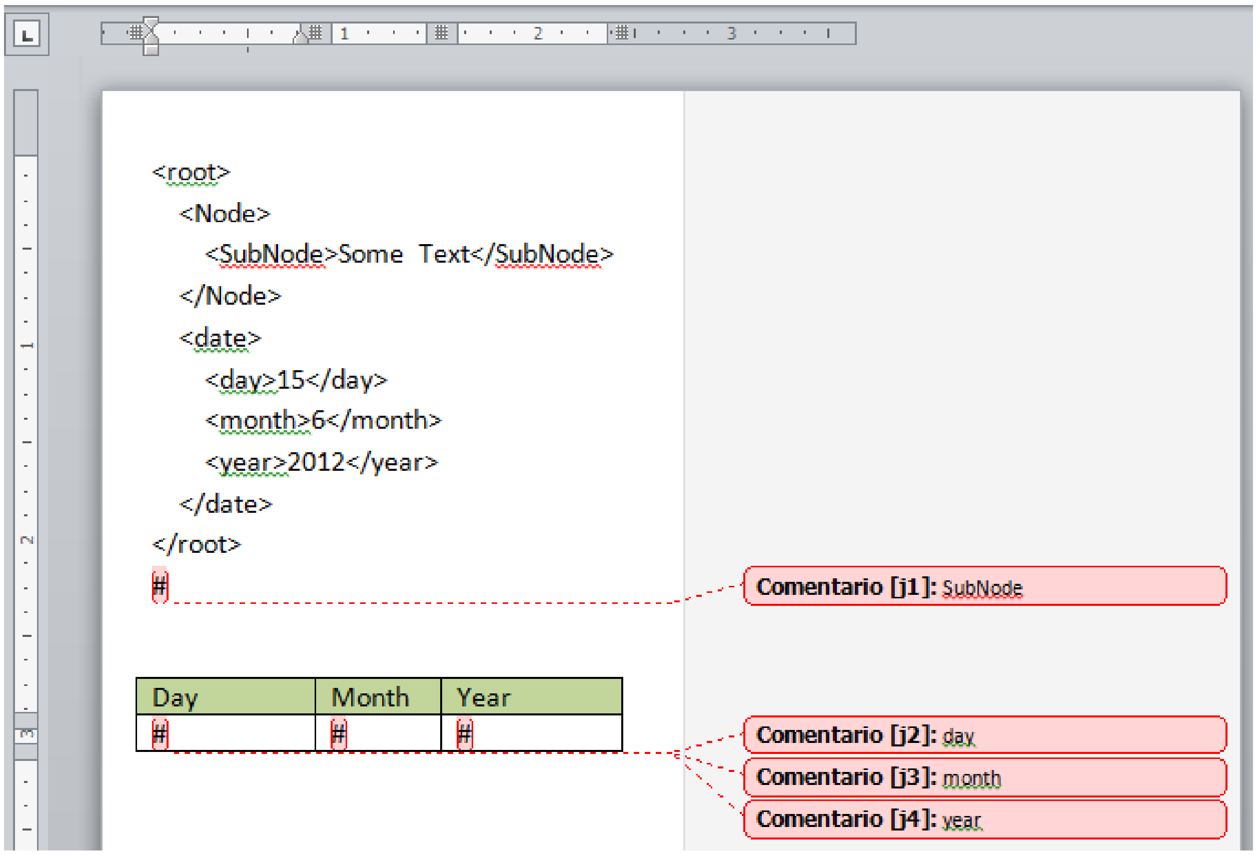
How to create a simple Report, testing your reports.

1st of all, you need to know the structure of the data XML your report will consume. Take a look at the example in the image. The values that the report will try to consume are denoted by a hash character (#) commented with the node name, in the example [SubNode, day, month, year]



```
<root>
  <Node>
    <SubNode>Some  Text</SubNode>
  </Node>
  <date>
    <day>15</day>
    <month>6</month>
    <year>2012</year>
  </date>
</root>
#
```

