SmartXLS
PivotTable framework – user manual
Author: vsubhuman

To use Excel pivot tables properly you cannot avoid configuration of the data fields.
But this framework simplifies this work.

PivotTable

PivotTable class is the central part of the framework. This class describes not the table itself, but configuration that will be used to create real pivot table. Instance of this class can be created and configured without presence of the SmartXLS WorkBook instance. You can instantiate PivotTable by this methods:

```
new PivotTable("Pivot table", new TableRange(0, 0, 10, 10), 0);
new PivotTable("Pivot table", "A1:K10");
new PivotTable("Pivot table");
new PivotTable();
```

In the first example PivotTable will be use sheet at the index 0 from the source document, to get data; it will use data in the range from the cell (0,0) to the cell (10,10) of this sheet; and after creating a new sheet with pivot table it will be renamed into «Pivot table».
In the second example number of sheet not specified, so table will use **currently selected sheet** from the source document; and it will use data from range «A1:K10» (Excel style ranging); and result sheet also will be renamed.
In the third example table range also not specified, so table will use **all data** on the selected sheet of the source document; result sheet also will be renamed.
In the last example no parameters specified, so result sheet will not be renamed.
All configuration of the first example could be set to the already created table like this:

```
PivotTable table = new PivotTable();
table.setName("Pivot table");
table.setSourceRange(0, 0, 10, 10);
table.setSourceRange("A1:K10");
table.setSourceRange(new TableRange("A1:K10"));
table.setSourceSheet(0);
```

In this example name, source range and source sheet are set to existing PivotTable. Note, that lines 3, 4 and 5 are doing exactly the same thing – they all set source range just by different methods. Object TableRange also can be created with 2 pairs of cell coordinates (startRow, startCol, endRow, endCol) or with Excel style string address («A1:K10»). You can set some other parameters of the pivot table:

```
table.setShowDataColumnsOnRow(false);
table.setShowHeader(true);
table.setShowRowButtons(true);
table.setShowTotalCol(true);
table.setShowTotalRow(true);
table.setTargetCell(0, 0);
```

Note that in this examples showed **default values** of this methods. Please read Java documentation for this class to get more info about all methods. Also you can set style of the result pivot table (Excel built-in style):

```
table.setStyle(PivotBuiltInStyles.PivotStyleMedium4);
```

Use SmartXLS enum class PivotBuiltInStyles to set value for this method.

Document

Document class is the entity of an Excel document. Document can has some format, path to file of that document and optionally a password, to read or write the document. You can create a Document like this:

```
new Document(DocumentFormat.XLS, "old table.xls");
new Document(DocumentFormat.XLSX, "new table.xls", "password");
```

Use enum class DocumentFormat to set format of the document. You can use document entities to set «source» and «target» documents of the table:

```
table.setSourceDocument(DocumentFormat.XLSX, "table.xlsx");

Document target = new Document(DocumentFormat.XLSX, "encripted.xlsx", "password");
table.setTargetDocument(target);
```

Quick version is available for both «setSourceDocument» and «setTargetDocument» methods, but it doesn't allow to set passwords. So if you want to create document with a password you should use class constructor. Source and target documents used in a PivotTable to simplify work with SmartXLS. Document class provides functionality to read SmartXLS WorkBook from file or to write existing WorkBook into file. So you can set source and target documents for PivotTable and they will be automatically read and written:

```
table.setSourceDocument(DocumentFormat.XLSX, "source data.xlsx");
table.setTargetDocument(DocumentFormat.XLSX, "pivot table.xlsx");
```

When all configuration set for PivotTable it's time to describe field that will be in the pivot table.

