Fop

What is FOP?

FOP is the world's first print formatter driven by XSL formatting objects. It is a Java application that reads a formatting object tree and then turns it into a PDF document. The formatting object tree, can be in the form of an XML document (output by an XSLT engine like XT or Xalan) or can be passed in memory as a DOM Document or (in the case of XT) SAX events.
The latest version of Fop is 0.18 and it supports the xsl:fo candidate release. You can download Fop including a precompiled version, the source code and many example files to get you started. Pointers to introductions into xsl:fo can be found in the section specifications . Please be aware, that Fop is at the moment not a full implementation of the basic conformance level of the xsl:fo standard. You can find a list of supported flow objects and properties in the section Features and in section Limitations in what way this support is limited.
FOP is part of Apache's XML project. The homepage of FOP is http://xml.apache.org/fop . A list of known bugs, the committers to this project and the tasks they are working on can be found in the file Status in the root of your Fop distribution. A history of the latest changes to the project can be found in CHANGES.

FOP Objectives

The goals of the Apache XML FOP Project are to deliver an XSL FO->PDF formatter that is compliant to at least the Basic conformance level described in the W3C Candidate Recommendation 21 November 2000, and that complies with the 11 March 1999 Portable Document Format Specification (Version 1.3) from Adobe Systems.
Conformance to the XML 1.0 Recommendation, XSLT 1.0 Recommendation and the XML Namespaces Recommendation is understood. Other relevant documents, such as the XPath and XLink Working Drafts, are referenced as necessary. The FOP Project will attempt to use the latest version of evolving specifications.
Secondary goals of the FOP Project (also, "FOP") are desirable requirements that also have a high priority.
One secondary goal of the FOP Project is to deliver a follow-on version of the formatter that is compliant to the Extended conformance level described in the XSL FO. Any formatting objects that cannot be translated into PDF will be explicitly identified as such.
Another secondary goal is to improve the conversion of SVG into PDF. This includes the use of FOP to work as a transcoder for Batik to convert an SVG document into a PDF document and the fully support embedding SVG inside fo documents in an fo:instream-foreign-object.
A final secondary goal is the continued refinement of the FOP design and implementation. In particular, maintaining the separation between formatting and rendering, continuing to support the AWT Viewer renderer (backend), and providing new renderers, are all desirable objectives.

TODO List for FOP

FOP TODO

Core Features - these are the areas that are important to getting fop to be useable for general use.
Processing improvements - these are mainly things that can be improved in the way fop works, eg. resources usage, pdf output etc.
Enhancements - these are added functionality that might be useful, no comment is made about the suitability of these suggestions.

Core Features

Layout

This is the crucial part to getting FOP to make it to the next step.

Webstart Demo

a demo using Java Webstart that runs the awt viewer and supports linking from an index

Website

Need to improve the website to better demonstrate what fop is and can do. Examples, screenshots, pdf documents.

Property Support

Currently all properties are in the xml file. We need to handle all default values properly (including ones that change depending on the element) and all possible values.

Property Resolution

This is support for the functions in property values that evaluate some expression. Better support for resolution including support for "inherit" and values with lists.

Validity Checking

Check the validity of children for a particular element to ensure there are no invalid children. Should help process the layout better.

Processing Improvements

Logging

Support for proper logging with logKit.

Image Handling

Needs to be a bit more solid and allow for direct insertion of images (jpeg) into the pdf.

Configuration

Support for avalon. Better multithread handling.

More PDF Outputs

Support for streaming and linearized pdf to help with different deployement of pdf documents.

i18n in awt viewer

Use proper i18n handling for awt viewer.

better abstraction of rendering classes

put all common rendering processes into an abstract class that has no dependancies on any specific renderers.

Enhancements

linking support in svg

support the a link rendering for the svg output.

input from url

be able to specify a url as the input fo (or xml, xsl) documents.

improve text handling in svg

support (better) the direct rendering of text into pdf graphics and other similair outputs

svg renderer (output to svg doc for slide presentation)

Create an SVGRenderer that will render all pages onto a single svg document suitable for slide show presentations (with batik) suggested by Vincent Hardy.

stream encoding

support for different encoding on different types of streams in pdf document.

svg features

currently patterns and gradients are not generated properly

Downloading FOP

You can download the latest release version from the distribution directory .
The file contains also the documentation (including some example fo files) and the source.
If you want to work with the latest and nicest code, you can use the cvs version. See the section on AnonCVS in the xml.apache.org documentation for details. Sometimes people have difficulties to access the cvs server; in this case you can download a snapshot from the cvs files here. In both cases you have to build Fop yourself - see Compiling Fop for details.
To run FOP from the command line, see Running FOP . If you are interested in embedding FOP in a Java application of your own, see Embedding FOP .

