Output

Notes about Output Formats: Renderers

by Keiron Liddle, Art Welch

1 Output Formats

FOP supports a number of different output formats. This is achieved by using different renderers that create the output.

Here we will explain some information for uses to be able to understand what the renderers are doing and what difference there may be between different renderers.

1.1 Common Information

Each renderer is given an area tree to render to its output format. The area tree is simply a representation of the pages and the placement of text and graphical objects on those pages.

The renderer will be given each page as it is ready and an output stream to write the data out. The renderer is responsible for managing the output format and associated data and flow.

Fonts and Layout - some formats (eg. PDF and AWT) rely on different font information. The fonts for these outputs have different sizes for the same point size. This means that the layout can be quite different for the same fo document.

DPI - This is an important issue when creating output for printing. The dpi is used to convert measurements into points. For example 1in = 2.54cm = 72 points. It is also used when determining the size of images and the rendering of certain graphics in the output. Currently FOP uses a value of 72dpi.

You may want to send your output directly to a printer. The Print renderer uses the java api to print the document or you might be able to send the output stream directly to a printer. If your printer supports postscript you could send the postscript to the printer. If you have a printer that supports PCL you could stream the PCL document to your printer. On Windows:

```
fop ... -ps \\computername\printer or fop ... -pcl \\computername\printer
```

On UNIX:

```
proc = Runtime.getRuntime().exec("lp -d" + print_queue + " -o -dp -");
out = proc.getOutputStream();
```

And give the OutputStream (out) to the PCLRenderer and it happily sends the PCL to the AIX print queue.

1.2 PDF

PDF is the best supported output format. It is also the most accurate with text and layout. This creates a PDF document that is streamed out as each page is rendered. This means that the internal page index information is stored near the end of the document. The PDF version supported is 1.3 which is currently the most popular version for Acrobat Reader (4.0), PDF versions are forwards/backwards compatible.

1.3 PCL

This format is for the Hewlett-Packard PCL printers. It should produce output as close to identical as possible to the printed output of the PDFRenderer within the limitations of the renderer, and output device.

The output created by the PCLRenderer is generic PCL 5 as documented in the "HP PCL 5 Printer Language Technical Reference Manual" (copyright 1990). This should allow any device fully supporting PCL 5 to be able to print the output generated by the PCLRenderer.

1.3.1 Limitations

- Text or graphics outside the left or top of the printable area are not rendered properly. In general things that should print to the left of the printable area are shifted to the right so that they start at the left edge of the printable area and an error message is generated.
- The Helvetica and Times fonts are not well supported among PCL printers so Helvetica is mapped to Arial and Times is mapped to Times New. This is done in the PCLRenderer, no changes are required in the FO's. The metrics and appearance for Helvetica/Arial and Times/Times New are nearly identical, so this has not been a problem so far.
- Only the original fonts built into FOP are supported.
- For the non-symbol fonts, the ISO 8859/1 symbol set is used (PCL set "0N").
- Multibyte characters are not supported.
- SVG support is limited. Currently only lines, rectangles (may be rounded), circles, ellipses, text, simple paths, and images are supported. Colors are supported (dithered black and white) but not gradients.
- Images print black and white only (not dithered). When the renderer prints a color image it uses a threshold value, colors above the threshold are printed as white and below are black. If you need to print a non-monochrome image you should dither it first.
- Image scaling is accomplished by modifying the effective resolution of the image data. The available resolutions are 75, 100, 150, 300, and 600 DPI.
- Color printing is not supported. Colors are rendered by mapping the color intensity to one

of the PCL fill shades (from white to black in 9 steps).
- SVG clipping is not supported.

1.3.2 Additional Features

There are some special features that are controlled by some public variables on the PCLRenderer class.

orientation
The logical page orientation is controlled by the public orientation variable. Legal values are:

curdiv, paperheight
The curdiv and paperheight variables allow multiple virtual pages to be printed on a piece of paper. This allows a standard laser printer to use perforated paper where every perforation will represent an individual page. The paperheight sets the height of a piece of paper in decipoints. This will be divided by the page.getHeight() to determine the number of equal sized divisions (pages) that will fit on the paper. The curdiv variable may be read/written to get/set the current division on the page (to set the starting division and read the ending division for multiple invocations).

topmargin, leftmargin
The topmargin and leftmargin may be used to increase the top and left margins for printing.

1.4 PostScript

The PostScript renderer is still in its early stages and therefore still missing some features. It provides good support for most text and layout. Images and SVG are not fully supported, yet. Currently, the PostScript renderer generates PostScript Level 3 with most DSC comments. Actually, the only Level 3 feature used is FlateDecode, everthing else is Level 2.

1.4.1 Limitations

- Images and SVG may not be display correctly. SVG support is far from being complete. No image transparency is available.
- Character spacing may be wrong.
- No font embedding is supported.
- Multibyte characters are not supported.
- PPD support is still missing.
- The renderer is not yet configurable.

1.5 RTF

This is currently not integrated with FOP but it will soon. This will create an rtf (rich text format) document that will attempt to contain as much information from the fo document as possible.

1.6 SVG

This format creates an SVG document that has links between the pages. This is primarily for slides and creating svg images of pages. Large documents will create SVG files that are far too large for and SVG viewer to handle. Since fo documents usually have text the SVG document will have a large number of text elements. The font information for the text is obtained from the jvm in the same way as the AWT viewer, if the svg is view where the fonts are different, such as another platform, then the page will appear wrong.

1.7 XML

This is for testing and verification. The XML created is simply a representation of the internal area tree put into XML. It does not perform any other purpose.

1.8 Print

It is possible to directly print the document from the command line. This is done with the same code that renders to the AWT renderer.

1.9 AWT

The AWT viewer shows a window with the pages displayed inside a java graphic. It displays one page at a time. The fonts used for the formatting and viewing depend on the fonts available to your JRE.

1.10 MIF

This format is the Maker Interchange Format which is used by Adobe Framemaker. This is currently not fully implemented.

1.11 TXT

Text as you could imagine does not work very well. It is an output format that you should expect bad results. The main purpose of this is to get a quick and dirty view of the document and the text inside it.

The TXTRenderer is a FOP renderer that produces plain ASCII text output that attempts to match the output of the PDFRenderer as closely as possible. This was originally developed to accommodate an archive system that could only accept plain text files. Of course when limited to plain fixed pitch text the output does not always look very good.

Output

The TXTRenderer works with a fixed size page buffer. The size of this buffer is controlled with the textCPI and textLPI public variables. The textCPI is the effective horizontal characters per inch to use. The textLPI is the vertical lines per inch to use. From these values and the page width and height the size of the buffer is calculated. The formatting objects to be rendered are then mapped to this grid. Graphic elements (lines, borders, etc) are assigned a lower priority than text, so text will overwrite any graphic element representations.