

# Customizing PDF output in the DITA Open Toolkit

WHITE PAPER



David J. Kelly  
Project Manager/  
Technical Consultant

**The default DITA Open Toolkit (DITA OT) provides XSL-FO stylesheets that create simple, generic PDF files. The default DITA OT PDF output is a good starting point, but it is an ugly place to stop.**

**To create better-looking PDF output with more publishing functions, you need to change XSL-FO stylesheets. What kinds of things can you do, and where do you start? This paper explains how to modify XSL-FO stylesheets in the DITA OT.**

## PDF production in the DITA OT

To create PDF files from DITA XML, the DITA OT first uses XSL stylesheets to convert the DITA markup to a form of XML called Formatting Objects (FO). FO is a markup language that describes page layout and formatting and that specifies placement of content within the formatting. The DITA OT then uses an FO processor to transform FO to PDF files.

The FO processor uses the FO input to determine the look of the PDF output. If you want to modify the PDF formatting, you need to modify the XSL stylesheets that produce FO. These stylesheets use a language called XSL-FO. In short, XSL-FO is standard XSL scripting that converts one form of XML (DITA) to another form of XML (FO).

The DITA OT is packaged with an FO processor, Formatting Object Processor (FOP). This is an open-source processor developed and maintained by the Apache XML Graphics project. You can learn more about it at <http://xmlgraphics.apache.org/fop/>. FOP is a reliable FO processor, but it does have limitations. The FOP compliance web page notes where it does not support functionality specified in the W3C XSL-FO 1.1 standard.

You can modify the DITA OT to use other FO processors with other feature sets. The following web sites provide lists of other FO processors:

- ❖ <http://www.scriptorium.com/whitepapers/dita2pdf/dita2pdf3.html>
- ❖ <http://www.sagehill.net/docbookxsl/FOprocessors.html>

Modifying the DITA OT to use another FO processor is beyond the scope of this document.



## What kinds of changes can you make?

As of release 1.5, the DITA OT uses the Idiom FO stylesheets in the `/demo/fo` directory as its default PDF transforms. This document references those stylesheets for examples.

This white paper also builds on lessons about altering the DITA OT provided in *Hacking the DITA Open Toolkit*, a Scriptorium white paper by Simon Bate (<http://www.scriptorium.com/whitepapers/hackingot/index.html>). Some of the topics discussed here assume you are familiar with that paper. You will also need a basic understanding of XSL, XSL-FO, and the DITA OT.

You can customize the DITA OT's XSL-FO stylesheets for a wide range of formatting and automation functions. This paper covers the following customizations:

- ❖ Updating the title page
- ❖ Updating headers and footers
- ❖ Handling fonts
- ❖ Altering element formatting in attribute sets
- ❖ Altering element formatting in templates
- ❖ Modifying page layouts
- ❖ Modifying chapter characteristics
- ❖ Modifying table characteristics

Possibilities for customizing the DITA OT are endless. Other characteristics you might want to modify include:

- ❖ Changing the wrapping characteristics of long lines in codeblocks
- ❖ Rotating text in table headings or other places
- ❖ Modifying note formats
- ❖ Changing to multicolumn pages or landscape pages for a topic at any level in a ditamap
- ❖ Autosizing graphics when height or width is not specified

### Customizing the Idiom plugin

You can create and use customizations to the Idiom plugin without altering the original stylesheets. This method involves placing stylesheet files with modified or added templates into the plugins directory in the DITA OT. There are numerous advantages to using this method, one of which is that it allows you to customize the DITA OT several different ways within the same instance of the toolkit files.

Before modifying the XSL-FO stylesheets, familiarize yourself with the instructions provided in the “About/Customization” section of the DITA OT's `demo/fo/readme.txt` file.

## Introduction to the book structure

In the `demo/fo/xsl/fo/root-processing.xml` file, the last template is the `rootTemplate`. The `rootTemplate` template provides the overall organization of the book:

```
<xsl:template name="rootTemplate">
  ❶ <xsl:call-template name="validateTopicRefs"/>
  ❷ <fo:root xsl:use-attribute-sets="__fo__root">
  ❸ <xsl:comment>
    <xsl:text>Layout masters = </xsl:text>
    <xsl:value-of select="$layout-masters"/>
  </xsl:comment>
  ❹ <xsl:call-template name="createLayoutMasters"/>
  ❺ <xsl:call-template name="createBookmarks"/>
  ❻ <xsl:call-template name="createFrontMatter"/>
  ❼ <xsl:call-template name="createToc"/>
  <!-- <xsl:call-template name="createPreface"/> -->
  ❽ <xsl:apply-templates/>
  ❾ <xsl:call-template name="createIndex"/>
  </fo:root>
</xsl:template>
```

The context for execution of this template is the root of your document—usually a ditamap, a bookmap, or a topic.

The first line (❶) inside the `rootTemplate` template calls a template that iterates through the `topicRefs`, if there are any, to ensure they are valid.

The second line (❷) creates the `fo:root` element, which is the top-level element for an XSL-FO document.

After a comment (❸) that records which set of layout masters is being used, several other `xsl:call-template` elements create additional portions of the XSL-FO document. The first item created (❹) is the `fo:layout-master-set`, which establishes some of the basic page formatting, such as page size, margin sizes, header and footer areas, and so forth, for different kinds of pages in the manual. “Modifying page layouts” on page 10 discusses these format characteristics.

The next `xsl:call-template` (❺) creates the bookmarks section for PDF bookmarks, which is beyond the scope of this introductory document. The third `xsl:call-template` (❻) invokes the templates that create front matter for the book. This is the call-template to follow to find the title page template. The fourth `xsl:call-template` (❼) creates the table of contents. Then, after a call to create the preface (currently commented out in the Idiom plugin), an `xsl:apply-templates` element (❽) processes the body of the document. Finally, an `xsl:call-template` (❾) creates the document index.



## Updating the title page

In the rootTemplate, notice the xsl:call-template for the createFrontMatter template. You will find the createFrontMatter template in two different files:

- ❖ front-matter.xsl
- ❖ front-matter\_1.0.xsl

When a stylesheet file name ends with “\_1.0,” this is an indication that it is used for processing DITA content that is at version 1.1 or higher. In this example, the template to change is in the front-matter\_1.0.xsl file.