Running FOP

Prerequisites

Following software must be installed:
a) Java 1.1.x or later (If you want to use the previewer (option -awt), you need Swing or Java 2)
b) All libraries you need are part of the Fop distribution and can be found in the xml-fop/lib directory. Look at the batch/shell script fop.bat/fop.sh to see, how Fop can be invoked easily. These libraries are included:

- An XML parser which supports SAX and DOM like Xerces-J. (Xerces is the default xml parser)
- An XSLT processor
- The SVG library batik.jar is the library from the batik project at xml.apache.org.
- The imaging library Jimi from Sun

Starting FOP as an standalone application

```
Fop [options] [-fo|-xml] infile [-xsl file] [-awt|-pdf|-mif|-pcl|-txt|-print] <outfile>
```

[OPTIONS]
-d debug mode -x dump configuration settings -q quiet mode -c cfg.xml use additional configuration file cfg.xml -l lang the language to use for user information
[INPUT]
infile xsl:fo input file (the same as the next) -fo infile xsl:fo input file -xml infile xml input file, must be used together with -xsl -xsl stylesheet xslt stylesheet
[OUTPUT]
outfile input will be rendered as pdf file into outfile -pdf outfile input will be rendered as pdf file (outfile req'd) -awt input will be displayed on screen -mif outfile input will be rendered as mif file (outfile req'd) -pcl outfile input will be rendered as pcl file (outfile req'd) -txt outfile input will be rendered as text file (outfile req'd) -print input file will be rendered and sent to the printer see options with "-print help"
[Examples]
Fop foo.fo foo.pdf Fop -fo foo.fo -pdf foo.pdf (does the same as the previous line) Fop -xsl foo.xsl -xml foo.xml -pdf foo.pdf Fop foo.fo -mif foo.mif Fop foo.fo -print or Fop -print foo.fo Fop foo.fo -awt

Problems

If you have problems running FOP, please have a look at the FOP FAQ. If you don't find a solution there, you can ask for help on the list fop-dev@xml.apache.org. Maybe it is a bug and maybe somebody is already working on it.

Features

What's Implemented?

The following formatting objects and properties of the xsl-fo candidate recommendation are implemented.Please have also a look at the section on limitations

1) Formatting Objects

This section follows the table "B Formatting Object Summary" in the xsl:fo specification. At the end of each sub-section you find listed what is not implemented.

B.1 Declaration and Pagination and Layout Formatting Objects

- root
- page-sequence
- page-sequence-master
- single-page-master-reference
- repeatable-page-master-reference
- repeatable-page-master-alternatives
- conditional-page-master-reference
- layout-master-set
- simple-page-master
- region-body
- region-before
- region-after
- region-start
- region-end
- flow
- static-content

Not implemented: declarations, color-profile, title

B.2 Block Formatting Objects

- block

Not implemented: block-container

B.3 Inline Formatting Objects

- character
- external-graphic
- inline
- instream-foreign-object

- leader
- page-number
- page-number-citation, see limitations

Not implemented: bidi-override, initial-property-set, inline-container

B.4 Table Formatting Objects

- table
- table-body
- table-cell
- table-column
- table-footer
- table-header
- table-row

Not implemented: table-and-caption, table-caption

B.5 List Formatting Objects

- list-block
- list-item
- list-item-body
- list-item-label

B.6 Link and Multi Formatting Objects

- basic-link (internal and external)

Not implemented: multi-switch, multi-case, multi-toggle, multi-properties, multi-property-set

B.7 Out-of-line Formatting Objects

- footnote
- footnote-body

Not implemented: float

B.8 Other Formatting Objects

- wrapper

Not implemented: marker, retrieve-marker

border and padding shorthand properties

2) Properties

Property values can be computed. Compound properties are also understood by Fop.

- background-color
- blank-or-not-blank
- border-after-color
- border-after-style
- border-after-width
- border-before-color
- border-before-style
- border-before-width
- border-bottom
- border-bottom-color
- border-bottom-style
- border-bottom-width
- border-color (only one value allowed)
- border-end-color
- border-end-style
- border-end-width
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-start-color
- border-start-style
- border-start-width
- border-style
- border-top
- border-top-color
- border-top-style
- border-top-width
- border-width
- bottom
- break-after
- break-before
- character
- color
- column-count
- column-gap
- column-width
- country
- end-indent

- extent
- external-destination
- flow-name
- font-family
- font-size
- font-style
- font-weight
- height
- hyphenate
- hyphenation-character
- hyphenation-push-character-count
- hyphenation-remain-character-count
- id
- initial-page-number
- internal-destination
- keep-with-next (broken)
- language
- leader-alignment (not value "page")
- leader-length (see limitations)
- leader-pattern (not value "use-content")
- leader-pattern-width
- left
- line-height
- margin-bottom (only on pages and regions)
- margin-left (only on pages and regions)
- margin-right (only on pages and regions)
- margin-top (only on pages and regions)
- master-name
- maximum-repeats
- number-columns-spanned
- odd-or-even
- padding (only one value allowed)
- padding-after
- padding-before
- padding-bottom
- padding-end
- padding-left
- padding-right
- padding-start
- padding-top
- page-height
- page-position
- page-width

