FOP: Embedding

How to Embed FOP in a Java application

$Revision: 490012 $

Table of contents

1 Overview....3
2 Basics....3
2.1 Logging....3
2.2 Logging (Upcoming FOP 1.0 Version only)....4
2.3 Processing XSL-FO....4
2.4 Processing XSL-FO generated from XML+XSLT....4
3 Controlling logging....5
4 Input Sources....6
5 Using a Configuration File....6
6 Setting the Configuration Programmatically....7
7 Hints....7
7.1 Object reuse....7
7.2 AWT issues....8
7.3 Getting information on the rendering process....8
8 Improving performance....8
9 Multithreading FOP....8
10 Examples....9
10.1 ExampleFO2PDF.java....9
10.2 ExampleXML2FO.java....9
10.3 ExampleXML2PDF.java....9
10.4 ExampleObj2XML.java....10
10.5 ExampleObj2PDF.java....10

10.6 ExampleDOM2PDF.java....11
10.7 ExampleSVG2PDF.java (PDF Transcoder example)....11
10.8 Final notes....11

1. Overview

Review Running FOP for important information that applies to embedded applications as well as command-line use, such as options and performance.

To embed FOP in your application, 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).

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.

2.1. Logging

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.

Call `setLogger(Logger)` always immediately after instantiating the Driver object. See here:

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

```
/*..*/

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

2.2. Logging (Upcoming FOP 1.0 Version only)

Logging is handled automatically via Jakarta Commons-Logging, which uses JDK logging by default. No special driver configuration is needed. For specialized configuration of Commons-Logging (e.g. to use a different logger or to change logging levels), please see the Jakarta Commons-Logging site.

2.3. Processing XSL-FO

Once the Driver is set up, one of the `render()` methods is called. Depending on whether DOM or an InputSource 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. You don't have to call `run()` or `render()` on the Driver object if you use `getContentHandler()`.

Here is an example use of Driver:

```
Driver driver = new Driver();
//Setup logging here: driver.setLogger(...
driver.setRenderer(Driver.RENDER_PDF);
driver.setInputSource(new FileInputSource(args[0]));
driver.setOutputStream(new FileOutputStream(args[1]));
driver.run();
```

2.4. Processing XSL-FO generated from XML+XSLT

If you want to process XSL-FO generated from XML using XSLT we recommend using standard JAXP to do the XSLT part and piping the generated SAX events directly through to FOP. Here's how this would look like:

```
Driver driver = new Driver();
//Setup logging here: driver.setLogger(...
driver.setRenderer(Driver.RENDER_PDF);

//Setup the OutputStream for FOP
```

```
driver.setOutputStream(new java.io.FileOutputStream(outFile));

//Make sure the XSL transformation's result is piped through to FOP
Result res = new SAXResult(driver.getContentHandler());

//Setup XML input
Source src = new StreamSource(xmlFile);

//Setup Transformer
Source xsltSrc = new StreamSource(xslFile);
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer(xsltSrc);

//Start the transformation and rendering process
transformer.transform(src, res);
```

Note:

There's no need to call `run()` or `render()`.

This may look complicated at first, but it's really just the combination of an XSL transformation and a FOP run. It's also easy to comment out the FOP part for debugging purposes, for example when you're tracking down a bug in your stylesheet. You can easily write the XSL-FO output from the XSL transformation to a file to check if that part generates the expected output.

For fully working examples of the above and hints to some interesting possibilities, see the examples section below.

3. Controlling logging

Current FOP 0.20.x production uses the Logger package from Apache Avalon Framework to do logging. See the Apache Avalon Framework for more information.

Per default FOP uses the SimpleLog 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.

4. Input Sources

The input XSL-FO document is always handled internally as SAX (see the Parsing Design Document for the rationale). However, the input itself can be provided in a variety of ways to FOP, which normalizes the input (if necessary) into SAX events:

- **SAX Events through SAX Handler**: `FOTreeBuilder` is the SAX Handler which is obtained through `getContentHandler` on `Driver`.
- **DOM (which is converted into SAX Events)**: The conversion of a DOM tree is done via the `render(Document)` method on `Driver`.
- **Data Source (which is parsed and converted into SAX Events)**: The `Driver` can take an `InputSource` as input. This can use a `Stream`, `String` etc.
- **XML+XSLT Transformation** (which is transformed using an XSLT Processor and the result is fired as SAX Events: `XSLTInputHandler` is used as an `InputSource` in the render(`XMLReader`, `InputSource`) method on `Driver`.

There are a variety of upstream data manipulations possible. For example, you may have a DOM and an XSL stylesheet; or you may want to set variables in the stylesheet. Interface documentation and some cookbook solutions to these situations are provided in Xalan Basic Usage Patterns.

See the Examples for some variations on input.

5. Using a Configuration File

To access an external configuration:

```
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.

