Customization of the *DocBook XSL Stylesheets* for the TCC

John W. Shipman

2009-01-28 17:09

Table of Contents

1. Introduction

DocBook is a generalized framework for writing documentation using XML. Here at the New Mexico Tech Computer Center (TCC), we use DocBook extensively for external and internal documentation. Among the many advantages of this system is that a single source document can be translated mechanically to both HTML and PDF representations.

This document contains all the files used for local customization, in "lightweight literate programming" format. For more information, see the author's *Lightweight literate programming* [1] page.

The local DocBook toolchain consists of these components:

- Norman Walsh's *DocBook XSL Stylesheets* provide a generic styling of DocBook that can be customized with local style conventions. These stylesheets consist of XSLT scripts. For more information about XSLT, see *XSL Transformations (XSLT) Version 1.0* [2].

  These style sheets can be downloaded from the SourceForge repository [3].

- The *xsltproc* package implements the XSLT language. Production of the HTML output from a DocBook document needs only XSLT.

- The *xep* processor is necessary to produce the printable, PDF (Adobe Page Description Format) form of a DocBook document. This transformation starts by using *xsltproc* to transform the original DocBook document into XSL, also known as XSL-FO (for Formatting Objects). The resulting `.fo` file is input to *xep*, which produces the PDF output file.

  For more information on XSL, see the standard: *Extensible Stylesheet Language (XSL) Version 1.0* [4].

1.1. Bob Stayton's *DocBook XSL: The complete guide*

Without customization, the stock *DocBook XSL Stylesheets* produce a pretty bland, generic output style. This document describes the TCC's local customization layers, built on top of the *DocBook XSL Stylesheets*, that give our documents their local style.

Documentation for the *DocBook XSL Stylesheets* is rather skeletal. Fortunately, there is an excellent book that describes the customization process in detail:

[1] http://www.nmt.edu/~shipman/soft/litprog/homepage.html
[2] http://www.w3.org/TR/xslt
[3] http://sourceforge.net/project/showfiles.php?group_id=21935
[4] http://www.w3.org/TR/xsl/

Stayton, Bob. DocBook XSL: The complete guide. Third edition, March 2005, ISBN 0-9741521-2-9.

This book will be referred to throughout this document as “Stayton.”

Another good resource is the DocBook XSL Stylesheet Wiki [5].

1.2. How to get this publication

This publication is available in Web form [6] and also as a PDF document [7]. Please forward any comments to **`tcc-doc@nmt.edu`**.

2. Skills you will need

You will need to know your way around these tools to work effectively on stylesheet customization:

- XSLT, the language in which the *DocBook XSL Stylesheets* are written. Doug Tidwell's book *XSLT* (O'Reilly, 2001, ISBN 0-596-00053-7) is an extremely valuable tutorial. See *XSLT Reference* [8] for a summary of the language. There are also some XSLT debugging tools, but the present author has not tried them.
- XSL, also known as XSL-FO (for Formatting Objects), is the language used to specify printed output. We don't recommend trying to learn this language, which is considerably bigger than XSLT, from the standard [9]. Fortunately, Dave Pawson's book *XSL-FO* (O'Reilly, 2002, ISBN 0-596-00355-2) is a well-written tutorial.
- A schema-aware text editor helps maintain the correctness of XSLT and XML files. We recommend *XML document authoring with emacs nxml-mode* [10].

3. What is a customization layer?

One of the great benefits of the *DocBook XSL Stylesheets* is their modularity. If you don't like a style, you can start your own *customization layer* that makes changes to someone else's style, leaving intact the parts you don't want to change. The TCC standard style is a customization layer built on top of the stock, uncustomized *DocBook XSL Stylesheets*.

Specifically, you can do any of these things:

- If you are writing TCC documentation, follow the procedures in *Writing documentation with DocBook-XML 4.2* [11].
- If you don't like some parts of the TCC style, you can add your customization layer on top of the TCC's customization layer. The top-level TCC templates live in this directory:

  ```
  /u/www/docs/tcc/doc/docbook42xep/
  ```

  The root HTML customization file is named `tcc_html.xsl` and the root PDF file is named `tcc_fo.xsl`.
- If you don't want any part of the TCC's style, you can build your own customization layer on top of the *DocBook XSL Stylesheets*. The stock stylesheets live in this directory:

[5] http://wiki.docbook.org/topic/DocBookXslStylesheets/
[6] http://www.nmt.edu/tcc/doc/docbook42xep/ims/
[7] http://www.nmt.edu/tcc/doc/docbook42xep/ims/tccstyle.pdf
[8] http://www.nmt.edu/tcc/help/pubs/xslt
[9] http://www.w3.org/TR/xsl/
[10] http://www.nmt.edu/tcc/help/pubs/nxml/
[11] http://www.nmt.edu/tcc/help/pubs/docbook42

```
/u/www/docs/tcc/doc/docbook42xep/mss/
```

The procedure for building a customization layer is well-described on p. 102 of Stayton. Briefly:

1. Create a file whose name ends in `.xsl` to hold your customization layer, and set up your `Makefile` or other procedures to use this file as input to *xsltproc*.
2. In this file, use **`xsl:import`** to read the layer you are building yours on—the TCC stylesheets or the stock *DocBook XSL Stylesheets*.
3. Write templates to replace the parts of the layer under you that you don't like.

4. Overall structure of the local customization layer

In general, to produce a local customization of the *DocBook XSL Stylesheets*, we need only write an XSLT file (with file extension `.xsl`) that imports the *DocBook XSL Stylesheets* and then adds local customizations.

However, there is another significant complication. Customizing the title page format for either HTML or FO output is a two-stage process. First, one must create an XML template file that describes the format of the title page. Our customization file is called `tcc-titlepage.xml`. Then, one runs it through *xsltproc* using a special stylesheet, `template/titlepage.xsl` in the *DocBook XSL Stylesheets*, that converts it into file `tcc-titlepage.xsl`, which is the part of the local customization layer that describes the title page layout.

These are the principal components of the TCC's local customization layer for the Modular Stylesheets. All file references are relative to the root of our current customization layer, which is currently at `/u/www/docs/tcc/doc/docbook42xep` on **`infohost`**.

- The `tcc_html.xsl` file is the root customization file for producing HTML output. See Section 6, "`tcc_html.xsl`: HTML customization layer" (p. 5).
- The `html-titlepage.xml` file is the template file used to produce the `html/html-titlepage.xsl` file that describes the HTML title page format. See Section 7, "`html-titlepage.xml`: HTML title page template" (p. 17).
- The `tcc_fo.xsl` file is the root customization file for producing PDF output. See Section 8, "`tcc_fo.xsl`: PDF customization layer" (p. 18).
- The `fo-titlepage.xml` file is the template file used to produce the `fo-titlepage.xsl` file that describes the PDF title page format. See Section 9, "`fo-titlepage.xml`: PDF title page template" (p. 31).

5. Title page customization: XSLT that builds XSLT

Before we move on to the details of our customization layer, let's look at one of the less obvious parts of the process, title page customization.

The term "title page" is a slight misnomer. There is no guarantee that this material will be on a separate page—just that it will be presented first in the document.

Before reading this section, you should read and understand Stayton's chapter 10, "*Title page customization*".

Here's what's different about title page customization. Most customization consists of writing XSLT that modifies the stock XSLT-based *DocBook XSL Stylesheets*. However, customization of the title page is indirect: you create an XML file that specifies what you want your title page to look like, and then use a special script named `template/titlepage.xsl` that transforms that XML file into the actual

XSLT script that plugs into the *DocBook XSL Stylesheets* structure and specifies the layout of the title page.

So, to customize the title page, follow these steps:

1. Prepare a title page template file. This file is an `.xml` file that uses namespace `http://nwalsh.com/docbook/xsl/template/1.0` to describe the layout of the title page. Start with a copy of the stock version, which is located at `html/titlepage.templates.xml` in the *DocBook XSL Stylesheets*, and modify it to suit.
2. Use *xsltproc* to transform this file into an `.xsl` file that is the actual title page customization file. The script that transforms the XML file to XSLT is at `template/titlepage.xsl` in the *DocBook XSL Stylesheets*.
3. Include the generated `.xsl` file as part of the customization layer.

6. `tcc_html.xsl`: HTML customization layer

The `tcc_html.xsl` file is an XSLT script that lies on top of the *DocBook XSL Stylesheets*, specifying the local customizations for our HTML presentation.