- position (allowed values: "static" (default),"relative", "absolute", fixed)
- provisional-distance-between-starts
- provisional-label-separation
- ref-id
- region-name
- right
- rule-style
- rule-thickness
- space-after.optimum
- space-before.optimum
- span
- src
- start-indent
- table-omit-footer-at-break
- table-omit-header-at-break
- text-align
- text-align-last
- text-decoration
- text-indent
- top
- white-space-collapse
- width
- wrap-option

All other properties are not implemented.

3)SVG Support

FOP uses Batik directly for its SVG support. Therefore FOP supports the same elements and properties as are supported by Batik. As FOP is designed for rendering to a static medium then only static SVG is rendered.

Due to some limitations in PDF some SVG images, particularly ones with effects or transparency, may not come out correctly. The images should still be rendered correctly for the AWT and Print renderers.

Limitations

FOP implements the fo objects and properties listed in features, sometimes it does so only in a limited way.

fo:leader

leader-length.minimum is not used at all

page-number-citation

Only works for table of contents without any problems. The case where the page number doesn't fit on a line isn't handled, and any text on the same line and after the page-number might not appear exactly where you want it to.

Padding

Padding works in conjunction with indents and spaces. It is only implemented for blocks. At the moment padding can't be used to make extra space (indents+spaces must be used), but only to control how much the background-color extends beyond the content rectangle.

Tables

There two limitations for tables: 1) FOP needs you to explicitly specify column widths 2) Cells have to contain block-level FOs. They can't contain straight character data.
A working basic example of a table looks like this:

```
<fo:table>
 <fo:table-column column-width="150pt"/>
 <fo:table-column column-width="150pt"/>
 <fo:table-body font-size="10pt" font-family="sans-serif">
  <fo:table-row>
   <fo:table-cell>
    <fo:block>text</fo:block>
   </fo:table-cell>
   <fo:table-cell>
    <fo:block>text</fo:block>
   </fo:table-cell>
  </fo:table-row>
  <fo:table-row>
```

```
    <fo:table-cell>
     <fo:block>text</fo:block>
    </fo:table-cell>
    <fo:table-cell>
     <fo:block>text</fo:block>
    </fo:table-cell>
   </fo:table-row>
   <fo:table-row>
    <fo:table-cell>
     <fo:block>text</fo:block>
    </fo:table-cell>
    <fo:table-cell>
     <fo:block>text</fo:block>
    </fo:table-cell>
   </fo:table-row>
  </fo:table-body>
</fo:table>
```

Examples

Examples for the use of xsl:fo can be found in the Fop distribution in the subdirectory xml-fop/docs/examples/fo. You can start transformation of all fo files into pdf files by starting xml-fop/docs/examples/runtests. The resulting test files can be found in xml-fop/docs/examples/tests

At the moment the following files are part of the distribution:

- simple.fo - a very simple file which gives use a first impression of the structure of an fo file
- normal.fo - a simple file showing the use of a 2 level of headings, normal text and a header.
- table.fo - some table examples
- list.fo - a short tutorial how to use list fo's and properties
- images.fo - shows how to embed gif and jpg images into the xsl:fo file using external-graphic.
- border.fo - a not so simple example how to use borders in tables
- extensive.fo - a longer test file containing a lot of different flow objects and properties. A good candidate to test your bugfix or new Fop code.
- leader.fo - shows different uses of fo:leader, p.e. as rule or in a table of content
- normalex.fo - shows the use of computed property values
- inhprop.fo - shows the use of inherited property values
- instream.fo - shows the use of fo:instream-foreign-object together with svg
- textdeko.fo - shows the use of the property textdecoration
- readme.fo - uses an old version of Fop documentation for a longer example
- Look also into the directory examples/svg. There you find some very extensive svg examples. Just start makedoc.
- In the directory examples/pagination you find a suite of examples showing the use of xsl:fo pagination.

Developers will find the first steps to a test suite for all implemented formatting objects and properties in xml-fop/test/xml/.

