RML Example 17: Graphics

RML (Report Markup Language) is ReportLab's own language for specifying the appearance of a printed page, which is converted into PDF by the utility rml2pdf.

These RML samples showcase techniques and features for generating various types of ouput and are distributed within our commercial package as test cases. Each should be self explanatory and stand alone.

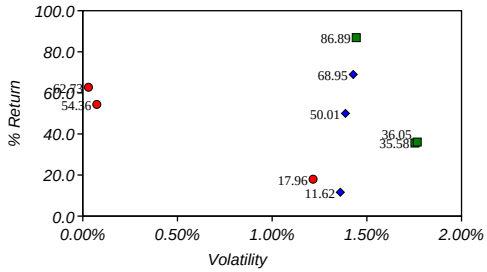
Diagra Integration

Diagra is ReportLab's charting and data graphics product. It allows charts and data graphics to be prepared visually in a Drawing Editor, and used in a variety of contexts including within RML, as bitmaps on the web, and for generating batches of EPS files.

Referring to a Drawing Module

The first stage is to use the drawing editor to create a drawing module. Take note of the class name as you generate it. You can then refer to it directly with a drawing tag. The drawing tag takes at least two parameters. The `module`attribute holds the name of the module the drawing is defined in. This is a normal python module reference as used with the import statement, and may contain dots to refer to items anywhere on the path e.g. `reportlab.graphics.charts.barcharts`. The directory where the RML document lives will always be on the path, so if your graphic is in the same directory, you can just use the filename (minus extension). The second attribute, `function`, is the name of the constructor to call. If you used the Drawing Editor, insert the class name of the drawing here. You could also call an arbitrary Python function which returns a Python object; often it is convenient to write helper functions to set up your drawings outside of RML. Finally, there is a third optional argument baseDir, which contains a directory name to look underThis must be escaped with double slashes on Windows e.g. C:\\mycharts. We included a chart module scatterplot.py in this test directory containing a class ScatterPlotDrawing, so for our example we will just refer so let's refer to it now:

```
<drawing module="test_014_scatterplot" function="ScatterPlotDrawing" />
```



It's important to recognize that the Diagra framework is completely general and not necessarily for charts. We use it to crank out ReportLab logos with variable sizes and colors! So, let's show one more example, which is about the simplest data graphic we have: a 'slidebox' which accepts one numeric parameter:

```
<drawing module="test_014_slidebox" function="SlideBoxDrawing" />
```

Source: ReportLab

Making it dynamic

Static charts are not much use to anyone. In most cases, you will want to pass in the numeric data at runtime, and perhaps change the title. The Diagra framework is completely general - not a charting framework per se - and lets you set any attribute of any object within the chart; so you could set the height of a bar or something similar. Let's start with an ultra-simple example. The above SlideBox takes a single numeric parameter. In general you should use the drawing editor to explore the available parameters.

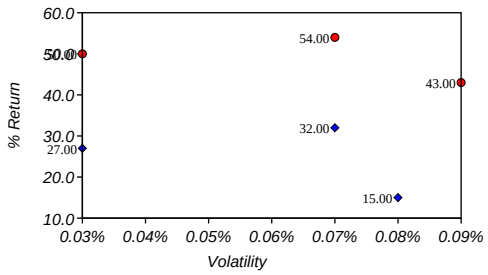
```
<drawing module="test_014_slidebox" function="SlideBoxDrawing" >
```

```
    <param name="SlideBox.trianglePosition">4</param>
</drawing>
```

Source: ReportLab

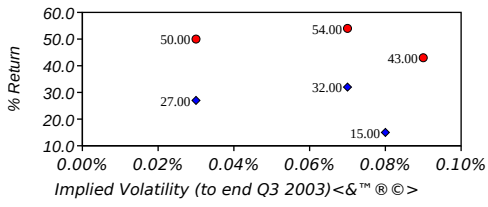
The most common use of the param tag will be to set the dynamic data for a chart. For example, let's take the preceding one and pass in some data. The Drawing Editor will reveal that the data parameter for scatter plot is a list of sequences of x-y pairs, so we make up our data nested this way (you can use square or round brackets, it doesn't matter):

```
<drawing module="test_014_scatterplot" function="ScatterPlotDrawing">
  <param name="ScatterPlot.data">[((0.03, 50), (0.04, 54), (0.05, 43)),
                                  ((0.03, 27), (0.04, 32), (0.055, 15))]</param>
  </drawing>
```