Comentario [j1]: SubNode

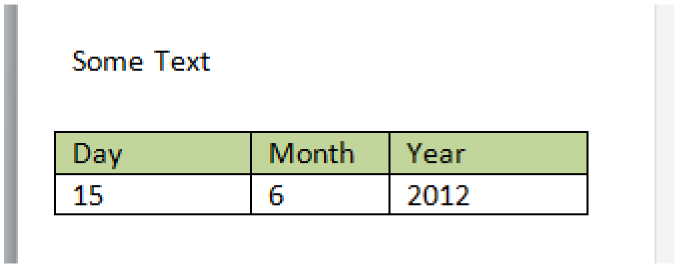
Day	Month	Year
#	#	#

Comentario [j2]: day

Comentario [j3]: month

Comentario [j4]: year

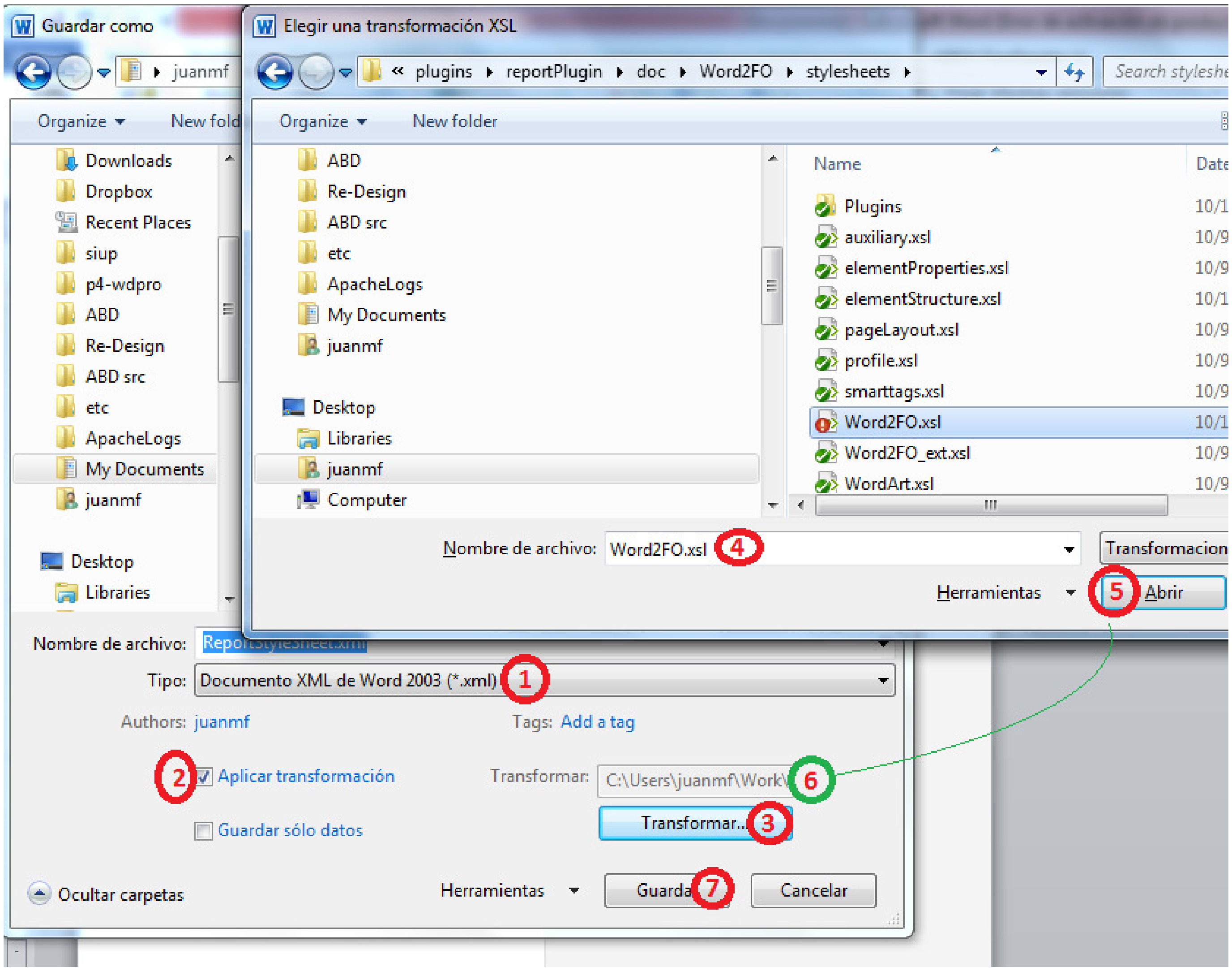
The previous template should render something like:



Some Text

Day	Month	Year
15	6	2012

To achieve that, we need to save the word document in Word 2003 XML format (WordML) (ReportStylesheet.xml). Once you saved the report template in WordML you can proceed to save it again but this time applying the transformation to generate the actual XSLT report template (ReportStylesheet.xsl) *follow the steps in the image*.



How to configure the Engine

Once you have ReportStylesheet.xsl saved, you need to add it to the new report configuration in conjunction with a provider method that must generate the XML data that will be handed over to the Report stylesheet.

The provider method is optional, if you use the engine as a library instead of as a controller with its own route, you can generate the XML data as you wish. More on that later…

Provider method example follows:

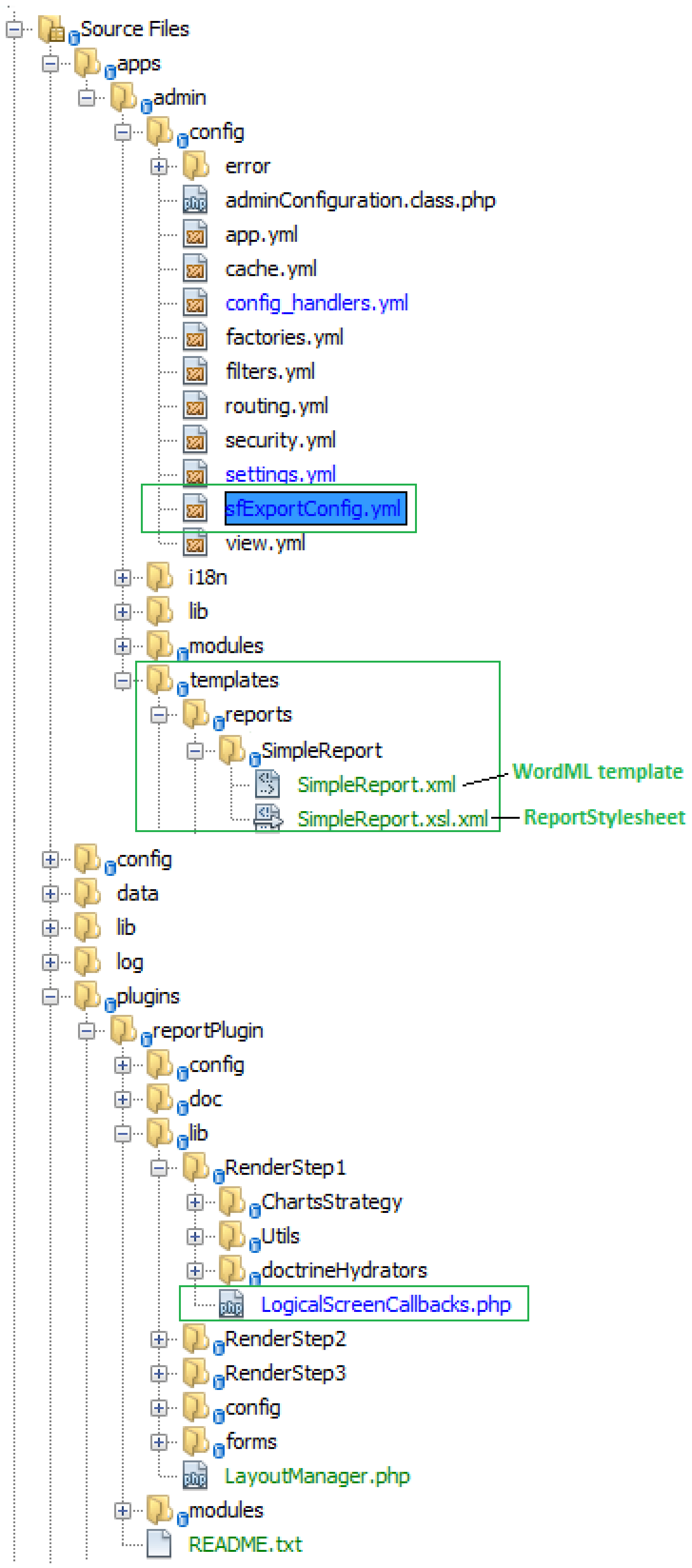
```
// projects\src\plugins\reportPlugin\lib\RenderStep1\LogicalScreenCallbacks.php
/**
 * This class helps in the Step1 of a three steps Process.
 *
 * The process consists of the following steps:
 *
 * Step 1 consists in generating the XML based raw data for the report. RawXMLData
 * Step 2 consists in merging this raw data with a XSL-FO template to give it
 * the presentation information. XSL-FO acts as an intermediate language used
 * to render the final report in any format with a last transformation.XSL-FO
 * Step 3 consists on rendering the XSL-FO representation (the intermediate
 * language) of the report to the desired output format. FinalReport
 *
 * This class should be used to add the methods that creates or access the
 * RawXMLData and returns it to be processed in Step2.
 *
 * To help extract data from database in XML format we have a Doctrine XML Hydrator
 * you can see a usage example in LogicalScreenCallbacks::helloworld().
 *
 * @author Juan Manuel Fernandez <juanmf@gmail.com>
 * @see    %sf_plugins_dir%/reportPlugin/config/sfExportConfig.yml
 */
class LogicalScreenCallbacks
{
    /**
     * This method generates the xml data for the 'How to create a simple Report'
     * topic of the Report Engine tutorial
     *
     */
    public static function howToSimple()
    {
        $xml = <<<XML
<root>
    <Node>
        <SubNode>Some Text</SubNode>
    </Node>
    <date>
        <day>15</day>
        <month>6</month>
        <year>2012</year>
    </date>
</root>
XML;
        $q = new DOMDocument();
        $q->loadXML($xml);
        return $q;
    }
}
```

