Customization of the 4.3
DocBook-XSL Stylesheets for
the TCC

John W. Shipman
2013-08-23 23:40

Abstract

Describes the customization of the DocBook Modular Stylesheets for the Tech Computer Center's
standard styles.

This publication is available in Web form\footnote{1} and also as a PDF document\footnote{2}. Please forward any
comments to tcc-doc@nmt.edu.

Table of Contents

1. Introduction ............................................................................................................................ 3
   1.1. Bob Stayton’s DocBook XSL: The complete guide ................................................................. 3
2. Online files .............................................................................................................................. 4
3. Skills you will need .................................................................................................................. 4
4. What is a customization layer? ................................................................. 4
5. Overall structure of the local customization layer ............................................................... 5
6. Title page customization: XSLT that builds XSLT ................................................................. 6
7. tcc_html.xsl: HTML customization layer ............................................................................. 7
   7.1. HTML general page layout ............................................................................................... 8
   7.2. HTML title page and table of contents ........................................................................... 9
   7.3. HTML headers and footers ............................................................................................ 10
      7.3.1. The header.navigation template .............................................................................. 10
      7.3.2. The top.nav.bar template ..................................................................................... 12
      7.3.3. The footer.navigation template ............................................................................. 14
      7.3.4. The bot.links template ......................................................................................... 15
      7.3.5. The colophon template ......................................................................................... 17
      7.3.6. The author.colophon.mode template ............................................................... 18
   7.4. HTML section and subsection headings .......................................................................... 19
   7.5. HTML inline element customizations ............................................................................ 19
      7.5.1. The inline.italicsansseq template ........................................................................... 19
      7.5.2. The inline.smallcapsseq template ........................................................................... 20
      7.5.3. application ............................................................................................................ 21
      7.5.4. Emphasis with boldface ..................................................................................... 21
      7.5.5. callout graphics ..................................................................................................... 21
      7.5.6. firstterm ............................................................................................................. 21
      7.5.7. guibutton, guiicon, guilabel, and guimenu ..................................................... 22
      7.5.8. The keysym element ............................................................................................ 22

\footnote{1}{http://www.nmt.edu/tcc/doc/docbook43/ims/}
\footnote{2}{http://www.nmt.edu/tcc/doc/docbook43/ims/tccstyle.pdf}
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 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.
  
  These style sheets can be downloaded from the SourceForge repository.

- 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.

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.

http://www.nmt.edu/~shipman/soft/litprog/
http://www.w3.org/TC/xsl
http://sourceforge.net/project/showfiles.php?group_id=21935
http://www.w3.org/TC/xslt
Documentation for the *DocBook-XSL Stylesheets* is rather skeletal. Fortunately, there is an excellent book that describes the customization process in detail:


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

2. Online files

Numerous files pertaining to this project are available online.

- `tcc_html.xsl`\(^7\): The localized HTML stylesheet.
- `html-titlepage.xml`\(^8\): The XML file that describes the layout of the HTML title page.
- `tcc_fo.xsl`\(^9\): The localized FO (PDF) stylesheet.
- `fo-titlepage.xml`\(^10\): The XML file that describes the layout of the PDF title page.
- The Makefile\(^11\) for this project.
- `tccstyle.xml`\(^12\): DocBook-XML source file for this document.

3. 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*\(^13\) 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\(^14\). 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*\(^15\).

4. 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*.

---

\(^7\) [http://www.nmt.edu/tcc/doc/docbook43/ims/tcc_html.xsl](http://www.nmt.edu/tcc/doc/docbook43/ims/tcc_html.xsl)
\(^11\) [http://www.nmt.edu/tcc/doc/docbook43/ims/Makefile](http://www.nmt.edu/tcc/doc/docbook43/ims/Makefile)
\(^12\) [http://www.nmt.edu/tcc/doc/docbook43/ims/tccstyle.xml](http://www.nmt.edu/tcc/doc/docbook43/ims/tccstyle.xml)
\(^13\) [http://www.w3.org/TC/xslt](http://www.w3.org/TC/xslt)
\(^14\) [http://www.w3.org/TC/xslt](http://www.w3.org/TC/xslt)
Specifically, you can do any of these things:

- If you are writing TCC documentation, follow the procedures in *Writing documentation with DocBook-XML 4.3*.