The createFrontMatter template is as follows:

```
<xsl:template name="createFrontMatter">
  <xsl:choose>
    ❶ <xsl:when test="$ditaVersion >= '1.1'">
      <xsl:call-template name="createFrontMatter_1.0"/>
    </xsl:when>
    ❷ <xsl:otherwise>
      ❸ <fo:page-sequence master-reference="front-matter"
        xsl:use-attribute-sets="__force__page__count">
        ❹ <xsl:call-template name="insertFrontMatterStaticContents"/>
        ❺ <fo:flow flow-name="xsl-region-body">
          ❻ <fo:block xsl:use-attribute-sets="__frontmatter">
            ❼ <!-- set the title -->
            <fo:block xsl:use-attribute-sets="__frontmatter__title">
              ❽ <xsl:choose>
                <xsl:when test="//*[contains(@class,' bkinfo/bkinfo ')]][1]">
                  <xsl:apply-templates select="//*[contains(@class,' bkinfo/bkinfo ')]][1]/
                    *[contains(@class,' topic/title ')]/node()"/>
                </xsl:when>
                <xsl:when test="//*[contains(@class,' map/map ')]/@title">
                  <xsl:value-of select="//*[contains(@class,' map/map ')]/@title"/>
                </xsl:when>
                <xsl:otherwise>
                  <xsl:value-of select="/descendant::*[contains(@class,' topic/topic ')]][1]/
                    *[contains(@class,' topic/title ')]"/>
                </xsl:otherwise>
              </xsl:choose>
            </fo:block>
            <!-- set the subtitle -->
            ❾ <xsl:apply-templates select="//*[contains(@class,' bkinfo/bkinfo ')]][1]/*[contains(@class,'
              bkinfo/bktitlealts ')]/*[contains(@class,' bkinfo/bksubtitle ')]"/>
            ❿ <fo:block xsl:use-attribute-sets="__frontmatter__owner">
              <xsl:choose>
                <xsl:when test="//*[contains(@class,' bkinfo/bkowner ')]">
                  <xsl:apply-templates select="//*[contains(@class,' bkinfo/bkowner ')]"/>
                </xsl:when>

```

```

        <xsl:otherwise>
            <xsl:apply-templates select="$map/*[contains(@class, ' map/topicmeta ')]"/>
        </xsl:otherwise>
    </xsl:choose>
</fo:block>
</fo:block>
<!--<xsl:call-template name="createPreface"/>-->
</fo:flow>
</fo:page-sequence>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

```

The third line of this template (❶) indicates that if the version of DITA being used is 1.1 or higher, use the template called `createFrontMatter_1.0`.

In the “otherwise” portion of the first choose statement (❷), the front matter section is created with the `fo:page-sequence` element (❸). An `xsl:call-template` element (❹) invokes code that creates the headers and footers for the front-matter page sequence. (“Updating headers and footers” on page 6 describes how to modify headers and footers.)

The template creates an `fo:flow` element (❺) to tell the FO processor to flow the contents of topics into the body area of the page. Then the template inserts an `fo:block` element (❻) containing the contents of the front matter section. This `fo:block` references the `__frontmatter` attribute set, which is contained in `demo/fo/cfg/fo/attrs/front-matter-attr.xml`. The default version of this attribute set specifies one style, which centers the alignment of the front matter. You can add any other attributes that are valid for an `fo:block`, such as default fonts, borders, background colors, and so forth. For example, here is the attribute set with alignment set to “left” and with an added “border” attribute specifying a solid black, 4-point border (changes shown in bold):

```

<xsl:attribute-set name="__frontmatter">
    <xsl:attribute name="text-align">left</xsl:attribute>
    <xsl:attribute name="border">4pt solid black</xsl:attribute>
</xsl:attribute-set>

```

The next section of code (❼) contains an `fo:block` that creates the title on the title page. This block refers to the `__frontmatter__title` attribute set, contained in `front-matter-attr.xml`, which you can alter with any attribute that is valid for an `fo:block`. This attribute set is where you change the font style and size for the title, change the alignment, and change vertical placement using, for example, the `padding-top` and the `valign` attributes.

Here is the attribute set modified with a larger font size, more line leading, and a serif font in italics:

```

<xsl:attribute-set name="__frontmatter__title">
    <xsl:attribute name="margin-top">80mm</xsl:attribute>
    <xsl:attribute name="font-family">Garamond</xsl:attribute>
    <xsl:attribute name="font-size">36pt</xsl:attribute>
    <xsl:attribute name="font-weight">bold</xsl:attribute>
    <xsl:attribute name="font-style">italic</xsl:attribute>
    <xsl:attribute name="line-height">160%</xsl:attribute>
</xsl:attribute-set>

```



After the `fo:block` for the title is an `xsl:choose` statement (8) that selects the text of the title from one of the following (in order):

1. The `bkinfo/title` node, if available
2. The `map` element's `title` attribute, if available
3. The first topic's title element

If you need a different order of selection or a different location for the title text, change the code in this section.

The line of code for the subtitle (9) is outside the `fo:block` for the title, so its format falls under the `fo:block` for the entire title page. If you want additional control over the subtitle, place it in its own `fo:block` and apply formatting attributes to it. Here's an example of line 9 in which the subtitle is placed in an `fo:block` with a new attribute set defined for it. The attribute set specifies right justification, 20-point Garamond regular text, and a red rule (using the block's top border) between the subtitle and the title. (This is an example to demonstrate XSL-FO, not an example to demonstrate tasteful title page design!)

```
<fo:block use-attribute-sets="__frontmatter_subtitle">
  <xsl:apply-templates select="//*[contains(@class,' bkinfo/bkinfo ')]*[1]/*[contains(@class,' bkinfo/bktitlealts
    ')]/*[contains(@class,' bkinfo/bksubtitle ')]"/>
</fo:block>

<xsl:attribute-set name="__frontmatter_subtitle">
  <xsl:attribute name="margin-top">12pt</xsl:attribute>
  <xsl:attribute name="font-family">Garamond</xsl:attribute>
  <xsl:attribute name="font-size">20pt</xsl:attribute>
  <xsl:attribute name="border-top">2pt solid red</xsl:attribute>
  <xsl:attribute name="font-style">normal</xsl:attribute>
</xsl:attribute-set>
```

The new `__frontmatter_subtitle` attribute set should be placed in the `front-matter-attr.xml` file for consistency and future access.