Now that we have both, **data** and **report Stylesheet** we need to tell the report engine that they go together. For that we use a YAML configuration file that associates the Stylesheet with the data provider method, *note that the callback is a* `[ClassName, methodName]` *notation*:

```
# project\src\apps\admin\config\sfExportConfig.yml
reports:
  SimpleReport:
    name: Simple Report
    step1: # Cooking Data, optional Step.
      logical_screen_callback: [LogicalScreenCallbacks, howToSimple]
      # e.g. [p1, p2, ..] using this key, the action can pass parameters too.
      callback_params: ~
    step2: # Merging data and Report XSLT Stylesheet
      structure:
        default:
          layout:
            # No need to know this for now.
            class: ~
            # An XPath expression selecting data nodes for this view.
            nodes: '/root'
            xslt_structure:
              # A XML to FO transformation adding structure. Path from %sf_app_dir%
              sheet: '/templates/reports/SimpleReport/SimpleReport.xsl.xml'
              # Associative array with relevant parameters for this xslt.
              params: {}
            xslt_style:
              # A FO to FO transformation adding style. Path from %sf_app_dir%
              sheet: ~
              # Associative array with relevant parameters for this xslt.
              params: {}
```

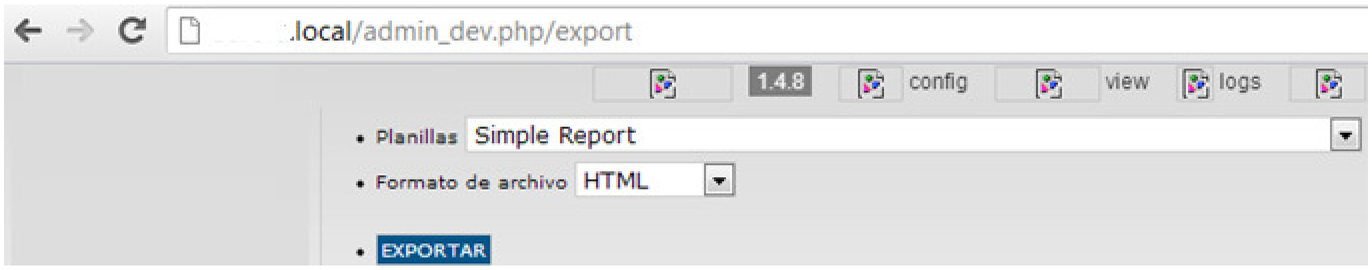
The files should be placed accordingly, as described in the configuration. Note that the paths in config are relative to the *apps* directory, in the following image you can see the relevant files.

File hierarchy:



