FOP Output Options

$Revision: 348211 $

by Keiron Liddle, Art Welch

Table of contents

FOP supports multiple output formats by using a different renderer for each format. The renderers do not all have the same set of capabilities, sometimes because of the output format itself, sometimes because some renderers get more development attention than others.

1. General Information

1.1. Fonts

Most FOP renderers use a FOP-specific system for font registration. However, the AWT and print renderers use the java awt package, which gets its font information from the operating system registration. This can result in several differences, including actually using different fonts, and having different font metrics for the same font. The net effect is that the layout of a given FO document can be quite different between renderers that do not use the same font information.

1.2. Output to a Printer or Other Device

The most obvious way to print your document is to use the FOP print renderer, which uses the Java API (AWT). However, you can also send output from the Postscript renderer directly to a Postscript device, or output from the PCL renderer directly to a PCL device.

Here are Windows command-line examples for Postscript and PCL:

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

Here is some Java code to accomplish the task in UNIX:

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

Set the OutputStream (out) to the PCLRenderer and it happily sends the PCL to the UNIX printer queue.

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.

Note that FOP does not currently support "tagged pdf".

2.1. Fonts

PDF has a set of fonts that are always available to all PDF viewers, to quote from the PDF Specification: *"PDF prescribes a set of 14 standard fonts that can be used without prior definition. These include four faces each of three Latin text typefaces (Courier, Helvetica, and Times), as well as two symbolic fonts (Symbol and ITC Zapf Dingbats). These fonts, or suitable substitute fonts with the same metrics, are guaranteed to be available in all PDF viewer applications."*

2.2. Post-processing

FOP does not currently support several desirable PDF features: document properties (title, author, etc.), and watermarks. One workaround is to use Adobe Acrobat (the full version, not the Reader) to process the file manually or with scripting that it supports.

Another popular post-processing tool is iText, which has tools for adding security features, document properties, watermarks, and many other features to PDF files.

Warning:

Caveat: iText may swallow PDF bookmarks. But Jens Stavnstrup tells us that this doesn't happen if you use iText's PDFStamper.

Here is some sample code that uses iText to encrypt a FOP-generated PDF. (Note that FOP now supports PDF encryption. However the principles for using iText for other PDF features are similar.)

```
public static void main(String args[]) {
  try {
    ByteArrayOutputStream fopout=new ByteArrayOutputStream();
    FileOutputStream outfile=new FileOutputStream(args[2]);
    Driver driver =new Driver();
    driver.setOutputStream(fopout);
    driver.setRenderer(Driver.RENDER_PDF);
    Transformer transformer=TransformerFactory
      .newInstance().newTransformer(new StreamSource(new File(args[1])));
    transformer.transform(new StreamSource(new File(args[0])),
       new SAXResult(driver.getContentHandler()));
    PdfReader reader = new PdfReader(fopout.toByteArray());
    int n = reader.getNumberOfPages();
    Document document = new Document(reader.getPageSizeWithRotation(1));
    PdfWriter writer = PdfWriter.getInstance(document, outfile);
    writer.setEncryption(PdfWriter.STRENGTH40BITS, "pdf", null,
      PdfWriter.AllowCopy);
    document.open();
```

```
    PdfContentByte cb = writer.getDirectContent();
    PdfImportedPage page;
    int rotation;
    int i = 0;
    while (i < n) {
      i++;
      document.setPageSize(reader.getPageSizeWithRotation(i));
      document.newPage();
      page = writer.getImportedPage(reader, i);
      rotation = reader.getPageRotation(i);
      if (rotation == 90 || rotation == 270) {
        cb.addTemplate(page, 0, -1f, 1f, 0, 0,
         reader.getPageSizeWithRotation(i).height());  }
      else {
        cb.addTemplate(page, 1f, 0, 0, 1f, 0, 0);
      }
      System.out.println("Processed page " + i);
    }
    document.close();
  }
  catch( Exception e) {
    e.printStackTrace();
  }
}
```

Check the iText tutorial and documentation for setting access flags, password, encryption strength and other parameters.

2.3. Watermarks

In addition to the PDF Post-processing options, consider the following workarounds:

- Use a background image for the body region.
- (submitted by Trevor_Campbell@kaz.com.au) Place an image in a region that overlaps the flowing text. For example, make region-before large enough to contain your image. Then include a block (if necessary, use an absolutely positioned block-container) containing the watermark image in the static-content for the region-before. Note that the image will be drawn on top of the normal content.

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.

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 is not supported.
- 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).

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.

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.

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.

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.

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.

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.

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.

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.

10. MIF

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

11. TXT

The text renderer 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, and is primarily useful for getting a quick-and-dirty view of the document text. The renderer is very limited, so do not be surprised if it gives unsatisfactory results.

The Text renderer 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.

Because FOP lays the text onto a grid during layout, there are frequently extra or missing spaces between characters and lines, which is generally unsatisfactory. Users have reported that the optimal settings to avoid such spacing problems are:

- font-family="Courier"

- font-size="7.3pt"
- line-height="10.5pt"