After the subtitle, the template inserts another `fo:block` element (10) containing the name of the document owner. The contents of this block are selected from the `bkinfo` element, if it is available, or from the `map`'s `topicmeta` element.

If you would like other items on the title page, insert them into the flow of XSL-FO objects within the `fo:block` (6) for the title page. Examples of additional content that commonly appear on a title page include graphics and the publisher's name.

## Updating headers and footers

XSL-FO templates for headers and footers are located in the `demo/fo/xsl/fo/static-content.xml` file.

Headers and footers in XSL-FO are defined as `fo:static-content` elements. These in turn are contained in areas that hold large sections of content (or the entire document) called `fo:page-sequences`. In general, `fo:page-sequences` are associated with large parts of a book.

In `demo/fo/xsl/fo/static-content.xml`, there are several templates with names such as `insertBodyStaticContents`, `insertTocStaticContents`, and so forth. Each of these templates contains references to the specific header and footer templates used for the corresponding part of the book.

For example, the table of contents section of a book has headers and footers defined in the `insertTocStaticContents` template. This template defines which headers and footers to use for this document section as follows:

```
<xsl:call-template name="insertTocOddFooter"/>
<xsl:call-template name="insertTocEvenFooter"/>
<xsl:call-template name="insertTocOddHeader"/>
<xsl:call-template name="insertTocEvenHeader"/>
```

The `insertTocOddFooter` template, located farther down in the file, contains the following code:

```
❶ <fo:static-content flow-name="odd-toc-header">
❷ <fo:block xsl:use-attribute-sets="__toc__odd__header">
❸   <xsl:call-template name="insertVariable">
     <xsl:with-param name="theVariableID" select="'Toc odd header'"/>
     <xsl:with-param name="theParameters">
       <proddname>
         <xsl:value-of select="$productName"/>
       </proddname>
       <pagenum>
         <fo:inline xsl:use-attribute-sets="__toc__odd__header__pagenum">
           <fo:page-number/>
         </fo:inline>
       </pagenum>
     </xsl:with-param>
   </xsl:call-template>
 </fo:block>
</fo:static-content>
```

Many aspects of this code example are common to the header and footer templates. First, as mentioned before, the `fo:static-content` element (❶) is the element that contains a header or footer area. The attribute `flow-name="odd-toc-header"` uses a naming convention provided by the DITA OT that indicates which area is created (in this case, the header of an odd-numbered page in the TOC section). In general, you should not change this name.

The `fo:block` element (❷) specifies that a physical space will be created within the `fo:static-content` area. The attribute set `__toc__odd__header` provides formatting characteristics for the `fo:block`. This attribute set is in the `demo/fo/cfg/fo/attrs/static-content-attr.xml` file. The template is as follows:

```
<xsl:attribute-set name="__toc__odd__header">
  <xsl:attribute name="text-align">right</xsl:attribute>
  <xsl:attribute name="margin-right">10pt</xsl:attribute>
  <xsl:attribute name="margin-top">10pt</xsl:attribute>
</xsl:attribute-set>
```

If you want to change the text alignment to left or centered, or if you want to change the margins, this is the place to do it. You can also create other block-level attributes for the header or footer on the appropriate attribute sets. (Refer to <http://www.w3.org/TR/xsl/#d0e9451>, “Block-level Formatting Objects,” for more information about `fo:block` and its properties.)



With the attribute set, you can also create border rules or create a default font style for the text. For example, if you want the header to be centered, in 12-point Eras bold type, and with a 1-point rule along its bottom edge, the changed attribute set is as follows:

```
<xsl:attribute-set name="__toc__odd__header">
  <xsl:attribute name="text-align">center</xsl:attribute>
  <xsl:attribute name="margin-right">10pt</xsl:attribute>
  <xsl:attribute name="margin-top">10pt</xsl:attribute>
  <xsl:attribute name="border-bottom-width">1pt</xsl:attribute>
  <xsl:attribute name="border-bottom-style">solid</xsl:attribute>
  <xsl:attribute name="border-bottom-color">black</xsl:attribute>
  <xsl:attribute name="font-family">Eras</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="font-size">12pt</xsl:attribute>
</xsl:attribute-set>
```

Going further down the header template, you see an `xsl:call-template` element (❸) that specifies the remainder of the header's content. Using this template, you can only specify the product name for a predefined header template. You can change or replace the content within the `xsl:call-template` element, or you can replace the entire element with XSL-FO structures of your own to get the content and format in the header that you would like. Other programming constraints at this point are that the XSL-FO element must be valid and well-formed within an `fo:block` element, and any variables used must be active within this template.

In the body footer templates (called `odd-footer` and `even-footer`), content is placed in a table to organize the text into different positions along the bottom of the page. One common change is to alter these templates to add a third column in the middle to contain your company name or other important information.

As you look through the other templates in `static-content.xml`, you will see that similar approaches apply to the other header and footer templates.

## Handling fonts

Using fonts beyond the default fonts in the DITA OT requires configuring the DITA OT's instance of FOP (the FO processor that converts XSL-FO into PDF format). The process is documented on the Apache FOP website (<http://xmlgraphics.apache.org/fop/0.95/configuration.html>) and further clarified in Scriptorium's white paper, *Configuring fonts in FOP and the DITA Open Toolkit* ([http://www.scriptorium.com/whitepapers/fop\\_fonts/index.html](http://www.scriptorium.com/whitepapers/fop_fonts/index.html)). Please refer to these resources for information about configuring fonts in the DITA OT.

## Altering element formatting in attribute sets

Attribute sets in XSL-FO provide a convenient way to refer to a set of related attributes for a given element or elements. Multiple elements can reference an attribute set. You can change an attribute set in one place, and all the elements that reference it have the change applied to them.

Many of the attribute sets in the DITA OT reside in XSL files in the `demo/fo/cfg/fo/attrs` directory. The names of the files give some indication as to what kinds of templates they contain, but in some cases, the file naming is not easy to decipher. For example, the file that contains elements used for highlighting text, such as `<b>` for bold and `<i>` for italics, is called `hi-domain-attr.xsl`.

The most direct way of finding an attribute set that affects a given DITA element is as follows:

1. Find an XSL template that selects the given attribute. For example, using a search tool that enables you to find text in a group of files, look for section titles by searching for the following string:

```
topic/title
```

This search returns a lot of results, but from it, you can find the template you are looking for.

