Displaying JpGraph in TYPO3

Laurent Foulloy

Polytech'Savoie - University of Savoie – France

yolf.typo3@orange.fr

1 Abstract

The aim of this paper is to present the extension sav_jpgraph available in the TER which enables the user to display graphs produced by the JpGraph library in TYPO3 web sites.

The extension introduces a new concept for implementing such graphs. It relies on XML templates which, in many cases, are easily written from the php examples provided with the JpGraph Library. Markers can be introduced in templates in order to change their behavior. Data can also be changed through XML either manually or by means of queries.

After having introduced JpGraph and existing extensions in the TER, in this paper we will focus on the technical aspects of the XML template processing. Examples of template writing and results will be provided.

2 Introduction

JpGraph is an Object-Oriented Graph creating Library for php. The library makes the creation of many types of graph possible. Information on JpGraph can be obtained at http://www.aditus.nu/jpgraph/. The software licence indicates: "*JpGraph is released under a dual license. QPL 1.0 (Qt Free Licensee) For non-commercial, open-source or educational use and JpGraph Professional License for commercial use. The professional version also includes additional features and support.*"

Five extensions concerning JpGraph are available in the TER.

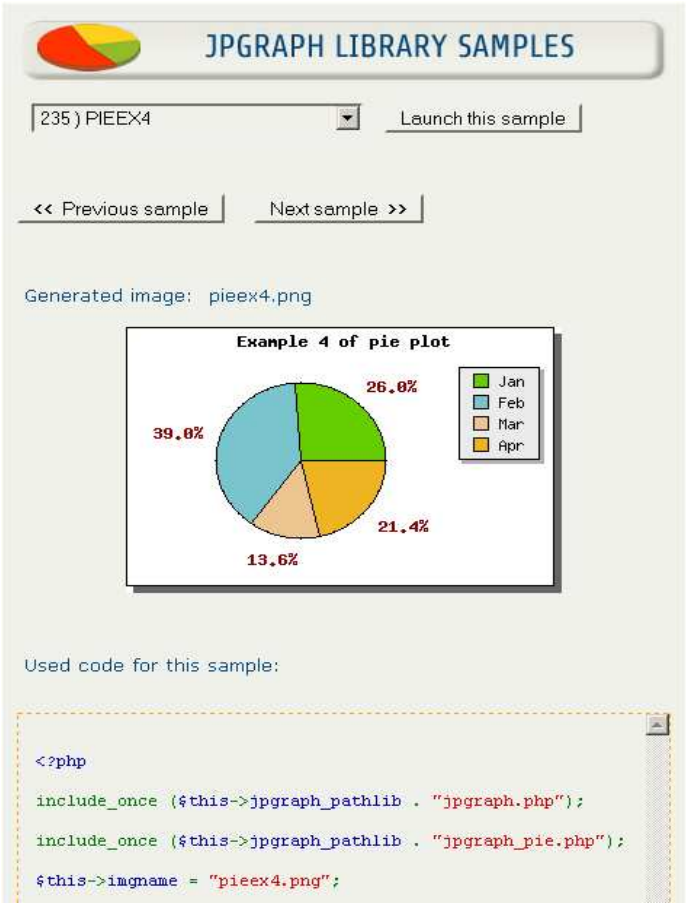
- The extension **JPGraphs Charts (jpgraph)** was released for the first time on March 2003. At the present time, the latest update is version 0.3.11 (June 2006). The JpGraph files are in the directory "jp". It is an old version of JpGraph1.x for php4. Let us note that JpGraph1.27 is the-end-of-life version for phh4.
- The extension **Database JpGraph (mh_omdbchart)** was released for the first time on August 2006. At the present time, the latest update is version 0.10.0 (may 2007). The JpGraph files are in the directory "pi1/jpgraph". It is also an old version of JpGraph1.x for php4 (version 1.20.3). The extension allows the use of queries. It requires the extension "adodb" to be installed.
- The extension **JPGraph Library V2.2 for PHP5 (rt_jpgraphlib)** was released in September 2007 under version 2.2.0 always available as it in the TER. This extension contains version 2.2 of JpGraph in the directory "jpgraph". Let us note that the latest version of the JpGraph Library was released in January 2009 as version 2.3.4.
- The extension **JPGraph Library V2.2 Tutorial (rt_jpgraphtutor)** was released at the same time as rt_jpgraphlib. It contains the large set of examples provided with the JpGraph Library. Of course, the extension requires rt_jpgraphlib to be installed. This extension is very useful to browse the examples.
- The extension **SAV JpGraph (sav_jpgraph)** was released in February 2009. At the present time, the latest release is the version 0.0.10 (August 2009). The extension contains the latest version of