Configuration

How to configure Fop

In the directory xml-fop/conf you will find two configuration files. One of them, config.xml, is only intended for Fop developers, who want to add new default values to some Fop feature. Don't change this file. For user configuration there is a file called userconfig.xml. It contains templates for all settings a user can change. Most of them are commented out. Uncomment the entry you want to set and change the value according to your wishes. Please regard any comments which specify the value range. And, well, the configuration files are xml files, so keep them at least well-formed ;-)
The file userconfig.xml is not read automatically, but the user must specify its use on the command line. See Running Fop for details.

setting up hyphenation

Fop comes already with some hyphenation pattern. If you need a hyphenation pattern which isn't included in the distribution, do the following:
1. get the TeX hyphenation pattern file and turn it into an xml file which conforms to the hyphenation.dtd in the sub directory /hyph
2. name this new file following this schema: languageCode_countryCode.xml. If you don't need a country code, leave it away, p.e. the file name for an American english hyphenation pattern would look like this: en_US.xml. For an Italian file: it.xml. Language and country codes must be the same as in xsl:fo, that is follow ISO 639 and ISO 3166 respectively. NOTE: The ISO 639/ISO 3166 convention is that language names are written in lower case, while country codes are written in upper case.
3. If you have build your new hyphenation pattern file successfully there are two ways to make it accessible to Fop.
a) Put this new file into the directory /hyph and rebuild Fop. The file will be picked up and added to the fop.jar.
b) Put the file into a directory of your choice and specify this directory in the userconfig.xml in the entry <hyphenation-dir>.

Font Support

Status

FOP (building PDF files) normally supports only the base 14 font package defined in the Adobe PDF specification. That includes the following fonts: Helvetica, Times, Courier, Symbol and ZapfDingbats.
Font support in FOP can be extended by the addition of font metric files (written in XML) created from Adobe Type 1 fonts and Truetype fonts. No other font types (Type 3, etc.) are supported at this time.

Adding additional Type 1 fonts

As mentioned above you need an XML file containing font metrics to be able to use an additional font. FOP contains a tool that can generate such a font metrics file from a PFM file, which normally comes with the font file.

Generating a font metrics file

Run the class org.apache.fop.fonts.apps.PFMReader to generate the XML file.

```
java -cp fop.jar;xerces.jar;xalan.jar;batik.jar
org.apache.fop.fonts.apps.PFMReader pfm-file xml-file
```

Note: The tool will construct some values (FontBBox, StemV and ItalicAngle) based on assumptions and calculations which are only an approximation to the real values. FontBBox and Italic Angle can be found in the human-readable part of the PFB file. The PFMReader tool does not yet interpret PFB files, so if you want to be correct, you may have to adjust the values in the XML file manually. The constructed values however appear to have no visible influence.

Register the fonts within FOP

Edit conf/userconfig.xml and add entries for the font if the fonts section, ie:

```
<font metrics-file="cyberbit.xml" kerning="yes"
embed-file="C:\WINNT\Fonts\Cyberbit.ttf"> <font-triplet name="Cyberbit"
style="normal" weight="normal"> </font>
```

Adding additional TrueType

Adding Truetype fonts is almost identical to the process of adding type 1 fonts. The main difference is in the first step.

Generating a font metrics file

As mentioned above you need an XML file containing font metrics to be able to use an additional font. FOP contains a tool that can generate such a font metrics file from your truetype font file.
Create metrics for the fontfile (we assume the file has the name cmr10.ttf and exists in c:\myfonts\):

```
java org.apache.fop.fonts.apps.TTFReader C:\myfonts\cmr10.ttf C:\myfonts\cmr10.ttf ttfcm.xml
```

TrueType collections

TrueType collections (.ttc files) contains more than one font. To create metrics for a ttc file you must specify the font in the collection with the -ttcname option to TTFReader.
To get a list of the fonts in a collection, just start the TTFReader as if it were a normal truetype file (without the -ttcname option). It will then display all the font names and exit with an Exception...
Example on generating metrics for a .ttc file:

```
java org.apache.fop.fonts.apps.TTFReader -ttcname "MS Mincho" msmincho.ttc msminch.xml
```

Register the fonts within FOP

Same as for Type 1 fonts.

Embedding fonts

Font embedding is enabled in the userconfig.xml file.

Remember to start fop with -c conf/userconfig.xml

Extensions to FOP

Sometimes it is desirable to have extensions to xsl:fo in order to support some feature of the output format which isn't covered by the xsl:fo specification.

Default Extensions

These extension are available by default. They are automatically loaded and you only need to provide the correct namespace for your embedded xml markup.

SVG Please see the SVG page for more details.