- 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/docbook43/`

  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:
  
  `/u/www/docs/tcc/doc/docbook43/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.

5. **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. Then, one runs it through `xsltproc` using a special stylesheet, `template/titlepage.xsl` in the *DocBook-XSL Stylesheets*, that converts it into an XSL file, which is the part of the local customization layer that describes the title page layout. Here, we are using an XSL file to transform an XML file into XSL.

Our title customization files are called `html-titlepage.xml` and `fo-titlepage.xml` for the HTML and FO versions, respectively. The transformed files are `html-titlepage.xsl` and `fo-titlepage.xsl`. These transformed files are referenced by the base stylesheets.

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/docbook43/` on infohost.

- The `tcc_html.xsl` file is the root customization file for producing HTML output. See Section 7, "tcc_html.xsl: HTML customization layer" (p. 7).

- The `html-titlepage.xml` file describes the HTML title page format. It is transformed by the `template/titlepage.xsl` file to produce the `html-titlepage.xsl` file that actually generates the HTML title page. See Section 9, "html-titlepage.xml: HTML title page template" (p. 26).

---

http://www.nmt.edu/tcc/help/pubs/docbook43/
6. 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 XSL stylesheet 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. That stylesheet lives here:

```
/u/www/docs/tcc/doc/docbook43/mss/template/titlepage.xsl
```

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://nw-alsh.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.

Here’s a picture of the dependencies reflected in this project’s Makefile.

---

17 http://www.nmt.edu/tcc/doc/docbook43/ims/Makefile
Caution

Note the feedback in this process: the `tcc_html.xsl` and `tccFo.xsl` files extracted from the document are used to build the HTML and PDF renderings.

If you manage to generate copies of these files that don’t work, just fix the problems and then issue the command `make code` before you type the full `make`.

Also, subdirectory `good/` under this directory should have some working copies you can use in an emergency.

7. `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 namespace.
- 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.

```xml
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:date="http://exslt.org/dates-and-times"
```
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).

A number of common strings such as “TCC Help System” are declared in a separate file, fo_params.xsl. This allows non-TCC users to supply their own branding for our style features. For the contents of this file, see Section 8, “The html_params.xsl file: HTML branding” (p. 24).

The remaining local customizations are divided into these sections:

- Section 7.1, “HTML general page layout” (p. 8).
- Section 7.2, “HTML title page and table of contents” (p. 9).
- Section 7.3, “HTML headers and footers” (p. 10).
- Section 7.4, “HTML section and subsection headings” (p. 19).
- Section 7.5, “HTML inline element customizations” (p. 19).
- Section 7.6, “HTML block element customizations” (p. 22).

### 7.1. 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">

```
<!-html.stylesheet-->```
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.

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

In early 2009, a lot of browsers started rendering the non-break space character (&nbsp;) as a 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.

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

7.2. 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 9, “html-titlepage.xml: HTML title page template” (p. 26).

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

This next bit is high wizardry. We often use tags such as code 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 code and userinput:

```xml
<xsl:template match="code|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 or code template to the element instead.

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

```xml
<!--Render selected inline elements in titles-->
<xsl:template match="filename|sgmltag|userinput|varname|code" mode="no.anchor.mode">
  <xsl:apply-templates select="." />
</xsl:template>
```

7.3. HTML headers and footers

Customization of the TCC header and footer content follows the TCC Documentation Guidelines\(^\text{18}\). The intent is to mimic the style of HTML pages generated by other tools, such as PyStyler\(^\text{19}\) [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.

7.3.1. 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.

```xml
<!--header.navigation: Navigation links at page top-->
<xsl:template name="header.navigation">
  <xsl:param name="prev" select="/foo"/>
  <xsl:param name="next" select="/foo"/>
  <xsl:param name="nav.context"/>
</xsl:template>
```

\(^\text{18}\) [http://www.nmt.edu/tcc/doc/plan/]
\(^\text{19}\) [http://www.nmt.edu/tcc/help/pubs/pystyler/pystyler.pdf]
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.