the JpGraph Library, that is the version 3.0.2. Files are in the directory “src” which also contains the directory “examples”. This extension has introduced a new concept of XML-based templates for JpGraph.

3 Building a template from php

3.1 Introduction

The JpGraph library includes hundreds of examples which are in the directory “src/examples”. They provide a very good basis to start the development of a template. We recommend to install “rt_jpgraphlib” and “rt_jpgraphtutor” to browse the examples. The extension “rt_jpgraphtutor” displays the php code and the resulting graph as shown below.



Introducing JpGraph in TYPO3 requires to execute the php code either by means of a generic extension or a specific one. The former is quite complex to develop because they are many possible configuration options in JpGraph. The extension “JPGraphs Charts” attempts to do so but it was not completed. Furthermore if a new graph class is added to the JpGraph Library, a new development has to be undertaken. The latter can be a solution, however it is not flexible when one wants to change one or several parameters.

In many cases, the general structure of the graph will not change, only very few parameters will be modified to go from one graph to another. The main idea developed in “SAV JpGraph” is to get rid of the php code by transforming it into XML. The XML code provides a template which will be easily modified by means of XML tags associated with markers, data, queries, ...

In this section we explain how to build an XML template from the php code. For the sake of simplicity, this process is illustrated from the above example of a pie plot (pieex4.php). The file content is given below.

```
1.      <?php
2.      include ("../jpgraph.php");
3.      include ("../jpgraph_pie.php");
4.
5.      $data = array(40,60,21,33);
6.
7.      $graph = new PieGraph(300,200,"auto");
8.      $graph->SetShadow();
9.
10.     $graph->title->Set("Example 4 of pie plot");
11.     $graph->title->SetFont(FF_FONT1,FS_BOLD);
12.
13.     $p1 = new PiePlot($data);
14.     $p1->value->SetFont(FF_FONT1,FS_BOLD);
15.     $p1->value->SetColor("darkred");
16.     $p1->SetSize(0.3);
17.     $p1->SetCenter(0.4);
18.     $p1->SetLegends(array("Jan","Feb","Mar","Apr","May"));
19.     $graph->Add($p1);
20.
21.     $graph->Stroke();
22.
23.     ?>
```

The transformation to XML relies on six steps:

1. Transforming the data,
2. Transforming the object creations,
3. Transforming the method calls,
4. Transforming the parameters,
5. Adding references,
6. Overloading attributes by references.

In the following paragraphs we will detail each step from the previous example.

3.2 Transforming the data

Data are pieces of information used for the graph display. In general they are in php arrays either in php variables or directly in the code. In our example, a php array variable is used in line 5 while the data for the legend are directly inserted in the code (cf. line 18).

Each array is transformed into a XML <data> tag. Data are identified by means of a reserved attribute “id”. Lines 5 and 18 become respectively:

```
<data id="defaultData">
   40,60,21,33
</data>
<data id="defaultLegend">
  Jan,Feb,Mar,Apr,May
</data>
```

3.3 Transforming the object creations

In our example, two objects are created in lines 7 and 13, respectively a “PieGraph” and a “PiePlot”. Let us note that because the “PiePlot” is used before the “PieGraph” object, it is more natural to define it first. In the transformation process to XML, we will associate each object with a new XML tag whose name is the object name. The objects are identified, as in the case of date, using the attribute “id” which can be a number, for example. Lines 7 and 13 become:

```
<PiePlot id="1">
...
</PiePlot>

<PieGraph id="1">
...
</PieGraph>
```