The relevant result for this example is as follows:

```
<xsl:template match=" *[contains(@class,' topic/section ')]/*[contains(@class,' topic/title ')]">
```

2. Examine the template and find a reference to an attribute set, if any. In the example for section titles, the template includes the following lines:

```
<fo:block xsl:use-attribute-sets="section.title" id="{@id}">
  <xsl:apply-templates/>
</fo:block>
```

The attribute set for section titles is named `section.title`.

3. When you find the attribute set name, search for it in the files. In most cases, the attribute set is in the `demo/fo/cfg/fo/attrs` directory, but there may be exceptions.
4. When you find an attribute set, you can change, add, or delete attributes for the `fo:block` element that the attribute set applies to. For example, here is the attribute set for section titles:

```
<xsl:attribute-set name="section.title">
  <xsl:attribute name="font-family">Sans</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="space-before.optimum">15pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-column">always</xsl:attribute>
</xsl:attribute-set>
```

The following example shows modifications that apply a 11-point regular Times Roman font and 6 points of additional line spacing before the text:

```
<xsl:attribute-set name="section.title">
  <xsl:attribute name="font-family">TimesRoman</xsl:attribute>
  <xsl:attribute name="font-weight">regular</xsl:attribute>
  <xsl:attribute name="space-before.optimum">11pt</xsl:attribute>
  <xsl:attribute name="keep-with-next.within-column">always</xsl:attribute>
  <xsl:attribute name="padding-before">6pt</xsl:attribute>
</xsl:attribute-set>
```

## Altering element formatting in templates

In many cases, formatting for an element is not provided in the attribute sets. Rather, it is specified within templates where the primary processing for an element takes place.



The function of attributes in a template is the same as the function of attributes in attribute sets. (If one or more attributes can be used in different templates, the attributes should go in an attribute set.)

Formatting may involve something other than an attribute. For example, you may want to format a note as a one-row table with an icon in the left cell. An example of this usage already exists in the DITA OT for notes, as follows:

```
<fo:table xsl:use-attribute-sets="note__table">
  <fo:table-column column-number="1"/>
  <fo:table-column column-number="2"/>
  <fo:table-body>
    <fo:table-row>
      <fo:table-cell xsl:use-attribute-sets="note__image__entry">
        <fo:block>
❶ <fo:external-graphic src="url({concat($artworkPrefix, $notelImagePath)})" xsl:use-attribute-sets="image"/>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell xsl:use-attribute-sets="note__text__entry">
        <xsl:call-template name="placeNoteContent"/>
      </fo:table-cell>
    </fo:table-row>
  </fo:table-body>
</fo:table>
```

The template inserts the graphic into the note at **❶** within the context of a table that it creates for the note. You might want the graphic to occur above the note text, in which case you could alter the template to place the graphic in an `fo:block` preceding the note, as follows:

```
<fo:block>
  <fo:external-graphic src="url({concat($artworkPrefix, $notelImagePath)})" xsl:use-attribute-sets="image"/>
</fo:block>
<xsl:call-template name="placeNoteContent"/>
```

## Modifying page layouts

Page layout characteristics you might want to modify include:

- ❖ Page size
- ❖ Page margins and side column width
- ❖ Number of columns
- ❖ Text indentation
- ❖ Header and footer characteristics, described in the section “Updating headers and footers” on page 6.

## Page size

Page margins are defined as global variables in the file `demo/fo/cfg/fo/attrs/basic-settings.xml`.

You can modify the existing page size and page margin variables in this file, or you can create new page size and page margin variables. (One reason for doing this is if different page sizes are required for different parts of a book.)

Following is an example of the settings changed from US Letter page size (the DITA default) to A4 page size:

```
<!-- The default of 215.9mm x 279.4mm is US Letter size (8.5x11in) -->
<xsl:variable name="page-width">210mm</xsl:variable>
<xsl:variable name="page-height">297mm</xsl:variable>
```

When you modify the existing variables, the changes affect `fo:page-sequences` whose `fo:page-sequence-master` uses an `fo:simple-page-master`. The `fo:simple-page-master` references the page size and page margin variables. For example, this snippet is from the `createDefaultLayoutMasters` template in `demo/fo/cfg/fo/layout-masters.xml`.

```
<fo:simple-page-master master-name="front-matter-first"
  page-width="{ $page-width }"
  page-height="{ $page-height }">
  <fo:region-body margin-top="{ $page-margin-top }"
    margin-bottom="{ $page-margin-bottom }"
    margin-left="{ $page-margin-left }"
    margin-right="{ $page-margin-right }"/>
</fo:simple-page-master>
```

If you create new page size variables, you can use these in new or existing `fo:simple-page-master` elements. If you create new `fo:simple-page-master` elements, you then need to update or create `fo:page-sequence-master` elements and `fo:page-sequence` elements to use the new settings.

Note that the page size and page margin variables in `demo/fo/cfg/fo/attrs/basic-settings.xml` are set in units of millimeters. The DITA OT supports a variety of units of length, including the following:

- ❖ cm (centimeters)
- ❖ mm (millimeters)
- ❖ in (inches)
- ❖ pt (points)
- ❖ pc (picas)
- ❖ px (pixels)
- ❖ em (em-space length)

You can modify the lengths and units for all of the dimension variables. For example, change the `page-width` and `page-height` to inches as follows:

```
<!-- The default of 215.9mm x 279.4mm is US Letter size (8.5x11in) -->
<xsl:variable name="page-width">8.5in</xsl:variable>
<xsl:variable name="page-height">11in</xsl:variable>
```



## Page margins and side column width

The page margins and side column width are defined in `demo/fo/cfg/fo/attrs/basic-settings.xml`. By default, all margins are set to the same measurement, but individual margins can be set as needed. Following is an example that shows the page margin variables set up for asymmetrical margins:

```
<!-- This is the default, but you can set the margins individually below. -->
  <xsl:variable name="page-margins">20mm</xsl:variable>

  <!-- Change these if your page has different margins on different sides. -->
  <xsl:variable name="page-margin-left" select="0.75in"/>
  <xsl:variable name="page-margin-right" select="0.5in"/>
  <xsl:variable name="page-margin-top" select="0.75in"/>
  <xsl:variable name="page-margin-bottom" select="1in"/>
```

Notice in this example that the `page-margin-left` variable and related variables are set to absolute values. In the default format, they are all set to the value of the `page-margins` variable, so they are all the same.