It starts with the usual **`xsl:stylesheet`** root element. Attributes include:

- The **`xsl:`** namespace is the usual XSLT namespage.
- We also use one of the EXSLT extensions to XSLT, the date package, as namespace **`date:`**
- The **`extension-element-prefixes`** attribute informs the XSLT processor that the **`date:`** namespace prefix extends XSLT.
- The **`exclude-result-prefixes`** attribute instructs the XSLT processor to process elements in the **`date:`** namespace rather than copying them to the output.

```
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:date="http://exslt.org/dates-and-times"
    extension-element-prefixes="date"
    exclude-result-prefixes="date">
  <!-- XSL-HTML stylesheet customization layer for the TCC
   !   For documentation, see:
   !       http://www.nmt.edu/tcc/doc/docbook42xep/ims/
   !-->
```

Next we import the `html/chunk.xsl` stylesheet from the stock *DocBook XSL Stylesheets*. Stayton discusses this in the chapter on HTML customization, the section on chunking: this file is the root HTML stylesheet for chunked HTML output (as opposed to one huge page).

```
<xsl:import
    href="http://www.nmt.edu/tcc/doc/docbook42xep/mss/html/chunk.xsl"/>
  <xsl:output method="html"/>
```

The remaining local customizations are divided into these sections:

- Section 6.1, “HTML general page layout” (p. 6).
- Section 6.2, “HTML title page and table of contents” (p. 6).
- Section 6.3, “HTML headers and footers” (p. 7).

- Section 6.4, “HTML section and subsection headings” (p. 14).
- Section 6.5, “HTML specific element customizations” (p. 14).

6.1. HTML general page layout

The first general customization we need is to link to the CSS stylesheet that customizes the appearance of all DocBook-generated Web pages. The content of the **`html.stylesheet`** parameter appears as ***`S`*** in the generated style sheet link: “**`<link rel="stylesheet" href="`*`S`*`" type="text/css">`**

```
<xsl:param name="html.stylesheet">/tcc/docbook.css</xsl:param>
```

The color scheme defined in the stock `html/docbook.xsl` file is black text on a white background. Since the TCC has different colors (defined in the CSS style sheet, `docbook.css`), we need to get rid of the stock color scheme. To do this, we define an empty **`body.attributes`** template to replace the template by that name that contains the color definitions.

```
<xsl:template name="body.attributes"/>
```

In early 2009, a lot of browsers started rendering the non-break space character (`&nbsp;`) as dark lozenge with a question mark inside it. This turns out to have been caused by an incorrect encoding scheme. This issue is discussed in Stayton, 3rd ed., p. 298, in the chapter “Languages, characters, and encoding,” the section entitled “Output encoding.” The default encoding is ISO-8859-1, but it should be UTF-8. The `chunker.output.encoding` variable controls the encoding that appears in the `meta` tag for chunked output.

```
<xsl:param name="chunker.output.encoding">UTF-8</xsl:param>
```

6.2. HTML title page and table of contents

The main customization for title pages is to include the generated `html-titlepage.xsl` template that came out of the title page customization process; see Section 7, “`html-titlepage.xml`: HTML title page template” (p. 17).

```
<xsl:include href="html-titlepage.xsl"/>
```

This next bit is high wizardry. We often use tags such as **`userinput`** and **`filename`** inside section and subsection titles. With the stock stylesheets, text marked up with these tags uses ordinary fonts. It wasn't obvious to me how to fix this, so I wrote to Bob Stayton, and got back this highly useful reply:

> It certainly isn't obvious, but this customization will achieve what you want for userinput:
>
> ```
> <xsl:template match="userinput" mode="no.anchor.mode">
>   <xsl:apply-templates select="." />
> </xsl:template>
> ```
>
> If you want to trace this through the templates:
>
> 1. Processes **`<section>`** with **`mode="toc"`** in `autotoc.xsl`. That applies templates in **`mode="title.markup"`**.
> 2. Matches on **`<section>`** in **`mode="title.markup"`** in `titles.xsl`. That sets variable **`$title`** to the content of the **`<title>`** element. Applies templates on **`$title`** in **`mode="title.markup"`**.
> 3. Matches on **`title`** in **`mode="title.markup"`** in `titles.xsl`. Applies templates in **`mode="no.anchor.mode"`**.

The **`'no.anchor.mode'`** prevents links and index entries. There are no templates in such mode for inlines, so it defaults to outputting just the text content of the element. The customization above applies the regular **`userinput`** template to the element instead.

So, here's the customization that does this for **`filename`**, **`sgmltag`**, **`userinput`**, and **`varname`** elements:

```
<xsl:template match="filename|sgmltag|userinput|varname"
    mode="no.anchor.mode">
  <xsl:apply-templates select="." />
</xsl:template>
```

6.3. HTML headers and footers

Customization of the TCC header and footer content follows the *TCC Documentation Guidelines* [12]. The intent is to mimic the style of HTML pages generated by other tools, such as *PyStyler* [13] [PDF file].

Because the TCC's navigational features are completely different in structure from the stock templates, we have elected to completely replace the **`header.navigation`** and **`footer.navigation`** templates. See page 140 of Stayton for a description of when these templates are called.

6.3.1. The `header.navigation` template

The first template we define is **`header.navigation`**, which sets up the top-of-page features in HTML. This template is a customized copy of the stock **`header.navigation`** template from `html/chunk-common.xsl`.

The template takes three parameters:

`prev`
The previous page's node, or empty if this is the first child of its parent.

`next`
The next page's node, or empty if there is none.

`nav.context`
Not used by this template. If you're curious about what it does in the stock template, see the stock template in `html/chunk-common.xsl`.

```
<xsl:template name="header.navigation">
  <xsl:param name="prev" select="/foo"/>
  <xsl:param name="next" select="/foo"/>
  <xsl:param name="nav.context"/>
```

Next are two variables internal to this template:

`home`
The root element of the DocBook document. The XPath expression **`"/*[1]"`** selects the first child of the document root, which is probably an **`article`**.

`up`
The parent element of this node. The XPath expression **`"parent::*"`** selects the parent node.

```
  <xsl:variable name="home" select="/*[1]"/>
  <xsl:variable name="up" select="parent::*"/>
```

[12] http://www.nmt.edu/tcc/doc/plan/
[13] http://www.nmt.edu/tcc/help/pubs/pystyler/pystyler.pdf

The first HTML header element is the top-of-page navigational bar, containing the links “Next / Prev / ...” and so forth. This is produced by calling the **tcc.top.nav.bar** template, and passing our **prev**, **next**, and **home** nodes to it.

```
  <xsl:call-template name="tcc.top.nav.bar">
    <xsl:with-param name="prev" select="$prev"/>
    <xsl:with-param name="next" select="$next"/>
    <xsl:with-param name="home" select="$home"/>
  </xsl:call-template>
```

Next is a small table, with one row and two columns, that positions the title on the top left and the TCC logo on the top right. This is wrapped in a **div** element with class **navheader** so that a CSS stylesheet can customize the appearance of this area.

```
  <div class="navheader">
    <table width="100%" summary="Navigation header">
      <tr valign="top">
        <td align="left" valign="top">
          <h1>
            <xsl:apply-templates select="$home" mode="object.title.markup"/>
          </h1>
        </td>
        <td>
          <img alt="Tech Computer Center logo"
               src="/tcc/images/logo.png"/>
        </td>
      </tr>
    </table>
```

The **mode="object.title.markup"** template call causes the title text to be inserted as content of the HTML **h1** element.

Lastly, we add a horizontal rule to set off the header from the page body.

```
    <hr/>
  </div>
</xsl:template>
```

6.3.2. The tcc.top.nav.bar template

The **tcc.top.nav.bar** template generates the standard TCC HTML page-top navigation bar. It takes three parameters:

prev
: Previous page's node, or empty if this is the first page.

next
: Next page's node, or empty if this is the last page.

home
: Node for the home page.

```
<xsl:template name="tcc.top.nav.bar">
  <xsl:param name="prev"/>
  <xsl:param name="next"/>
  <xsl:param name="home"/>
```

The nav bar is wrapped in a **div** element to put it all in a separate block. This element has class **topnavbar** so that a CSS stylesheet can change the appearance of this element.

```
  <div class="topnavbar">
```

The word “Next” always appears: it is a link if there is a next page, or a placeholder if not.