3.4 Transforming the methods

In line 8, the method “SetShadow” is called from the object “$graph”. To convert the method call, a XML tag whose name is the name of the method, is inserted inside the object tag. Therefore, the tag <SetShadow> is inserted in the tag <PieGraph> as shown below.

```
<PieGraph id="1">
  <SetShadow />
</PieGraph>
```

When the method is called from an internal object as in lines 10 and 11 in which the methods “Set” and “SetFont” are called from the internal object “title”, the XML tag associated with the method is inserted inside the internal object tag. The latter is itself inserted inside the object tag as shown below.

```
<PieGraph id="1">
  <SetShadow />
  <title
    <Set ... />
    <SetFont ... />
  </title>
</PieGraph>
```

3.5 Transforming the parameters

Parameters associated with objects or methods are simply transformed as XML tag attributes. The name of the attribute is not important. However, it is a good practice to have attributes that make sense. The order of the attributes is the same as in the method or object calls.

Let us analyze the cases of the “PieGraph”. The first transformation concerns line 7 in which the creation of the object “PieGraph” requires two paragraphs. The XML code becomes:

```
<PieGraph id="1" width="300" height="200" type="auto">
...
</PieGraph>
```

The second transformation concerns lines 10 and 11 in which parameters are associated with the method calls. The XML code becomes:

```
<PieGraph id="1" width="300" height="200" type="auto">
  <SetShadow />
  <title>
    <Set title="Example 4 of pie plot" />
    <SetFont family="FF_FONT1" style="FS_BOLD" />
  </title>
  ...
</PieGraph>
```

3.6 Adding references

References are the way to use existing objects, data, queries, ... when they are required. In our example, it occurs in lines 13 and 19. The former is related to the use of data in the “PiePlot” while the latter concerns the use of the “PiePlot” object which should be added to the “PieGraph”.

The reference is identified by means of a reserved attribute “ref”. The syntax of the attribute value is “tagName#id”.

Line 13 concerns the creation of the object “PiePlot” which refers to the “$data” variable. Since this variable was transformed into a XML tag <data> with the “id” attribute sets to “defaultData”, the creation of the “PiePlot” in XML becomes:

```
<PiePlot id="1" ref=”data#defaultData” >
...
</PiePlot>
```

Similarly, line 19, which concerns the call of the method “Add” from the variable “$graph” associated with the object “PieGraph”, refers to the variable “$pi1” associated with the object “PiePlot”. Since the “PiePlot” object was transformed into a XML tag <PiePlot> with the “id” attribute set to 1, the definition of the “PieGraph” in XML becomes:

```
<PieGraph id="1" width="300" height="200" type="auto">
  <SetShadow />
  <title>
    <Set title="Example 4 of pie plot" />
    <SetFont family="FF_FONT1" style="FS_BOLD" />
  </title>
  <Add ref=”PiePlot#1” />
  ...
</PieGraph>
```

3.7 Overloading attributes with references

Attributes can be easily overloaded. Replacing them by specific values to adapt the template to the user's requirements becomes possible. To overload the attribute “myAttribute”, the syntax is:

ref_myAttribute=“reference” where “reference” is the same as in the previous paragraph.

For example, let us illustrate how to overload the title of the graph, i.e. “Example 4 of pie plot” by an external marker whose “id” is “title”. The title was introduced as the attribute “title” of the “Set” method as shown below:

```
<PieGraph id="1" width="300" height="200" type="auto">
  ...
    <Set title="Example 4 of pie plot" />
  ...
</PieGraph>
```

The attribute “title” will be overloaded as follows:

```
<PieGraph id="1" width="300" height="200" type="auto">
  ...
    <Set title="Example 4 of pie plot" ref_title="marker#title" />
  ...
</PieGraph>
```

