Running Apache FOP

$Revision: 955884 $

Table of contents

1. System Requirements

The following software must be installed:

- Java 1.4.x or later Runtime Environment.
  - Many JREs >=1.4 contain older JAXP implementations (which often contain bugs). It's usually a good idea to replace them with a current implementation.
- Apache FOP. The FOP distribution includes all libraries that you will need to run a basic FOP installation. These can be found in the [fop-root]/lib directory. These libraries include the following:
  - Apache XML Graphics Commons, an shared library for Batik and FOP.
  - Apache Batik, an SVG library.
  - Apache Commons Logging, a logger abstraction kit.
  - Apache Commons IO, a library with I/O utilities.
  - Apache Excalibur/Avalon Framework, for XML configuration handling.

The following software is optional, depending on your needs:

- Graphics libraries. Generally, FOP contains direct support for the most important bitmap image formats (including PNG, JPEG and GIF). See FOP: Graphics Formats for details.
- PDF encryption. See FOP: PDF Encryption for details.

In addition, the following system requirements apply:

- If you will be using FOP to process SVG, you must do so in a graphical environment. See FOP: Graphics (Batik) for details.

2. Installation

2.1. Instructions

Basic FOP installation consists of first unzipping the `.gz` file that is the distribution medium, then unarchiving the resulting `.tar` file in a directory/folder that is convenient on your system. Please consult your operating system documentation or Zip application software documentation for instructions specific to your site.

2.2. Problems

Some Mac OSX users have experienced filename truncation problems using Stuffit to unzip and unarchive their distribution media. This is a legacy of older Mac operating systems, which had a 31-character pathname limit. Several Mac OSX users have recommended that Mac OSX users use the shell command `tar -xzf` instead.

3. Starting FOP as a Standalone Application

3.1. Using the fop script or batch file

The usual and recommended practice for starting FOP from the command line is to run the batch file fop.bat (Windows) or the shell script fop (Unix/Linux). These scripts require that the environment variable JAVA_HOME be set to a path pointing to the appropriate Java installation on your system. Macintosh OSX includes a Java environment as part of its distribution. We are told by Mac OSX users that the path to use in this case is `/Library/Java/Home`. **Caveat:** We suspect that, as Apple releases new Java environments and as FOP upgrades the minimum Java requirements, the two will inevitably not match on some systems. Please see Java on Mac OSX FAQ for information as it becomes available.

```
USAGE
Fop [options] [-fo|-xml] infile [-xsl file]
[-awt|-pdf|-mif|-rtf|-tiff|-png|-pcl|-ps|-txt|-at [mime]|-print] <outfile>
 [OPTIONS]
  -version          print FOP version and exit
  -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
  -r                relaxed/less strict validation (where available)
  -dpi xxx          target resolution in dots per inch (dpi) where xxx is a number
  -s                for area tree XML, down to block areas only
  -v                run in verbose mode (currently simply print FOP version and
continue)

  -o [password]     PDF file will be encrypted with option owner password
  -u [password]     PDF file will be encrypted with option user password
  -noprint          PDF file will be encrypted without printing permission
  -nocopy           PDF file will be encrypted without copy content permission
  -noedit           PDF file will be encrypted without edit content permission
  -noannotations    PDF file will be encrypted without edit annotation permission
  -a                enables accessibility features (Tagged PDF etc., default off)
  -pdfprofile prof  PDF file will be generated with the specified profile
                    (Examples for prof: PDF/A-1b or PDF/X-3:2003)

  -conserve         Enable memory-conservation policy (trades memory-consumption for
disk I/O)
                    (Note: currently only influences whether the area tree is
serialized.)

 [INPUT]
  infile            xsl:fo input file (the same as the next)
                    (use '-' for infile to pipe input from stdin)
  -fo  infile       xsl:fo input file
  -xml infile       xml input file, must be used together with -xsl
  -atin infile      area tree input file
  -ifin infile      intermediate format input file
  -imagein infile   image input file (piping through stdin not supported)
  -xsl stylesheet   xslt stylesheet

  -param name value <value> to use for parameter <name> in xslt stylesheet
                    (repeat '-param name value' for each parameter)

  -catalog          use catalog resolver for input XML and XSLT files
```

```
 [OUTPUT]
  outfile           input will be rendered as PDF into outfile
                    (use '-' for outfile to pipe output to stdout)
  -pdf outfile      input will be rendered as PDF (outfile req'd)
  -pdfa1b outfile   input will be rendered as PDF/A-1b compliant PDF
                    (outfile req'd, same as "-pdf outfile -pdfprofile PDF/A-1b")
  -awt              input will be displayed on screen
  -rtf outfile      input will be rendered as RTF (outfile req'd)
  -pcl outfile      input will be rendered as PCL (outfile req'd)
  -ps outfile       input will be rendered as PostScript (outfile req'd)
  -afp outfile      input will be rendered as AFP (outfile req'd)
  -tiff outfile     input will be rendered as TIFF (outfile req'd)
  -png outfile      input will be rendered as PNG (outfile req'd)
  -txt outfile      input will be rendered as plain text (outfile req'd)
  -at [mime] out    representation of area tree as XML (outfile req'd)
                    specify optional mime output to allow the AT to be converted
                    to final format later
  -if [mime] out    representation of document in intermediate format XML (outfile
req'd)
                    specify optional mime output to allow the IF to be converted
                    to final format later
  -print            input file will be rendered and sent to the printer
                    see options with "-print help"
  -out mime outfile input will be rendered using the given MIME type
                    (outfile req'd) Example: "-out application/pdf D:\out.pdf"
                    (Tip: "-out list" prints the list of supported MIME types)
  -svg outfile      input will be rendered as an SVG slides file (outfile req'd)
                    Experimental feature - requires additional fop-sandbox.jar.

  -foout outfile    input will only be XSL transformed. The intermediate
                    XSL-FO file is saved and no rendering is performed.
                    (Only available if you use -xml and -xsl parameters)


 [Examples]
  fop foo.fo foo.pdf
  fop -fo foo.fo -pdf foo.pdf (does the same as the previous line)
  fop -xml foo.xml -xsl foo.xsl -pdf foo.pdf
  fop -xml foo.xml -xsl foo.xsl -foout foo.fo
  fop -xml - -xsl foo.xsl -pdf -
  fop foo.fo -mif foo.mif
  fop foo.fo -rtf foo.rtf
  fop foo.fo -print
  fop foo.fo -awt
```

PDF encryption is only available if FOP was compiled with encryption support **and** if compatible encryption support is available at run time. Currently, only the JCE is supported. Check the Details.

