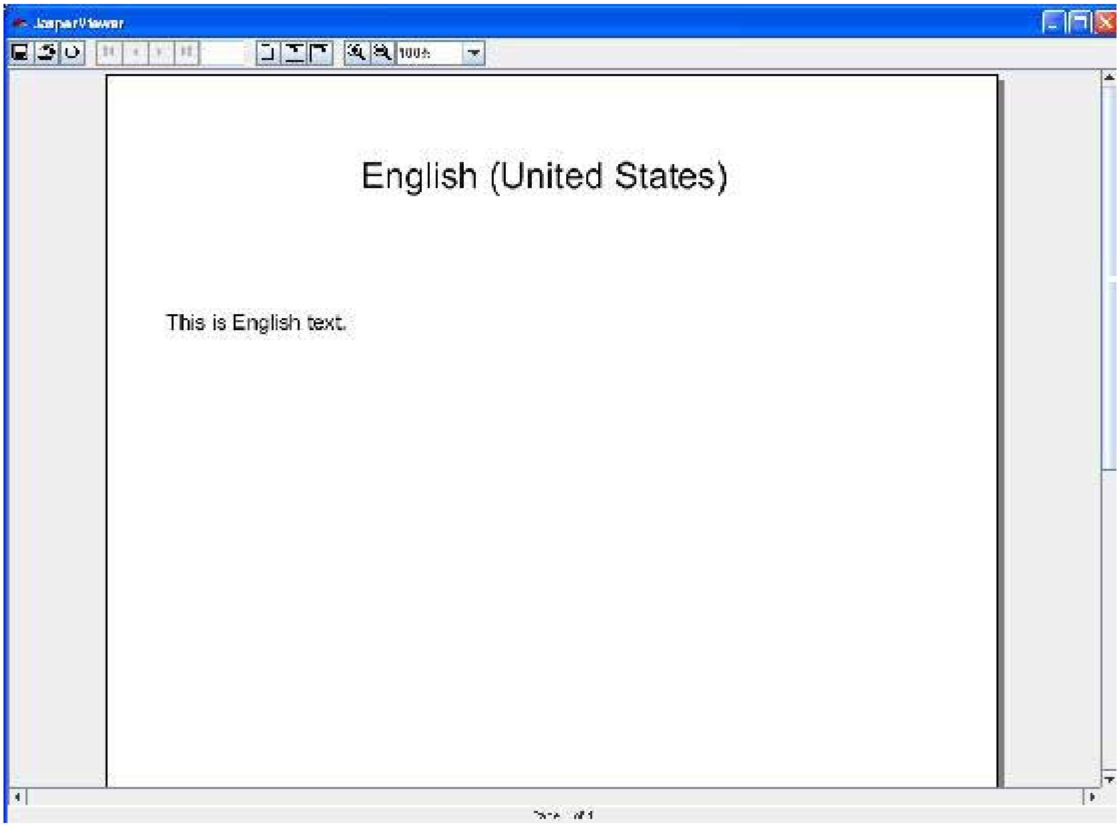
The import file - baseBuild.xml is picked up from the chapter Environment Setup and should be placed in the same directory as the build.xml.

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

<project name = "JasperReportTest" default = "viewFillReport" basedir = ".">
   <import file = "baseBuild.xml" />

   <target name = "viewFillReport" depends = "compile,compilereportdesing,run"
      description = "Launches the report viewer to preview the report stored
      in the .JRprint file.">

      <java classname = "net.sf.jasperreports.view.JasperViewer" fork = "true">
         <arg value = "-F${file.name}.JRprint" />
         <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 line window and go to the directory where build.xml is placed. Finally, execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFillI18** (viewFullReport is the default target) as follows −

```
C:\tools\jasperreports-5.0.1\test>ant  -Dmain-class=com.tutorialspoint.JasperReportFillI18
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
   [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
   [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
   warning: 'includeantruntime' was not set, defaulting to
   [javac] Compiling 1 source file to C:\tools\jasperreports-5.0.1\test\classes
   [javac] Note: C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint\
      JasperReportFillI18.java
   uses unchecked or u
   [javac] Note: Recompile with -Xlint:unchecked for details.

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.

run:
   [echo] Runnin class : com.tutorialspoint.JasperReportFillI18
   [java] log4j:WARN No appenders could be found for logger
   (net.sf.jasperreports.extensions.ExtensionsEnvironment).
   [java] log4j:WARN Please initialize the log4j system properly.

viewFillReport:
   [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: 3 minutes 28 seconds
```

As a result of above compilation, a JasperViewer window opens up as shown in the screen given below −



English (United States)

This is English text.

Previous Page Next Page

FAQ's Cookies Policy Contact