- The **xsl:when** tests to see if there is a next page.
- The **href.target** template returns the URL corresponding to a given node.

```
    <xsl:choose>
      <xsl:when test="count($next) &gt; 0">  <!-- Is there a prev? -->
        <a>
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="object" select="$next"/>
            </xsl:call-template>
          </xsl:attribute>
          <xsl:text>Next</xsl:text>
        </a>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>Next</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
```

Next we output a slash, with spaces around it, to separate “Next” from “Previous”.

```
    <xsl:text> / </xsl:text>
```

Generation of the “Previous” link is similar: it is an actual link if there is a previous page, otherwise it is the word “Previous” as a placeholder.

```
    <xsl:choose>
      <xsl:when test="count($prev) &gt; 0">  <!-- Is there a prev? -->
        <a>
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="object" select="$prev"/>
            </xsl:call-template>
          </xsl:attribute>
          <xsl:text>Previous</xsl:text>
        </a>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>Previous</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
```

Again, we output a slash to set off following elements.

```
    <xsl:text> / </xsl:text>
```

The “home” link here is called “Contents.” Again, it is followed by a slash.

```
    <a>
      <xsl:attribute name="href">
        <xsl:call-template name="href.target">
          <xsl:with-param name="object" select="$home"/>
        </xsl:call-template>
      </xsl:attribute>
      <xsl:text>Contents</xsl:text>
    </a>
    <xsl:text> / </xsl:text>
```

Lastly we output links to the TCC help system and the Tech homepage.

```
    <a href="http://www.nmt.edu/tcc/">TCC Help System</a>
    <xsl:text> / </xsl:text>
    <a href="http://www.nmt.edu/">NM Tech homepage</a>
  </div>
</xsl:template>  <!-- End of tcc.top.nav.bar -->
```

6.3.3. The `footer.navigation` template

This template replaces the stock **`footer.navigation`** template from `html/chunk-common.xsl`. It takes the same three parameters as the ones passed to Section 6.3.1, “The **`header.navigation`** template” (p. 7). It has the same two internal variables: **`home`**, the **`article`** node of the document; and **`up`**, the parent node.

```
<xsl:template name="footer.navigation">
  <xsl:param name="prev" select="/foo"/>
  <xsl:param name="next" select="/foo"/>
  <xsl:param name="nav.context"/>

  <xsl:variable name="home" select="/*[1]"/>
  <xsl:variable name="up" select="parent::*"/>
```

First we output a horizontal rule to set off the page body from the footer content.

```
  <hr/>
```

The content of the footer is enclosed in a **`div`** element, with **`class="navfooter"`** so that we can apply a CSS rule to just this **`div`**.

```
  <div class="navfooter">
```

The standard TCC page-bottom links are output by a separate template; see Section 6.3.4, “The **`tcc.bot.links`** template” (p. 11).

```
    <xsl:call-template name="tcc.bot.links">
      <xsl:with-param name="prev" select="$prev"/>
      <xsl:with-param name="next" select="$next"/>
      <xsl:with-param name="home" select="$home"/>
    </xsl:call-template>
```

The colophon, or author credit information, is output by yet another template; see Section 6.3.5, “The **`tcc.colophon`** template” (p. 12).

```
    <xsl:call-template name="tcc.colophon">
      <xsl:with-param name="home" select="$home"/>
    </xsl:call-template>
  </div>
</xsl:template>
```

6.3.4. The `tcc.bot.links` template

This template outputs the standard TCC page-bottom navigational links. These links start with a bold-faced term such as “Next” that match the links on the top of the page, followed by a link to the target page using that page's title as the link text. Each link line is wrapped in its own **`div`** element, and the entire set of links is wrapped in another **`div`** with **`class="botlinks"`** to allow CSS markup.

```
<xsl:template name="tcc.bot.links">
  <xsl:param name="prev"/>
  <xsl:param name="next"/>
  <xsl:param name="home"/>
  <div class="botlinks">
```

First comes the “Next” link.

- The **`xsl:if`** tests to see whether there is a node in the **`$next`** variable.
- The entire structure is wrapped in a **`div`** element with **`class="bot-next"`**.
- The **`href.target`** template takes a node as a parameter and returns the URL of the corresponding location in the generated document.
- The **`object.title.markup`** template extracts the title of the node we're pointing to.

```
    <xsl:if test="count($next) &gt; 0">
      <div class="bot-next">
        <b>Next: </b>
        <a>
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="object" select="$next"/>
            </xsl:call-template>
          </xsl:attribute>
          <xsl:apply-templates select="$next" mode="object.title.markup"/>
        </a>
      </div>
    </xsl:if>
```

Next comes the “Contents” link, similarly, wrapped in a **`div`** with **`class="bot-contents"`**.

```
    <xsl:if test="$home != .">
      <div class="bot-contents">
        <b>Contents: </b>
        <a>
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="object" select="$home"/>
            </xsl:call-template>
          </xsl:attribute>
          <xsl:apply-templates select="$home" mode="object.title.markup"/>
        </a>
      </div>
    </xsl:if>
```

The “Previous” link works just like the “Next” link.

```
    <xsl:if test="count($prev) &gt; 0">
      <div class="bot-prev">
        <b>Previous: </b>
        <a>
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="object" select="$prev"/>
            </xsl:call-template>
          </xsl:attribute>
          <xsl:apply-templates select="$prev" mode="object.title.markup"/>
        </a>
      </div>
    </xsl:if>
```

The last two links, “Help” and “Home”, always point to the same places.

```
    <div><b>Help: </b> <a href="http://www.nmt.edu/tcc/help/">Tech
       Computer Center: Help System</a></div>
    <div><b>Home: </b>
       <a href="http://www.nmt.edu/">About New Mexico Tech</a></div>
  </div>
</xsl:template>
```

6.3.5. The `tcc.colophon` template

The colophon section contains:

1. A horizontal rule to separate it from the page-bottom links.
2. An HTML **`address`** element containing lines for each author, and a **`mailto:`** link for reader feedback.
3. A “Last updated” timestamp.
4. The document's URL.

The template takes one parameter, **`home`**, the **`article`** node. It needs that to generate the “Contents” link.

```
<xsl:template name="tcc.colophon">
  <xsl:param name="home"/>
```

First comes the horizontal rule, then the start of a **`div`** element with **`class="colophon"`** so we can write CSS rules for this section, then the start of the **`address`** element.

```
  <hr/>
  <div class="colophon">
    <address>
```

To generate all the author names, we call a separate template; see Section 6.3.6, “The **`author.colophon.mode`** template” (p. 13).

```
      <xsl:apply-templates select="$home/articleinfo/authorgroup"
        mode="author.colophon.mode"/>
```

The **mailto:** link forwards mail to **tcc-doc@nmt.edu**, which is the mailing alias for the TCC Documentation Group. This ends the HTML **address** element.

```
      <div class="colophon-mailto">
        <xsl:text>Comments welcome: </xsl:text>
        <a href="mailto:tcc-doc@nmt.edu">tcc-doc@nmt.edu</a>
      </div>
    </address>
```

The "Last updated" line is wrapped in a **div** to put it on a separate line. The **datetime.format** template is located in `common/pi.xsl`. It uses the EXSLT extension module **http://exslt.org/dates-and-times**; see the online documentation for **dates-and-times** [14].

```
    <div class="colophon-date">
      <xsl:text>Last updated: </xsl:text>
      <xsl:call-template name="datetime.format">
        <xsl:with-param name="date" select="date:date-time()"/>
        <xsl:with-param name="format" select="'Y-m-d H:M'"/>
      </xsl:call-template>
    </div>
```

We use Server Side Includes to generate the page's own URL; see *Using HTML Server Side Includes (SSI)* [15]. SSI commands are syntactically represented as comments whose first character is **"#"**. The **echo** command causes insertion of some variable's value into the web page, and the **DOCUMENT_URI** variable is always the URL of the web page itself.

```
    <div class="colophon-url">
      <xsl:text>URL: </xsl:text>
      <span class="colophon-uri">
        <xsl:text>http://www.nmt.edu</xsl:text>
        <xsl:comment>#echo var="DOCUMENT_URI"</xsl:comment>
      </span>
    </div>
  </div>
</xsl:template>
```

6.3.6. The `author.colophon.mode` template

This template takes as input a DocBook **authorgroup** element, extracts each author's name, and wraps each in a **div** element to put it on a separate line.

- The **xsl:for-each** iterates over the **author** elements.
- Each author's name is wrapped in a **div** element with **class="colophon-author"**.

```
<xsl:template match="authorgroup" mode="author.colophon.mode">
  <xsl:for-each select="author">
    <div class="colophon-author">
      <xsl:value-of select="./firstname"/>
      <xsl:text> </xsl:text>
```

[14] http://exslt.org/date/index.html
[15] http://www.nmt.edu/tcc/help/html/ssi.html

```
      <xsl:value-of select="./surname"/>
    </div>
  </xsl:for-each>
</xsl:template>
```

6.4. HTML section and subsection headings

All of the local customizations of section headings can be done by setting XSLT variables.

