FontLibrary

Here we discuss the problem of using java.awt.Font within a restricted environment, and the requirements on both FOP and Batik to resolve this.

java.awt.Font provides a map of character sequences to glyph sequences, together with the means to render the glyph sequence to a Java2D graphic object. The class coordinates with the JVM to load font data from OS configured fonts (so-called 'system fonts'). Unfortunately this mechanism has a few short falls: Managing system fonts becomes problematic within a Cloud-like environment#. System fonts must be available to all JVM hosts increasing deployment complexities. Further to this, the JVM is only guaranteed to support PS Type 1 and TrueType font formats and as such awt.Font only caters for these formats. FOP supports a wider set of font formats (for example AFP) and so a more general purpose font representation is required.

FOP uses font data for performing text layout and font embedding, and in both cases font parsing is handled without the AWT. Batik also performs some custom font handling. Both projects independently parse TrueType fonts suggesting another need for a common font library.

A solution will provide a font source neutral font abstraction that can be shared by both FOP and Batik. In traditional environments, the AWT system may be utilised but a centralised mechanism should be available that allows the configuration of alternative font loading schemes. Alternative loading strategies could also leverage awt.Font since the createFont method reads font data from an InputStream that could be provisioned in a manner that supports restricted IO and a more granular access control.

Since Batik heavily relies on Java2D to render SVG, we will still have to use AWT fonts. It should be possible to create subclasses of java.awt.Font that will encapsulate other fonts formats (TrueType for embedding in PDF by FOP, AFP, etc.).

Batik

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="bdc8996b-5326-4e4c-9406-0b2a5bdd700e"><ac:plain-text-body><![CDATA[	**File [Method]**	**Description**	]]></ac:plain-text-body></ac:structured-macro>
gvt/font/FontFamilyResolver.java	Loads system fonts - we can make this depend upon an configurable font-loading component		
gvt/font/AWTGVTFont.java AWTGVTFont(Font, double)	Controlled - only called in StrokingTextPainter.createModifiedACIForFontMatching - we plan to bypass this		
gvt/font/AWTGVTFont.java AWTGVTFont(String, int, int)	Controlled - called in FontFamilyResolver		
gvt/font/AWTGVTFont.java AWTGVTFont(Map)	gvt/text/GlyphLayout.getFont() calls this only as a fallback. This method is called by the GlyphLayout constructor. All calls of this constructor originate from StrokingTextPainter, so that's under control. extension/svg/GlyphIterator (AttributedCharacterIterator, !GVTGlyphVector), extension/svg/GlyphIterator.nextChar: ultimately called only by StrokingTextPainter, so should be under control		
bridge/FontFace.java getFontFamily (BridgeContext, ParsedURL)	Font.createFont(Font.TRUETYPE_FONT, InputStream!) - will have to be modified to make use of the URI resolution		
WMF to SVG: used by FOP to render WMF images			
transcoder/wmf/tosvg/WMFHeaderProperties.java readRecords	creates an AWT font out of the font properties stored in the WMF file; Will have to be adapted to the new font infrastructure (although not referenced in Batik code)		
transcoder/wmf/tosvg/WMFPainter.java	no way to pass a custom font resolver (spi ImageConverter thingie). Might have to leave it there for now		
transcoder/wmf/tosvg/WMFFont.java	just passed an AWT Font whose creation is controlled in the two methods above		
transcoder/wmf/tosvg/AbstractWMFPainter.java getAttributedString()	font obtained from possibly passed Graphics2D. Normally the font has been set earlier to a controlled value		
The following classes have been moved to XGC			
ext/awt/g2d/GraphicContext.java	Font member: default font will have to be obtained from font sub-system		
ext/awt/g2d/DefaultGraphics2D.java getFontMetrics()	may have to bypass fmg and get the metrics directly from the font		
In XGC			
java2d/GraphicContext.java	default font will have to be obtained from font sub-system, or systematically overridden by calling setFont. The constructor is called at 17 different places, which may involve quite some work		
java2d/DefaultGraphics2d.java	dodgy things with fmg but this class is not used anywhere		

The following classes use java.awt.Font, but can be left as is

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="1cdda35f-ccc5-4422-8136-93b73a60a498"><ac:plain-text-body><![CDATA[	**File [Method]**	**Description**	]]></ac:plain-text-body></ac:structured-macro>
svggen API for Java 2D -> SVG Dom			
svggen/SVGFont.java	Creates an SVG font from an awt. Font		
svggen/SVGGeneratorContext.java			
svggen/SVGGraphics2D.java	Used to output SVG only, does not need to be adapted		
Used with Swing API			
apps/svgbrowser/JSVGViewerFrame.java			

apps/svgbrowser/PreferenceDialog.java	
apps/svgbrowser/Main.java	
util/gui/MemoryMonitor.java	
util/gui/xmleditor/XMLContext.java	
util/gui/xmleditor/XMLEditorKit.java	
util/gui/xmleditor/XMLTextEditor.java	Used for swing, does not need to be adapted
util/PreferenceManager.java	

FOP

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="7a8070f6-66bd-4a19-882c-d196379b91c0"><ac:plain-text-body><![CDATA[	**File [Method]**	**Description**	]]></ac:plain-text-body></ac:structured-macro>
afp/AFPGraphics2D.java	Implements Graphics2D for generating GOCA (e.g rendering SVG)		
afp/svg/AFPTextHandler.java drawString	gets a FOP font matching an AWT font. Needs change.		
fonts/FontInfo.java getFontInstanceForAWTFont()	should disappear		
Java2D Renderers: inherently dependent on AWT fonts, normally no changes necessary			
render/java2d/CustomFontMetricsMapper.java			
render/java2d/InstalledFontCollections.java			
render/java2d/Java2DFontMetrics.java			
render/java2d/Java2DGraphicsState.java			
PCL Renderer: normally nothing to do, PCL uses AWT fonts			
render/ps/NativeTextHandler.java	gets a FOP font matching an AWT font. Needs change.		
svg/PDFDocumentGraphics2D.java drawString()	AWT font used when stroking is needed. Can be left as is		
svg/PDFGraphics2D.java drawString()	drawString(): gets a FOP font matching an AWT font. Needs change; getFontMetrics: used only by Java2D renderers		
svg/SVGUtilities	not used anywhere		