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

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 `top.nav.bar` template, and passing our `prev`, `next`, and `home` nodes to it. See Section 7.3.2, “The `top.nav.bar` template” (p. 12).

```xml
<!--Top-of-page navigation bar-->
<xsl:call-template name="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 organization’s 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.

```xml
<div class="navheader">
  <table width="100%" summary="Navigation header">
    <tr valign="top">
      <td align="left" valign="top">
        <h1>
          <xsl:apply-templates select="$home" mode="object.title.markup"/>
        </h1>
      </td>
      <td>
        <img alt="organizational logo" src="{$html.logo}"/>
      </td>
    </tr>
  </table>
</div>
```

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.

```xml
<hr/>
</div>
</xsl:template>
```
7.3.2. The top.nav.bar template

The 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.

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.

```
<--Generates the top-of-page navigational bar-->
<xsl:template name="top.nav.bar">
  <xsl:param name="prev"/>
  <xsl:param name="next"/>
  <xsl:param name="home"/>

  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.

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

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.
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.

```xml
<!--“Prev” link-->
<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.

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

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

```xml
<!--“Contents” link-->
<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>
```

The Help link is optional. The help.url parameter defines the URL, and the help.short.name parameter is the link text. If help.url is not defined, there will be no such link. See Section 8.4, “help.url: Help link URL” (p. 25) and Section 8.3, “help.short.name: Help link short text” (p. 25).

```xml
<!--“Help” link (optional)-->
<xsl:if test="$help.url">
  <a href="{$help.url}" 
    <xsl:value-of select="$help.short.name"/>
  </a>
</xsl:if>
<xsl:text> / </xsl:text>
```
Lastly we output a link to the homepage: its link text is home.short.name, and its target is home.url. See Section 8.6, “home.short.name: Homepage short link text” (p. 26) and Section 8.7, “home.url: Homepage URL” (p. 26).

```xml
<!--“Home” link-->
<a href="{$home.url}"/>
   <xsl:value-of select="$home.short.name" />
</a>
</div>
</xsl:template> <!-- End of top.nav.bar -->
```

### 7.3.3. 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 7.3.1, “The header.navigation template” (p. 10). It has the same two internal variables: home, the article node of the document; and up, the parent node.

```xml
<!--footer.navigation--><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.

```
```xml
<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.

```xml
<!--Navigation links at the page bottom--> <div class="navfooter">
```

The standard TCC page-bottom links are output by a separate template; see Section 7.3.4, “The bot.links template” (p. 15).

```xml
<xsl:call-template name="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 7.3.5, “The colophon template” (p. 17).
7.3.4. The 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.

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.

Next comes the “Contents” link, similarly, wrapped in a div with class="bot-contents".
The “Previous” link works just like the “Next” link.

The Help link is optional. When the help.url parameter is defined, the link points there, with the link text defined by help.long.name. See Section 8.4, “help.url: Help link URL” (p. 25) and Section 8.2, “help.long.name: Full link text for the help link” (p. 25).

The text for the Home link is given by the home.url parameter, and its target by the home.url parameter. See Section 8.5, “home.long.name: Full home link text” (p. 25) and Section 8.7, “home.url: Homepage URL” (p. 26).
7.3.5. The 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.

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.

To generate all the author names, we call a separate template; see Section 7.3.6, “The author.colophon.mode template” (p. 18).

The mailto: link forwards mail to the email address defined in Section 8.1, “email.to.author: Email address for reader comments” (p. 25). This ends the HTML address element.
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.

We use Server Side Includes to generate the page’s own URL; see Using HTML Server Side Includes (SSI). 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.

For CSS markup, see Section 10.4.9, “div.colophon-url” (p. 33) and Section 10.3.1, “span.colophon-uri” (p. 30).

7.3.6. 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".

---

http://exslt.org/date/index.html
7.4. 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.

