Report-engine tutorial

Contents

[What is report-engine? 3](#_Toc336018512)

[How to build report-engine? 4](#_Toc336018513)

[Flat Reports 5](#_Toc336018514)

[Your first report 5](#_Toc336018515)

[Report Input 5](#_Toc336018516)

[Report Output 5](#_Toc336018517)

[Report Configuration 5](#_Toc336018518)

[Totals and aggregations 5](#_Toc336018519)

[Pivot Tables /Crosstab reports 6](#_Toc336018520)

[Your first Pivot table report 6](#_Toc336018521)

[The Input 6](#_Toc336018522)

[The Output 6](#_Toc336018523)

[Configuration 6](#_Toc336018524)

[Totals and aggregations 6](#_Toc336018525)

[Advanced features 7](#_Toc336018526)

[Spring integration 7](#_Toc336018527)

[Writing a custom input for your reports 7](#_Toc336018528)

[Writing a custom output for your reports 7](#_Toc336018529)

[FAQ 8](#_Toc336018530)

[Useful links 9](#_Toc336018531)

# What is report-engine?

Report Engine is a set of JAVA classes for reports and pivot tables with support for groupings, totals/subtotals, aggregation. It accepts input from text files, databases, excel or you can write your custom input and exports the report in a multitude of formats (HTML, RTF, PDF, TXT, SVG etc.)

# How to build report-engine?

# Flat Reports

## Your first report

Each report needs three elements: input, column configuration and output. Let’s have a look at the report below:

FlatReport flatReport = new FlatReport();

//set input

IReportInput input = new StreamReportInput(new FileInputStream("input.txt"));

flatReport.setIn(input);

//set output

IReportOutput output = new ExcelReportOutput("c:/output.xls");

flatReport.setOut(output);

//report configuration

IDataColumn[] reportColumns = new IDataColumn[]{

new DefaultDataColumn("Country", 0),

new DefaultDataColumn("City", 1),

new DefaultDataColumn("Population", 2)

};

flatReport.setDataColumns(reportColumns);

//start execution

flatReport.execute();

## Report Input

The main input classes are:

* *StreamReportInput* - handles input from streams (any kind) and reads data columns separated by a specific user-defined separator (comma, tab, etc.)

IReportInput reportInput = new StreamReportInput(

new FileInputStream("c:\commaSeparated.csv"),",");

flatReport.setIn(input);

…

* *DbQueryReportInput* - executes a query and takes the result as input for your reports

1. *For an existing database connection*

java.sql.Connection dbConnection = ...

DbQueryReportInput dbReportInput = new DbQueryReportInput();

dbReportInput.setConnection(connection);

dbReportInput.setSqlStatement("select id, country, region, city, population from DB\_POPULATION\_TABLE ");

1. Or if you don’t have the connection, report-engine can create one for you given the right parameters:

DbQueryReportInput dbReportInput = new DbQueryReportInput();

dbReportInput.setDbConnString("jdbc:hsqldb:mem:countriesDB");

dbReportInput.setDbDriverClass("org.hsqldb.jdbcDriver");

dbReportInput.setDbUser("sa");

dbReportInput.setDbPassword("secret");

dbReportInput.setSqlStatement("select id, country, region, city, population from DB\_POPULATION\_TABLE");

* *MemoryReportInput* - takes an array of objects as input

Object[][] REPORT\_DATA = new Object[][]{

new String[]{"a","b","c","d"},

new String[]{"1","2","3","4"},

new String[]{"x","y","z","t"}

}

IReportInput reportInput = new MemoryReportInput(REPORT\_DATA);

* If these classes don’t cover your needs you can always write your own input by implementing the *IReportInput* interface

## Report Output

The predefined output formats for your reports are:

* *HtmlReportOutput* - fast html output

HtmlOuputhtmlOut **= new** HtmlOutput(**new** FileOutputStream("employees.html")

* *ExcelReportOutput* – creates an excel output

ExcelOuput output = **new** ExcelOutput(**new** FileOuputStream("employees.xls"));

* *StaxReportOutput* - xml output based on STax
* *XsltReportOutput -* output based on an XSLT template - can result in HTML, TXT, SVG, etc.
* *XslFoReportOutput* - output based on XSL-FO framework - can result in PDF, PNG, TXT, ghostscript. Actually, everything supported by [apache fop project](http://xmlgraphics.apache.org/fop/trunk/output.html).

XslFoOutput pdfOutput = **new** XslFoOutput(

**new** FileOutputStream("employees.pdf"));

XslFoOutput pngOutput = **new** XslFoOutput(

**new** FileOutputStream("employees.png"), MimeConstants.*MIME\_PNG*);

* *XmlDOMReportOutput* - xml output based on DOM
* Of course you can always write your own report output by implementing the *IReportOutput* interface

## Report Configuration

There are two kinds of columns: data columns and group columns.