Let us note that the principle can be applied to any attribute. Therefore if a reference attribute, that is an attribute whose name is “ref”, has to be overloaded, one must use the attribute “ref_ref”. It is the case, for example, to overload default data by external data for the PiePlot or the legend which leads to:

```
<PiePlot id="1" ref="data#defaultData" ref_ref="data#data" >
  ...
    <SetLegends ref="data#defaultLegend" ref_ref="data#legend" />
  ...
</PiePlot>
```

3.8 Final template

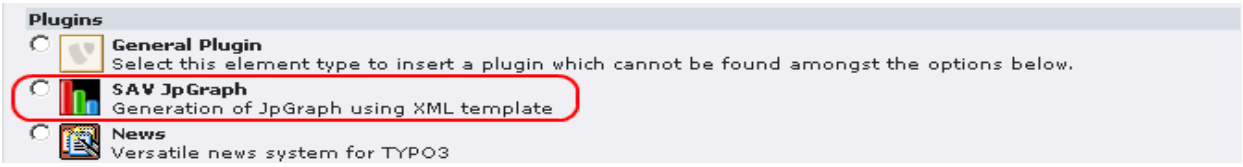
The final template associated with the example is provided below. Let us note that it is a good practice to introduce a comment which details the external tags to overload attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<jpgraph>

  <!-- External tags that can be used for the template pieex4.xml
<data id="data"></data>                    // data set
<data id="legend"></data>                  // Legend data set
<marker id="title"></marker>               // title of the graph
<marker id="backgroundColor"></marker>     // Color of the graph background
<marker id="xCenter"></marker>             // x coord for the center
<marker id="yCenter"></marker>             // y coord for the center
<marker id="width"></marker>               // Graph width
<marker id="height"></marker>              // Graph height
<file id="1"><file>                        // File name where to save the image
   -->


  <!-- Define the data -->
  <data id="default">
    40,60,21,33
  </data>
  <data id="defaultLegend">
    Jan,Feb,Mar,Apr,May
  </data>

  <!-- Create the pie plot  -->
  <PiePlot id="1" ref="data#default" ref_ref="data#data">
    <SetSize param="0.3" />
    <SetCenter xCenter="0.4" ref_xCenter="marker#xCenter" yCenter="0.5" ref_yCenter="marker#yCenter"/>
    <value>
      <SetFont family="FF_ARIAL" style="FS_BOLD" />
      <SetColor color="darkred" />
    </value>
    <SetLegends ref="data#defaultLegend" ref_ref="data#legend" />
  </PiePlot>

  <!-- Set up the graph -->
  <PieGraph width="300" ref_width="marker#width" height="200" ref_height="marker#height" type="auto">
    <SetShadow />
    <SetMarginColor color="white" ref_color="marker#backgroundColor" />
    <title>
      <Set title="Example 4 of pie plot" ref_title="marker#title"/>
      <SetFont family="FF_ARIAL" style="FS_BOLD" />
    </title>
    <Add ref="PiePlot#1" />
    <Stroke ref="file#1" />
  </PieGraph>