Pivot fields

There's 4 areas in pivot table: page area, row area, column area and data area. Pivot fields is build upon some source data and can be placed in one of those areas. PivotTable framework provides special classes to work with all of them:

1. PivotField class

PivotField is the base for any other class. It describes basic functionality of any field. But you should use this class only to describe fields for «Page» area or «Column» area, because fields in this areas doesn't use any additional functionality. Example:

```
PivotField page = new PivotField(PivotArea.PAGE, "Page data");
table.addField(page);

PivotField column = new PivotField(PivotArea.COLUMN, "Column data");
table.addField(column);
```

Here we created two pivot fields. One of them will be placed in the «Page» area, and the other in the «Column» area. To specify the area of the field you will use enum class PivotArea. String specified after pivot area indentifier is the name of the source data for the field. PivotField allows you to set some basic configuration for a field:

```
page.setColumnWidth(SizeUnit.PIXEL.create(150));
page.setColumnWidthPx(150);
page.setColumnWidth(new Size(SizeUnit.PIXEL, 150));
page.setSortType(SortType.ASCEND);
```

In first 3 rows we set width of the column. Note, that result of any of that operations will be the same, it's just 3 ways to set the same value. In the 4th row we set sorting type for the field data. Use enum class SortType to specify such a value. Please read javadocs for all this classes for better understanding.

2. RowField class

RowField is the PivotField that will be placed in the «Row» area. This class provides additional functionality to configure the rows:

```
RowField row1 = new RowField("Row data");
row1.setOutline(true);
row1.setCompact(true);
row1.setSubtotalTop(true);

RowField row2 = new RowField("Row data", true, true, true);
```

Here we create two row fields. They get the same name of the source data. For the field «row1» we set additional parameters: outline (field will be shown in separate line), compact (child field will be shown in the same column) and subtotal top (subtotal values will be shown in the top line). Note, that second two parameters work only if first one set as true (only outlined field can be compact and have subtotal top). When we create second field «row2» we set all those parameters right in the constructor (in the same order). Also all those parameters are set as true by default, so you should set them only if you want false.
You cannot change PivotArea of a RowField (same with a DataField and FunctionField). But you can set same parameters as for simple PivotField:

```
row2.setColumnWidthPx(150);
row2.setSortType(SortType.ASCEND);
```

Note, that width of the column can be changed downwards only once. It means that if we will create two row fields «Big data» and «Small data» and set «Big data» as compact – they will be placed in the same column. If we set width of the «Big data» as 150px and width of the «Small data» as 100px – when they will be placed in the table, firstly column width will be set to 150px and then width of second row field will be ignored. If we change places of width («Big data» - 100px, «Small data» - 150px) – firstly column width will be set as 100px and then changed to 150px.

3. DataField class

This is the most interesting of all fields. DataField is a PivotField that will be placed into «Data» area. It also provides additional functionality only for that kind of field:

```
DataField data1 = new DataField("Data 1");
data1.setName("First data");
data1.setNumberFormatting("0.0");
data1.setSummarizeType(SummarizeType.SUM);
data1.setColumnWidthPx(150);

DataField data2 = new DataField("Data 2", "Data 2", "0.00", SummarizeType.AVERAGE);
data2.setColumnWidthPx(100);
```

Here we created two data fields. First field «data1» use source data by the name «Data 1», name of the data field in the pivot table will be changed to «First data», number values in this field will be formatted by pattern «0.0» (round to first number after decimal point), subtotal result will be calculated as sum of all values and width of the column will be set to 150px.
For the second field «data2» we set most of the same parameters in the constructor. Note, that pivot table cannot have source data fields and pivot fields with the same name (case insensitive). If you specify the same name of the source data and name of the field (like we did for the «data2») - then name of the field will be changed by adding one dot (.) at the end of the name. Name of the pivot field «data2» will be «Data 2.»
Note that if in you final configuration there will be two fields with the same name – you will get an error.
Also you can set sort type for the data field but it won't have any effect, because sorting of the rows described by row fields.