Bookmarks

To use this standard Fop extension, you need to add a namespace entry for http://xml.apache.org/fop/extensions on the root element.
You can provide outlines inside the root object (but outside any page-sequences or other formatting objects). Here's an example of an outline entry:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:fox="http://xml.apache.org/fop/extensions"> <fox:outline
internal-destination="sec3"> <fox:label>Running FOP</fox:label> <fox:outline
internal-destination="sec3-1"> <fox:label>Prerequisites</fox:label>
</fox:outline> <fox:outline> </fo:root>
```

It works similarly to a basic-link. There is also an external-destination property, but it isn't supported currently. See the pdfoutline.fo file in docs/examples/fo for a more complete example.

Adding Your Own

To add your own extension you need to do the following things.
Write code that implements your extension functionality. The easiest place to start is by looking at the code in org.apache.fop.extension.
Create a jar file with your classes, it must also include the following file "/META-INF/services/org.apache.fop.fo.ElementMapping". In this file you need to put the fully qualified classname of your element mappings class. This class must implement the "org.apache.fop.fo.ElementMapping" interface.
Create your fo file with the extra xml data embedded in the file with the correct name space. The examples for svg and pdfoutline.fo show how this can be done.
Put your jar file in the classpath and then run fop over the fo file.

SVG in FOP

Introduction

FOP uses the SVG library from Batik to handle SVG. This format can be handled as an `fo:instream-foreign-object` or in a separate file referenced with `fo:external-graphic`. Either way the SVG document will be read in and converted into a DOM in Batik. This DOM will then be used by the renderer to create the graphical image.
The AWT and Print renderers simply use batik to draw the SVG into a graphic.
In the case of the PDF renderer there is a PDFGraphics2D class that Batik uses to render the image into. This class converts the drawing instructions into PDF markup which is placed into the current PDF document.

Converting SVG to a PDF Document

It is possible to convert a standalone SVG document directly into a simple page PDF document. This is possible through the use of Batik's transcoder mechanism. `java org.apache.batik.apps.rasterizer.Main -m application/pdf document.svg` This will output the svg document as "document.pdf" containing a PDF rendering of the SVG file.
It is also possible to specify the width and/or height of the PDF document on the command line with -w and -h or if you are using the transcoder api you can use the transcoding hints.
Currently the SVG image is drawn at the SVG document size and simply scaled in PDF to the new size. So the result may not be the best possible. For example if you have any images or effects it will draw them at the original resolution of the svg document. When this is viewed in the pdf it will have an incorrect resolution for the size of the pdf.
The size of the pdf file will also remain the same regardless of what size the page is.
For more information see Batik for how transcoders work.
These are the relevant classes, found in the package org.apache.fop.svg :

- ***PDFGraphics2D*** used for drawing onto a Graphics2D into an existing pdf document, used internally to draw the svg.
- ***PDFDocumentGraphics2D*** used to create a pdf document and inherits from PDFGraphics2D to do the rest of the drawing. Used by the transcoder to create a standalone pdf document from an svg. Can be used independantly the same as any Graphics2D.
- ***PDFTranscoder*** used to transcode an svg document into a standalone pdf, via PDFDocumentGraphics2D.

Compiling FOP

Compilation is started by executing build, either as a batch file on win32 (build.bat) or as a shell script on unix. Before you can start one of these scripts, you have to setup your classpath and the environment variable JAVA_HOME (see below).
The compilation uses Ant, a replacement of make (you can find more information about Ant at jakarta.apache.org). build.xml is the replacement of makefile. Look there for detailed information on the build process and different targets.
A help screen is shown by calling "build usage".
If you only want to use Fop, you don't need to build it. A fop.jar comes with the distribution.

Setting up your classpath

You don't have to setup your classpath; all libraries needed to compile Fop are coming with the distribution and are referenced by the build script, so you only need to care about them, if you build Fop in any other way. See build.bat/build.sh for details.

Setting of JAVA_HOME

You have to set the enviroment variable JAVA_HOME. It must point to your local JDK root directory. This is true, even if you use JDK 1.2 or above, which normally don't need this setting. It is used by Ant, the compilation software.

Problems

If you have problems compiling Fop, please try this first:

- delete the build directory completely and try a new build from scratch
- check, whether you have an older version of xerces.jar, xalan.jar, batik.jar somewhere in you classpath.

If you still have problems, please look at the page bugs, for further help.

Embedding FOP

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).

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(RENDER_PDF); driver.run();
```

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.

A third possibility may be used to build the FO Tree, namely calling getContentHandler() and firing 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.

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

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.

Testing FOP

Running and Using Tests

Testing is an important part of getting FOP to operate correctly and conform to the necessary standards.
A testing system has been set up that works with as a build target when developing with FOP. A developer can run the tests after making changes to the code, the aim is to have the tests run to verfiy that nothing working has been broken.
To setup the testing the developer must place a reference fop.jar in the "<cvs_repository>/test/reference/" directory. This jar will be dynamically loaded to create the reference output.

W3C TestSuite

The testing is set up so that you can download the testsuite from http://www.w3.org/Style/XSL/TestSuite/ , unzip the file into the base directory of FOP. Then you can uncomment the lines in the build.xml file in the test target and it will run through all the tests in the testsuite distribution.

Writing a Test

A test belongs to one of a few catagories. A basic test should excercise one element in a number of situations such as changing a property. This should have at least one normal value, one border value and one invalid value. If the property can be of different types then this should also be included.
A bug test is a test that is specifically aimed at a problem with FOP. That is, the test is not excercising the specification but rather a problem with FOP in handling a particular situation that is not exposed with the other testing.
A system test is one that tests the abitlity of FOP to handle a number of different elements together.
A test can consist of a complete fo document or a part of the document such as some elements that will be placed into the flow of a standard document.