</jpgraph>
```

4 Using the template

4.1 Markers and data

Once the template design is completed, it can be displayed in the front end. A new content has to be inserted in the page and the SAV JpGraph plugin selected.

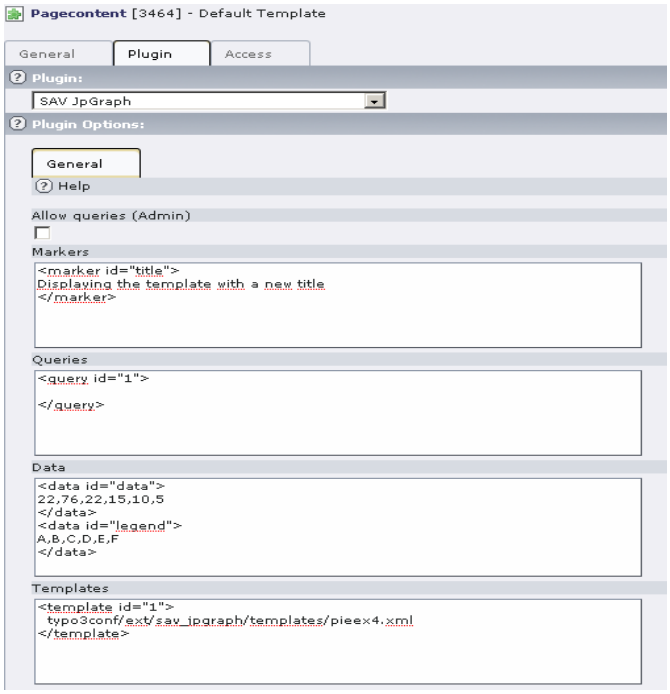


The extension comes with a flexform which has predefined XML elements. Filling the “Templates” part of the flexform with the template path is the only information required to display it. In our example, the template path is “typo3conf/ext/sav_jpgraph/templates/pieex4.xml”.

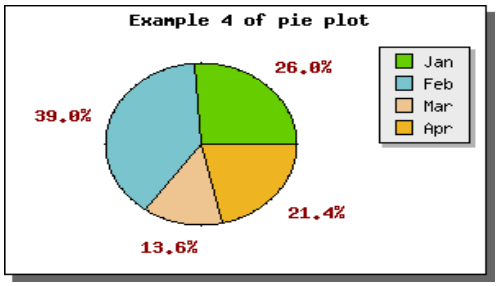
In the following figure, representing the back end flexform, the title of the graph is modified using the marker “title”, the data and the legend are modified using the XML <data> tags with the “id” attribute respectively sets to “data” and “legend”. As explained in section 3.7, it will overload respectively the

attributes “defaultData” and “defaultLegend”.

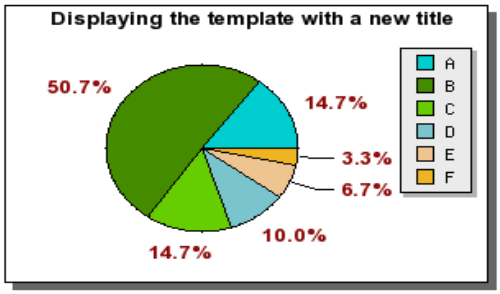
Let us note that the documentation of the SAV JpGraph includes a reference section which details all options associated with XML tags.



The page can now be displayed in the front end which leads to the following output.



Original template



Result using the marker and data tags

4.2 Queries

Queries are part of the SAV JpGraph templating concept. They are used to generated data.

Let us illustrate the use of queries in an example. Assume that we want to display the percentage of each valid type content for a web site developed with TYPO3. Two queries are necessary.

The first one will select in the table tt_content the number of records for a given Ctype.

```
SELECT COUNT(*) AS total FROM tt_content WHERE NOT deleted AND not hidden GROUP BY CType ORDER BY Ctype
```

The second one will get all distinct Ctype in tt_content in order to build the legends.

```
SELECT DISTINCT CType FROM tt_content WHERE NOT deleted AND not hidden ORDER BY CType
```

The queries are converted into XML tags. Each clause of the query is transformed into a tag.

It leads to the following XML code which has to be put in the queries section of the flexform:

```
<query id="1">
  <select>COUNT(*) AS total</select>
  <from>tt_content</from>
  <where>NOT deleted AND NOT hidden</where>
  <groupby>CType</groupby>
  <orderby>CType</orderby>
</query>