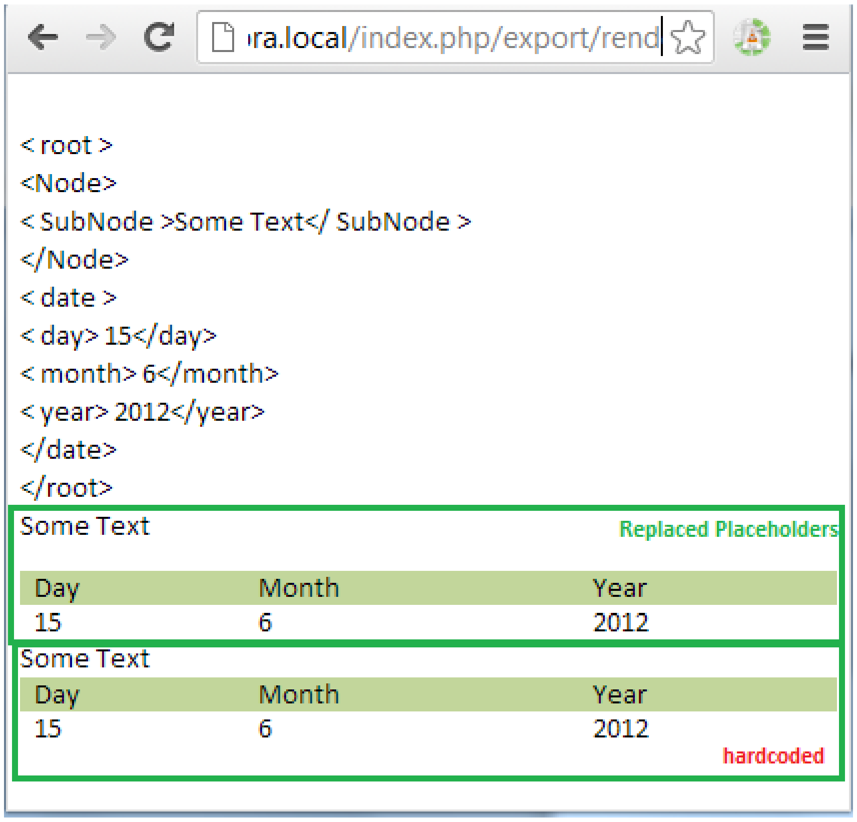
```
Source Files
  apps
    admin
      config
        error
        adminConfiguration.class.php
        app.yml
        cache.yml
        config_handlers.yml
        factories.yml
        filters.yml
        routing.yml
        security.yml
        settings.yml
        sfExportConfig.yml
        view.yml
      i18n
      lib
      modules
      templates
        reports
          SimpleReport
            SimpleReport.xml
            SimpleReport.xsl.xml
  config
  data
  lib
  log
  plugins
    reportPlugin
      config
      doc
      lib
        RenderStep1
          ChartsStrategy
          Utils
          doctrineHydrators
          LogicalScreenCallbacks.php
        RenderStep2
        RenderStep3
        config
        forms
        LayoutManager.php
      modules
      README.txt
```

Then you go to the "http://**virtualHost.local/index.php/export**" route to see the report selection form, which now contains the new report descriptor, remember that *Simple Report* is the **name** configuration value:



- Planillas: Simple Report
- Formato de archivo: HTML
- EXPORTAR

If you click "EXPORTAR" you get this:



```
< root >
<Node>
< SubNode >Some Text</ SubNode >
</Node>
< date >
< day> 15</day>
< month> 6</month>
< year> 2012</year>
</date>
</root>
```

Some Text

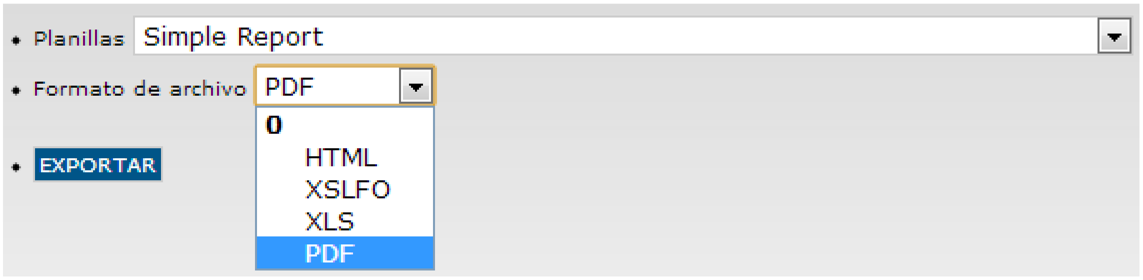
Day	Month	Year
15	6	2012

Some Text

Day	Month	Year
15	6	2012

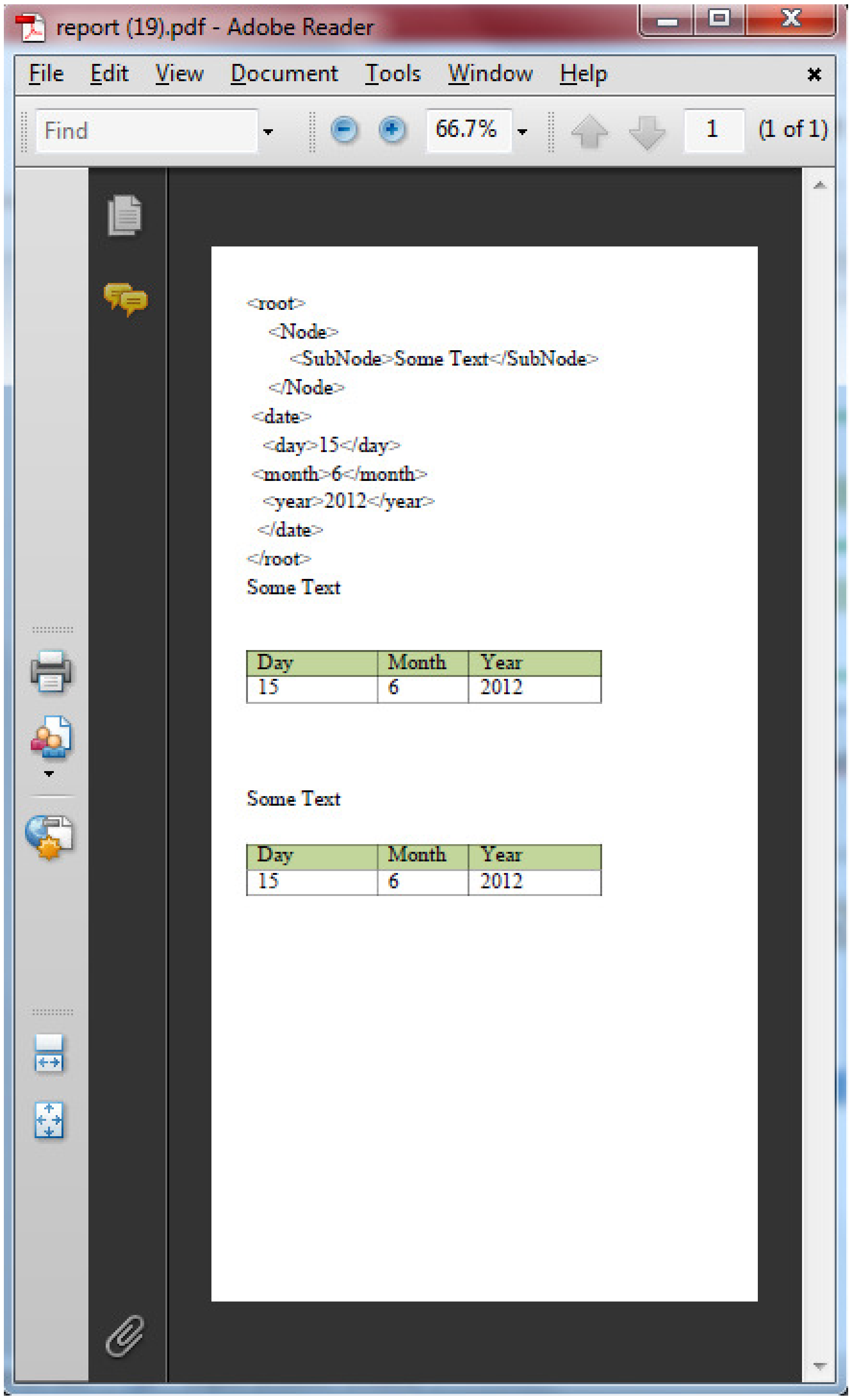
Note: Chrome doesn't show borders if they are thinner than "1pt;", but Firefox'd show them and they are in the HTML markup.