```xml
<!--Turn on section numbering-->
<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.

```xml
<!--Suppress chunk generation messages-->  
<xsl:param name="chunk.quietly" select="1"/>
```

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.

```xml
<!--Name the HTML files after the corresponding section ID.-->  
<xsl:param name="use.id.as.filename">1</xsl:param>
```

7.5. HTML inline element customizations

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

7.5.1. 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.

```xml
<--inline.italicssansseq: Italic sans-serif font-->
<xsl:template name="inline.italicsansseq">
  <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:call-template>
</xsl:param>
</xsl:template>
```

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\(^\text{22}\)) 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.

```xml
<--Drop an anchor, and implement XLink features-->
<xsl:param name="content">
  <xsl:call-template name="anchor"/>
  <xsl:call-template name="simple.xlink">
    <xsl:apply-templates />
  </xsl:call-template>
</xsl:param>
```

The content is then wrapped in an HTML `span` element with 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.

```xml
<--Wrap the content in a span with the DocBook element name-->
<span class="{local-name(.)}">
  <xsl:copy-of select="$content"/>
</span>
</xsl:template>
```

### 7.5.2. The **inline.smallcapsseq** template

The next template is **inline.smallcapsseq**, 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.

```xml
<--inline.smallcapsseq: Caps-and-small-caps font-->
<xsl:template name="inline.smallcapsseq">
  <xsl:param name="content">
    <xsl:call-template name="anchor"/>
  </xsl:param>
</xsl:template>
```

\(^\text{22}\) [http://www.w3.org/TR/2001/REC-xlink-20010627/](http://www.w3.org/TR/2001/REC-xlink-20010627/)
7.5.3. **application**

We want DocBook application elements to be set in italic sans-serif font. For the font markup template, see Section 7.5.1, “The inline.italicsansseq template” (p. 19).

```xml
<!--application: Use italic sans font-->
<xsl:template match="application">
  <xsl:call-template name="inline.italicsansseq"/>
</xsl:template>
```

7.5.4. **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.

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

7.5.5. **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, ...

```xml
<!--Define the subdirectory with graphics for programlistco-->
<xsl:param name="callout.graphics.path">callouts/</xsl:param>
```

7.5.6. **firstterm**

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

```xml
<!--firstterm: Use italics-->
<xsl:template match="firstterm">"
7.5.7. **guibutton**, **guiicon**, **guilabel**, and **guimenu**

These elements are rendered in italic sans-serif font. See Section 7.5.1, “The inline.italicsansseq template” (p. 19).

```xml
<!--gui items: Use italic sans font--> 
<xsl:template match="guibutton|guiicon|guilabel|guimenu">
    <xsl:call-template name="inline.italicsansseq"/>
</xsl:template>
```

7.5.8. **The keysym element**

The DocBook keysym element denotes names of keys on the keyboard. We render those in small caps in HTML. See Section 7.5.2, “The inline.smallcapsseq template” (p. 20).

```xml
<!--keysym: Use caps-and-small caps for key names-->  
<xsl:template match="keysym">
    <xsl:call-template name="inline.smallcapsseq"/>
</xsl:template>
```

7.5.9. **HTML inline math**

Page 324 of Section 1.1, “Bob Stayton’s DocBook XSL: The complete guide” (p. 3) recommends a style rule for typesetting simple inline math formulae: enclose them in `<phrase role='math'>...<phrase>`. The corresponding rule sets letters in italics:

```xml
<!--Math: Use italics-->  
<xsl:template match="phrase[role='math']">
    <xsl:call-template name="inline.italicseq"/>
</xsl:template>
```

7.6. **HTML block element customizations**

At this point there is only one customized block element, **programlisting**.

7.6.1. **The programlisting element**

When using the literate programming features of the local DocBook toolchain, we look for `programlisting` elements that have a `role` attribute that start with "outFile:". In that case:

1. We output the file name from the `role` attribute, in small type, right-justified, to identify the file to which the following block is written.
2. To format the literate block, we call the template from Section 7.6.2, “The html-lit-block template: Formatting a literate block” (p. 23).

Here is the template that handles HTML *programlisting* elements. If it is a literate code block, that is, if the *role* attribute is present and its initial characters are "outfile:", we save the rest of the attribute value in the variable *fileName*.