<query id="2">
  <select>DISTINCT CType</select>
  <from>tt_content</from>
  <where>NOT deleted AND NOT hidden</where>
  <orderby>CType</orderby>
</query>
```

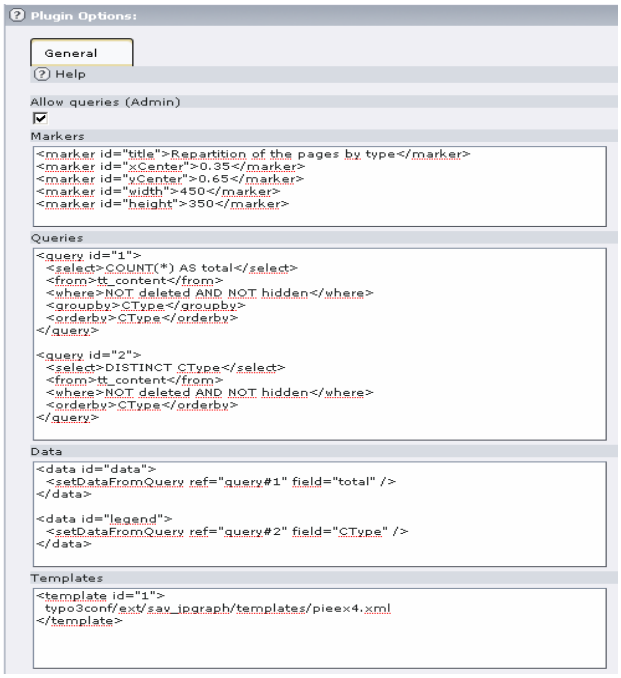
Finally, data must refer to the queries which is realized using “setDataFromQuery”. It extracts a field, given by the “field” attribute, from a query known by the reference provided in the “ref” attribute. It gives the following XML syntax which has to be put in the data section of the flexform:

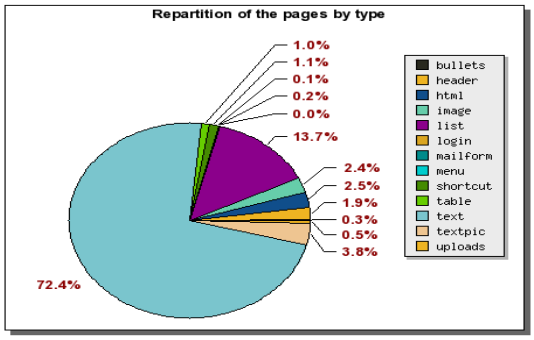
```
<data id="data">
  <setDataFromQuery ref="query#1" field="total" />
</data>

