

Configuring fonts in FOP and the DITA Open Toolkit

WHITE PAPER



David J. Kelly
Project Manager/
Technical Consultant

The DITA Open Toolkit includes an installation of FOP, a tool used to convert DITA content to PDF. A user must configure both FOP and the Open Toolkit to use any other fonts beyond a few basic defaults. This paper supplements the Apache instructions for configuring FOP and provides additional information for integration of FOP with the Open Toolkit.

A box of beautifully enticing promise

Creating a documentation production system from a new DITA Open Toolkit reminds me of my son's new Lego kit. Viewing it on the shelf at the store, he was enticed by the possibilities—it was a challenge (which he enjoys), and it would yield a beautiful product when finished.

The Lego Taj Mahal is a massive kit. On the shelf, the box was huge, with attractive color photographs of the finished project. The cashier smiled when we brought up the empty display box. He said he would bring one out of the back. What he brought was the same size, but it was a brown cardboard box and must have weighed forty pounds. We paid for it, and then we had to lug it all over the mall and back to the car. Buyer's remorse was already starting.

At home, my son opened the cardboard box and dumped out a jumble of tiny parts. Surely no graceful order could come of this (Figure 1).

Figure 1: The pieces



The extended analogy

Unlike the Taj Mahal Lego kit, the DITA Open Toolkit arrives in functional form (at least technically speaking). The default Open Toolkit produces generic output in HTML, PDF, and other formats. However, in my years working with the DITA Open Toolkit, I have yet to see any company accept the standard output. Everyone needs and wants changes to the DITA Open Toolkit. And once you open that box, the pieces spill to the floor like a Legomaniac's fondest dreams.

But with modern technology and expert advice, the Emperor Shah Jahan could have created the Taj Mahal in a fraction of time. So—what about those fonts?

The story on PDF files and fonts

By default, FOP and the Open Toolkit only recognize a few fonts (basically, serif, sans serif, and monospace). If you want to use others, you need to configure both FOP and the Open Toolkit to use them.

NOTE Each PDF rendering engine has different font configuration requirements. The following discussion is for creating PDF files using the FOP renderer bundled with the DITA Open Toolkit.

Configuration of fonts for FOP is described on the [Apache web pages for FOP](#). At Scriptorium, we have found that these instructions, while good for the standalone FOP, do not address all the issues for FOP with the DITA Open Toolkit.

FOP is not initially configured to recognize specific fonts. It defaults to a font-family value of “any.” According to the Apache FOP web site, “any” is mapped to the Times family. In my case, Times was not available, so my initial output was sans serif.

For the DITA Open Toolkit, customizing font usage requires changes in three places:

- ❖ The build file in the DITA Open Toolkit that starts the FOP processing
- ❖ The FOP user configuration file
- ❖ The XSL-FO stylesheets that specify what fonts to use for different parts of the document

Configuring the DITA Open Toolkit build file for FOP

FOP is bundled with the DITA Open Toolkit in the `demo/fo/fop` folder (in version 1.4.3 and later). Although there is a parameter in the Ant build files to reference the location of FOP's user configuration file, the parameter is optional, and has no value by default. Configuration settings for FOP are stored in the `fop.xconf` file. FOP supplies a default `fop.xconf` file, but the DITA Open Toolkit does not reference it, and no configuration file is used by default. To address this, you must add a parameter that specifies the location of the FOP configuration file.

Here's what you do:

1. Verify the location of the FOP configuration file. In DITA Open Toolkit 1.4.3 and (so far) in DITA Open Toolkit 1.5, this is the path and filename:

demo/fo/fop/conf/fop.xconf

NOTE: In earlier versions of the OT, fop.xconf file is not included. It can be added. I would recommend a new, full installation of FOP for earlier versions of the DITA Open Toolkit, or use DITA Open Toolkit 1.4.3 if possible. Integration of a new install of FOP with the DITA Open Toolkit is beyond the scope of this document. You could add the fop.xconf file somewhere and continue to use the instructions in this document. The location of the fop.xconf file will be specified in the Ant build files as described later in step 4.