```xml
<!--programlisting: Process code; implement literate features--><xsl:template match="programlisting">
  <xsl:choose>
    <xsl:when test="@role and starts-with(@role, 'outfile:')">
      <xsl:variable name='fileName' select="substring(@role,9)"/>
      The div element used to right-align the output file name is formatted using the Section 10.4.4, "div.code-block-label" (p. 32). Then we call Section 7.6.2, “The html-lit-block template: Formatting a literate block” (p. 23).
    </xsl:when>
    <xsl:otherwise>
      If it is not a literate block, the *apply-imports* operation punts the formatting to the default *programlisting* template.
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

7.6.2. The *html-lit-block* template: Formatting a literate block

This template is a slightly modified version of the *programlisting* template from the stock version in html/verbatim.xsl. We generate an HTML *pre* element with a class='executable' attribute, which is styled by the rule in Section 10.4.1, "pre.executable" (p. 31).

```xml
<!--html-lit-block: Format a literate code block--> <xsl:template name="html-lit-block">
  <xsl:param name="suppress-numbers" select="'0'"/>
  <xsl:variable name="id">
    <xsl:call-template name="object.id"/>
  </xsl:variable>
</xsl:template>
```
7.7. Epilogue for the HTML stylesheet

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

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

8. The html_params.xsl file: HTML branding

This section contains file html_params.xsl, which contains text strings, URLs, and other branding information specific to the TCC. Non-TCC users of these stylesheets may replace these parameters to customize their own branding.

This file is imported by tcc_html.xsl, so it must have a root xsl:stylesheet node with the usual namespace declarations.

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <!-- File html_params.xsl: Branding parameters for HTML. 
  ! Do not edit this file directly. It is mechanically extracted
  ! from the documentation: 
  ! http://www.nmt.edu/tcc/doc/docbook43/ims/
  !-->
```

The parameters are presented here in alphabetical order for easy location.
8.1. email.to.author: Email address for reader comments

```xml
<!--email.to.author-->
<xsl:param name="email.to.author"
    >tcc-doc@nmt.edu</xsl:param >
```

8.2. help.long.name: Full link text for the help link

This parameter defines the link text to a Help link on the bottom of the page, where there is plenty of room.

```xml
<!--help.long.name: Help link text in the page bottom-->
<xsl:param name="help.long.name"
    >Tech Computer Center: Help System</xsl:param >
```

8.3. help.short.name: Help link short text

The help.short.name parameter is the text for the Help link in the top navigational bar, where space is tight.

```xml
<!--Help link text in the top nav bar-->
<xsl:param name="help.short.name"
    >TCC Help System</xsl:param >
```

8.4. help.url: Help link URL

The help.url parameter is the URL of the TCC help system. If this is defined with no value, no “Help” links will be placed on generated Web pages.

```xml
<!--URL of the TCC help system-->
<xsl:param name="help.url"
    >http://www.nmt.edu/tcc/help/</xsl:param >
```

8.5. home.long.name: Full home link text

This is the link text for the Home link at the bottom of the page, where there is plenty of room.

```xml
<!--home.long.name-->
<xsl:param name="home.long.name"
    >About New Mexico Tech</xsl:param >
```
8.6. **home.short.name**: Homepage short link text

The `home.short.name` is the link text for the *Home* navigational link at the top of the page, where space is tight.

```xml
<!--Link text for "Home" links-->
<xsl:param name="home.short.name">
  NM Tech homepage</xsl:param>
```

8.7. **home.url**: Homepage URL

The `home.url` parameter is the URL of *Home* navigational link.

```xml
<!--URL of "Home" links-->
<xsl:param name="home.url">
  http://www.nmt.edu/</xsl:param>
```

8.8. **html.logo**: Organizational logo image

This is the value of the `<image src="...">` attribute for the organizational logo.