The problem with asymmetrical margins is that the values are used as “left” and “right” measurements, not “inner” and “outer” measurements, which is what you might prefer for page margins. One way to handle this is shown in the following example for the `fo:simple-page-master` on the even-side, front-matter page (in the `createDefaultLayoutMasters` template in `demo/fo/cfg/fo/layout-masters.xml`):

```
<fo:simple-page-master master-name="front-matter-even"
  page-width="{ $page-width }"
  page-height="{ $page-height }">
  <fo:region-body margin-top="{ $page-margin-top }"
    margin-bottom="{ $page-margin-bottom }"
    margin-left="{ $page-margin-right }"
    margin-right="{ $page-margin-left }"/>
  <fo:region-before extent="{ $page-margin-top }"
    display-align="before"
    region-name="even-frontmatter-header"/>
  <fo:region-after extent="{ $page-margin-bottom }"
    display-align="after"
    region-name="even-frontmatter-footer"/>
</fo:simple-page-master>
```

The `page-margin-right` variable is now used for the left margin, and the `page-margin-left` variable is used for the right margin.

If this arrangement makes you semantically uncomfortable, you can also create your own `page-margin-inner` and `page-margin-outer` variables and substitute them for the `margin-left` and `margin-right` attributes in the `fo:region-body` elements where needed.

## Number of columns

To change the number of columns on a page, you create new `fo:simple-page-master` elements in the `fo:layout-masters` element and then you create a new `fo:page-sequence-master` that uses the `fo:simple-page-master` elements. In the structure of the book, you create a condition that makes a new `fo:page-sequence` that uses the specific `fo:page-sequence-master` defined with the required number of columns. You can only specify the number of columns in a page layout for an `fo:page-sequence`. This element can occur only at the top level of the `fo:root` element for the document. This level corresponds to large document sections such as chapters, appendixes, the index, and so forth. You cannot specify the number of columns at a lower level in the document without significant restructuring of the XSL-FO for document organization.

Following is a short example of the technique for applying multiple columns to a document section.

In this example, you change the number of columns in the `createDefaultLayoutMasters` template in `demo/fo/cfg/fo/layout-masters.xml`. Inside the `fo:layout-master-set` element, create the following `fo:simple-page-master`. (Bold type indicates where this template is changed from the `fo:simple-page-master` called `body-first`.)

```
<fo:simple-page-master
  master-name="double-column"
  page-width="{ $page-width}"
  page-height="{ $page-height}">
  ❶ <fo:region-body
    column-count="2"
    column-gap="24pt" margin-top="{ $page-margin-top}"
    margin-bottom="{ $page-margin-bottom}"
    margin-left="{ $page-margin-left}"
    margin-right="{ $page-margin-right}"/>
    <fo:region-before extent="{ $page-margin-top}"
      display-align="before"
      region-name="first-body-header"/>
    <fo:region-after extent="{ $page-margin-bottom}"
      display-align="after"
      region-name="first-body-footer"/>
  </fo:region-body>
</fo:simple-page-master>
```

You will find the controls for columns in the `column-count` and `column-gap` attributes on the `fo:region-body` element (❶). Set the attribute values as needed for your application.

Also within the `fo:simple-page-master` in the same file, create the `fo:page-sequence-master` for double-column pages, as follows:

```
<fo:page-sequence-master master-name="two-column">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference master-reference="double-column" odd-or-even="odd"
      page-position="first"/>
    <fo:conditional-page-master-reference master-reference="double-column" odd-or-even="odd"/>
    <fo:conditional-page-master-reference master-reference="double-column" odd-or-even="even"/>
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```



This example uses the same `fo:simple-page-master` for all the pages. You could create different `fo:simple-page-masters` for odd and even pages.

At this point, you can also apply the double-page layout to a specific chapter. To do this, you need to specify which chapter should use the two-page column layout. The complete details of this technique are beyond the scope of this paper.

For the sake of simplicity, the following example applies the two-column layout to all chapter pages. In the `processTopicChapter` template in `demo/fo/xsl/fo/commons.xsl`, the first line in the template is as follows:

```
<fo:page-sequence master-reference="body-sequence" xsl:use-attribute-sets="__force__page__count">
```

Change this line as follows:

```
<fo:page-sequence master-reference="two-column" xsl:use-attribute-sets="__force__page__count">
```

The `master-reference` attribute now applies the `fo:pages-sequence-master` with `master-name="two-column"` to the text for all of the chapters.

## Text indentation

Text indentation is controlled in a number of templates in the Idiom plugin. The best way to find all instances of text indentation is to search for the following string in all the files in the `demo/fo` directory:

```
<xsl:attribute name="margin-left">
```

The search results return several instances in the `demo/fo/cfg/fo/attrs/commons-attr.xsl` file, as well as other files. For example, the `margin-left` attribute appears in the following attribute set for the body text of a top-level topic:

```
<xsl:attribute-set name="body__toplevel">
  <xsl:attribute name="margin-left">25pt</xsl:attribute>
  <xsl:attribute name="font-size"><xsl:value-of select="$default-font-size"/></xsl:attribute>
</xsl:attribute-set>
```

You could set the text indentation by changing the value of the `margin-left` attribute in each location, such as this one. This approach allows you to create different indents for each type of text that is associated with a given attribute set.

Alternatively, you could use a variable that is already set up in the Idiom plugin and affect multiple instances with a single value. In the `demo/fo/cfg/fo/attrs/basic-settings.xsl` file, find the following variable definition:

```
<xsl:variable name="side-col-width">25pt</xsl:variable>
```

Currently, this variable is not actually used in the templates. To use this variable, modify all the `margin-left` attributes in `commons-attr.xsl` and other files (that is, all the instances you want to change) similar to the change shown in bold for the `body__toplevel` attribute set:

