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 Basics

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:

```
import org.apache.fop.apps.Driver;

/*..*/

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

In the example above, args[0] contains the path to an XSL-FO file, while args[1] contains a path for the target PDF file.

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 below for more information.

```
import org.apache.avalon.framework.logger.Logger;
import org.apache.avalon.framework.logger.ConsoleLogger;
```

```
/*..*/

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

To setup the user config file you can do the following

```
import org.apache.fop.apps.Options;

/*..*/

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. Also, have a look at the examples at the bottom of this page.

Note:

If your XSL-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 JDK 1.4

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.

Examples showing this can be found at the bott

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/examples/servlet you can find a working example how to use FOP in a servlet. After building the 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

The source code for the servlet can be found under xml-fop/examples/servlet/src/FopServlet.java.

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.

2 Examples

The directory "xml-fop/examples/embedding" contains several working examples. In contrast of the examples above the examples here primarily use JAXP for XML access. This may be easier to understand for people familiar with JAXP.

2.1 ExampleFO2PDF.java

This example demonstrates the basic usage pattern to transform an XSL-FO file to PDF using FOP.

Example XSL-FO to PDF

2.2 ExampleXML2FO.java

This example has nothing to do with FOP. It is there to show you how an XML file can be converted to XSL-FO using XSLT. The JAXP API is used to do the transformation. Make sure you've got a JAXP-compliant XSLT processor in your classpath (ex. Xalan).

Example XML to XSL-FO

2.3 ExampleXML2PDF.java

This example demonstrates how you can convert an arbitrary XML file to PDF using XSLT and XSL-FO/FOP. It is a combination of the first two examples above. The example uses JAXP to transform the XML file to XSL-FO and FOP to transform the XSL-FO to PDF.

Example XML to PDF (via XSL-FO)

The output (XSL-FO) from the XSL transformation is piped through to FOP using SAX events. This is the most efficient way to do this because the intermediate result doesn't have to be saved somewhere. Often, novice users save the intermediate result in a file, a byte array or a DOM tree. We strongly discourage you to do this if it isn't absolutely necessary. The performance is significantly higher with SAX.

2.4 ExampleObj2XML.java

This example is a preparatory example for the next one. It's an example that shows how an arbitrary Java object can be converted to XML. It's an often needed task to do this. Often people create a DOM tree from a Java object and use that. This is pretty straightforward. The example here however shows how to do this using SAX which will probably be faster and not even more complicated once you know how this works.

Example Java object to XML

For this example we've created two classes: ProjectTeam and ProjectMember (found in

xml-fop/examples/embedding/java/embedding/model). They represent the same data structure found in xml-fop/examples/embedding/xml/xml/projectteam.xml. We want to serialize a project team with several members which exist as Java objects to XML. Therefore we created the two classes: ProjectTeamInputSource and ProjectTeamXMLReader (in the same place as ProjectTeam above).

The XMLReader implementation (regard it as a special kind of XML parser)is responsible for creating SAX events from the Java object. The InputSource class is only used to hold the ProjectTeam object to be used.

Have a look at the source of ExampleObj2XML.java to find out how this is used. For more detailed information see other resources on JAXP (ex. An older JAXP tutorial).

2.5 ExampleObj2PDF.java

The last example here combines the previous and the third to demonstrate how you can transform a Java object to a PDF directly in one smooth run by generating SAX events from the Java object that get fed to an XSL transformation. The result of the transformation is then converted to PDF using FOP as before.

Example Java object to PDF (via XML and XSL-FO)

2.6 Final notes

These examples should give you an idea of what's possible. It should be easy to adjust these examples to your needs. For examples, you can use a DOMSource instead of a StreamSource to feed a DOM tree as input for an XSL transformation.

If you think you have a decent example that should be here, contact us via one of the mailing lists and we'll see to it that it gets added. Also, if you can't find the solution to your particular problem drop us a message on the fop-user mailing list.