Submitting a Test

If you have a test which you think would be useful you should supply the test and a diff to the appropriate test suite xml file. Make sure that the test works as would be expected against the current build.

How Testing Works

The tests are stored in the "<cvs_repository>/test" directory.
You can run the tests by specifying the build target "test" ie: `build.sh test`
This will then compare the current code in the local src directory to a specified release of FOP. Any differences between the current code and the output from the reference version will be reported. If the test previously passed then the test run will have failed.
The testing is done by reading a test suite xml file, which corresponds to the standard testsuite.dtd supplied from w3c. This xml file contains a test xml file and an xsl file (which may simply copy the file). It also contains information such as if the test has passed and any comments.
For FOP the testing is done by rendering all the testing documents using the XML renderer. The XML files are then compared to see if there are any differences.

SVG Testing

The testing of SVG is not part of this testing system. SVG is tested for its rendering accuracy by using the transcoding mechanism via Batik. So that the only part that needs testing is how the SVG image is embedded inside the flow of the fo document.

Getting involved

Read the Status file

The Status file contains the list of features people are working on at the moment. And an outline what steps are next.

Subscribe to the fop discussion list

You can subscribe to fop-dev@xml.apache.org by sending an email to fop-dev-subscribe@xml.apache.org
Sending bug reports and feature requests to the list is a welcome and important contribution to developing Fop.
Read also the archive of the discussion list fop-dev to get an idea of the issues being discussed.

Look at the developer's code using cvs

Between releases the newest code can be accessed via cvs. To do this you need to install a cvs client on your computer, if it is not already there. An explanation how to connect to the Fop source repository can be found at http://xml.apache.org/cvs.html . An introduction into cvs and the cvs manual can be found in the reference library .
All changes to the code repository are annonced in a special discussion group. You can subscribe to fop-cvs@xml.apache.org by sending an email to fop-cvs-subscribe@xml.apache.org . If you want to contribute to the development of Fop you should subscribe, because it is important that you follow changes being made.

Contributing code, tests and documentation

If you want to contribute code (p.e. a bugfix), a test or documentation (p.e. an additional example), please do the following:
1) Make sure your code doesn't break the existing one and that Fop still compiles.
2) Create a file which shows the differences to the existing code.
3) Send this file to fop-dev@xml.apache.org.
One of the committers will test your code and commit it to the code repository.
If you have a test or useful bug test you should read this page .
BTW: The Apache project knows different roles for contributors, namely 'users', 'developers', 'committers' and the 'Project Management Committee' (An explanation of these roles can be

found here).

Get familiar with the Fop related standards

At the moment Fop is mainly a tool to render XSL:FO files to pdf. Therefore if you want to contribute to Fop you should become familiar with these standards. You can find their internet addresses on our website.

Fop's architecture

A bird's eye view on the way Fop operates can be found in the document FOP Mechanics
If you want to extend the functionality of FOP by adding new formatting objects, you should do the following:
1. FO Object: Write a class which contains the description of your formatting object and put it into the package fop.fo.flow, fop.fo.pagination (if it is a property it goes to fop.fo.properties. The classes in this package are generated via an xslt stylesheet located in codegen/properties.xml)
2. Element Mapping: Add it to the list in fop.fo.StandardElementMapping (if it is a property you need to add it to fop.fo.PropertyListBuilder)
3. Area: Either your need can be fulfilled within one of the existing classes in fop.layout, then just add the code to handle the new fo/property or you must write a new one.
4. Renderer: Choose the renderer you are interested in. If you worked on an existing layout class you must add code to handle the new features to the already existing area specific method in the renderer class. Otherwise you have to add a new method.

FOP Mechanics

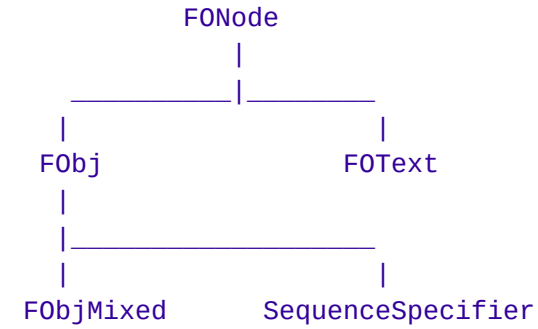
Introduction

The overall process is controlled by ***org.apache.fop.apps.Driver*** . In this class, a typical sequence is:

```
Driver driver = new Driver();
driver.setRenderer("org.apache.fop.render.pdf.PDFRenderer", version);
driver.setOutputStream(new FileOutputStream(args[1])); driver.render(parser,
inputHandler.getInputSource());
```

