Apache FOP: Graphics Formats

$Revision: 706570 $

Table of contents

1. Introduction

After the Apache FOP 0.94 release, the image handling subsystem has been rewritten in order to improve the range of supported images and image subtypes, to lower the overall memory consumption when handling images, to produce smaller output files and to increase the performance in certain areas. Of course, this causes a few changes most of which the user will probably not notice. The most important changes are:

- The image libraries Jimi and JAI are no longer supported. Instead, Apache FOP uses the Image I/O API that was introduced with Java 1.4 for all bitmap codecs.
- Some bitmap images are no longer converted to a standardized 24 bit RGB image but are instead handled in their native format.
- A plug-in mechanism offers a possibility to add support for new formats without changing the FOP's source code.

The actual image loading framework no longer resides in Apache FOP, but was instead placed in XML Graphics Commons.

2. 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 Through" columns for more details.

Format	Type	Support Through		
		Apache FOP (native)	Apache Batik	Image I/O
BMP (Microsoft Windows Bitmap)	bitmap			X [1]
EMF (Windows Enhanced Metafile)	vector (with embedded bitmaps)	(X)		
EPS (Encapsulated PostScript)	metafile (both bitmap and vector), most frequently used for vector drawings	(X)		
GIF (Graphics Interchange Format)	bitmap			X
JPEG (Joint	bitmap	(X)		X

Photographic Experts Group)				
PNG (Portable Network Graphic)	bitmap			X
SVG (Scalable Vector Graphics)	vector (with embedded bitmaps)		X	
TIFF (Tag Image Format File)	bitmap	(X)		X [1]
WMF (Windows Metafile)	vector (with embedded bitmaps)		(X)	

Legend:

- "(X)" means restricted support. Please see the details below.
- [1]: Requires the presence of JAI Image I/O Tools (or an equivalent Image I/O compatible codec) in the classpath. JAI Image I/O Tools also adds support for JPEG 2000, WBMP, RAW and PNM. Other Image I/O codecs may provide support for additional formats.

Note:

JAI Image I/O Tools is not the same as the JAI library! The former simply exposes JAI's codecs using the Image I/O API but does not include all the image manipulation functionality.

2.1. Map of supported image formats by output format

Not all image formats are supported for all output formats! For example, while you can use EPS (Encapsulated PostScript) files when you generate PostScript output, this format will not be supported by any other output format. Here's an overview of which image formats are supported by which output format:

Image Format	PDF	PostScript	Java2D, PNG, TIFF, AWT	PCL	AFP	RTF
BMP (Microsoft Windows Bitmap)	X	X	X	X	X	X
EMF (Windows Enhanced Metafile)						X [1]
EPS (Encapsulate PostScript)		X [1]				

GIF (Graphics Interchange Format)	X	X	X	X	X	X
JPEG (Joint Photographic Experts Group)	X [1]	X [1]	X	X	X [1]	X
PNG (Portable Network Graphic)	X	X	X	X	X	X
SVG (Scalable Vector Graphics)	X	X	X	X	X	X
TIFF (Tag Image Format File)	X [2]	X [2]	X	X	X [2]	X
WMF (Windows Metafile)	X	X	X	X	X	X

Legend:

- [1]: Supported without the need to decode the image.
- [2]: Supported without the need to decode the image, but only for certain subtypes.

3. Graphics Packages

3.1. XML Graphics Commons Native

XML Graphics Commons supports a number of graphic file formats natively as basic functionality: all bitmap formats for which there are Image I/O codecs available (JPEG, PNG, GIF, TIFF, etc.), EPS and EMF.

3.2. FOP Native

FOP has no native image plug-ins for the image loading framework of its own but currently hosts the Batik-dependent SVG and WMF plug-ins until they can be moved to Apache Batik.

3.3. Apache Batik

Apache Batik will later receive the SVG and WMF plug-ins for the image loading framework that are

currently hosted inside FOP.

Current FOP distributions include a distribution of the Apache Batik. 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:

- Start Java 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.

3.4. Image I/O

The image loading framework in XML Graphics Commons provides a wrapper to load images through the JDK's Image I/O API (JSR 015). Image I/O allows to dynamically add additional image codecs. An example of such an add-on library are the JAI Image I/O Tools available from Sun.