```
<xsl:attribute-set name="body__toplevel">
  <xsl:attribute name="margin-left"><b><xsl:value-of select="$side-col-width"/></b></xsl:attribute>
  <xsl:attribute name="font-size"><xsl:value-of select="$default-font-size"/></xsl:attribute>
</xsl:attribute-set>
```

Now you can change the value of the variable in the `demo/fo/cfg/fo/attrs/basic-settings.xsl` file and have the new value reflected in multiple attribute sets.

You can also create additional variables in the `basic-settings.xsl` file if there are different groups of elements that should receive similar indentation. For example, there is a set of indented, low-level titles that you might want to have all at the same indentation, but at a different measurement than the text indentation. Create a new variable in the `demo/fo/cfg/fo/attrs/basic-settings.xsl` file and use it in the templates for the indented titles.

## Modifying chapter characteristics

A chapter in a DITA document usually corresponds to a top-level `topicref` in a ditamap. Following is an example of a ditamap for a book consisting of three chapters:

```
<map id="bbq-1" rev="DRAFT" title="Elements of Barbecue">
  <topicref href="meat.xml" format="dita" scope="local" navtitle="Cuts of Meat">
    <topicref href="pork.xml" format="dita" scope="local"/>
    <topicref href="beef.xml" format="dita" scope="local"/>
    <topicref href="poultry.xml" format="dita" scope="local"/>
    <topicref href="seafood.xml" format="dita" scope="local"/>
  </topicref>
  <topicref href="fire.xml" format="dita" scope="local" navtitle="Types of Fire">
    <topicref href="wood.xml" format="dita" scope="local"/>
    <topicref href="charcoal.xml" format="dita" scope="local"/>
    <topicref href="gas.xml" format="dita" scope="local"/>
  </topicref>
  <topicref href="sauce.xml" format="dita" scope="local" navtitle="Varieties of Sauce">
    <topicref href="tomato.xml" format="dita" scope="local"/>
    <topicref href="vinegar.xml" format="dita" scope="local"/>
    <topicref href="soy.xml" format="dita" scope="local"/>
    <topicref href="sweet.xml" format="dita" scope="local"/>
  </topicref>
</map>
```

The chapters are “Cuts of Meat,” “Types of Fire,” and “Varieties of Sauce.” In the Idiom plugin, these chapter-level `topicrefs` result in the creation of new `fo:page-sequences` for each chapter, which includes the content of all the descendant topics.

The `fo:simple-page-master` elements for the default chapter `fo:page-sequence` are associated with the `fo:page-master-sequence` called `body-sequence` in `demo/fo/cfg/fo/layout-masters.xsl`.

To create a new type of chapter (for instance, a chapter with a different column width), add a new `fo:page-sequence` and associate a top-level topic to this `fo:page-sequence` using some kind of testing or selection criteria. This selection or choice would need to take into account the selection provided by the topic template in `demo/fo/xsl/fo/commons.xsl`, which begins with the following line:

```
<xsl:template match="*[contains(@class, ' topic/topic ')]">
```



In this template, a variable invokes the `determineTopicType` template to determine what type of topic is being processed. Topic types defined by the DITA OT include `topicChapter`, `topicAppendix`, `topicPart`, `topicPreface`, `topicNotices`, `topicSimple`, `topicAbstract`, and “otherwise.” Each topic type is processed differently. Use this template to set up processing to define a different topic type or chapter type.

The top-level topic element is processed with the `processTopicChapter` template, which is as follows:

```
<xsl:template name="processTopicChapter">
  ❶ <fo:page-sequence master-reference="body-sequence" xsl:use-attribute-sets="__force__page__count">
    <xsl:call-template name="startPageNumbering"/>
    <xsl:call-template name="insertBodyStaticContents"/>
  ❷ <fo:flow flow-name="xsl-region-body">
  ❸ <fo:block xsl:use-attribute-sets="topic">
    <xsl:attribute name="id">
      <xsl:value-of select="@id"/>
    </xsl:attribute>
    <xsl:if test="not(ancestor::*[contains(@class,'topic/topic')])">
      <fo:marker marker-class-name="current-topic-number">
        <xsl:number format="1"/>
      </fo:marker>
      <fo:marker marker-class-name="current-header">
        <xsl:for-each select="child::*[contains(@class,'topic/title')]">
          <xsl:call-template name="getTitle"/>
        </xsl:for-each>
      </fo:marker>
    </xsl:if>
    <xsl:apply-templates select="*[contains(@class,'topic/prolog')]">
  ❹ <xsl:call-template name="insertChapterFirstpageStaticContent">
      <xsl:with-param name="type" select="'chapter'"/>
    </xsl:call-template>
  ❺ <fo:block xsl:use-attribute-sets="topic.title">
      <xsl:call-template name="pullPrologIndexTerms"/>
      <xsl:for-each select="child::*[contains(@class,'topic/title')]">
        <xsl:call-template name="getTitle"/>
      </xsl:for-each>
    </fo:block>
  ❻ <xsl:call-template name="createMiniToc"/>
      <xsl:apply-templates select="*[contains(@class,'topic/topic')]">
    </fo:block>
  </fo:flow>
</fo:page-sequence>
</xsl:template>
```

The `processTopicChapter` template controls the chapter format setup, the chapter head static content, chapter-level TOC, and prolog. For some items, such as the chapter static content and chapter-level TOC, additional templates are called. This is the template that creates the `fo:page-sequence` for the chapter.

In the previous set of FO templates for the DITA OT, a single fo:page-sequence was used for the entire book. The Idiom plugin now creates a new fo:page-sequence for each top-level topic.

For the fo:page-sequence (❶), the fo:flow (❷), and the top-level fo:block (❸) in the fo:flow element, you can specify formatting characteristics that apply to the entire chapter. Some of these characteristics include page numbering, default font settings, and so forth.

For example, the line after ❶ calls the startPageNumbering template. If you want the page numbering to start at 1 with the first page of the first chapter, go to the startPageNumbering template. There you can take the code out of comments to restart the page numbering.

In the chapter head static content, which includes the word “Chapter” or “Part” and the chapter or part number, the line at ❹ invokes the insertChapterFirspageStaticContent template. The chapter title itself is placed inside the fo:block at line ❺, which derives its formatting from the topic.title attribute set located in demo/fo/cfg/fo/attrs/common-attrs.xsl.

The “miniTOC,” or chapter-level table of contents, is created automatically when the chapter is created. If you do not want a miniTOC to appear, or you want it to appear only when some condition is met, you can control its insertion at line ❻. If you want to change the appearance or functionality of the miniTOC, you modify it in the createMiniToc template, which is located further down in the commons.xsl file.

## Modifying table characteristics

The Idiom plugin in DITA OT 1.5 provides a default table format as shown in the following example:

Name	Description
guitar	A stringed instrument played by plucking or strumming.
piano	A stringed instrument whose strings sound when they are struck by a hammer mechanism that is activated when a player depresses a key.
flute	A woodwind instrument that sounds when air is blown across a hole in a tube.

Using the same source document, here is the table processed by the older, “legacy” FO stylesheets in DITA OT 1.4.3:

Name	Description
guitar	A stringed instrument played by plucking or strumming.
piano	A stringed instrument whose strings sound when they are struck by a hammer mechanism that is activated when a player depresses a key.
flute	A woodwind instrument that sounds when air is blown across a hole in a tube.

Modifications to the Idiom plugin’s table stylesheets include:

- ❖ Adding rules between rows and between columns.
- ❖ Changing the indentation for text in cells.
- ❖ Centering the header text.



- ❖ Changing the background color.
- ❖ Adding a table rule where a table breaks across a page. (The default behavior, not shown in the preceding example, is for table rules to be omitted at the bottom and top where the table is broken across a page.)

## Adding rules between rows and between columns

The Idiom plugin implements user controls for table rules, giving the user more control over the appearance of rules in a table. These controls require adding the `frame`, `colsep`, and `rowsep` attributes to different elements in the DITA table markup. Without these attributes, rows and columns do not get rules, by default. As shown in the preceding example, the default behavior in the earlier stylesheets was that all rows and columns received rules.

While the new controls are an improvement over the legacy version of DITA OT FO processing, the use of these controls presents problems. For one thing, insertion of required attributes adds extra work to the effort of creating tables. Existing source documents may not use the `frame`, `rowsep`, and `colsep` attributes. For documents containing hundreds or thousands of tables, this could lead to considerable effort to update the tables to use the new convention. Also, requiring users to specify the presence or absence of rules increases the chances of having inconsistent table formatting in different parts of a document.

One solution is to modify the stylesheets to insert all rules unless the `frame`, `rowsep`, and `colsep` attributes are present. Tables in legacy markup continue to have rules, and authors still have more individualized control over table formatting by including the attributes.

A simple and admittedly heavy-handed approach to adding rules to tables is inserting borders for all sides if the `frame` attribute is not present in the table element. The template to modify is in `demo/fo/xsl/fo/tables_1.0.xsl`. The `generateTableEntryBorder` template controls the presence of borders. (Because this is a long template, the following example shows the template with several elements elided.)