Formatting Object Tree

The class ***org.apache.fop.fo.FOTreeBuilder*** is responsible for actually constructing the FO tree. The key SAX events used are
`startElement()`,
`endElement()` and `characters()`.
All formatting objects derive from abstract class ***org.apache.fop.fo.FONode*** . The other FO classes inherit from ***FONode*** as follows:



FO's extending FObj:
Package org.apache.fop.fo.pagination:
`LayoutMasterSet`
`PageSequence`
`RegionAfter`
`RegionBefore`
`RegionBody`
`Root`
`SequenceSpecification`
`SimplePageMaster`
Package org.apache.fop.fo.flow:
`BlockContainer`

`DisplayGraphic`
`DisplayRule`
`DisplaySequence`
`Flow`
`InlineGraphic`
`ListBlock`
`ListItem`
`ListItemBody`
`ListItemLabel`
`PageNumber`
`StaticContent`
`Table`
`TableBody`
`TableCell`
`TableColumn`
`TableRow`

FO's extending SequenceSpecifier:
Package org.apache.fop.fo.pagination:

`SequenceSpecifierAlternating`
`SequenceSpecifierRepeating`
`SequenceSpecifierSingle`

FO's extending FObjMixed:
Package org.apache.fop.fo.flow:

`Block`
`Inline`
`BasicLink`

FONode

The class inheritance described above only describes the nature of the content. Every FO in FOP also has a parent, and a Vector of children. The parent attribute (in the Java sense), in particular, is used to enforce constraints required by the FO hierarchy.
FONode, among other things, ensures that FO's have a parent, that they have children, that they maintain a marker of where the layout was up to (for FObj's it is the child number, and for FOText's it is the character number), and that they have a `layout()` method.

Making FO's

Every FO class has code that looks something like this:

```
public static class Maker extends FObj.Maker {
```

```
    public FObj make(FObj parent, PropertyList propertyList)
      throws FOPException
    {
      return new SimplePageMaster(parent, propertyList);
    }
}
```

The class also has a static method that resembles

```
public static FObj.Maker maker()
    {
      return new PageSequence.Maker();
    }
```

A hash 'fobjTable' exists in ***FOTreeBuilder***, and maps the FO names (such as 'fo:table') to object references to the appropriate factories (such as ***Table.Maker***).

Properties (recall that FO's have properties, areas have traits, and XML nodes have attributes) are also a concern of ***FOTreeBuilder***. It accomplishes this by using a ***PropertyListBuilder***. There is a separate ***PropertyListBuilder*** for each namespace encountered while building the FO tree. Each Builder object contains a hash of property names and their respective makers. It may also contain element-specific property maker hashes; these are based on the ***local name*** of the flow object, ie. ***table-row***, not ***fo:table-row***. If an element-specific property mapping exists, it is preferred to the generic mapping.

The base class for all properties is ***Property***, and all the property makers extend ***Property.Maker***. A more complete discussion of the property architecture may be found in Properties.

FO Formatting

FOTreeBuilder calls `format()` on the root FO, passing it the ***AreaTree*** reference. In turn, ***Root*** calls `format()` on each ***PageSequence***, passing it the ***AreaTree*** reference.

The ***PageSequence*** `format()` method does the following things:

1. Makes a ***Page***, using ***PageMasterFactory*** to produce a ***PageMaster***, and using `makePage()` in the latter class. In the simplest picture, a ***Page*** has 5 areas represented by ***AreaContainers***;
2. Handles layout for ***StaticContent*** objects in the 'before' and 'after' regions, if set. This uses the `layout()` method in ***StaticContent***;
3. If a page break is not forced, it will continue to layout the flow into the body area (***AreaContainer***) of the current page;
4. It continues with (1) when layout into the current page is done, but the flow is not empty.

Area Layout

FO's that represent actual areas, starting with ***Flow*** and ***StaticContent***, have a `layout()`

method, with the following signature:

```
public Status layout(Area area)
```

The fundamental role of the `layout()` method is to manage the layout of children and/or to generate new areas.

Example: the `layout()` method for ***Flow*** generates no new areas - it manages the layout of the flow children.

Example: the `layout()` method for ***Block*** generates a new ***BlockArea*** in and of itself, and also manages the layout of the block children, which are added to the ***BlockArea*** before that is itself added to its parent ***Area***.

`Layout()` methods are subject to the general constraint that possibly not all of their children can be accommodated, and they report back accordingly with an appropriate ***Status***.

Rendering

This is a separate process. The `render()` method in ***Driver*** is invoked (say, by ***CommandLine***) with the laid-out ***AreaTree*** and a ***PrintWriter*** as arguments. This actually calls the `render()` method in a specific implementation of the ***Renderer*** interface, typically ***PDFRenderer*** or ***AWTRenderer***.

