OpenHtmlToPdf Feature Documentation

This documentation tries to show the advanced features of OpenHtmlToPdf. To generate the documentation a FreeMarker template is used to finally generate the HTML to render with OpenHtmlToPdf. FreeMarker allows to do conditionals and calculation within the template. This is not pure MVC but very useful to generate a report HTML. In combination with the excellent FreeMarker support in the (expensive) IntelliJ IDEA IDE this is a very productive environment to build reports. Please lookup the source of this document if you want to know some tricks not explicit mentioned in this documentation.

Pagebreak Tuning

In a perfect world `style="page-break-inside: avoid"` would just work and all reports would look beautiful. OpenHtmlToPdf tries its best to avoid a page break inside. But this is not always possible and also rather complex. If you know or can calculate how much space a block will take you can use the special `-fs-page-break-min-height` CSS property on the block. You can specify the minimum height needed on the page any not relativ CSS unit (e.g. in cm but not in %). If that amount of space is not remaining on the page, a pagebreak happens before the block is drawn. Example:

```
<div style="-fs-page-break-min-height:5cm">
        My nice content, which should not break... But should
        also not be higher than 5cm.
</div>
```

MathML & LaTeX

To display math you can use MathML and latex. To do so you need the dependencies:

```
<dependency>
        <groupId>com.openhtmltopdf</groupId>
        <artifactId>openhtmltopdf-mathml-support</artifactId>
        <version>...</version>
</dependency>
<dependency>
        <groupId>com.openhtmltopdf</groupId>
        <artifactId>openhtmltopdf-latex-support</artifactId>
        <version>...</version>
</dependency>
```

If you don't use the LaTeX feature you don't need to include the openhtmltopdf-latex-support. You must activate the support in the Builder to use it:

```
builder.useMathMLDrawer(new MathMLDrawer());
builder.addDOMMutator(LaTeXDOMMutator.INSTANCE);
```

The LaTeX support translates a LaTeX fragment using SnuggleTeX to HTML+MathML, which is then rendered using the MathML support.

This is a small inline formular: $a^2 + b^2 = c^2$. You can use many LaTeX features and environments. The exact amount of supported features is depending on StruggleTex and JEuclid which are the backing libraries for the LaTeX and MathML support.

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\prod_{i=1}^{n} x = x^n$$

```
<latex>
        This is a small inline formular: $$a^2 + b^2 = c^2$$. You can use many LaTe
        features and environments. The exact amount of supported features is depend
        StruggleTex and JEuclid which are the backing libraries for the LaTeX and M

        $$\sum\limits_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

        $\prod\limits_{i=1}^n x = x^n$
</latex>
```

Here is some pure MathML: $a \times (b+c)$
If you are writing the document by hand it may be just simpler to use LaTeX: $ax(b+c)$

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
        <mrow>
                <mi>a</mi>
                <mo>x</mo>
                <mfenced open="(" close=")">
                        <mrow>
                                <mi>b</mi>
                                <mo>+</mo>
                                <mi>c</mi>
                        </mrow>
                </mfenced>
        </mrow>
</math>

<br/>
If you are writing the document by hand it may be just simpler to use LaTeX:
<latex>$$a x (b+c)$$</latex>
```

Objects

OpenHtmlToPdf comes with some builtin objects, which you can use to quickly create diagrams, add background PDF images and so on. To use them include the openhtmltopdf-objects dependency in your pom:

```
<dependency>
        <groupId>com.openhtmltopdf</groupId>
        <artifactId>openhtmltopdf-objects</artifactId>
        <version>...</version>
</dependency>
```

Merge Background PDF

You can add a watermark / background to your document. To do so you should place