### Data columns

Data columns are normal report columns displaying formatted data and totals. For each column there are a few parameters to set: header, values displayed, calculator-if totals are needed … and that’s about it.

Now, let’s see an example:

For report-engine API the order in which you define your columns is very important because **it defines the output order of your columns**.

### Your first report

It’s now time to build our first report: Expenses report.

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.InputStream;

**import** net.sf.reportengine.FlatReport;

**import** net.sf.reportengine.config.DefaultDataColumn;

**import** net.sf.reportengine.config.IDataColumn;

**import** net.sf.reportengine.in.StreamReportInput;

**import** net.sf.reportengine.out.HtmlOutput;

/\*\*

\* this is your first report

\*

\*/

**public** **class** FirstReport {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

**try** {

FlatReport flatReport = **new** FlatReport();

flatReport.setReportTitle("Mothly Expenses report");

//the input

InputStream fileStream = **new** FileInputStream("expenses.csv");

StreamReportInput reportInput = **new** StreamReportInput(fileStream,",");

flatReport.setIn(reportInput);

//the output

HtmlOutput output = **new** HtmlOutput(**new** FileOutputStream("expenses.html"));

flatReport.setOut(output);

//data columns configuration

IDataColumn[] dataColumns = **new** IDataColumn[]{

**new** DefaultDataColumn("Month",0),

**new** DefaultDataColumn("Day",1),

**new** DefaultDataColumn("Spent on",2),

**new** DefaultDataColumn("Amount",3)

};

flatReport.setDataColumns(dataColumns);

flatReport.execute();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

}

}

### Group columns

Group columns are helpful when displaying totals on data columns. At each change in the values of a group column the totals are displayed.

Let’s check the following example: My list of expenses

|  |  |  |  |
| --- | --- | --- | --- |
| August |  | food | 500$ |
| August |  | transportation | 300$ |
| September |  | food | 567$ |
| September |  | transportation | 154$ |
| September |  | dinner | 200$ |

If we declare the first column as a group column then report-engine will make sure to display totals, averages (or whatever you’ve set ) at each change in the values of the first column:

|  |  |  |  |
| --- | --- | --- | --- |
| August |  | food | 500$ |
| August |  | transportation | 300$ |
| **Total August** |  |  | **800$** |
| September |  | food | 567$ |
| September |  | transportation | 154$ |
| September |  | dinner | 200$ |
| **Total September** |  |  | **921$** |

### Your first report containing a group Column

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.InputStream;

**import** net.sf.reportengine.FlatReport;

**import** net.sf.reportengine.config.DefaultDataColumn;

**import** net.sf.reportengine.config.DefaultGroupColumn;

**import** net.sf.reportengine.config.IDataColumn;

**import** net.sf.reportengine.config.IGroupColumn;

**import** net.sf.reportengine.core.calc.Calculator;

**import** net.sf.reportengine.in.StreamReportInput;

**import** net.sf.reportengine.out.HtmlOutput;

/\*\*

\* The first report containing a group column.

\* The month column is declared as a group column so

\* after each change in this column the totals will

\* be displayed on the other columns.

\*/

**public** **class** FirstGroupReport {

**public** **static** **void** main(String[] args) {

**try** {

FlatReport flatReport = **new** FlatReport();

flatReport.setShowTotals(**true**);

flatReport.setShowGrandTotal(**true**);

flatReport.setReportTitle("Mothly Expenses");

//define the input

InputStream fileInput = **new** FileInputStream("expenses.csv");

StreamReportInput reportInput = **new** StreamReportInput(fileInput,",");

flatReport.setIn(reportInput);

//define the output

HtmlOutput output = **new** HtmlOutput(**new** FileOutputStream("xpenses.html"));

flatReport.setOut(output);

//group column configuration

IGroupColumn[] groupColumns = **new** IGroupColumn[]{

**new** DefaultGroupColumn("Month", 0, 0)

};

flatReport.setGroupColumns(groupColumns);

//data columns configuration

IDataColumn[] dataColumns = **new** IDataColumn[]{

**new** DefaultDataColumn("Day",1),

**new** DefaultDataColumn("Spent on",2),

**new** DefaultDataColumn("Amount",3,Calculator.*AVG*)

};

flatReport.setDataColumns(dataColumns);

//start executing the report

flatReport.execute();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

}

}

### Totals and aggregations

## Examples

# Pivot Tables /Crosstab reports

## Your first Pivot table report

## The Input

## The Output

## Configuration

## Totals and aggregations

# Advanced features

## Spring integration

## Writing a custom input for your reports

## Writing a custom output for your reports

# FAQ

# Limitations

# Useful links