The content of each param tag is evaluated literally. This is more compact and easier to generate and parse than generating thousands of series and data-point tags.
You can also modify other parameters which are nothing to do with numeric data, just as you do in the Drawing Editor. Let's force the x axis to include the zero, so the leftmost points do not dangle in the margin, and change the title below the y axis:
Parameters passed through to charts may now contain the standard XML escapes for '&','<' and '>'. However, unicode font handling for charts and graphics is not yet complete; non-ASCII characters such as copyright and trademark which are passed in through RML may be displayed as multiple bytes of garbage when displayed in a Type 1 Font. We believe that the graphics are the last remaining area of our framework that needs unicode-enabling and hope to complete this in a release next week. The PDF rendering for the standard numeric escapes eg ࡊ is carried out, but will only work in param tags if the relevant objects font understands them. This should be the case for TTF fonts if the document encoding is "utf-8". See the example below where you should see <&™ ® ©> in the x axis label.

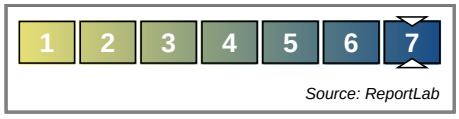
```
<drawing module="test_014_scatterplot" function="ScatterPlotDrawing">
  <param name="ScatterPlot.height">50</param>
  <param name="ScatterPlot.xValueAxis.labels.fontName">VeraItalic</param>
  <param name="ScatterPlot.xLabel">Implied Volatility (to end Q3 2003)&lt;&amp;&#x2122;&#174;&#169;&gt;
  <param name="ScatterPlot.xValueAxis.forceZero">1</param>
  <param name="ScatterPlot.data">
    [((0.03, 50), (0.07, 54), (0.09, 43)), ((0.03, 27), (0.07, 32), (0.08, 15))]
  </param>
</drawing>
```



Alignment and Boundaries

It is convenient to be able to align the drawings, and sometimes you want to see where they are on the page. The hAlign attribute takes values of LEFT, RIGHT, Center and (for the sake of the Brits who maintain the software) CENTRE. The default is CENTER. Here is a right aligned drawing with a boundary:

```
<drawing module="test_014_slidebox" function="SlideBoxDrawing"
        hAlign="RIGHT" showBoundary="1"/>
```



If you want fine-grained control of the border, we suggest to implement suitable rectangles within your Drawing.

Adding widgets and dynamic graphic creation

There is a widget tag which permitted on-the-fly creation of drawings and adding things to groups.

What follows is advanced and will probably be easiest understood if you (a) have Python skills, or (b) look at one of the source files in the drawin editor.

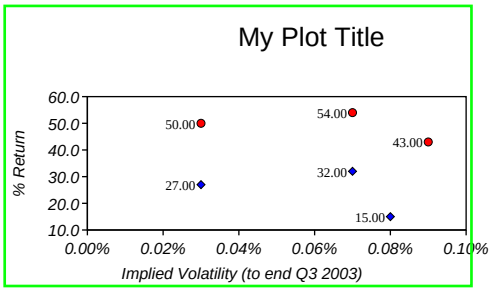
You can add shapes and widgets to drawings at runtime. In fact, you can even start with a bare-bones drawing and build it up from nothing. In general it is MUCH easier to do this in the Drawing Editor and just change the numeric data at runtime.

In the example below, we have taken our Scatter Plot and added a title at the top. In addition, to make it a bit easier to see where the boundaries are, I have added a thin border around the edge of the drawing using a rectangle widget. The param tag is used to set the added widgets.

```
<drawing module="test_014_scatterplot" function="ScatterPlotDrawing">
  <param name="ScatterPlot.height">50</param>
  <param name="ScatterPlot.xLabel">Implied Volatility (to end Q3 2003)</param>
  <param name="ScatterPlot.xValueAxis.forceZero">1</param>
  <param name="ScatterPlot.data">[((0.03, 50), (0.07, 54), (0.09, 43)),
           ((0.03, 27), (0.07, 32), (0.08, 15))]</param>

  <widget module="reportlab.graphics.shapes" function="String" name="title"
        initargs="87.5,90,'My Plot Title'"/>
  <param name="title.fontName">Helvetica</param>

  <widget module="reportlab.graphics.shapes" function="Rect" name="border"
        initargs="0,0,175,105"/>
  <param name="border.fillColor">None</param>
  <param name="border.strokeColor">green1</param>
</drawing>
```



This last one illustrates a "feature": it is possible for drawings to draw out of the box, as the data point does on the left. Obviously, if you ask Diagra to make a bitmap file on disk, this is cut off; but in RML documents it leaks out. You should design your drawings to leave adequate space for the longest expected data labels.

Drawings in graphics mode

The **<drawingGraphic>** is the equivalent of the **<drawing>** tag that can be used in graphics mode. It has extra attributes **x**, **y** and **anchor**.