FOP Graphics Formats

1. Overview of Graphics Support

The table below summarizes the *theoretical* support for graphical formats within FOP. In other words, within the constraints of the limitations listed here, these formats *should* work. However, many of them have not been tested, and there may be limitations that have not yet been discovered or documented. The packages needed to support some formats are not included in the FOP distribution and must be installed separately. Follow the links in the "Support Thru" column for more details.

Format	Type	Support Thru
BMP (Microsoft Windows Bitmap)	bitmap	FOP native
EPS (Encapsulated PostScript)	metafile (both bitmap and vector), probably most frequently used for vector drawings	FOP native (limited support, see restrictions below)
GIF (Graphics Interchange Format)	bitmap	FOP native
JPEG (Joint Photographic Experts Group)	bitmap	FOP native
PNG (Portable Network Graphic)	bitmap	JIMI or JAI
SVG (Scalable Vector Graphics)	vector (with embedded bitmaps)	Batik
TIFF (Tag Image Format File)	bitmap	FOP native or JAI, depending on the subformat. See TIFF for more details.(JIMI also supports TIFF, but this has not been implemented within FOP).

2. Graphics Packages

2.1. FOP Native

FOP has native ability to handle some graphic file formats.

2.2. JIMI

Because of licensing issues, the JIMI image library is not included in the FOP distribution. First, download and install it. Then, copy the file "JimiProClasses.zip" from the archive to {fop-install-dir}/lib/jimi-1.0.jar. Please note that FOP binary distributions are compiled with JIMI support, so there is no need for you to build FOP to add the support. If jimi-1.0.jar is installed in the right place, it will automatically be used by FOP, otherwise it will not.

2.3. JAI (Java Advanced Imaging API)

Warning:
JAI support is available for Release 0.20.5 and later. The comments in this section do not apply to releases earlier than 0.20.5.

FOP has been compiled with JAI support, but JAI is not included in the FOP distribution. To use it, install JAI, then copy the jai_core.jar and the jai_codec.jar files to {fop-install-dir}/lib. JAI is much faster than JIMI, but is not available for all platforms. See What platforms are supported? on the JAI FAQ page for more details.

2.4. Batik

Current FOP distributions include a distribution of the Apache Software Foundation's Batik version 1.5beta4. It is automatically installed with FOP. Because Batik's API changes frequently, it is highly recommended that you use the version that ships with FOP, at least when running FOP.

Warning:
Batik must be run in a graphical environment.

Batik must be run in a graphical environment. It uses AWT classes for rendering SVG, which in turn require an X server on Unixish systems. If you run a server without X, or if you can't connect to the X server due to security restrictions or policies (a so-called "headless" environment), SVG rendering will fail.

Here are some workarounds:

- If you are using JDK 1.4, start it with the `-Djava.awt.headless=true` command

line option.
- Install an X server which provides an in-memory framebuffer without actually using a screen device or any display hardware. One example is Xvfb.
- Install a toolkit which emulates AWT without the need for an underlying X server. One example is the PJA toolkit, which is free and comes with detailed installation instructions.

3. BMP

FOP native support for BMP images is limited to the RGB color-space.

4. EPS

FOP provides support for two output targets:

- PostScript (full support).
- PDF (partial support). Due to the lack of a built-in PostScript interpreter, FOP can only embed the EPS file into the PDF. Acrobat Reader will not currently display the EPS (it doesn't have a PostScript interpreter, either) but it will be shown correctly when you print the PDF on a PostScript-capable printer. PostScript devices (including GhostScript) will render the EPS correctly.

Other output targets can't be supported at the moment because FOP lacks a PostScript interpreter.

5. JPEG

FOP native support of JPEG does not include all variants, especially those containing unusual color lookup tables and color profiles. If you have trouble with a JPEG image in FOP, try opening it with an image processing program (such as Photoshop or Gimp) and then saving it. Specifying 24-bit color output may also help. For the PDF and PostScript renderers most JPEG images can be passed through without decompression. User reports indicate that grayscale, RGB, and CMYK color-spaces are all rendered properly.

6. PNG

If using JAI for PNG support, only RGB and RGBA color-spaces are supported for FOP rendering.

7. SVG

7.1. Introduction

FOP uses Batik for SVG support. This format can be handled as an `fo:instream-foreign-object` or in a separate file referenced with `fo:external-graphic`.

Note:
Batik's SVG Rasterizer utility may also be used to convert standalone SVG documents into PDF. For more information please see the SVG Rasterizer documentation on the Batik site.

7.2. Placing SVG Graphics into PDF

The SVG is rendered into PDF by using PDF commands to draw and fill lines and curves. This means that the graphical objects created with this remain as vector graphics.

There are a number of SVG things that cannot be converted directly into PDF. Parts of the graphic such as effects, patterns and images are inserted into the PDF as a raster graphic. The resolution of this graphic may not be ideal depending on the FOP dpi (72dpi) and the scaling for that graphic. We hope to improve this in the future.