First, we turn on section numbering, so that each section and subsection has a number such as 3.1.4 that identifies where it fits into the overall structure.

```
  <xsl:param name="section.autolabel" select="1"/>
```

We set the **`chunk.quietly`** variable to suppress messages that would normally be written as each chunk is generated.

```
<xsl:param name="chunk.quietly" select="1"></xsl:param>
```

The **`use.id.as.filename`** variable instructs XSLT to use each section's **`id`** attribute as its file name. Without this variable, chunk file names are assigned according to section numbering, but this means any changes in the section structure change the file names of chunks. With the variable set, for example “**`<section id='foo'>`**” will be placed in a file named `foo.html`. This makes it easier for external documents to link to specific locations in the HTML structure.

```
<xsl:param name="use.id.as.filename">1</xsl:param>
```

6.5. HTML specific element customizations

This part of the stylesheet makes changes to the appearance of certain specific DocBook elements.

6.5.1. The `inline.italicsansseq` template

Some later parts of the stylesheet need the ability to select new font types. For example, we want application names (the **`application`** element) to be presented in an italic, sans-serif font, so that they really stand out from other elements such as **`userinput`**.

Accordingly, we need templates that set their arguments using such font variants. These templates are modeled on the ones from the stock stylesheets, such as the **`inline.italicseq`** template for italics, which is located in `html/inline.xsl`.

The first one is **`inline.italicsansseq`**. This template is used to format content using italic sans-serif font.

```
<xsl:template name="inline.italicsansseq">
```

This template takes one argument, **`content`**, which by default is the context node's content.

- The **`anchor`** template, from `html/html.xsl`, defines an HTML anchor (**`<a name="`*`I`*`"/>`**) if its context node has an **`id="`*`I`*`"`** attribute.
- The **`simple.xlink`** template implements a basic XLink (see the XLink standard [16]) if the context node has an **`xlink:href`** attribute.

This **`xsl:param`** element, then, sets the **`content`** parameter to the context node's content, optionally preceded by an anchor, and optionally wrapped in an XLink.

[16] http://www.w3.org/TR/2001/REC-xlink-20010627/

```
  <xsl:param name="content">
    <xsl:call-template name="anchor"/>
    <xsl:call-template name="simple.xlink">
      <xsl:with-param name="content">
        <xsl:apply-templates/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:param>
```

The **`content`** is then wrapped in an HTML **`span`** element whose **`style`** attribute specifies the **`sans-serif`** font family and the **`italic`** font style. It also attaches a **`class="N"`** attribute where ***N*** is the element's local name. For example, if this template is used to wrap a DocBook **`application`** element, it will generate a **`span`** with **`class="application"`**. This makes it possible to write CSS rules that apply to that element type.

```
  <span style="font-family: sans-serif; font-style: italic"
        class="{local-name(.)}"
  ><xsl:copy-of select="$content"/></span>
</xsl:template>
```

6.5.2. The `inline.smallcapsseq` template

The next template is **`inline.smallcappsseq`**, which uses a caps-and-small-caps font. The structure of this template is the same as that of the **`inline.italicsansseq`** template; only the **`style`** attribute of the **`span`** element is different.

```
<xsl:template name="inline.smallcapsseq">
  <xsl:param name="content">
    <xsl:call-template name="anchor"/>
    <xsl:call-template name="simple.xlink">
      <xsl:with-param name="content">
        <xsl:apply-templates/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:param>
  <span style="font-variant: small-caps"
        class="{local-name(.)}"
  ><xsl:copy-of select="$content"/></span>
</xsl:template>
```

6.5.3. Verbatim elements: `programlisting`, `screen`, and `literallayout`

Because we make heavy use of **`programlisting`** in literate programming and other software-related documentation, such elements are set off by shading. This can be turned on by setting the XSLT variable **`shade.verbatim`**.

```
<xsl:param name="shade.verbatim" select="1"></xsl:param>
```

6.5.4. `application`

We want DocBook **`application`** elements to be set in italic sans-serif font. For the font markup template, see Section 6.5.1, “The **`inline.italicsansseq`** template” (p. 14).

```
<xsl:template match="application">
  <xsl:call-template name="inline.italicsansseq"/>
</xsl:template>
```

6.5.5. Emphasis with boldface

We use the tag **`<emphasis role='strong'>`** to mean strong emphasis. This is rendered in boldface. The **`inline.boldseq`** template is part of the stock stylesheets, residing in `html/inline.xsl`.

```
<xsl:template match="emphasis[@role='strong']">
  <xsl:call-template name="inline.boldseq"/>
</xsl:template>
```

6.5.6. `callout` graphics

We stipulate that if a DocBook user uses callouts such as **`programlistingco`**, they must have a subdirectory named **`callout/`** containing images named `1.png`, `2.png`, and so forth, to be used as the graphic for callouts 1, 2, ….

```
<xsl:param name="callout.graphics.path">callouts/</xsl:param>
```

6.5.7. `firstterm`

Normally the DocBook **`firstterm`** element receives no special markup. As we use this element rather heavily, we want its content italicized.

```
<xsl:template match="firstterm">
  <xsl:call-template name="inline.italicseq"/>
</xsl:template>
```

6.5.8. `guibutton`, `guiicon`, `guilabel`, and `guimenu`

These elements are rendered in italic sans-serif font. See Section 6.5.1, “The **`inline.italicsansseq`** template” (p. 14).

```
<xsl:template match="guibutton|guiicon|guilabel|guimenu">
  <xsl:call-template name="inline.italicsansseq"/>
</xsl:template>
```

6.5.9. The `keysym` element

The DocBook **`keysym`** element denotes names of keys on the keyboard. We render those in small caps in HTML. See Section 6.5.2, “The **`inline.smallcapsseq`** template” (p. 15).

```
<xsl:template match="keysym">
  <xsl:call-template name="inline.smallcapsseq"/>
</xsl:template>
```

6.6. Epilogue for the HTML stylesheet

This is the end of the HTML stylesheet file, `tcc_html.xsl`.

```
</xsl:stylesheet>
```

7. `html-titlepage.xml`: HTML title page template

Very little customization has been done to the stock version of this file. Because we are using chunked HTML output, the title page material appears at the top of every chunk—so we don't really need a title page. The title has been moved to the page header, produced by the **`header.navigation`** template. The **`authorgroup`** and **`author`** content have been moved to the page footer, produced by the `footer.navigation` template. The remaining title page elements are discarded.

Here is the `html-titlepage.xml` template file, with comments. This file is a modified copy of file `html/titlepage.templates.xml` in the stock *DocBook XSL Stylesheets* distribution.

First is the root **`templates`** element, which describes the several namespaces used in this file.

```
<!--File html-titlepage.xml.  For documentation, see:
 !    http://www.nmt.edu/tcc/doc/docbook42xep/ims/
 !-->
<t:templates xmlns:t="http://nwalsh.com/docbook/xsl/template/1.0"
    xmlns:param="http://nwalsh.com/docbook/xsl/template/1.0/param"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

The namespaces are:

- **`t:`** is the namespace used to write title page template files.
- Attributes in the **`param:`** namespace will be passed as arguments to the generated template. For example, an attribute “**`param:node="ancestor-or-self::chapter[1]"`**” describes a parameter that would be passed as:

```
    <xsl:with-param name="node" select="ancestor-or-self::chapter[1]"/>
```

- **`xsl:`** is the usual XSLT namespace.

The remainder of the file consists of **`t:titlepage`** elements; each describes the title page for a particular context. The only one we've customized here is the first one, which governs the title page layout of an **`<article>`**. Here is the start tag:

```
<t:titlepage t:element="article" t:wrapper="div" class="titlepage">
```

Within each **`t:titlepage`** element are several child elements that describe the generated content. Most are empty. The one that used to contain all the title page content we're deleting is this one:

```
  <t:titlepage-content t:side="recto">
    <!--All items removed were here-->
  </t:titlepage-content>
```

The **`t:titlepage-content`** element describes generated content. Its **`t:side="recto"`** attribute means that this applies to the right-hand (recto) page, that is the “front #x201d; title page, as opposed to the “verso” page on the back. These terms obviously apply to printed pages and not to HTML; however, the recto page in the HTML is still the “front” page.

Here is the content that was removed from the stock `titelepage.templates.xml` file:

```
    <title/>
    <subtitle/>
    <corpauthor/>
    <authorgroup/>
    <author/>
    <othercredit/>
    <releaseinfo/>
    <copyright/>
    <legalnotice/>
    <pubdate/>
    <revision/>
    <revhistory/>
    <abstract/>
```

These elements, if present, would produce the title, subtitle, corporate author, and so forth.

Next in our `html-titlepage.xml` file is the template for the verso page, which we are not customizing:

```
  <t:titlepage-content t:side="verso">
  </t:titlepage-content>
```

Only one customization remains. In the stock file, there is a horizontal rule (**`<hr/`**) following the title page content. Since we aren't generating any title page content, we don't need the horizontal rule either. Here is the customized **`t:titlepage-separator`** element that used to contain the horizontal rule:

```
  <t:titlepage-separator>
    <!--hr removed here; reinstate if there is any content in
     !  the titlepage-content element above.
     !-->
  </t:titlepage-separator>
```

Next we finish off the **`t:titlepage`** element, supplying empty **`t:titlepage-before`** elements. If there were any content in these elements, that content would appear before the title page.