```
<xsl:template name="generateTableEntryBorder">
  ...
  <xsl:if test="number($rowsep) and (../parent::node())[contains(@class, ' topic/thead ')]">
    ...
  </xsl:if>
  <xsl:if test="number($rowsep) and ((../following-sibling::*[contains(@class, ' topic/row ')] or
  (../parent::node())[contains(@class, ' topic/tbody ') and ancestor::*[contains(@class, ' topic/tgroup ')]][1]/
  *[contains(@class, ' topic/tfoot ')])">
    ....
  </xsl:if>
  <xsl:if test="$needTopBorderOnBreak = 'true'">
    ...
  </xsl:if>
  <xsl:if test="number($colsep) and following-sibling::*[contains(@class, ' topic/entry ')]">
    ...
  </xsl:if>
```

```

<xsl:if test="number($colsep) and not(following-sibling::*[contains(@class, ' topic/entry ')]) and
((count(preceding-sibling::*)+1) &lt;&lt; ancestor::*[contains(@class, ' topic/tgroup ')][1]/@cols)">
...
</xsl:if>
</xsl:template>

```

To modify the template so it always creates rules if the frame attribute is not available, add an `xsl:choose` element with the `xsl:otherwise` element around the current contents of the template. Before the `xsl:otherwise` element, add an `xsl:when` element that executes if the frame attribute is not available. Inside the `xsl:when` attribute, add XSL-FO elements that specify that all borders should have rules. The template now looks like this, with the original contents collapsed:

```

<xsl:template name="generateTableEntryBorder">
  <xsl:variable name="colsep">
    <xsl:call-template name="getTableColsep"/>
  </xsl:variable>
  <xsl:variable name="rowsep">
    <xsl:call-template name="getTableRowsep"/>
  </xsl:variable>
  <xsl:variable name="frame" select="ancestor::*[contains(@class, ' topic/table ')][1]/@frame"/>
  <xsl:variable name="needTopBorderOnBreak">
    ...
  </xsl:variable>
  <xsl:choose>
    <xsl:when test="not(ancestor::*[contains(@class, ' topic/table ')][1]/@frame)">
      <xsl:call-template name="processAttrSetReflection">
        <xsl:with-param name="attrSet" select=" '__tableframe__top' "/>
        <xsl:with-param name="path" select="$tableAttrs"/>
      </xsl:call-template>
      <xsl:call-template name="processAttrSetReflection">
        <xsl:with-param name="attrSet" select=" 'thead__tableframe__bottom' "/>
        <xsl:with-param name="path" select="$tableAttrs"/>
      </xsl:call-template>
      <xsl:call-template name="processAttrSetReflection">
        <xsl:with-param name="attrSet" select=" '__tableframe__right' "/>
        <xsl:with-param name="path" select="$tableAttrs"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:if test="number($rowsep) and (../parent::node()[contains(@class, ' topic/thead ')])">
        ...
      </xsl:if>
      <xsl:if test="number($rowsep) and ((../following-sibling::*[contains(@class, ' topic/row ')]) or (../
parent::node()[contains(@class, ' topic/tbody ')]) and ancestor::*[contains(@class, ' topic/tgroup ')][1]/
*[contains(@class, ' topic/tfoot ')])">
        ...
      </xsl:if>
      <xsl:if test="$needTopBorderOnBreak = 'true'">

```



```

...
</xsl:if>
<xsl:if test="number($colsep) and following-sibling::*[contains(@class, ' topic/entry ')]">
...
</xsl:if>
<xsl:if test="number($colsep) and not(following-sibling::*[contains(@class, ' topic/entry ')] and
((count(preceding-sibling::*)+1) &lt; ancestor::*[contains(@class, ' topic/tgroup ')]][1]/@cols)">
...
</xsl:if>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

```

In the `xsl:call-template` elements in this example, the `processAttrSetReflection` template hands off the processing to attribute sets that specify the rule to use for the top, bottom, and right rules. You could also place the attributes for creating rules directly in this template, or you could create your own attribute sets with different values for the rules.

## Changing indentation for text in cells

By default, the table elements do not specify any text indent. However, because of how XSL-FO elements inherit properties from their ancestors, the table inherits a `margin-left` attribute property from the body of the topic. This indent is specified in `demo/fo/cfg/fo/attrs/commons-attr.xml` in one of three attribute sets: `body__toplevel`, `body__secondlevel`, and `body`. These templates set the `margin-left` attribute to the same value, 25pt. This attribute and its value are inherited and used by the `fo:table-cell` element.

Table cells typically do not need this much indentation. For very dense tables the indent wastes a lot of space. You can remove the indent by overriding the `margin-left` attribute inheritance in the table cell.

The following example code shows the `tbody.row.entry` attribute set (located in `demo/fo/cfg/fo/attrs/tables-attr.xml`) with a new entry. This entry specifies that the `margin-left` attribute value is 0pt.