2. In the DITA Open Toolkit's root directory, open the build_dita2pdf.xml file in a text editor or an XML editor.
3. At the top of the file, find these two lines (lines 7 and 8 in Open Toolkit 1.4.3):

```
<project name="dita2pdf" default="dita2pdf">
<property name="transtype" value="pdf"/>
```

4. Add a new property after the two lines shown in step 3:

```
<property name="args.fo.userconfig" value="${basedir}/demo/fo/fop/conf/fop.xconf"/>
```

NOTE: You may have to adjust the path in the value attribute to match the path to your copy of fop.xconf. The parameter `${basedir}` is the absolute path location of the DITA Open Toolkit you are using, so that's where your path should start.

5. Save and close the file. Now the Open Toolkit directs FOP to read your changes in the FOP configuration file. Changes to the FOP configuration file are described in the next section.
6. To verify that the configuration file is now configured to work with the Open Toolkit, use the Open Toolkit to create PDF output from DITA source files. In the output messages from the build, you should find the entry for the dita.fo2pdf.userconfig Ant task. Following the line that says "dita.fo2pdf.userconfig" should be additional entries that begin with "[fop]." For example, the first few lines of my output for the dita.fo2pdf.userconfig task looked like this:

```
dita.fo2pdf.userconfig:
[fop] Oct 2, 2009 2:11:57 PM org.apache.fop.apps.FopFactoryConfigurator configure
[fop] INFO: Default page-height set to: 11in
[fop] Oct 2, 2009 2:11:57 PM org.apache.fop.apps.FopFactoryConfigurator configure
[fop] INFO: Default page-width set to: 8.26in
```

Defining fonts in the FOP configuration file

The description of fonts in the FOP configuration file is documented in the [FOP fonts page](#). The remaining discussions in this document assume that you have read that page and have familiarized yourself with the concepts and procedures on it.

Automated font registration

Font registration is the process by which FOP reads information about fonts so it can describe the fonts properly as it creates PDF. The information FOP needs about a font are stored in the font name, font weight, and font style attributes. These values are referred to as the font-triplet. FOP also needs to know the location of the font file and, if a font metrics file is needed, the location of the font metrics file.



Automated registration of fonts makes it difficult for you to reference the fonts accurately, so this method is not recommended.

Creating font metrics files (not needed for FOP 0.94 and later)

In FOP 0.94 and later, you do not need to supply metrics files to register fonts in most cases. For FOP 0.94 and later, you can skip to the next section, Individual registration of fonts.

The instructions on the Apache web page are a little sketchy on this part of the installation. For example, to create the metrics files, you are instructed to run the PFMReader for Type 1 fonts and the TTFReader for TrueType fonts using a command line for the PFMReader. The example command line shown on the Apache web site is not exactly right.

I found that I had to list all (or almost all) of the .jar files in the fop/lib directory in the -cp value for this command line rather than just the few specified by the web site. Also, the names of the .jar files shown in the example are generic names. The real names include the version numbers of the jar files.

The Apache instructions assume that you are running the command line from the fop directory.

Once I made these adjustments, my command line for generating the Palatino Linotype metrics file was as follows:

```
java -cp "build/fop.jar;lib/avalon-framework-4.2.0.jar;lib/commons-logging-1.0.4.jar;lib/commons-io-1.3.1.jar;lib/serializer-2.7.0.jar;lib/xmlgraphics-commons-1.3.1.jar" org.apache.fop.fonts.apps.TTFReader C:/Windows/Fonts/pala.ttf C:/fontmetrics/ttfpala.xml
```

In this example, the command reads the font file located at C:/Windows/Fonts/pala.ttf and creates a font metrics file at C:/fontmetrics/ttfpala.xml.

Individual registration of fonts

To register a font in the FOP configuration file, you use multiple elements inside the <fonts> element, as instructed on the Apache web site. In addition to the information on the Apache web site, there are a couple of other points to consider:

- ❖ In FOP 0.94 and above, you do not need the metrics-url attribute for most fonts.
- ❖ The Apache web site's example for the use of relative references is a little misleading. From the Apache examples, it appears that specifying <font-base>.</font-base> in the <fop> element will cause FOP to use the directory containing the file as the base location for relative references. If this were true, I should be able to put the metrics file in the conf folder with only the filename in the metrics-url attribute. I could not get that approach to work.