```xml
<!--Image source for the organizational logo-->
<xsl:param name="html.logo">
  /tcc/images/logo.png</xsl:param>
```

8.9. **html.stylesheet**: URL of the CSS stylesheet

The `html.stylesheet` parameter is the URL of the CSS-2 stylesheet presented in Section 10, “docbook.css: The CSS stylesheet for the HTML rendering” (p. 29).

```xml
<!--URL of our CSS-2 stylesheet-->
<xsl:param name="html.stylesheet">
  http://www.nmt.edu/tcc/docbook.css</xsl:param>
```

8.10. **html_params.xsl**: Epilogue

Finally, here is the closing tag for the `html_params.xsl` file.

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

9. **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
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.

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. We removed most of them: for example, author information will appear in the page colophon. However, we do retain the **abstract** element in the main title; this is the standard location for pointers to the HTML and PDF forms of the document.

```
  <t:titlepage-content t:side="recto">
    <abstract/>
    <!--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” 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 **titlepage.templates.xml** file:
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:

```xml
<tt:titlepage-content t:side="verso">
  <!--hr removed here; reinstate if there is any content in the titlepage-content element above.-->
</tt: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 `tt:titlepage-separator` element that used to contain the horizontal rule:

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

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

```xml
<tt:titlepage-before t:side="recto">
  <!--hr removed here; reinstate if there is any content in the titlepage-content element above.-->
</tt:titlepage-before>
<tt:titlepage-before t:side="verso">
  <!--hr removed here; reinstate if there is any content in the titlepage-content element above.-->
</tt:titlepage-before>
</tt:titlepage>
```

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 `tt: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 `tt:titlepage t:element="book" t:wrapper="div" class="titlepage">`, insert it here, and customize it as appropriate.
10. docbook.css: The CSS stylesheet for the HTML rendering

The term “stylesheet” has two meanings in the context of this document.

• XSLT\(^{23}\) stands for Extensible Stylesheet Language Transformations, which is the language used in the various .xsl files in this document. In this sense, an XSLT script is a program that transforms an XML file into some other form.

• The file docbook.css, shown below, uses CSS Level 2\(^{24}\), the Cascading Style Sheets standard to style the appearance or branding of HTML pages that come out of this DocBook toolchain. See also the local reference guide, Styling Web pages with CSS-2\(^{25}\).

For the rationale behind the design of TCC web pages, see the TCC Documentation Guidelines\(^{26}\).

10.1. docbook.css: Prologue

Here is the docbook.css file. Install the working copy at this location:

```
/u/www/docs/tcc/docbook.css
```

which has this URL:

```
http://www.nmt.edu/tcc/docbook.css
```

The file starts with a header that points back at this documentation. In CSS, comments are enclosed within “//* ... */” characters.

```
/* Stylesheet for NM Tech Computer Center web pages from DocBook
 * Do not edit this file. It is extracted automatically from the
documentation here:
 * http://www.nmt.edu/tcc/doc/docbook43/ims/
 */
```

10.2. docbook.css: General page layout

The first rule applies to the HTML `body` tag, and sets up the general color scheme. The background is a pale tan and the text a coffee-brown, appropriate for a university in the desert. The `max-width` rule keeps paragraph widths to a reasonable maximum, yet still allows room for the standard 75-character width limit for verbatim output. The `margin-left` rule keeps the text from slamming up against the left side.

```
body
{
  background-color: #ffffff;
  color: #663300;
  max-width: 50em;
  margin-left: 1em;
}
```
We don't use HTML h1 titles, but the various levels of DocBook section tags do generate HTML h2, h3, and h4 titles. To make these titles stand out better, they are placed inside a ruled box. The background color inside the box is a darker tan, with the borders using the text color. We don't change the font sizes, which should be smaller at each level, but we do use less padding around the h3 and h4 headings.

```css
h2.title /* sect1 titles */
{
  background-color: #ffe2a0;
  border-style: solid;
  border-color: #663300;
  padding: 4px;
}

h3.title, h4.title /* sect2 and sect3 titles */
{
  background-color: #ffe2a0;
  border-style: solid;
  border-color: #663300;
  padding: 2px;
}
```

Next we stipulate a minimum padding within table cells.