At the highest level ***PDFRenderer***, for example, begins by rendering each ***Page***. The `render()` method in ***Page*** (as is the case for other areas), invokes a particular method in the renderer of choice, e.g. `renderPage()`. ***NOTE***: this system is bypassed for ***Page***, incidentally.

Renderers

PrintRenderer

The PrintRenderer is an abstract base class for print type renderers. Currently the PCL, PDF, and TXT renderers extend from this. This allows as much common functionality to be contained in one place as possible (at least as much as I could consolidate fairly quickly). Unfortunately I have not yet been able to make the renderPage and renderWordArea methods common. This is unfortunate because these methods seem to experience the most activity. Maybe soneone else will have a clever solution to this (without breaking them into a bunch of little bits).

It is my hope that this base class will be useful for other renderers as well.

PCLRenderer

The PCLRenderer is a FOP renderer that 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.

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 support is limited. Currently only lines, rectangles (may be rounded), circles, ellipses, text, simple paths, and images are supported. Colors are supported (dithered black and white) but not gradients.
- 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).
- SVG clipping is not supported.

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:

- 0 Portrait
- 1 Landscape
- 2 Reverse Portrait
- 3 Reverse Landscape

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.

TXTRenderer

The TXTRenderer is a FOP renderer that 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. Of course when limited to plain fixed pitch text the output does not always look very good.
The TXTRenderer 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.

UML Diagrams

You can find UML diagramms for all Fop packages (latest release version) here.

SVG

FOP supports svg rendering. SVG is supported as an instream-foreign-object embedded in an FO document or as an external SVG image.
If the svg is embedded in an instream-foreign-object then all the elements and attributes are read directly and converted into an SVG DOM representation using the Batik library. This is then stored as a DOM until required for rendering. The rendering process depends on the what type of renderer is being used.
The SVG DOM is rendered in the PDF renderer by using the abitlity of Batik to render DOM to a Graphics2D. First the DOM is converted into an intermediate representation then this is rendered to a PDFGraphics2D graphic object which writes the drawing instructions directly as PDF markup.
The AWTRenderer and the PrintRenderer use Batik directly to draw the SVG image into the current java Graphics2D context.
For more information see the SVG documentation.

Bugs

How to report bugs

Please report bugs to bugzilla, the Apache bug database. A copy of your bug report is sent automatically to the discussion list fop-dev@xml.apache.org.
Please make sure, before you report a bug, that it is not mentioned in the FAQ or in the list of open bugs at bugzilla.
Please make your description as concise as possible and add an example fo file with your report, which just demonstrates the problem. Thanks for your help!

Known bugs

A list of known bugs can be found at bugzilla.

Frequently Asked Questions

Introduction

Here we have some answers to common questions about FOP. This only covers general information about getting started with FOP and pointers to more information.
For up to date information or more details please visit the Fop FAQ site. The site uses Jyve to provide an interactive FAQ: http://www.OWAL.co.uk:8090/

Questions

Answers

What is FOP

FOP is a print formatter for XSL formatting objects.
It can be used to render an XML file containing XSL formatting objects into a page layout. The main target is PDF but other rendering targets are supported, such as AWT, PCL, text and direct printing.

How does FOP interact with other Apache Projects?

FOP is distributed with Cocoon as a PDF serializer for XSL (FO) documents.
Batik can be used with FOP to transcode an SVG image into a PDF document. The mime type for PDF is "application/pdf".

What is XSL (FO)

What can I do with FOP

How can I contribute

How do I author XSL documents

How can I see a demo

There will be a Java Webstart demo sometime in the future.

FOP Relevant Specifications and Links

Specifications

- XSL-FO Candidate Recommendation (21 November 2000)
- A dtd for the XSL-FO CR from November provided by N. Grigoriev from RenderX
- Supported SVG Candidate Recommendation (02 November 2000)
- XML Recommendation
- XSLT Recommendation
- Portable Document Format (PDF) 1.3 Reference Manual
- Simple API for XML (SAX)
- Document Object Model (DOM)
- Namespaces in XML Recommendation
- Java JDK 1.1 Documentation

Tutorials/Articles

- Elliotte Rusty Harold: Chapter 15 on xsl:fo from his excellent xml book
- Paul Sandoz: Using formatting objects with the slides dtd
- J. David Eisenberg: Using XSL Formatting Objects
- Miloslav Nic: XSL FO reference

Other resources

- Apache archive of fop-dev@apache.org
- External, but easier to browse archive fop-dev@apache.org
- There is an xsl:fo mailing list: www-xsl-fo@w3.org. Subscription info can be found here: http://www.w3.org/Mail/Request . And the archive can be found here: http://lists.w3.org/Archives/Public/www-xsl-fo/

License

The Apache Software License, Version 1.1

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
4. The names "FOP" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLU- DING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally created by James Tauber <jtauber@jtauber.com>. For more information on the Apache Software Foundation, please see http://www.apache.org/ .