4. FunctionField class

This class provides functionality to create data field, that weren't initially in the model. You can do that by setting a formula. New field will be created by calculating this formula for every row of other data fields. Note, that you should use names of source data fields in your formula and not names of the result pivot fields, or column letters:

```
DataField data1 = new DataField("Data 1", "First data");
DataField data2 = new DataField("Data 2", "Second data");

FormulaField formula1 = new FormulaField(" 'Data 1' + 'Data 2' ");
formula1.setName("Formula 1");
formula1.setNumberFormatting("0");
formula1.setSummarizeType(SummarizeType.PRODUCT);

String f2 = " 'Data 1' * 'Data 2' ";
FormulaField formula2 = new FormulaField(f2, "Formula 2", "0", SummarizeType.PRODUCT);
```

Here we created two «Data» fields that uses fields «Data 1» and «Data 2» as source, and names of result fields will be: «First data» and «Second data». Then we create two formula fields with specified formulas. As you can see, in the formula you should use name of the **source data field**. You can set all parameters of the DataField to the FormulaField (because FormulaField is a DataField, just with formula instead of source).

Any type of a field you can add to the PivotTable by the same method:

```
table.addField(new PivotField(PivotArea.PAGE, "Page data"));
table.addField(new RowField("Row data"));
table.addField(new DataField("Data", "Data"));
table.addField(new DataField("Values", "Value"));
table.addField(new FormulaField("Data * Value"));
```

So, when you've configured your PivotTable, and added all fields that you need in it. It time to convert!

PivotTableConverter

Special class «PivotTableConverter» provides all functionality to convert simple excel table into pivot table. It has 3 main methods:

1. `void convert(WorkBook source, PivotTable table)`

This method gives you ability to convert SmartXLS WorkBook that you already have, by the rules that you have set in your PivotTable (Remember, that PivotTable class it's just the rules how to convert real table. It doesn't have any real data in it.)
To use this method you should get yourself instance of the WorkBook, and PivotTable (we will use variable «table» that we have created in the earlier examples):

```
WorkBook wb = new WorkBook();
wb.readXLSX("source data.xlsx");

PivotTableConverter.convert(wb, table);

wb.writeXLSX("pivot table.xlsx");
```

As you can see, with this method, work of reading and writing actual excel files is lying on you.

2. `WorkBook convert(PivotTable table, boolean writeTarget)`

This method doesn't asks you for WorkBook, because he tries to creates one of his own. To use this method you should have «source document» option set in your PivotTable (otherwise exception will be thrown). Method reads configuration of the source document for PivotTable (format and the file path), reads this file, loads WorkBook, and calls previous method.
Method also can write WorkBook into file after converting. For that you should set «writeTarget» parameter as true, and «target document» option should be set in PivotTable (otherwise – exception, yes). If «writeTarget» is true and «target document» is set – method reads configuration of the document and write converted WorkBook into file.
After all of this converted WorkBook is returned as result of the method. So you can call this method if you want to load WorkBook, convert it and use without writing. Example:

```
table.setSourceDocument(DocumentFormat.XLSX, "source data.xlsx");
WorkBook wb = PivotTableConverter.convert(table, false);
```

3. `WorkBook convert(PivotTable table)`

This method is just calling previous one with «writeTarget» parameter as true. Example:

```
table.setSourceDocument(DocumentFormat.XLSX, "source data.xlsx");
table.setTargetDocument(DocumentFormat.XLSX, "pivot table.xlsx");
PivotTableConverter.convert(table);
```

Note that converted WorkBook is also returned as result of this method.

Saving PivotTable

Now you already know hot to create PivotTable, how to configure it, how to add fields, and how to convert real tables. Bu there's one function left in this framework. You can save PivotTable as file, or load it from a file.
If you need to convert pivot tables just once in a while, you can create configuration that you need, and export it to the file:

```
XMLProvider prov = new XMLProvider();
prov.saveConfiguration("table.xml", table);
```

When you would need to use it again, you can load table from file just by two lines of code:

```
XMLProvider prov = new XMLProvider();
PivotTable table = prov.loadConfiguration("table.xml");
```

Or even with one:

```
PivotTable table = new XMLProvider().loadConfiguration("table.xml");
```

XML is a quite simple and human readable format, so once you've exported the state you can use this one file, just by adding changes that you need, and loading new table every time.

That's it! Good luck with this framework, and happy using!