```css
th, td
{
  padding: 5px;
}
```

### 10.3. Docbook.css: Inline elements

We use a number of different HTML span elements to mark up segments of text that are inline, that is, they may be included within a paragraph. They are shown here in alphabetical order by the name of the associated class attribute.

```css
/*================================================================
* Inline elements, sorted by class.
*/
```

#### 10.3.1. span.colophon-uri

Sets the URL in the colophon section in very small, monospaced font. For its invocation, see Section 7.3.5, “The colophon template” (p. 17).

```css
span.colophon-uri
{
  font-family: monospace;
  font-size: xx-small;
}
```

#### 10.3.2. span.application

We use a slanted sans-serif font so application names stand out from the background.

```css
span.application
{
  font-family: sans-serif;
}
10.3.3. The span.gui{button|icon|label|menu} group

The general TCC convention is to set the names of GUI elements in oblique sans-serif bold type. We also set them in green type. See Section 7.5.7, “guibutton, guiicon, guilabel, and guimenu” (p. 22).

```css
span.guibutton, span/guiicon, span.guilabel, span.guimenu
{
    font-family: sans-serif;
    font-style: italic;
    font-weight: bold;
    color: #238b22; /* Forest green */
}
```

10.3.4. span.term

Within a DocBook variablelist, set the term in bold type.

```css
span.term
{
    font-weight: bold;
}
```

10.4. docbook.css: Block elements

Here are the CSS rules for block-level elements in the HTML rendering. First are the three types of pre (preformatted, monospaced) element.

- Section 10.4.1, “pre.executable” (p. 31). This is for blocks of code that use the literate programming features.
- Section 10.4.2, “pre.programlisting and pre.screen” (p. 32).

Rules for div elements are enumerated in alphabetical order by the value of the class attribute.

- Section 10.4.3, “div.abstract” (p. 32).
- Section 10.4.4, “div.code-block-label” (p. 32).
- Section 10.4.5, “div.colophon” (p. 32).
- Section 10.4.6, “div.colophon-author” (p. 33).
- Section 10.4.7, “div.colophon-date” (p. 33).
- Section 10.4.8, “div.colophon-mail-to” (p. 33).
- Section 10.4.9, “div.colophon-url” (p. 33).

```css
/*================================================================
* Block elements, sorted by class.
*/
```

10.4.1. pre.executable

This is a special block for displaying literate code fragments; see Section 7.6.1, “The programlisting element” (p. 22). We use a pale green background and enclose the block in a hairline black border, and
put five pixels of padding around the contents. The margin-top is reduced to one point so that the preceding div.code-block-label, which names the output file, is very close to the code block.

```css
pre.executable
{
    margin-top: 1px;
    background-color: #ddeecc;
    border: 1px solid;
    padding: 5px;
}
```

### 10.4.2. pre.programlisting and pre.screen

These block elements are to be set verbatim in monospaced type. We add a light gray screen so that these are visually distinct from ordinary text. We also enclose them in a hairline border and add a bit of padding.

```css
pre.programlisting, pre.screen
{
    background-color: #e0e0e0;
    border: 1px solid;
    padding: 5px;
}
```

### 10.4.3. div.abstract

This element encloses the abstract that appears above the table of contents on the top-level HTML page. The rule sets this text with wider margins.

```css
div.abstract
{
    margin-left: 6em;
    margin-right: 6em;
}
```

### 10.4.4. div.code-block-label

This element encloses the tiny label that identifies each block of literate code, hovering just above the top right corner of the pre.executable containing the code.

```css
div.code-block-label
{
    text-align: right;
    font-size: xx-small;
    font-family: monospace;
}
```

### 10.4.5. div.colophon

Used to enclose the entire colophon section. See Section 7.3.5, “The colophon template” (p. 17). Since this section is of interest to few, it is set in small type.
10.4.6. **div.colophon-author**