If you select PDF and you have ApacheFOP webservice running (or the local executable, which still needs a calling rederer, @see Step3);



- Planillas: Simple Report
- Formato de archivo: PDF
- EXPORTAR

You'd get this (*note that the 2nd table is hardcoded in the report template, the 1st hast the placeHolders*):


report (19).pdf - Adobe Reader
File Edit View Document Tools Window Help
Find
66.7%
1
(1 of 1)

```
<root>
   <Node>
      <SubNode>Some Text</SubNode>
   </Node>
 <date>
   <day>15</day>
 <month>6</month>
   <year>2012</year>
  </date>
</root>
```

Some Text

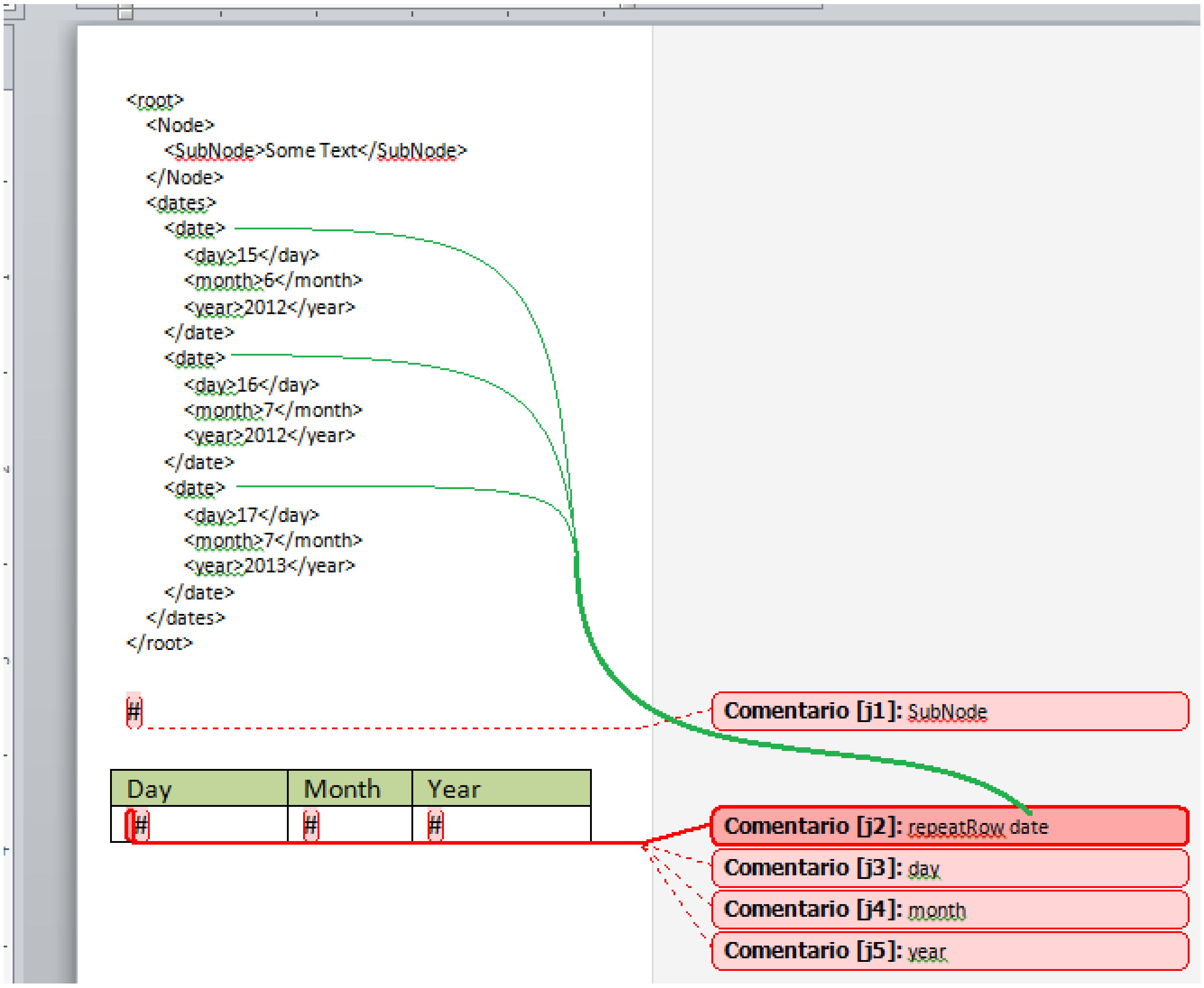
Day	Month	Year
15	6	2012

Some Text

Day	Month	Year
15	6	2012

A Slightly Harder Report, With Repeating Rows

The only difference (*besides the irrelevant fact that I removed the hardcoded "expected" table*) with the *Simple Report* example is that this time I added a new Comment surrounding a **leading white space in the 1st cell** with a command that gets interpreted by the modified Word2FO.xsl stylesheet and generates a new XSLT template that will repeat for each **date** and will be called from inside the table body. It's important to know that whenever you use a repeat command, either **repeatTable** , **repeatRow** or **repeatCell**, the context for the repeating block (table, Row or Cell) is the current repeating node being iterated. In the following example that'd be **1st date** first, **2nd date** after and finally **3rd date** node. Therefor the **day**, **month** and **year** node selectors in the following Word Comments are relative to their **date ancestor** (that also happens to be parent, but that doesn't matter).



```
<root>
  <Node>
    <SubNode>Some Text</SubNode>
  </Node>
  <dates>
    <date>
      <day>15</day>
      <month>6</month>
      <year>2012</year>
    </date>
    <date>
      <day>16</day>
      <month>7</month>
      <year>2012</year>
    </date>
    <date>
      <day>17</day>
      <month>7</month>
      <year>2013</year>
    </date>
  </dates>
</root>
```

