Tools for Adobe PostScript

Table of contents

1 Overview 2

2 The PostScript generator 2

3 Java2D: Graphics2D implementation for generating PostScript and EPS 2

3.1 Creating an EPS file 2

4 DSC parser/processor 3

1 Overview

Apache™ XML Graphics Commons contains various tools for writing and processing Adobe PostScript files. This includes:

- A PostScript generator class which helps writing PostScript files from scratch.
- Two Graphics2D implementations, one for plain PostScript and one for writing Encapsulated PostScript (EPS).
- A DSC-parser/processor: Parse, post-process and change DSC-compliant PostScript files.

> **Note:**
> We don't currently include a PostScript interpreter though we would love to have one. A Java-based PostScript interpreter to keep an eye on is the one from the FOray project.

2 The PostScript generator

The "PSGenerator" class can help writing PostScript files. It deals with things like escaping, saving/tracking/restoring graphics state, writing DSC comments and tracking of DSC resources.

You will rarely interact with the PS generator itself, as it is probably more interesting to generate a PostScript file using Java2D which is described in the following section.

3 Java2D: Graphics2D implementation for generating PostScript and EPS

We provide two classes (PSDocumentGraphics2D and EPSDocumentGraphics2D) which you can use to generated complete PostScript files using normal Java2D means. The difference between the two classes is that the EPS variant creates a fully compliant Encapsulated PostScript file while the PS variant simply creates a normal DSC-compliant level 2 PostScript file. It depends on your requirement which variant you choose. The PS variant is mostly for printing purposes while the EPS variant is better suited for inclusion in other documents.

3.1 Creating an EPS file

Creating an EPS file using the Graphics2D implementation is easy. Instantiate EPSDocumentGraphics2D, set a GraphicContext and set up the output document. Here's an example:

```
import org.apache.xmlgraphics.java2d.ps.EPSDocumentGraphics2D;

[..]

EPSDocumentGraphics2D g2d = new EPSDocumentGraphics2D(false);
g2d.setGraphicContext(new org.apache.xmlgraphics.java2d.GraphicContext());

//Set up the document size
g2d.setupDocument(out, 400, 200); //400pt x 200pt
//out is the OutputStream to write the EPS to

g2d.drawRect(10, 10, 50, 50); //paint a rectangle using normal Java2D calls

g2d.finish(); //Wrap up and finalize the EPS file
```

A complete example for generating an EPS files can be found in the "examples" directory in the distribution.

4 DSC parser/processor

Many PostScript files use special comments to structure a document. This allows manipulation of PostScript files without interpreting them. These special comments are defined in the Document Structuring Conventions. The code in Commons is designed to work with DSC 3.0. For details on how DSC is used, please take a look at the DSC specification.

The DSC support in Commons was primarily developed to implement resource optimization features in Apache FOP's PostScript output support. Resources like images which are used repeatedly in a document should not be written to the PostScript file each time it is used. Instead it is written once at the beginning of the file as a PostScript form. The form is then called whenever the image needs painting.

But the DSC parser could potentially be used for other purposes. The most obvious is extracting a subset of pages from a DSC-compliant file. Assume you want to print only page 45 to 57 of a particular document. There's an example that demonstrates exactly this. Check out the "examples" directory in the distribution. Other potential use cases for the DSC parser are:

- Patching PostScript files, for example, adding OMR marks for automatic packaging
- Imposition (2-up, n-up, rotation, etc.)
- EPS graphic extraction
- Inspecting the page count
- etc. etc.

The DSC parser (DSCParser) was designed as a pull parser, i.e. you fetch new events from the parser inspecting them and acting on them as they are found. If you prefer to work with a push parser, you can pass the DSCParser a DSCHandler implementation and the parser will send you all the events.

The best example to understand how to use the DSC parser is the PageExtractor class that implements the page extraction functionality mentioned above.

> **Note:**
> The DSC parser is not considered feature-complete. The basic infrastructure is there but, for example, not all DSC comments are available as concrete Java classes. If you need to extend the DSC parser for your own use cases, please send us your patches.