```
  <t:titlepage-before t:side="recto">
  </t:titlepage-before>

  <t:titlepage-before t:side="verso">
  </t:titlepage-before>
</t:titlepage>

</t:templates>
```

The rest of the the stock `html/titlepage.templates.xml` file has been deleted. This rather long section includes declarations for the format of title content for elements such as **`set`**, **`book`**, **`sect1`**, and so forth. We won't need these unless we later decide to add a “title page” to elements other than **`article`**.

If you want to define new kinds of title content for some elements, go to the stock `html/titlepage.templates.xml` file, find the **`t:titlepage`** element for the desired DocBook element, and paste it in here.

For example, if we ever define a **`book`** style, find the element that starts with **`<t:titlepage t:element="book" t:wrapper="div" class="titlepage">`**, insert it here, and customize it as appropriate.

8. `tcc_fo.xsl`: PDF customization layer

The PDF customization layer, which produces its output using XSL-FO (Formatting Objects), starts out with an **`xsl:stylesheet`** element similar to the HTML customization layer (see Section 6, "`tcc_html.xsl`: HTML customization layer" (p. 5)), with one difference: it includes the XSL-FO namespace as prefix "**`fo:`**".

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:date="http://exslt.org/dates-and-times"
    extension-element-prefixes="date"
    exclude-result-prefixes="date">
  <!-- XSL-FO stylesheet customization layer for the TCC
   !   For documentation, see:
   !       http://www.nmt.edu/tcc/doc/docbook42xep/ims/
   !-->
```

As usual, we import the stock FO templates, and set the output method to XML.

```
  <xsl:import
    href="http://www.nmt.edu/tcc/doc/docbook42xep/docbook-xsl-1.65.1/fo/docbook.xsl"
  <xsl:output method="xml"/>
```

The PDF customizations are divided into these sections:

- Section 8.1, "General page layout" (p. 19).
- Section 8.2, "Templates for title elements" (p. 21).
- Section 8.3, "PDF headers and footers" (p. 23).
- Section 8.4, "PDF section and subsection headings" (p. 27).
- Section 8.5, "PDF inline element customizations" (p. 28).
- Section 8.6, "PDF block element customizations" (p. 30).

8.1. General page layout

This section contains parameter settings that affect the overall PDF page format. To make changes here, you need to understand a number of XSL-FO concepts such as regions, blocks and inlines, block and inline progression directions, and such. Dave Pawson's book *XSL-FO* is absolutely indispensable for this background information; see Section 2, "Skills you will need" (p. 3).

First, we turn on double-sided formatting, which places the "outer" margin on the left of even pages and the right of odd pages, and the "inner" margin in the opposite positions.

```
<xsl:param name="double.sided">1</xsl:param>
```

The margins are defined next. For double-sided formatting, the default inner margin is 1.25" and the outer 0.75". To save paper, we set the inner margin to 1".

```
<xsl:param name="page.margin.outer">0.75in</xsl:param>
<xsl:param name="page.margin.inner">1in</xsl:param>
```

In the stock style, the **`xref`** element does not include a page number. We want internal cross-references to use the page number.

```
<xsl:param name="insert.xref.page.number">1</xsl:param>
```

The next parameter is a bit obscure. The stock style defaults to "draft mode", which overlays each page with a user-supplied image named `draft.png`. We do not support draft mode, and would prefer not to see a bunch of messages about how it can't find `draft.png`.

```
<xsl:param name="draft.mode">no</xsl:param>
```

Because the current style has no running header, we reclaim most of the space allocated to the header, but leave 0.25" so that the FO processor won't complain about inadequate space for the (empty) content that goes there. The area occupied by the header is called the “region before”, and its size paramater is **`region.before.extent`**.

```
<xsl:param name="region.before.extent">0.25in</xsl:param>
```

The next few sections set up the vertical spacing used in page makeup. Each of these is an **`xsl:attribute-set`** that contain attributes that define the minimum, optimum, and maximum spacing before or after a given element.

The values we use here were determined by trying various values to see how they look. This is a tradeoff: smaller spacing saves paper, and tends to put more on a page, reducing the number of cases where blocks of related content are divided across page breaks. On the other hand, larger spaces can make the document more readable.

First, the **`normal.para.spacing`** attribute set defines the spacing around paragraphs. The stock (minimum, optimal, maximum) spaces are (0.8em, 1em, 1.2em), but those are rather generous. To save paper we scrunch them down a bunch. (The “em” is the usual printer's measure—the point size of the font, so in a 12-point font, 0.5em is 6 points.)

```
<xsl:attribute-set name="normal.para.spacing">
  <xsl:attribute name="space-before.minimum">0.50em</xsl:attribute>
  <xsl:attribute name="space-before.optimum">0.60em</xsl:attribute>
  <xsl:attribute name="space-before.maximum">0.70em</xsl:attribute>
</xsl:attribute-set>
```

Next we set up the spacing for **`itemizedlist`** and similar elements. The **`list.block.spacing`** attribute set describes the space before and after the entire list. The default spacing is (0.8em, 1.0em, 1.2em) before and after.

```
<xsl:attribute-set name="list.block.spacing">
  <xsl:attribute name="space-before.minimum">0.70em</xsl:attribute>
  <xsl:attribute name="space-before.optimum">0.75em</xsl:attribute>
  <xsl:attribute name="space-before.maximum">0.80em</xsl:attribute>
  <xsl:attribute name="space-after.minimum">0.70em</xsl:attribute>
  <xsl:attribute name="space-after.optimum">0.75em</xsl:attribute>
  <xsl:attribute name="space-after.maximum">0.80em</xsl:attribute>
</xsl:attribute-set>
```

The **`list.item.spacing`** attribute set describes the space around individual items in the list. The default values are also (0.8em, 1.0em, 1.2em).

```
<xsl:attribute-set name="list.item.spacing">
  <xsl:attribute name="space-before.minimum">0.50em</xsl:attribute>
  <xsl:attribute name="space-before.optimum">0.60em</xsl:attribute>
  <xsl:attribute name="space-before.maximum">0.70em</xsl:attribute>
</xsl:attribute-set>
```

Finally, the **`verbatim.properties`** attribute set defines the vertical spacing around the various verbatim-type elements (**`programlisting`**, **`literallayout`**, and **`screen`**).

```
<xsl:attribute-set name="verbatim.properties">
  <xsl:attribute name="space-before.minimum">0.4em</xsl:attribute>
  <xsl:attribute name="space-before.optimum">0.5em</xsl:attribute>
  <xsl:attribute name="space-before.maximum">0.6em</xsl:attribute>
  <xsl:attribute name="space-after.minimum">0.4em</xsl:attribute>
  <xsl:attribute name="space-after.optimum">0.5em</xsl:attribute>
  <xsl:attribute name="space-after.maximum">0.6em</xsl:attribute>
```

We add a very thin black border (0.1mm) around verbatim elements. This has the advantage of showing clearly when a verbatim element is broken across a page boundary: the side facing the break will have no border. The default border style is **`none`**, so we must set the style to **`solid`**.

```
  <xsl:attribute name="border-width">0.1mm</xsl:attribute>
  <xsl:attribute name="border-style">solid</xsl:attribute>
```

The **`padding`** attribute insures that the content does not actually touch the border.

```

  <xsl:attribute name="padding">1mm</xsl:attribute>
</xsl:attribute-set>
```

8.2. Templates for title elements

The first customization is to force **`userinput`**, **`filename`**, and several other inlines to have their usual font markup inside the table of contents. This trick is discussed further in Section 6.2, “HTML title page and table of contents” (p. 6).

```
<xsl:template
    match="filename|sgmltag|userinput|varname|application"
    mode="no.anchor.mode">
  <xsl:apply-templates select="." />
</xsl:template>
```

To produce the standard TCC title page, we want to set up a table with one row and two columns, with the document's title in the left-hand column and the TCC logo in the right-hand column.

As page 161 of Stayton points out, there are three ways to customize titles. In ascending order by precedence:

1. Customize the title page specification file; see Section 5, “Title page customization: XSLT that builds XSLT” (p. 4).
2. Customize the attribute set named **`component.title.properties`**.
3. Customize the template named **`component.title`**.

Although the TCC style currently uses only one title element (the **`article`** element's **`title`** element), customizing **`component.title`** affects all title elements, so that approach is a bit too brute-force.