#

Day	Month	Year
#	#	#

Comentario [j1]: SubNode

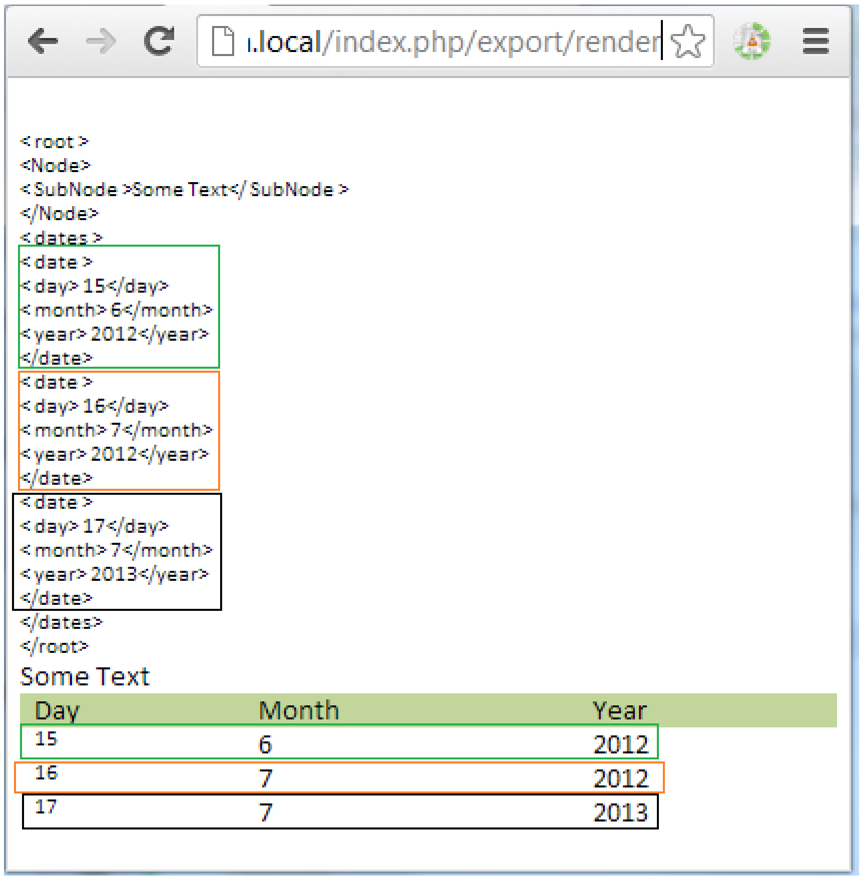
Comentario [j2]: repeatRow date

Comentario [j3]: day

Comentario [j4]: month

Comentario [j5]: year

When you select the HTML output you get this:



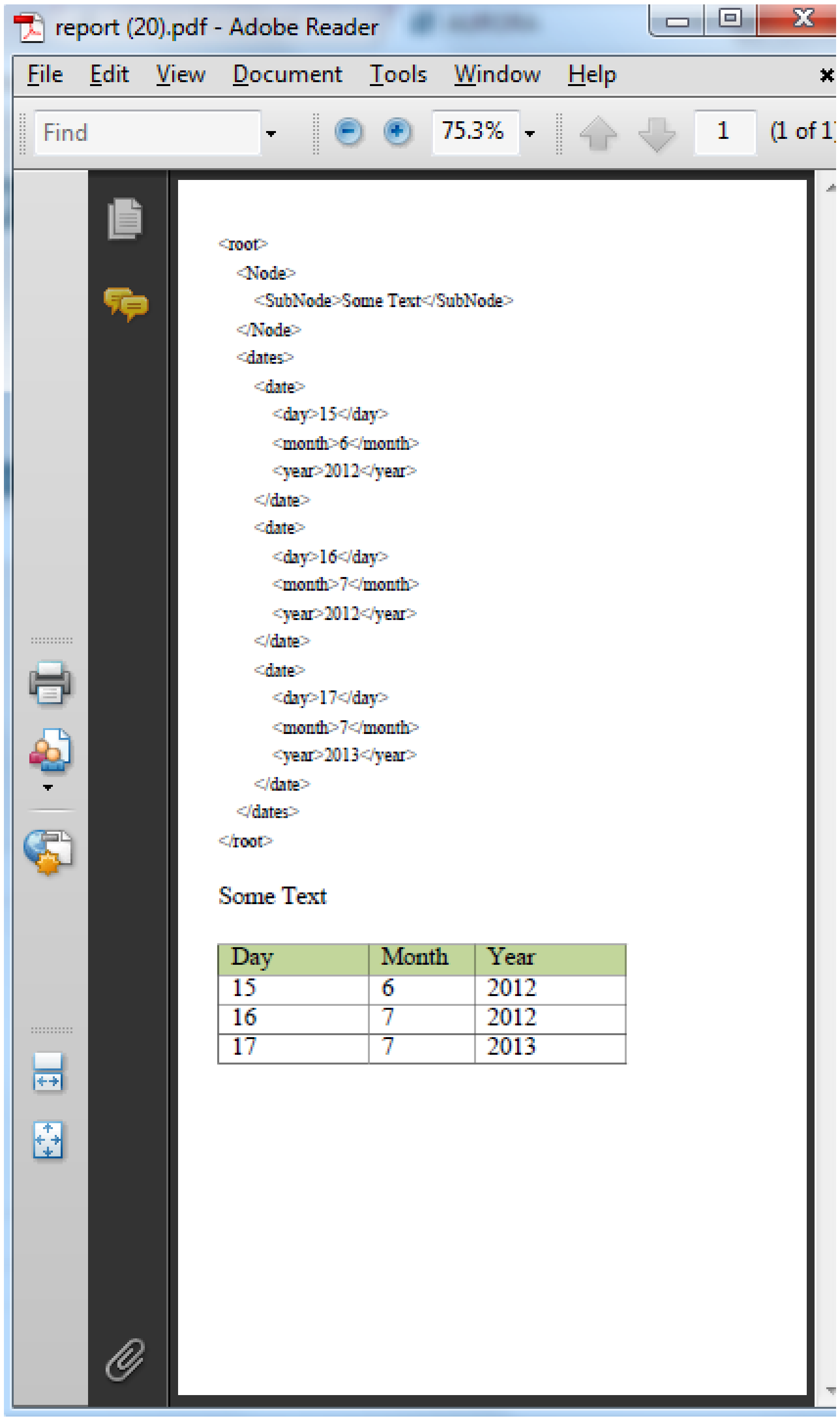
ı.local/index.php/export/render

```
< root >
<Node>
< SubNode >Some Text</ SubNode >
</Node>
< dates >
< date >
< day> 15</day>
< month> 6</month>
< year> 2012</year>
</date>
< date >
< day> 16</day>
< month> 7</month>
< year> 2012</year>
</date>
< date >
< day> 17</day>
< month> 7</month>
< year> 2013</year>
</date>
</dates>
</root>
```