No further reference to the `options` variable is necessary. The "options = " is actually not even necessary.

See Multithreading FOP for issues related to changing configuration in a multithreaded environment.

6. Setting the Configuration Programmatically

If you wish to set configuration options from within your embedded application, use the `Configuration.put` method. Here is an example that sets the "baseDir" configuration in a Unix environment:

```
org.apache.fop.configuration.Configuration.put("baseDir","/my/base/dir");
```

Here is another that sets baseDir in a Windows environment:

```
org.apache.fop.configuration.Configuration.put("baseDir","C:\my\base\dir");
```

See Multithreading FOP for issues related to changing configuration in a multithreaded environment.

7. Hints

7.1. 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.

7.2. AWT issues

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 may not automatically quit when finished, you will need to call `System.exit()`. These issues should be fixed in the upcoming JDK 1.4.

If you run into trouble running FOP on a head-less server, please see the notes on Batik.

7.3. 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.

8. Improving performance

There are several options to consider:

- Whenever possible, try to use SAX to couple the individual components involved (parser, XSL transformer, SQL datasource etc.).
- Depending on the target OutputStream (in case of an FileOutputStream, but not for a ByteArrayOutputStream, for example) it may improve performance considerably if you buffer the OutputStream using a BufferedOutputStream:
  `driver.setOutputStream(new java.io.BufferedOutputStream(out));`
  Make sure you properly close the OutputStream when FOP is finished.
- Cache the stylesheet. If you use the same stylesheet multiple times you can setup a JAXP `Templates` object and reuse it each time you do the XSL transformation. (More information can be found here.)
- Use an XSLT compiler like XSLTC that comes with Xalan-J.

9. Multithreading FOP

FOP is not currently completely thread safe. Although the relevant methods of the Driver object are synchronized, FOP uses static variables for configuration data and loading images. Here are

some tips to mitigate these problems:

- To avoid having your threads blocked, create a Driver object for each thread.
- If possible, do not change the configuration data while there is a Driver object rendering. Setup the configuration only once, preferably in the `init()` method of the servlet.
- If you must change the configuration data more often, or if you have multiple servlets within the same webapp using FOP, consider implementing a singleton class to encapsulate the configuration settings and to run FOP in synchronized methods.

There is also a known issue with fonts being jumbled between threads when using the AWT renderer (which is used by the -awt and -print output options). In general, you cannot safely run multiple threads through the AWT renderer.

10. Examples

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

10.1. ExampleFO2PDF.java

This example (current 0.20.5) (future 1.0dev) demonstrates the basic usage pattern to transform an XSL-FO file to PDF using FOP.

Example XSL-FO to PDF

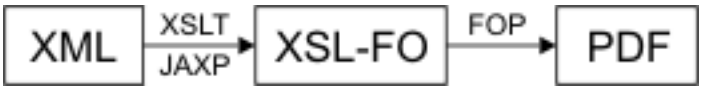
10.2. ExampleXML2FO.java

This example (current 0.20.5) (future 1.0dev) 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

10.3. ExampleXML2PDF.java

This example (current 0.20.5) (future 1.0dev) 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.

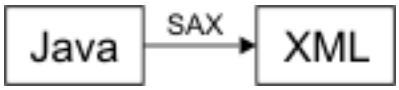


Example XML to PDF (via XSL-FO)

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.

10.4. ExampleObj2XML.java

This example (current 0.20.5) (future 1.0dev) 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).

10.5. ExampleObj2PDF.java

This example (current 0.20.5) (future 1.0dev) 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)

10.6. ExampleDOM2PDF.java

This example (current 0.20.5) (future 1.0dev) has FOP use a DOMSource instead of a StreamSource in order to use a DOM tree as input for an XSL transformation.

10.7. ExampleSVG2PDF.java (PDF Transcoder example)

This example (applies to 0.20.5 and future 1.0dev) shows use of the PDF Transcoder, a sub-application within FOP. It is used to generate a PDF document from an SVG file.

10.8. Final notes

These examples should give you an idea of what's possible. It should be easy to adjust these examples to your needs. Also, if you have other examples that you think should be added here, please let us know via either the FOP-USER or FOP-DEV mailing lists. Finally, for more help please send your questions to the FOP-USER mailing list.