Currently transparency is not supported in PDF so many svg images that contain effects or graphics with transparent areas will not be displayed correctly.

7.3. Placing SVG Text into PDF

If possible, Batik will use normal PDF text when inserting text. It does this by checking if the text can be drawn normally and the font is supported. This example svg text.svg / text.pdf shows how various types and effects with text are handled. Note that tspan and outlined text are not yet implemented.

Otherwise, text is converted and drawn as a set of shapes by batik, using the stroking text painter. This means that a typical character will have about 10 curves (each curve consists of at least 20 characters). This can make the pdf files large and when the pdf is viewed the viewer does not normally draw those fine curves very well (turning on Smooth Line Art in the Acrobat preferences will fix this). If the text is inserted into the PDF using the inbuilt text commands for PDF it will use a single character.

For PDF output, there is a configuration option to force SVG text to be rendered as text. The drawback to this approach is that it is effective only for available fonts (including embedded fonts). Font sizes are rounded to the next integer point size. This will be improved in the future.

Note that because SVG text can be rendered as either text or a vector graphic, you may need to consider settings in your viewer for both. The Acrobat viewer has both "smooth line art" and "smooth text" settings that may need to be set for SVG images to be displayed nicely on your screen (see Edit / Preferences / Display). This setting will not affect the printing of your document, which should be OK in any case, but will only affect the quality of the screen display.

7.4. Scaling

Currently, SVG images are rendered with the dimensions specified *in the SVG file*, within the viewport specified in the fo:external-graphic element. For everything to work properly, the two should be equal. The SVG standard leaves this issue as an implementation detail. FOP will probably implement a scaling mechanism in the future.

7.5. Known Problems

- soft mask transparency is combined with white so that it looks better on pdf 1.3 viewers but this causes the soft mask to be slightly lighter or darker on pdf 1.4 viewers
- there is some problem with a gradient inside a pattern causing a pdf error when viewed in acrobat 5
- text is not always handled correctly, it may select the wrong font especially if characters have multiple fonts in the font list
- more pdf text handling could be implemented It could draw the string using the attributed character iterator to handle tspans and other simple changes of text.
- JPEG images are not inserted directly into the pdf document This area has not been implemented yet since the appropriate method in batik is static
- Uniform transparency for images and other svg elements that are converted into a raster graphic are not drawn properly in PDF. The image is opaque.

8. TIFF

FOP-native TIFF support is limited to PDF and PostScript output only. Also, according to user reports, FOP's native support for TIFF is limited to images with the following characteristics (all must be true for successful rendering):

- single channel images (i.e., bi-level and grayscale only)
- uncompressed images, or images using CCITT T.4, CCITT T.6, or JPEG compression
- images using white-is-zero encoding in the TIFF PhotometricInterpretation tag

JAI: Supports RGB and RGBA only for FOP rendering.

9. Graphics Resolution

Some bitmapped image file formats store a dots-per-inch (dpi) or other resolution value. Since PDF and most output formats do not have a concept of resolution, but only of absolute image units (i.e. pixels) FOP ignores the resolution values as well. Instead, FOP uses the dimensions of the image as specified in the fo:external-graphic element to render the image:

- If no dimensions are given, FOP uses a default value of 72 dpi to compute the graphic's dimensions. For example, suppose a graphic 300 pixels wide and 400 pixels high. FOP will render the graphic at 4.167 inches wide, 5.555 inches high, with an apparent resolution of 72 dpi.
- If only one dimension is given, FOP by default uses the same aspect ratio to compute the other dimension (to avoid the appearance of stretching). For example, suppose a graphic 300 pixels wide and 400 pixels high, for which content-width = ".5in". FOP will compute the content-height = .667 inches, and will render the graphic at that size, with an apparent resolution of 600 dpi.
- If both dimensions are given, FOP simply renders the image in that space. For example, suppose a graphic 300 pixels wide and 400 pixels high, for which content-width = "3in" and content-height = "4in". FOP will render the graphic at that size, with an apparent resolution of 100 dpi.

If you need a higher apparent output resolution for bitmapped images, first make sure that at least one dimension of the image is defined in your XSL-FO input. Apart from that, resolution problems are in the image file itself, and must be corrected there: use or create a higher-resolution image file.

Note:
The explanation above describes only the basic default behavior. There are other attributes of the fo:external-graphic element that can affect the behavior described above.

10. Image caching

FOP caches images between runs. The URL is used as a key to identify images which means that when a particular URL appears again, the image is taken from the cache. If you have a servlet that generates a different image each time it is called with the same URL you need to use a constantly changing dummy parameter on the URL to avoid caching.

Currently, the images are not automatically released when an OutOfMemoryError is imminent. The image cache can grow to a considerable size over time when a lot of different URLs are in use. Starting with version 0.20.5 you can call `org.apache.fop.image.FopImageFactory.resetCache()` to manually empty the cache. Image caching will be improved as part of our redesign effort.