However, I could get FOP to recognize a relative path outside the fop folder. For example, I specified this for my installation of FOP:

```
<font-base>../../../../../../../../Windows/Fonts</font-base>
```

When I established this path as the relative path for fonts, I could enter the filename for a font file in the embed-url attribute without having to specify the entire path to the file. Following is an example for the Eras Bold font:

```
<font kerning="yes" embed-url="erasbd.ttf">
```

```
<font-triplet name="ErasB" style="normal" weight="normal"/>
</font>
```

You can also set the font base URL when executing FOP as an embedded application by using `fopFactory.setFontBaseURL`. That option is documented in the Apache pages.

- ❖ The name attribute in the font-triplet element does not have to be the name of the font in the font file or the font metrics file. In fact, you may not want it to be.

For reference, the `` element is structured like this:

```
<font kerning="yes" embed-url="file:///C:/myfonts/FTL____.pfb">
  <font-triplet name="FrutigerLightItalic" style="italic" weight="normal"/>
</font>
```

Note in the preceding sample that the font is `FrutigerLightItalic` and the font-style is `italic`. Suppose your XSL-FO code has `<p>` tags generating the FO element `<fo:block font-family="FrutigerLight">`. Then, within a given paragraph, the author inserts a `<varname>` or an `<i>` element. In the default XSL-FO implementation of the DITA Open Toolkit, `<varname>` or `<i>` creates an inline element like this:

```
<fo:inline font-style="italic">
```

Because no font-family name or font-weight is supplied, the `fo:inline` element takes the current defaults:

```
name="FrutigerLight" style="italic" weight="normal"
```

This combination of values does not match the font-triplet combination specified in the preceding font-triplet example (`<font-triplet name="FrutigerLightItalic" style="italic" weight="normal"/>`). The results would not be what you expect.

One solution is to name all style variants of `FrutigerLight`, `FrutigerBold`, and so on, just `Frutiger` (or `Fred` for that matter), regardless of the official font name in the font file. Then the font variants can be distinguished by the font-weight and font-style attributes, both in the configuration file and the XSL-FO. In general, the naming of fonts with the font-triplet can be tricky—familiarity with the XSL-FO stylesheets in the DITA Open Toolkit is helpful.

- ❖ When you supply `font-weight="bold"` in the XSL-FO, it is sometimes reported to FOP as `font-weight="700"`, and `font-weight="light"` is reported to FOP as `font-weight="200"`. In these cases you may need to specify two font-triplet elements to cover both possibilities, as shown in the following code for a bold font:

```
<font metrics-url="sab.xml" kerning="yes"
  embed-url="file:///C:/Windows/Fonts/sab____.pfb">
  <font-triplet name="Sabon" style="normal" weight="bold"/>
  <font-triplet name="Sabon" style="normal" weight="700"/>
</font>
```

In this example, the Sabon Bold font is defined with a font-name of `Sabon` and a font-weight of `bold` or `700`. When the XSL-FO creates an object with the attributes `font-name="Sabon"`, `font-weight="bold"`, and `font-style="normal"`, FOP will recognize that the font file specified in this font element should be used.



XSL-FO stylesheets and FOP-registered fonts

With the FOP user configuration file configured and integrated with the OT, your document comes out looking pretty much the way it did before you started.

You need to update the XSL-FO stylesheets to use font-family, font-style, and font-weight attribute values that correspond to the values of the name, weight, and style in the font-triplet in the configuration file. I have already pointed out one consideration in the stylesheets for matching the font-family name, font-weight, and font-style values in “Individual registration of fonts” on page 4.

Many of the font settings are established using <attribute-set> elements in the dita2fo-parms.xml file. The <attribute-set> element defines a group of style attributes that can be used by an element. The XSL-FO code that handles a given element then refers to the <attribute-set> to apply those styles. The following example clarifies this arrangement.