Some Text

Day	Month	Year
15	6	2012
16	7	2012
17	7	2013

For the PDF output of the same report you get:


report (20).pdf - Adobe Reader
File Edit View Document Tools Window Help
Find
75.3%
1
(1 of 1

```
<root>
  <Node>
    <SubNode>Some Text</SubNode>
  </Node>
  <dates>
    <date>
      <day>15</day>
      <month>6</month>
      <year>2012</year>
    </date>
    <date>
      <day>16</day>
      <month>7</month>
      <year>2012</year>
    </date>
    <date>
      <day>17</day>
      <month>7</month>
      <year>2013</year>
    </date>
  </dates>
</root>
```

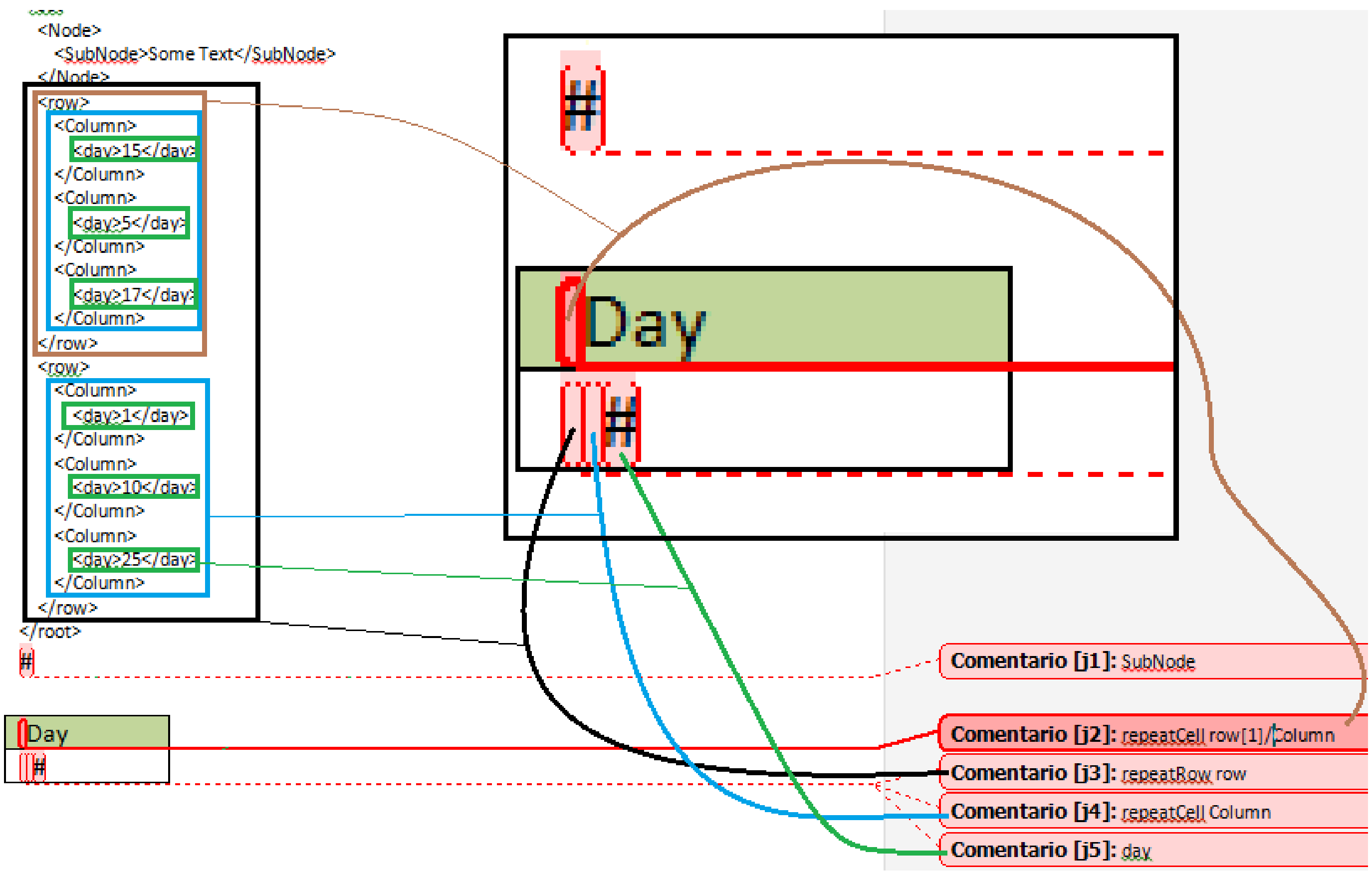
Some Text

Day	Month	Year
15	6	2012
16	7	2012
17	7	2013

As you can see, page metrics are preserved in the PDF output, margins, page size and orientation, etc.