```
<object type="pdf/background" pdfsrc="background.pdf" pdfpage="1" style="width:1px;
```

into the header or footer of the document. The document will be placed unscaled in the PDF origin, i.e. in the left lower corner.

- **pdfsrc**: URI of the PDFFile to use
- **pdfpage**: Page to import from the PDF file.

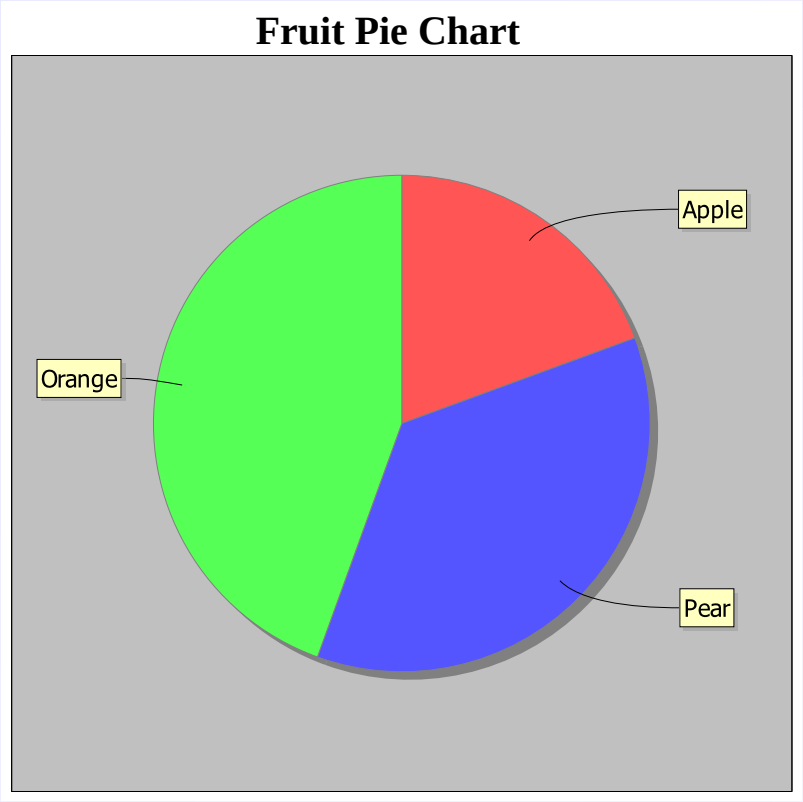
JFreeGraph

For simple charts you can use the builtin objects for JFreeGraph. Note: You must specify the dependency to JFreeMarker in your POM, because it is declared as a optional dependency on openhtmltopdf-objects.

```
<dependency>
        <groupId>org.jfree</groupId>
        <artifactId>jfreechart</artifactId>
        <version>1.5.0</version>
</dependency>
```

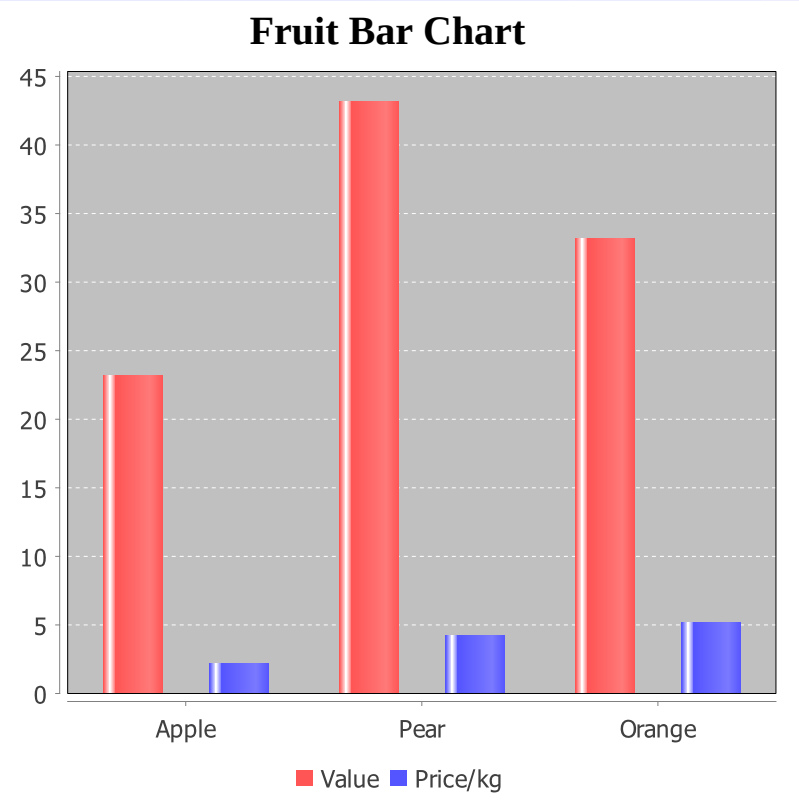
If you specify a URL for a data point then the segment in the diagram used for that datapoint is a link to that URL. Note: This only works in Acrobat Reader, all other PDF Viewer ignore this feature.

The Pie Diagram




```
<object type="jfreechart/pie"
                style="width:400px;height:400px;-fs-page-break-min-height:400px"
                title="Fruit Pie Chart">
        <data name="Apple" value="23.2" url="https://www.google.de?q=apple-pie"/>
        <data name="Pear" value="43.2" url="https://www.google.de?q=pear-pie"/>
        <data name="Orange" value="53.2" url="#start"/>
</object>
```

The Bar Diagram




```
<object type="jfreechart/bar"
                style="width:400px;height:400px; -fs-page-break-min-height:400px"
                title="Fruit Bar Chart"
                categories-title="Category" series-title="Series">
        <data series="Value" category="Apple" value="23.2" url="#apple"/>
        <data series="Value" category="Pear" value="43.2" url="#pear"/>
        <data series="Value" category="Orange" value="33.2" url="#orange"/>
        <data series="Price/kg" category="Apple" value="2.2"/>
        <data series="Price/kg" category="Pear" value="4.2"/>
        <data series="Price/kg" category="Orange" value="5.2"/>
</object>
```

Table of Content