Page 162 of Stayton's book gives the procedure for customizing the title of a particular element:

1. Copy the stock **`component.title`** template from `fo/component.xsl` to your customization layer, and give it a name. Here, we are customizing the title of an **`article`** element, so we'll call our copy **`article.title`**. For this template, see Section 8, “`tcc_fo.xsl`: PDF customization layer” (p. 18).
2. Modify the new copy to give the desired format.
3. In the appropriate part of the title page customization file, replace the reference to **`component.title`** with a reference to the new name—in this case, **`article.title`**.

So, here is the customized **`article.title`** template. The template takes these arguments:

node
: The context node containing the title.

pagewide
: Originally used to specify whether the title should be stretched to the full page width. Not used here.

id
: The unique identifier of the title block. If the title element has no unique ID, one will be generated for it.

title
: The content of the title.

The following code is taken from the original **component.title** from `fo/component.xsl`. Some of it is relatively inscrutable, such as the part that mentions the FoTeX extensions (which might matter if we used the PassiveTeX package to produce PDF output).

```
<xsl:template name="component.title">
  <xsl:param name="node" select="."/>
  <xsl:param name="pagewide" select="0"/>
  <xsl:variable name="id">
    <xsl:call-template name="object.id">
      <xsl:with-param name="object" select="$node"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:variable name="title">
    <xsl:apply-templates select="$node" mode="object.title.markup">
      <xsl:with-param name="allow-anchors" select="1"/>
    </xsl:apply-templates>
  </xsl:variable>

  <xsl:if test="$passivetex.extensions != 0">
    <fotex:bookmark xmlns:fotex="http://www.tug.org/fotex"
                    fotex-bookmark-level="2"
                    fotex-bookmark-label="{$id}">
      <xsl:value-of select="$title"/>
    </fotex:bookmark>
  </xsl:if>
```

The entire title content is wrapped in a **fo:block** container. The **keep-with-next** attribute stipulates that we would prefer not to break a page or column right after it. The **hyphenate** attribute discourages hyphenation in the title block.

```
  <fo:block keep-with-next.within-column="always"
            hyphenate="false">
```

Here we start the table. Support for the various table models may vary, but **table-layout="fixed"** definitely works. We use a four-inch column for the title and a two-inch column for the graphic (which was sized for that space).

```
    <fo:table table-layout="fixed" padding-bottom="0.2in">
      <fo:table-column column-number="1" column-width="4in"/>
      <fo:table-column column-number="2" column-width="2in"/>
      <fo:table-body>
        <fo:table-row>
```

The left-hand column contains the title text, left-justified.

```
          <fo:table-cell>
            <fo:block text-align="left">
              <xsl:copy-of select="$title"/>
            </fo:block>
          </fo:table-cell>
```

The right-hand column contains the TCC logo graphic. And that's the end of the template.

```
          <fo:table-cell>
            <fo:block>
              <fo:external-graphic src="url(logo.jpg)"
                content-width="2in"/>
            </fo:block>
          </fo:table-cell>
        </fo:table-row>
      </fo:table-body>
    </fo:table>
  </fo:block>
</xsl:template>
```

Below the table is the **`revhistory`** (revision history) element. Rather than using the RCS timestamp from the DocBook source file, we'll just show the current date and time using the EXSLT date package. This template replaces the stock **`revhistory`** template that appears in `fo/titlepage.xsl`.

```
<xsl:template match="revhistory" mode="titlepage.mode">
  <fo:block text-align="center">
    <xsl:call-template name="datetime.format">
      <xsl:with-param name="date" select="date:date-time()"/>
      <xsl:with-param name="format" select="'Y-m-d H:M'"/>
    </xsl:call-template>
  </fo:block>
</xsl:template>
```

8.3. PDF headers and footers

By default, the header area on each page is set off from the content with a ruled line. Because the TCC style does not use a running head, we turn off **`header.rule`** to eliminate this rule.

```
<xsl:param name="header.rule" select="0"/>
```

The stock **`header.content`** template is in `fo/pagesetup.xsl`; this template generates the content of the running head. We replace that here with one that has no content.

```
<xsl:template name="header.content">
  <xsl:param name="pageclass" select="''"/>
  <xsl:param name="sequence" select="''"/>
  <xsl:param name="position" select="''"/>
  <xsl:param name="gentext-key" select="''"/>
</xsl:template>
```

Footer customization is discussed at length in Chapter 12 of Stayton under the heading “Running headers and footers.”

First we change the appearance of the text in the running footer to make it look different from the body text. This is specified by the **`footer.content.properties`** attribute set; the stock version is in `fo/param.xsl`. The TCC Documentation Plan mandates nine-point italic text.

```
<xsl:attribute-set name="footer.content.properties">
  <xsl:attribute name="font-style">italic</xsl:attribute>
  <xsl:attribute name="font-size">9pt</xsl:attribute>
```

The rest of this attribute set is from the stock version.

```
  <xsl:attribute name="font-family">
    <xsl:value-of select="$body.fontset"/>
  </xsl:attribute>
  <xsl:attribute name="margin-left">
    <xsl:value-of select="$title.margin.left"/>
  </xsl:attribute>
</xsl:attribute-set>
```

The footer is formatted as a one-row, three-column table. The footer's content depends on two variables:

- The **`pageclass`** describes the general type of page. Values include:

`titlepage`	The title page.
`lot`	List-of-titles pages, including the table of contents, list of figures, and such.
`front`	Front matter: preface, dedication, etc.
`body`	Main content pages.
`back`	Back matter such as appendices and glossaries.
`index`	Book-style index pages.

- Pages in each **`pageclass`** are also divided by position. This is called the **`sequence`** attribute of the page, with these values:

`first`	First page of this class.
`odd`	An odd-numbered, nonfirst page.
`even`	An even-numbered, nonfirst page.
`blank`	A blank page added to even out the page count.

In our format, the content of the three columns of the footer table is shown by this table. The “Case” column describes which case applies, and the three cells of the footer table are called **`left`**, **`center`**, and **`right`**.

Case	**`left`**	**`center`**	**`right`**
`pageclass` is **`titlepage`**	blank	blank	blank
Not **`titlepage`**, single-sided	title	folio	logo
Not **`titlepage`**, double-sided, odd/first pages	title	folio	logo
Not **`titlepage`**, double-sided, even/blank pages	title	folio	logo

- “title” is the document's title.

- “folio” is the page number.
- “logo” is “New Mexico Tech Computer Center”.
- “blank” denotes a blank cell.

This layout puts the page number in the center of single-sided pages, the outside of double-sided pages.

So, here's the **`footer.content`** template. It is called once for each of the three different footer positions, and returns the content that should go in that position. It takes four parameters: **`pageclass`**, **`sequence`**, and **`position`** are described above. The **`gentext-key`** parameter is not used; its function is discussed on p. 195 of **`Stayton`**.

```
<xsl:template name="footer.content">
  <xsl:param name="pageclass" select="''"/>
  <xsl:param name="sequence" select="''"/>
  <xsl:param name="position" select="''"/>
  <xsl:param name="gentext-key" select="''"/>
```

The entire footer content is wrapped in a block container.

```
  <fo:block>
```

First we eliminate the **`titlepage`** case: there is no footer on title pages. (The current **`article`** style has no separate title page, if you're wondering how come all the TCC documents have a running footer on the first page.)

```
    <xsl:choose>
      <xsl:when test="$pageclass = 'titlepage'">
        <!--no footer on title pages-->
      </xsl:when>
```

For non-title pages, the next important case is single-sided output.

```
      <xsl:otherwise>       <!--Not a title page-->
        <xsl:choose>
          <xsl:when test="$double.sided = 0">   <!-- Single-sided -->
            <xsl:choose>
              <xsl:when test="$position = 'left'">
                <xsl:apply-templates select="." mode="titleabbrev.markup"/>
              </xsl:when>
              <xsl:when test="$position = 'center'">
                <fo:page-number/>
              </xsl:when>
              <xsl:when test="$position = 'right'">
                <xsl:text>New Mexico Tech Computer Center</xsl:text>
              </xsl:when>
            </xsl:choose>
          </xsl:when>       <!-- Single-sided -->
```

The **`titleabbrev.markup`** mode selects a **`titleabbrev`** if there is one, defaulting to the document's **`title`**.