```

<xsl:attribute-set name="tbody.row.entry">
  <!--body cell-->
  <xsl:attribute name="margin-left">0pt</xsl:attribute>
</xsl:attribute-set>

```

You also need to supply the `margin-left` attribute in the `thead.row.entry` attribute set to remove the indent from the table header cells.

## Centering the header text

The `demo/fo/cfg/fo/attrs/tables-attr.xml` file contains the template that controls the position of the header text. To center the header text, find the `thead.row.entry__content` template and add a new entry:

```

<xsl:attribute-set name="thead.row.entry__content">
  <!--head cell contents-->
  <xsl:attribute name="margin">3pt 3pt 3pt 3pt</xsl:attribute>

```

```

<xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
</xsl:attribute-set>

```

## Changing the background color

Modify the background color in the `demo/fo/cfg/fo/attrs/tables-attr.xml` file. Find the `thead.row.entry` template, which is as follows:

```

<xsl:attribute-set name="thead.row.entry">
  <!--head cell-->
  <xsl:attribute name="background-color">antiquewhite</xsl:attribute>
</xsl:attribute-set>

```

Replace the value `antiquewhite` value with another named color or with an RGB value. Please refer to the XSL-FO specification (<http://www.w3.org/TR/xsl/>) for more details on valid values for the `background-color` attribute.

## Adding a table rule where a table breaks across a page

Some style guides for technical documentation recommend omitting the table rule at the bottom of a page where a table breaks to the next page, and also the table rule at the top of the next page where the table continues. For others, however, this is not the preferred style. The Idiom plugin for XSL-FO implements the style that omits the table rules across a page break. This section describes how to “fix” this behavior.

**NOTE:** This modification is not needed for tables without the `frame` attribute when you have made the modification in “Adding rules between rows and between columns” on page 18.

Depending on the value of the `frame` attribute on the DITA table element in the source document, any of several attribute sets could be called for setting the table border behavior. The following example shows the modification for the attribute set that is called when `frame="all"`. When the `frame` attribute is equal to `all`, the attribute set that affects the border rules is `table__tableframe__all` in the `demo/fo/cfg/fo/attrs/tables-attr.xml` file. If your tables use other values for the `frame` attribute, modify the attribute sets associated with those values:

```

<xsl:attribute-set name="table__tableframe__all">
  <xsl:attribute name="border-top-style">solid</xsl:attribute>
  <xsl:attribute name="border-top-width">1pt</xsl:attribute>
  <xsl:attribute name="border-top-color">black</xsl:attribute>
  <xsl:attribute name="border-bottom-style">solid</xsl:attribute>
  <xsl:attribute name="border-bottom-width">1pt</xsl:attribute>
  <xsl:attribute name="border-bottom-color">black</xsl:attribute>
  <xsl:attribute name="border-left-style">solid</xsl:attribute>
  <xsl:attribute name="border-left-width">1pt</xsl:attribute>
  <xsl:attribute name="border-left-color">black</xsl:attribute>
  <xsl:attribute name="border-right-style">solid</xsl:attribute>
  <xsl:attribute name="border-right-width">1pt</xsl:attribute>
  <xsl:attribute name="border-right-color">black</xsl:attribute>
</xsl:attribute-set>

```



To control the border across the page margin, you use the “conditionality” property of lengths for XSL-FO. The conditionality property applies to the top and bottom border, but specifying the conditionality property for the border-bottom-width and border-top-width attributes does not work. Change the attributes to border-before-width and border-after-width (changing the attributes for color and style to match), and then add the conditionality property to the attributes. The modified attribute set is as follows:

```
<xsl:attribute-set name="table__tableframe__all">
  <xsl:attribute name="border-before-style">solid</xsl:attribute>
  <xsl:attribute name="border-before-width">1pt</xsl:attribute>
  <xsl:attribute name="border-before-width.conditionality">retain</xsl:attribute>
  <xsl:attribute name="border-before-color">black</xsl:attribute>
  <xsl:attribute name="border-after-style">solid</xsl:attribute>
  <xsl:attribute name="border-after-width">1pt</xsl:attribute>
  <xsl:attribute name="border-after-width.conditionality">retain</xsl:attribute>
  <xsl:attribute name="border-after-color">black</xsl:attribute>
  <xsl:attribute name="border-left-style">solid</xsl:attribute>
  <xsl:attribute name="border-left-width">1pt</xsl:attribute>
  <xsl:attribute name="border-left-color">black</xsl:attribute>
  <xsl:attribute name="border-right-style">solid</xsl:attribute>
  <xsl:attribute name="border-right-width">1pt</xsl:attribute>
  <xsl:attribute name="border-right-color">black</xsl:attribute>
</xsl:attribute-set>
```

With these changes, the table border is completed at page breaks, as shown in the following example:

Name	Description
guitar	A stringed instrument played by plucking or strumming.

---

#### 4 | OpenTopic | Garage Tasks

Name	Description
piano	A stringed instrument whose strings sound when they are struck by a hammer mechanism that is activated when a player depresses a key.
flute	A woodwind instrument that sounds when air is blown across a hole in a tube.

## Summary

This paper describes how to make basic changes to the format of PDF files produced by the default DITA Open Toolkit. You can make numerous other changes to create PDF documents that are more complex or visually satisfying. The DITA OT is capable of supporting a broad range of output customizations, but understanding of XSL-FO and the DITA OT stylesheets does require a bit of time and effort. The introduction provided by this paper is intended to help you make some of your own customizations.

## 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](mailto: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.