In the fop.xconf file, the Eras-Bold ITF font is defined with this code:

```
<font metrics-url="ttferasbd.xml" kerning="yes"
      embed-url="file:///C:/Windows/Fonts/erasbd.ttf">
  <font-triplet name="Eras" style="normal" weight="bold"/>
</font>
```

In the dita2fo-parms.xml file, I modified the “topictitle1” attribute set to change first-level headings as follows:

```
<xsl:attribute-set name="topictitle1" >
  <xsl:attribute name="break-before">page</xsl:attribute>
  <xsl:attribute name="margin-top">0.5pc</xsl:attribute>
  <xsl:attribute name="margin-bottom">0.5pc</xsl:attribute>
  <xsl:attribute name="font-size">18pt</xsl:attribute>
  <xsl:attribute name="font-family">Eras</xsl:attribute>
  <xsl:attribute name="font-style">normal</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-page">always</xsl:attribute>
</xsl:attribute-set>
```

In the dita2fo_titles.xml file, the following code already exists for processing first-level headings. It does not need to be modified because it already uses the “topictitle1” attribute set.

```
<!-- h1 -->
<xsl:template match="*[contains(@class,' topic/topic')]/*[contains(@class,' topic/title')]" priority="2">
  <fo:block xsl:use-attribute-sets="topictitle1" padding-top="1.4pc">
    <fo:block border-top-color="black" border-top-width="3pt" line-height="100%"
      border-left-width="0pt" border-right-width="0pt">
      <xsl:call-template name="get-title"/>
    </fo:block>
  </fo:block>
</xsl:template>
```

With these values in place, the XSL-FO code will create the following XSL-FO code when it encounters a title in a top-level topic; the following example shows the top `fo:block` element only, because that is the only one relevant to this discussion. The attributes from the `topic:title1` attribute-set have been substituted for the `xsl:use-attribute-sets="topic:title1"`.

```
<fo:block break-before="page"
  margin-top="0.5pc"
  margin-bottom="0.5pc"
  font-size="18pt"
  font-family="Eras"
  font-style="normal"
  font-weight="bold"
  keep-with-next.within-page="always"
  padding-top="1.6pc">
```

This is the XSL-FO code that FOP reads. FOP matches the font-family, font-style, and font-weight values in this `<fo:block>` element and finds the `` element with a matching `<font-triplet>` value. When it finds a match, it uses the font file specified in the `` element to create the PDF code.

Some PDF files may need to use 10pt Times New Roman for the `<p>` tag; others might call for 11pt Veljovic. You may have some sophisticated programming built into your stylesheets. If you made a separate DITA Open Toolkit for every type of PDF file you wanted to create, you would wind up with a lot of code duplication that would be difficult to maintain. To solve this problem, you can create a DITA Open Toolkit plug-in that contains the changes and add it to the DITA Open Toolkit's plugins directory. For guidance in this work, I recommend Scriptorium's white paper on hacking the DITA Open Toolkit.

Best wishes for implementing your fonts. And for encouragement, here's a shot of the Taj Mahal after a few hours of work (Figure 2). Yes, there's more to be done, but it appears that graceful order can indeed be imposed on the chaos.

Figure 2: Well on the way...



About the author

David Kelly has worked since 1978 as a technical writer, publications coordinator, documentation manager, software project manager, program manager, and technical consultant. Currently, he manages various Scriptorium projects and spends time working on new documentation automation solutions.

Always interested in the use of new tools to improve documentation processes and quality, David has devised several enhancements to the DITA Open Toolkit for Scriptorium's customers, from processing trademarked terms to translating Microsoft Word documents to DITA output. He also contributes to the Scriptorium blog, Palimpsest.

About Scriptorium

Scriptorium Publishing provides expert advice on how to develop, deploy, and manage content. Our typical customer has thousands of pages of information, which needs to be delivered in print, PDF files, HTML, and other media, often in dozens of languages. Our mission is to automate formatting and production tasks, usually through XML technologies, so that authors can write more efficiently.

Our consultants have experience in traditional publishing workflows, including typesetting, book design, copyfitting, and production editing. This understanding influences our approach to creating state-of-the-art publishing systems with modern tools and technologies, such as XML, HTML, DITA, the DITA Open Toolkit, XSLT, XSL-FO, FrameMaker, Ant, Perl, FrameScript, Flash, InDesign, XMetaL, oXygen, and many more.

Our customers include federal and state government as well as companies in defense, consumer electronics, telecommunications, health care, pharmaceutical, and other industries.

If you are facing a difficult publishing challenge, we want to hear from you. Contact us at info@scriptorium.com or 919-481-2701 x105.

Scriptorium Publishing is based in the Research Triangle area of North Carolina and has been in business since 1997.

Scriptorium Publishing Services, Inc.
PO Box 12761
Research Triangle Park, NC 27709-2761
USA
info@scriptorium.com
919-481-2701
www.scriptorium.com