The remaining cases are for the two double-sided formats. First, the left position, which is the logo for odd pages and the page number for even pages.

```
          <xsl:otherwise>   <!--Double-sided-->
            <xsl:choose>
```

```
              <xsl:when test="$position = 'left'">
                <xsl:choose>
                  <xsl:when test="$sequence = 'even' or
                                  $sequence = 'blank'">
                    <fo:page-number/>
                  </xsl:when>
                  <xsl:otherwise> <!-- left/odd -->
                    <xsl:text>New Mexico Tech Computer Center</xsl:text>
                  </xsl:otherwise>
                </xsl:choose>
              </xsl:when>
```

The center position always contains the running title.

```
              <xsl:when test="$position = 'center'">
                <xsl:apply-templates select="." mode="titleabbrev.markup"/>
              </xsl:when>
```

The right position on a double-sided page contains the folio on odd pages, the logo on even pages.

```
              <xsl:when test="$position = 'right'">
                <xsl:choose>
                  <xsl:when test="$sequence = 'even' or
                                  $sequence = 'blank'">
                    <xsl:text>New Mexico Tech Computer Center</xsl:text>
                  </xsl:when>
                  <xsl:otherwise> <!-- left/odd -->
                    <fo:page-number/>
                  </xsl:otherwise>
                </xsl:choose>
              </xsl:when>
            </xsl:choose>
          </xsl:otherwise>  <!-- Double-sided -->
        </xsl:choose>
      </xsl:otherwise>      <!--Not a title page-->
    </xsl:choose>
  </fo:block>
</xsl:template>
```

8.3.1. Other header/footer options to consider

Why did we decide to have no running head? The principal motivation was aesthetics, and also to reclaim a bit more page for functional content.

However, one annoying property of this layout is that the running title has to fit in a fairly restricted space. If the document title is too long, it will be folded in the footer, which is pretty ugly. The cure for that is to add a **`titleabbrev`** element just after the document's **`title`** element; the content of that element will be substituted for the full title in the running footer.

Here is a more conventional plan that would get around this problem:

- Place the running title in the header. It would be left-justified on even pages, right-justified otherwise. To set it off from the body of the page with a ruled line, **`header.rule`** should be turned back on.
- In the running footer, put the folio on the outside and the logo on the inside.

8.4. PDF section and subsection headings

Headings are “unindented” by a fair amount to make them stand out from the body text. The default value in `fo/param.xsl` is -4pc (four picas). We decrease that to avoid running the titles too far into the margins.

```
<xsl:param name="title.margin.left">-3pc</xsl:param>
```

The **`section.autolabel`** variable turns on section numbering, so that for example a subsection will have a number like 3.4.

```
<xsl:param name="section.autolabel" select="1"/>
```

Next we set a number of properties of section titles. The **`section.title.level1.properties`** attribute set is for the largest, top-level titles. It inherits all the other attributes from the **`section.properties`** attribute set that defines properties common to section titles of all levels.

```
<xsl:attribute-set name="section.title.level1.properties"
                   use-attribute-sets="section.properties">
```

The TCC style mandates a fairly thick (1-point) rule under first-level section titles. This effect is achieved by using the **`border-bottom`** properties. Because the default border style is **`none`**, we have to specify both a style (**`solid`**) and a width.

```
  <xsl:attribute name="border-bottom-style">solid</xsl:attribute>
  <xsl:attribute name="border-bottom-width">1pt</xsl:attribute>
```

The first section title property we have to adjust is the font size. In the stock stylesheets, top-level titles have a font size of “mag step 4” (where each mag step is a factor of 1.2 larger than the previous size, so mag step 2 is 1.2 × 1.2 or a factor of 1.44), second-level titles are mag step 3, and third-level titles are mag step 2. This seems to us a bit larger, so we'll go down one whole mag step. The base value to which these magnifications are applied is **`body.font.master`**, which defaults to 10.

```
  <xsl:attribute name="font-size">
    <xsl:value-of select="$body.font.master * 1.728"/>
    <xsl:text>pt</xsl:text>
  </xsl:attribute>
</xsl:attribute-set>
```

The attribute sets for second- and third-level headings are the same except for the font size multiplier.

```
<xsl:attribute-set name="section.title.level2.properties">
  <xsl:attribute name="font-size">
    <xsl:value-of select="$body.font.master * 1.44"/>
    <xsl:text>pt</xsl:text>
  </xsl:attribute>
</xsl:attribute-set>
<xsl:attribute-set name="section.title.level3.properties">
  <xsl:attribute name="font-size">
    <xsl:value-of select="$body.font.master * 1.2"/>
    <xsl:text>pt</xsl:text>
  </xsl:attribute>
</xsl:attribute-set>
```

8.5. PDF inline element customizations

This section describes the customization of various inline elements for PDF output.

First we define templates for type styles that are not part of the stock stylesheets. As with the HTML inline customizations (see Section 6.5, "HTML specific element customizations" (p. 14)), we define templates similar to the stock **`inline.italicseq`** template and others defined in `fo/inline.xsl`.

8.5.1. The `inline.italicsansseq` template

The **`inline.italicsansseq`** template selects italic sans-serif font. It takes one parameter, the content to be marked up.

```
<xsl:template name="inline.italicsansseq">
  <xsl:param name="content">
    <xsl:apply-templates/>
  </xsl:param>
```

The content is wrapped in an inline container that selects an italic **`font-style`** and a sans-serif **`font-family`**.

```
  <fo:inline font-style="italic" font-family="sans-serif">
    <xsl:copy-of select="$content"/>
  </fo:inline>
</xsl:template>
```

8.5.2. The `inline.smallcaps` template

The next template was supposed to be for a caps-and-small-caps font, but this template is not used currently because we couldn't find an appropriate, decent-looking public-domain font.

```
<xsl:template name="inline.smallcaps">
  <xsl:param name="content">
    <xsl:apply-templates/>
  </xsl:param>
  <fo:inline font-variant="small-caps">
    <xsl:copy-of select="$content"/>
  </fo:inline>
</xsl:template>
```

8.5.3. `application` and the `gui*` group

The **`application`**, **`guibutton`**, **`guiicon`**, **`guilabel`**, and **`guimenu`** elements are marked up in italic sans-serif type to make them stand out from the body font.

```
<xsl:template match="application|guibutton|guiicon|guilabel|guimenu">
  <xsl:call-template name="inline.italicsansseq"/>
</xsl:template>
```

8.5.4. Bold emphasis

The stock formatting for the DocBook **`emphasis`** element is italics. We define a stronger degree of emphasis, encoded as **`<emphasis role='strong'>…</emphasis>`**, and format it as boldface.

```
<xsl:template match="emphasis[@role='strong']">
  <xsl:call-template name="inline.boldseq"/>
</xsl:template>
```

8.5.5. Callouts

A *callout* is a location in a display that is flagged with a number that refers to an explanatory paragraph elsewhere.

In DocBook, the writer embeds a **`co`** element at some location inside a display (typically a **`programlistingco`** element) to associate its “**`id`**” attribute with that location. Then, elsewhere in the document, they use a **`calloutlist`** element as a container for **`callout`** elements that contain explanatory material for that location.

Our local DocBook DocBook documentation [17] stipulates that the user must have a subdirectory named `callouts/` containing graphics that are used to mark the locations of the **`co`** elements in the display. Typically these graphics are colored circles or rectangles with numerals inside them. We made our own using Gimp, and here's the procedure:

1. Create a small rectangle, maybe 20×15 pixels, and paint it some solid color, such as **`#ff8080`** (a darkish pink).
2. Use the Text tool to put in the numerals.
3. Save in PNG and PDF format, with the file name equal to the callout number. For example, `16.png` and `16.pdf` would be the Web and PDF callout images, respectively, for the 16th callout.

In the customization layer, we use **`callout.graphics.path`** to specify where to find the callout graphics.

```
<xsl:param name="callout.graphics.path">callouts/</xsl:param>
```

Then we specify that for PDF output, the callout images will also be in PDF format.

```
<xsl:param name="callout.graphics.extension">.pdf</xsl:param>
```

The default number of callouts is 15. We made 20. It would be easy to make more.

```
<xsl:param name="callout.graphics.number.limit">15</xsl:param>
```

8.5.6. `firstterm`

The DocBook **`firstterm`** element, denoting the first use of a term, should be italicized.

```
<xsl:template match="firstterm">
  <xsl:call-template name="inline.italicseq"/>
</xsl:template>
```

8.5.7. `keysym`

The TCC Document Plan specifies caps-and-small-caps font for names of keyboard keys, but as discussed in Section 8.5.2, “The **`inline.smallcaps`** template” (p. 28), there is no suitable public domain font at this point. So we use italic sans-serif font instead.

[17] http://www.nmt.edu/tcc/help/pubs/docbook42/

```
<xsl:template match="keysym">
  <xsl:call-template name="inline.italicsansseq"/>
</xsl:template>
```

8.5.8. `xref`: Page cross-reference format

By default, a page number citation in an **`xref`** element is shown in square brackets, e.g., “[12]”. We'd prefer to format them as “(p. 12)” to avoid confusion with the many technical uses of square brackets.