4. Details on image formats

4.1. BMP

BMP images are supported through an Image I/O codec. There may be limitations of the codec which are outside the control of Apache FOP.

4.2. EMF

Windows Enhanced Metafiles (EMF) are only supported in RTF output where they are embedded without decoding.

4.3. EPS

Apache FOP allows to use EPS files when generating PostScript output only.

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

Furthermore, FOP is currently not able to parse the preview bitmaps sometimes contained in EPS files.

4.4. GIF

GIF images are supported through an Image I/O codec. Transparency is supported but not guaranteed to work with every output format.

4.5. JPEG

FOP native support (i.e. the handling of undecoded images) 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. However, for other output formats, the JPEG images have to be decompressed. Tests have shown that there are some limitation in some Image I/O codecs concerning images in the CMYK color space. Work-arounds are in place but may not always work as expected.

4.6. PNG

PNG images are supported through an Image I/O codec. Transparency is supported but not guaranteed to work with every output format.

4.7. SVG

4.7.1. Introduction

FOP uses Apache 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.

4.7.2. 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. The same applies to PostScript output. For other output formats the SVG graphic may be converted to a bitmap image.

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 these raster images can be controlled through the "target resolution" setting in the configuration.

Currently transparency is limited in PDF so many SVG images that contain effects or graphics with transparent areas may not be displayed correctly.

4.7.3. Placing SVG Text into PDF and PostScript

If possible, Batik will use normal PDF or PostScript 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 output files large and when it is viewed the viewer may not normally draw those fine curves very well (In Adobe Acrobat, turning on "Smooth Line Art" in the preferences will fix this). Copy/paste functionality will not be supported in this case. If the text is inserted into the output file using the inbuilt text commands it will use a single character.

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.

4.7.4. 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. Additional scaling options are available through XSL-FO means.

If you use pixels to specify the size of an SVG graphic the "source resolution" setting in the configuration will be used to determine the size of a pixel. The use of pixels to specify sizes is discouraged as they may be interpreted differently in different environments.

4.7.5. 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 which may cause 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.
- 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.

4.8. TIFF

FOP can embed TIFF images without decompression into PDF, PostScript and AFP if they have either CCITT T.4, CCITT T.6, or JPEG compression. Otherwise, a TIFF-capable Image I/O codec is necessary for decoding the image.

There may be some limitation concerning images in the CMYK color space.

4.9. WMF

Windows Metafiles (WMF) are supported through classes in Apache Batik. At the moment, support for this format is experimental and may not always work as expected.

5. Graphics Resolution

Some bitmapped image file formats store a dots-per-inch (dpi) or other resolution values. FOP tries to use this resolution information whenever possible to determine the image's intrinsic size. This size is used during the layout process when it is not superseded by an explicit size on fo:external-graphic (content-width and content-height properties).

Please note that not all images contain resolution information. If it's not available the source resolution set on the FopFactory (or through the user configuration XML) is used. The default here is 72 dpi.

Bitmap images are generally embedded into the output format at their original resolution (as is). No resampling of the image is performed. Explicit resampling is on our wishlist, but hasn't been implemented, yet. Bitmaps included in SVG graphics may be resampled to the resolution specified in the "target resolution" setting in the configuration if SVG filters are applied. This can be used as a work-around to resample images in FO documents.

6. Page selection for multi-page formats

Some image formats such as TIFF support multiple pages/sub-images per file. You can select a particular page using a special URI fragment in the form: <uri>#page=<nr> (for example: `http://localhost/images/myimage.tiff#page=3`)

7. Image caching

FOP caches images between runs. There is one cache per FopFactory instance. The URI is used as a key to identify images which means that when a particular URI 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 URI you need to use a constantly changing dummy parameter on the URI to avoid caching.

The image cache has been improved considerably in the redesigned code. Therefore, resetting the image cache should be a thing of the past. If you still experience OutOfMemoryErrors, please notify us.

If all else fails, the image cache can be cleared like this: `fopFactory.getImageManager().getCache().clearCache();`