3.2. Writing your own script

FOP's entry point for your own scripts is the class `org.apache.fop.cli.Main`. The general pattern for the command line is: `java -classpath <CLASSPATH> org.apache.fop.cli.Main <arguments>`. The arguments consist of the options and infile and outfile specifications as shown above for the standard scripts. You may wish to review the standard scripts to make sure that you get your environment properly configured.

3.3. Running with java's -jar option

As an alternative to the start scripts you can run `java -jar path/to/build/fop.jar <arguments>`, relying on FOP to build the classpath for running FOP dynamically, see below. If you use hyphenation, you must put `fop-hyph.jar` in the `lib` directory.

You can also run `java -jar path/to/fop.jar <arguments>`, relying on the `Class-Path` entry in the manifest file. This works if you put `fop.jar` and all jar files from the `lib` directory in a single directory. If you use hyphenation, you must also put `fop-hyph.jar` in that directory.

In both cases the arguments consist of the options and infile and outfile specifications as shown above for the standard scripts.

3.4. FOP's dynamical classpath construction

If FOP is started without a proper classpath, it tries to add its dependencies dynamically. If the system property `fop.home` contains the name of a directory, then FOP uses that directory as the base directory for its search. Otherwise the current working directory is the base directory. If the base directory is called `build`, then its parent directory becomes the base directory.

FOP expects to find `fop.jar` in the `build` subdirectory of the base directory, and adds it to the classpath. Subsequently FOP adds all `jar` files in the lib directory to the classpath. The lib directory is either the `lib` subdirectory of the base directory, or, if that does not exist, the base directory itself.

If the system property `fop.optional.lib` contains the name of a directory, then all `jar` files in that directory are also added to the classpath. See the methods `getJARList` and `checkDependencies` in `org.apache.fop.cli.Main`.

4. Using Xalan to Check XSL-FO Input

FOP sessions that use -xml and -xsl input instead of -fo input are actually controlling two distinct conversions: Tranforming XML to XSL-FO, then formatting the XSL-FO to PDF (or another FOP output format). Although FOP controls both of these processes, the first is included merely as a convenience and for performance reasons. Only the second is part of FOP's core processing. If a user has a problem running FOP, it is important to determine which of these two processes is causing the problem. If the problem is in the first process, the user's stylesheet is likely the cause. The FOP development team does not have resources to help with stylesheet issues, although we have included links to some useful Specifications and Books/Articles. If the problem is in the second process, FOP may have a bug or an unimplemented feature that does require attention from the FOP development team.

Note:
The user is always responsible to provide correct XSL-FO code to FOP.

In the case of using -xml and -xsl input, although the user is responsible for the XSL-FO code that is FOP's input, it is not visible to the user. To make the intermediate FO file visible, the FOP distribution includes the "-foout" option which causes FOP to run only the first (transformation) step, and write the

results to a file. (See also the Xalan command-line below)

Note:
When asking for help on the FOP mailing lists, *never* attach XML and XSL to illustrate the issue. Always run the XSLT step (-foout) and send the resulting XSL-FO file instead. Of course, be sure that the XSL-FO file is correct before sending it.

The -foout option works the same way as if you would call the Xalan command-line:

```
java org.apache.xalan.xslt.Process -IN xmlfile -XSL file -OUT outfile
```

Note that there are some subtle differences between the FOP and Xalan command-lines.

5. Memory Usage

FOP can consume quite a bit of memory, even though this has been continually improved. This is partly inherent to the formatting process and partly caused by implementation choices. All FO processors currently on the market have memory problems with certain layouts.

If you are running out of memory when using FOP, here are some ideas that may help:

- Increase memory available to the JVM. See the -Xmx option for more information.

Warning:
It is usually unwise to increase the memory allocated to the JVM beyond the amount of physical RAM, as this will generally cause significantly slower performance.

- Avoid forward references. Forward references are references to some later part of a document. Examples include page number citations which refer to pages which follow the citation, tables of contents at the beginning of a document, and page numbering schemes that include the total number of pages in the document ("page N of TOTAL"). Forward references cause all subsequent pages to be held in memory until the reference can be resolved, i.e. until the page with the referenced element is encountered. Forward references may be required by the task, but if you are getting a memory overflow, at least consider the possibility of eliminating them. A table of contents could be replaced by PDF bookmarks instead or moved to the end of the document (reshuffle the paper could after printing).
- Avoid large images, especially if they are scaled down. If they need to be scaled, scale them in another application upstream from FOP. For many image formats, memory consumption is driven mainly by the size of the image file itself, not its dimensions (width*height), so increasing the compression rate may help.
- Use multiple page sequences. FOP starts rendering after the end of a page sequence is encountered. While the actual rendering is done page-by-page, some additional memory is freed after the page sequence has been rendered. This can be substantial if the page sequence contains lots of FO elements.

6. Problems

If you have problems running FOP, please see the "How to get Help" page.