This customization is a little tricky because it involves generated text, that is, text that may depend on the language of the document. Fortunately, on page 244 of Stayton there is an example of exactly this customization. Page 108 of Stayton has a good general introduction to generated text. You might be amused to learn that “**`l10n`**” stands for “localization”, and “i18n” stands for “internationalization.” The “**`%p`**” in the **`text`** attribute of the **`l:template`** element is replaced by the actual page number in the rendered document.

```
<xsl:param name="local.l10n.xml" select="document('')"/>
<l:i18n xmlns:l="http://docbook.sourceforge.net/xmlns/l10n/1.0">
  <l:l10n language="en">
    <l:context name="xref">
      <l:template name="page.citation" text=" (p. %p)"/>
    </l:context>
  </l:l10n>
</l:i18n>
```

8.6. PDF block element customizations

Customizations of block-level elements are presented here in alphabetical order.

8.6.1. Admonitions: `caution`, `important`, `note`, `tip`, and `warning`

These five elements, collectively known as admonitions, are formatted in the stock templates as slightly narrower blocks with the admonition type (e.g., “Warning”) as a larger heading above the content.

To make these elements stand out more (and in hopes that someone might actually read them!) we wrap them in a gray border. This customization is covered on p. 209ff of Stayton; the template is a copy of the one from `fo/admon.xsl`.

```
<xsl:template name="nongraphical.admonition">
  <xsl:variable name="id">
    <xsl:call-template name="object.id"/>
  </xsl:variable>
```

The only change we make to the stock template is the lines starting with “**`border="…"`**”, which add a solid gray border 4 points wide, and an extra 4 points of padding to keep the contents from touching the inside of the border.

```

  <fo:block space-before.minimum="0.8em"
            space-before.optimum="1em"
            space-before.maximum="1.2em"
            start-indent="0.25in"
            end-indent="0.25in"
            border="4pt solid #d0d0d0"
            padding="4pt"
            id="{$id}">
```

```
    <xsl:if test="$admon.textlabel != 0 or title">
      <fo:block keep-with-next='always'
                xsl:use-attribute-sets="admonition.title.properties">
         <xsl:apply-templates select="." mode="object.title.markup"/>
      </fo:block>
    </xsl:if>

    <fo:block xsl:use-attribute-sets="admonition.properties">
      <xsl:apply-templates/>
    </fo:block>
  </fo:block>
</xsl:template>
```

8.6.2. `programlisting` and other verbatim elements

By “verbatim element,” we mean the elements that are rendered with all line breaks and whitespace untouched: **`programlisting`**, **`screen`**, and **`literallayout`**.

Such elements are heavily used in TCC documentation. To make them stand out from narrative, we set the **`shade.verbatim`** option to show them with a light gray background.

```
<xsl:param name="shade.verbatim" select="1"></xsl:param>
```

8.6.3. `variablelist`: Variable list entry term

There are two ways to render DocBook's **`variablelist`**. The default is a two-column table, with the terms on the left and the definitions on the right. Our preference is the way HTML **`dl`** elements are rendered: with each term unindented, followed by the definition as an indented block. A single option, **`variablelist.as.blocks`**, selects the latter behavior.

```
<xsl:param name="variablelist.as.blocks" select="1"></xsl:param>
```

Because sometimes the indentation doesn't really make the term stand out enough, we boldface the term.

```
<xsl:template match="varlistentry/term">
  <xsl:call-template name="inline.boldseq"/>
</xsl:template>
```

8.7. PDF epilogue

Here's the end tag for **`xsl:stylesheet`**.

```
</xsl:stylesheet>
```

9. `fo-titlepage.xml`: PDF title page template

Here is the `fo-titlepage.xml` template file. This file is a modified copy of file `fo/titlepage.templates.xml` in the stock *DocBook XSL Stylesheets* distribution.

The `fo-titlepage.xml` file starts with the usual **`DOCTYPE`** declaration. This declaration contains a number of entity declarations used to specify a series of font sizes, each 20% larger than the one before, and also a series of three-quarter-em spaces.

```
<!DOCTYPE t:templates [
<!ENTITY hsize0 "10pt">
<!ENTITY hsize1 "12pt">
<!ENTITY hsize2 "14.4pt">
<!ENTITY hsize3 "17.28pt">
<!ENTITY hsize4 "20.736pt">
<!ENTITY hsize5 "24.8832pt">
<!ENTITY hsize0space "7.5pt"> <!-- 0.75 * hsize0 -->
<!ENTITY hsize1space "9pt"> <!-- 0.75 * hsize1 -->
<!ENTITY hsize2space "10.8pt"> <!-- 0.75 * hsize2 -->
<!ENTITY hsize3space "12.96pt"> <!-- 0.75 * hsize3 -->
<!ENTITY hsize4space "15.552pt"> <!-- 0.75 * hsize4 -->
<!ENTITY hsize5space "18.6624pt"> <!-- 0.75 * hsize5 -->
]>
```

Next comes the root element, which contains declarations for the various namespaces.

```
<!--File fo-titlepage.xml.  For documentation, see:
 !    http://www.nmt.edu/tcc/doc/docbook42xep/ims/
 !-->
<t:templates xmlns:t="http://nwalsh.com/docbook/xsl/template/1.0"
    xmlns:param="http://nwalsh.com/docbook/xsl/template/1.0/param"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Most of these namespaces are the same ones used in the HTML titlepage template (see Section 7, “`html-titlepage.xml`: HTML title page template” (p. 17)). The one new namespace, “**`fo:`**”, is the Formatting Objects namespace.

Next comes a **`t:titlepage`** element that specifies the title page layout for the DocBook **`article`** element.

`t:element="article"`
: Specifies the DocBook element to which this format applies.

`t:wrapper="fo:block"`
: The content of this title page will be wrapped in a FO **`block`** element.

`font-family="{$title.fontset}"`
: Specifies a set of font attributes to be used to display the title. Defined in `fo/pagesetup.xsl` in the *DocBook XSL Stylesheets*.

```
<t:titlepage t:element="article" t:wrapper="fo:block"
             font-family="{$title.fontset}">
```

The **`t:titlepage-content`** element specifies what to put on the right-hand (recto) page, and also specifies centering for the titles.

```
  <t:titlepage-content t:side="recto"
             text-align="center">
```

Next is the **`title`** element, which describes how to format the DocBook **`title`** element.

`t:named-template="article.title"`
: This attribute tells the XSL processor to use the template named **`article.title`** to format the title. This is a local customization; see Section 8.2, “Templates for title elements” (p. 21). The stock template calls the uncustomized **`component.title`** here.

`param:node="ancestor-or-self::article[1]"`
: The created template will match the first (and typically the only) **`article`** element in the DocBook source.

`keep-with-next="always"`
: This XSL-FO attribute specifies that there should be no page break after the title.

`font-size="&hsize5;"`
: Set the title in the largest-size font.

`font-weight="bold"`
: Set the title in boldface font.

```
    <title t:named-template="article.title"
        param:node="ancestor-or-self::article[1]"
        keep-with-next="always"
        font-size="&hsize5;"
        font-weight="bold"/>
```

The remainder of this file is exactly as it looks in the stock `fo/titlepage.templates.xml` file.

```
    <subtitle/>

    <corpauthor space-before="0.5em"
                font-size="&hsize2;"/>
    <authorgroup space-before="0.5em"
                 font-size="&hsize2;"/>
    <author space-before="0.5em"
            font-size="&hsize2;"/>

    <othercredit space-before="0.5em"/>
    <releaseinfo space-before="0.5em"/>
    <copyright space-before="0.5em"/>
    <legalnotice text-align="start"
                 margin-left="0.5in"
                 margin-right="0.5in"
                 font-family="{$body.fontset}"/>
    <pubdate space-before="0.5em"/>
    <revhistory space-before="0.5em"/>
    <revision space-before="0.5em"/>
    <abstract space-before="0.5em"
        text-align="start"
        margin-left="0.5in"
        margin-right="0.5in"
        font-family="{$body.fontset}"/>
  </t:titlepage-content>

  <t:titlepage-content t:side="verso">
  </t:titlepage-content>

  <t:titlepage-separator>
  </t:titlepage-separator>

  <t:titlepage-before t:side="recto">
  </t:titlepage-before>

  <t:titlepage-before t:side="verso">
  </t:titlepage-before>
</t:titlepage>

</t:templates>
```

As in Section 7, "`html-titlepage.xml`: HTML title page template" (p. 17), the many other title elements are omitted here. If it is later necessary to add titlepage content for other DocBook elements such as **`book`**, they can be added here, copied from the stock `fo/titlepage.templates.xml` file.