**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 −

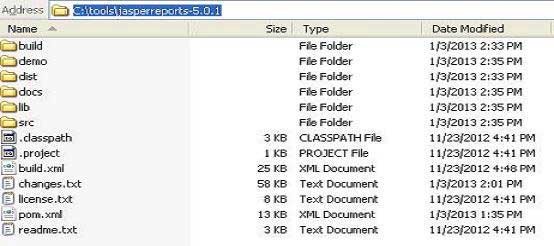
* It has a flexible report layout.
* It can present data either textually or graphically.
* Developers can supply data in multiple ways.
* It can accept data from the multiple data sources.
* It can generate watermarks (A watermark is like a secondary image that is laid over the primary image).
* It can generate sub reports.
* It is capable of exporting reports in a variety of formats.

JasperReports 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](http://sourceforge.net/projects/jasperreports/files/jasperreports/). 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 −



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"

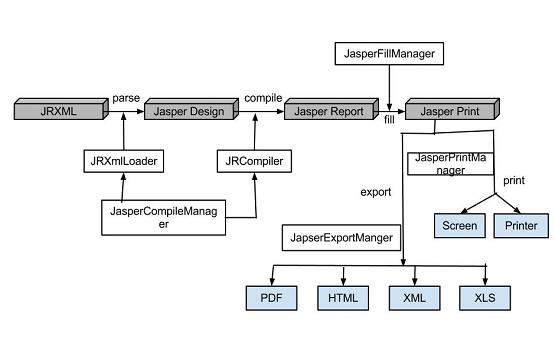
+ " parameter, for example," + " com.tutorialspoint.HelpMe ");

}

}

**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.



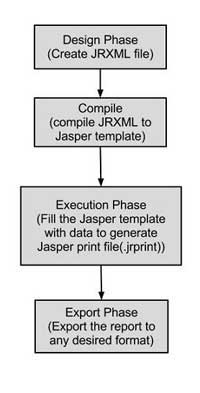
The list of these classes is −

* 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](https://www.tutorialspoint.com/ant/ant_environment.htm) chapter to setup Apache ANT on your system.

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](https://www.tutorialspoint.com/jasper_reports/jasper_report_designs.htm) − 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](http://community.jaspersoft.com/wiki/ireport-designer-getting-started) 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](https://www.tutorialspoint.com/jasper_reports/jasper_compiling_report_design.htm) − 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)](https://www.tutorialspoint.com/jasper_reports/jasper_filling_reports.htm) − 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](https://www.tutorialspoint.com/jasper_reports/jasper_exporting_reports.htm) − 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.