Embedding FOP

Notes about embedding FOP in your Java application

1 Embedding FOP

1.1 Overview

Instantiate org.apache.fop.apps.Driver. Once this class is instantiated, methods are called to set the Renderer to use and the OutputStream to use to output the results of the rendering (where applicable). In the case of the Renderer and ElementMapping(s), the Driver may be supplied either with the object itself, or the name of the class, in which case Driver will instantiate the class itself. The advantage of the latter is it enables runtime determination of Renderer and ElementMapping(s).

1.2 Examples

The simplest way to use Driver is to instantiate it with the InputSource and OutputStream, then set the renderer desired and call the run method.

Here is an example use of Driver which outputs PDF:

```
Driver driver = new Driver(new InputSource (args[0]),
                           new FileOutputStream(args[1]));
driver.setRenderer(Driver.RENDER_PDF);
driver.run();
```

You also need to set up logging. Global logging for all FOP processes is managed by MessageHandler. Per-instance logging is handled by Driver. You want to set both using an implementation of org.apache.avalon.framework.logger.Logger. See Jakarta Avalon Framework for more information.

```
Logger logger = new ConsoleLogger(ConsoleLogger.LEVEL_INFO);
MessageHandler.setScreenLogger(logger);
driver.setLogger(logger);
```

To setup the user config file you can do the following

```
userConfigFile = new File(userConfig);
options = new Options(userConfigFile);
```

Note:
This is all you need to do, it sets up a static configuration class.

Once the Driver is set up, the render method is called. Depending on whether DOM or SAX is being used, the invocation of the method is either render(Document) or render(Parser, InputSource) respectively.

Another possibility may be used to build the FO Tree. You can call getContentHandler() and fire the SAX events yourself.

Once the FO Tree is built, the format() and render() methods may be called in that order.

Here is an example use of Driver:

```
Driver driver = new Driver();
driver.setRenderer(Driver.RENDER_PDF);
driver.setInputSource(new FileInputSource(args[0]));
driver.setOutputStream(new FileOutputStream(args[1]));
driver.run();
```

You can also specify an xml and xsl file for the input.

Here is an example use of Driver with the XSLTInputHandler:

```
Driver driver = new Driver();
driver.setRenderer(Driver.RENDER_PDF);
InputHandler inputHandler = new XSLTInputHandler(xmlFile, xslFile);
XMLReader parser = inputHandler.getParser();
driver.setOutputStream(new FileOutputStream(outFile));
driver.render(parser, inputHandler.getInputSource());
```

Have a look at the classes CommandLineStarter or FopServlet for complete examples.

Note:
If your FO files contain SVG then batik will be used. When batik is initialised it uses certain classes in `java.awt` that intialises the java AWT classes. This means that a daemon thread is created by the jvm and on unix it will need to connect to a DISPLAY. The thread means that the java application will not automatically quit when finished, you will need to call `System.exit`. These issues should be fixed in the upcoming JDK1.4

1.3 Controlling logging

FOP uses Jakarta Avalon's Logger interface to do logging. See the Jakarta Avalon project for more information.

Per default FOP uses the ConsoleLogger which logs to System.out. If you want to do logging using a logging framework (such as LogKit, Log4J or JDK 1.4 Logging) you can set a

different Logger implementation on the Driver object. Here's an example how you would use LogKit:

```
    Hierarchy hierarchy = Hierarchy.getDefaultHierarchy();
    PatternFormatter formatter = new PatternFormatter(
       "[%{priority}]: %{message}\n%{throwable}" );

    LogTarget target = null;
    target = new StreamTarget(System.out, formatter);

    hierarchy.setDefaultLogTarget(target);
    log = hierarchy.getLoggerFor("fop");
    log.setPriority(Priority.INFO);

    driver.setLogger(new org.apache.avalon.framework.logger.LogKitLogger(log));
```

The LogKitLogger class implements the Logger interface so all logging calls are being redirected to LogKit. More information on Jakarta LogKit can be found here.

Similar implementations exist for Log4J (org.apache.avalon.framework.logger.Log4JLogger) and JDK 1.4 logging (org.apache.avalon.framework.logger.Jdk14Logger).

If you want FOP to be totally silent you can also set an org.apache.avalon.framework.logger.NullLogger instance.

If you want to use yet another logging facility you simply have to create a class that implements org.apache.avalon.framework.logging.Logger and set it on the Driver object. See the existing implementations in Avalon Framework for examples.

1.4 Hints

1.4.1 XML/XSL/DOM Inputs

You may want to supply you input to FOP from different data sources. For example you may have a DOM and XSL stylesheet or you may want to set variables in the stylesheet. The page here: http://xml.apache.org/xalan-j/usagepatterns.html describes how you can do these things.

You can use the content handler from the driver to create a SAXResult. The transformer then can fire SAX events on the content handler which will in turn create the rendered output.

1.4.2 Object reuse

If FOP is going to be used multiple times within your application it may be useful to reuse certain objects to save time.

The renderers and the driver can both be reused. A renderer is reusable once the previous render has been completed. The driver is reuseable after the rendering is complete and the

reset method is called. You will need to setup the driver again with a new OutputStream, IntputStream and renderer.

1.4.3 Getting information on the rendering process

To get the number of pages that were rendered by FOP you can call Driver.getResults(). This returns a FormattingResults object where you can lookup the number of pages produced. It also gives you the page-sequences that were produced along with their id attribute and their number of pages. This is particularly useful if you render multiple documents (each enclosed by a page-sequence) and have to know the number of pages of each document.

1.5 Using Fop in a servlet

In the directory xml-fop/docs/examples/embedding you can find a working example how to use Fop in a servlet. You can drop the fop.war into the webapps directory of Tomcat, then go to a URL like this:

http://localhost:8080/fop/fop?fo=/home/path/to/fofile.fo

http://localhost:8080/fop/fop?xml=/home/path/to/xmlfile.xml&xsl=/home/path/to/xslfile.xsl

You can also find the source code there in the file FopServlet.java

To compile this code you will need servlet_2_2.jar (or compatible), fop.jar and the sax api in your classpath.

Note:

Some browsers have problems handling the PDF result sent back to the browser. IE is particularly bad and different versions behave differently. Having a ".pdf" on the end of the url may help.