Repeating Cells, for variable Column number

Now we’ll make cells repetitive, inside repetitive columns, for that I changed the data so we have a matrix of days, with an XPath view like this “`//row/Column/day`“



```
 <Node>
   <SubNode>Some Text</SubNode>
 </Node>
 <row>
   <Column>
     <day>15</day>
   </Column>
   <Column>
     <day>5</day>
   </Column>
   <Column>
     <day>17</day>
   </Column>
 </row>
 <row>
   <Column>
     <day>1</day>
   </Column>
   <Column>
     <day>10</day>
   </Column>
   <Column>
     <day>25</day>
   </Column>
 </row>
</root>
```

Comentario [j1]: SubNode

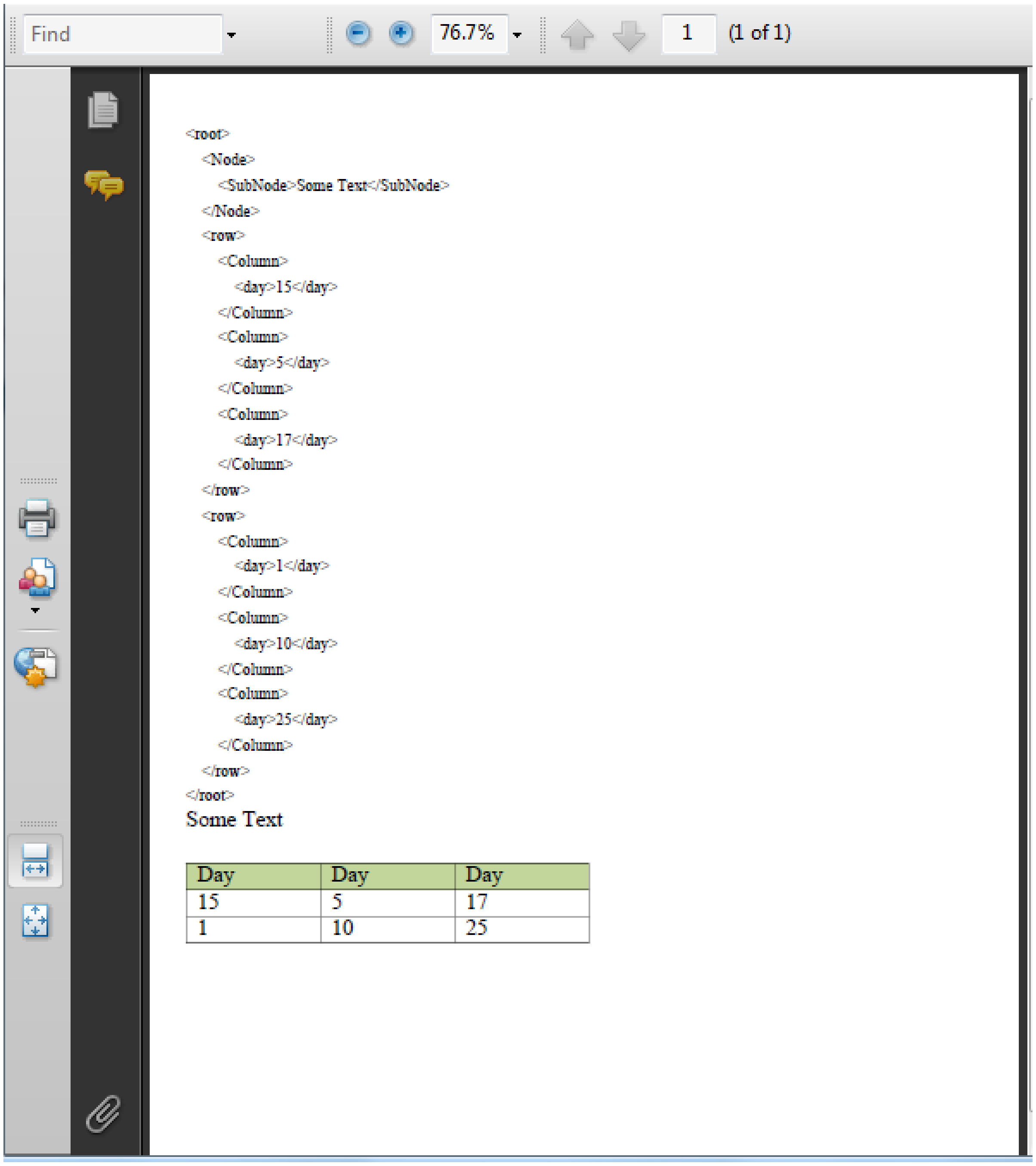
Comentario [j2]: repeatCell row[1]/Column

Comentario [j3]: repeatRow row

Comentario [j4]: repeatCell Column

Comentario [j5]: day

The PDF output of this Report StyleSheet looks like this.



```
<root>
  <Node>
    <SubNode>Some Text</SubNode>
  </Node>
  <row>
    <Column>
      <day>15</day>
    </Column>
    <Column>
      <day>5</day>
    </Column>
    <Column>
      <day>17</day>
    </Column>
  </row>
  <row>
    <Column>
      <day>1</day>
    </Column>
    <Column>
      <day>10</day>
    </Column>
    <Column>
      <day>25</day>
    </Column>
  </row>
</root>
```

Some Text

Day	Day	Day
15	5	17
1	10	25

As you can see, Header cells repeated three times, once for each **Column** node inside 1st **row** (that's "`repeatCell row[1]/Column`"). The two body rows obey to the two **row** nodes inside **root** (that's "`repeatRow row`"). On each row in turn, we get three cells (that's "`repeatCell Column`"), this works because in a current **row** context, XPath finds only 3 **Column** nodes.

Obviously, these repetitive cells would also work without repetitive rows. This example shows both toghether.

Adding Images