<data id="legend">
  <setDataFromQuery ref="query#2" field="CType" />
</data>
```

Let us note that an admin has to authorize the execution of queries by checking the “Allow queries” checkbox.

Let us also note that when queries are used, the plugin will run as a USER_INT while it runs as a USER in the other cases which means that caching is enabled.

The following figure represents the back end configuration and the resulting graph.

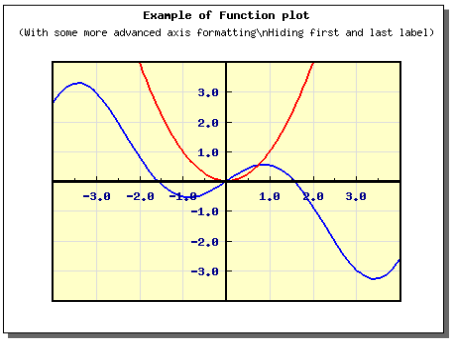




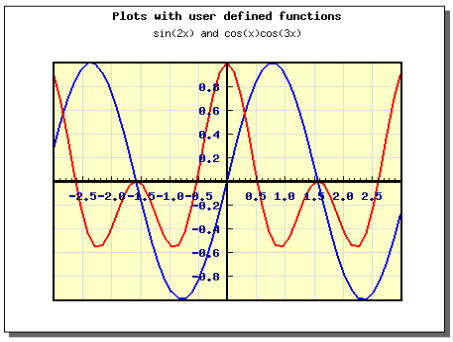
5 Other examples

The SAV JpGraph extension includes some XML template examples. The following pictures illustrates their use.

5.1 Function template (funcex1.xml)



Original template

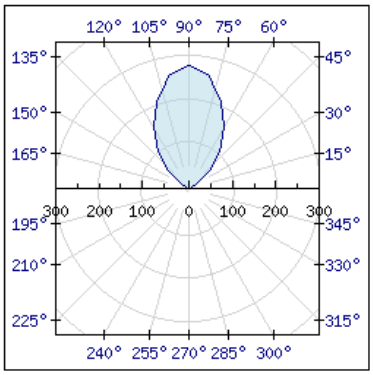


Modification of the markers

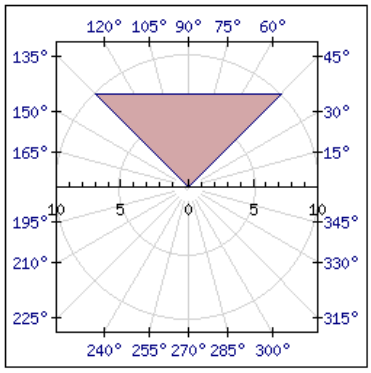
5.2 Led template (ledex1.xml)

Original template and modification of the markers for the text and the radius of the leds

5.3 Polar template (polarex0.xml)



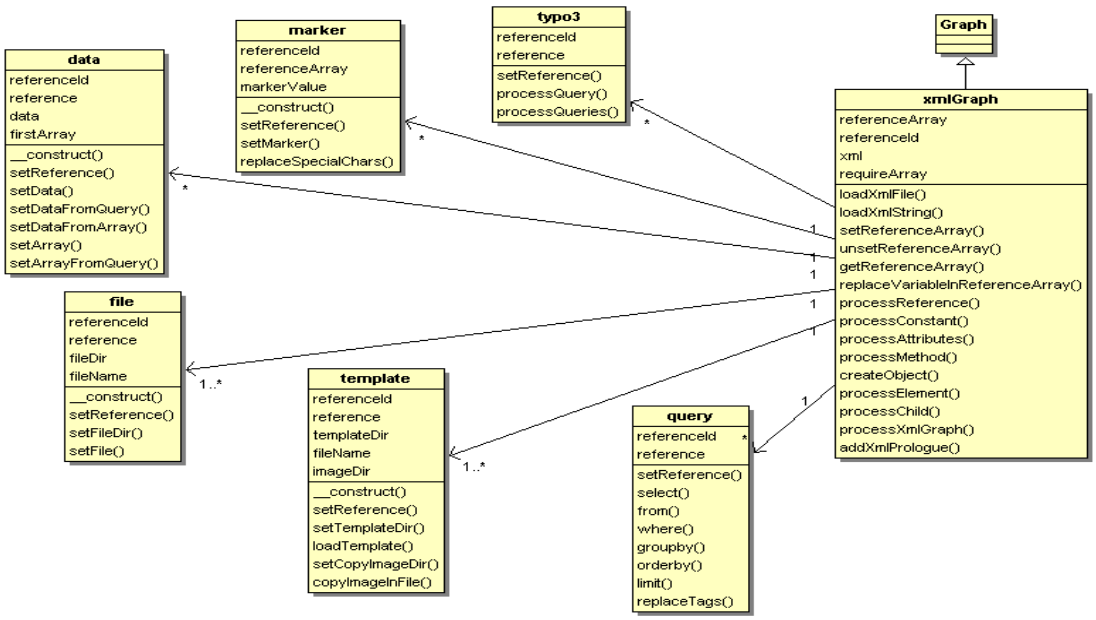
Original template



Modification of the data and color

6 Implementation aspects

The SAV JpGraph extension relies on a set of classes which have been grouped in the file “class.xmlgraph.php”. These classes are independent of the TYPO3 API and therefore give a quite general code. The file “class.typo3.php” contains a class to execute specific actions like query processing using the TYPO3 API. The UML class diagram is given in the following figure. The “graph” class is inherited from the JpGraph Library.



The XML template is processed by means of a small interpreter based on the simpleXML extension of the php language. Basically, the template is processed by the method “processXmlGraph” of the class “xmlGraph”. This method analyzes each child of the XML template by the recursive method “processChild”.

7 Conclusion

The SAV JpGraph extension provides an easy means to display graphs built with the JpGraph Library. A templating facility was introduced which makes it possible to develop parametrized graph by means of markers. The transformation from php to XML relies on simple rules. The execution time is slightly increased due to XML interpreter, especially when queries are used since, in that case, the plugin runs as a USER_INT.

Finally, let us note that the SAV JpGraph templating has been introduced in the SAV Library Extension Generator (sav_library).