Encloses the names of authors in the colophon. See Section 7.3.6, “The author.colophon.mode template” (p. 18).

```css
div.colophon-author
{
  font-style: italic;
}
```

10.4.7. **div.colophon-date**

Encloses the date of last update in the colophon section. See Section 7.3.5, “The colophon template” (p. 17).

```css
div.colophon-date
{
  font-size: x-small;
}
```

10.4.8. **div.colophon-mail-to**

Encloses the mailto: link in the colophon section. See Section 7.3.5, “The colophon template” (p. 17).

```css
div.colophon-mail-to
{
  font-family: monospace;
  font-size: xx-small;
}
```

10.4.9. **div.colophon-url**

Encloses the URL in the colophon section. See Section 7.3.5, “The colophon template” (p. 17).

```css
div.colophon-url
{
  font-size: xx-small;
}
```

11. **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 7, “tcc_html.xsl: HTML customization layer” (p. 7)), with one difference: it includes the XSL-FO namespace as prefix “fo:”.
We import two files: the stock FO templates and the parameters defined in Section 12, “The fo_params.xsl file: PDF branding” (p. 51). The output method to XML.

The PDF customizations are divided into these sections:

- Section 11.1, “General page layout” (p. 34).
- Section 11.2, “Templates for title elements” (p. 36).
- Section 11.3, “PDF headers and footers” (p. 39).
- Section 11.4, “PDF section and subsection headings” (p. 43).
- Section 11.5, “PDF inline element customizations” (p. 44).
- Section 11.6, “PDF block element customizations” (p. 48).

### 11.1. 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 3, “Skills you will need” (p. 4).

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.

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".
In the old DocBook 4.2 customizations, titles were unindented relative to the left margin of the page body. However, since version 1.68.1 of the style sheets, the titles are not indented; instead, the body indentation is set by the parameter `body.start.indent`, which defaults to 4pc. We'll reduce that a bit, to 3pc.

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

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`.

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 parameter is `region.before.extent`.

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 trade-off: 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.)

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.
11.2. 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 7.2, “HTML title page and table of contents” (p. 9).
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 6, “Title page customization: XSLT that builds XSLT” (p. 6).
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 11, “tcc_fo.xsl: PDF customization layer” (p. 33).
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).

```xml
<!--component.title: Customize the title page-->
<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:template>
```
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.

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).

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

The right-hand column contains the TCC logo graphic; see Section 12.1, “pdf.logo: Image source for the PDF organizational logo” (p. 52). The `padding-left` attribute ensures a separation from the title.
And that's the end of the 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**.

```xml
<!--revhistory: Generate a timestamp block-->
<xsl:template match="revhistory" mode="titlepage.mode">
  <xsl:call-template name="timestamp.block"/>
</xsl:template>

<!--timestamp.block: Display a timestamp as YYYY-MM-DD hh:mm-->
<xsl:template name="timestamp.block">
  <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>  
<!--timestamp.block-->"'
```

### 11.3. 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.

```xml
<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.

```xml
<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.

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

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

```xml
<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:

<table>
<thead>
<tr>
<th>pageclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>titlepage</td>
<td>The title page.</td>
</tr>
<tr>
<td>lot</td>
<td>List-of-titles pages, including the table of contents, list of figures, and such.</td>
</tr>
<tr>
<td>front</td>
<td>Front matter: preface, dedication, etc.</td>
</tr>
<tr>
<td>body</td>
<td>Main content pages.</td>
</tr>
<tr>
<td>back</td>
<td>Back matter such as appendices and glossaries.</td>
</tr>
<tr>
<td>index</td>
<td>Book-style index pages.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>First page of this class.</td>
</tr>
<tr>
<td>odd</td>
<td>An odd-numbered, nonfirst page.</td>
</tr>
<tr>
<td>even</td>
<td>An even-numbered, nonfirst page.</td>
</tr>
<tr>
<td>blank</td>
<td>A blank page added to even out the page count.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Case</th>
<th>left</th>
<th>center</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pageclass is titlepage</strong></td>
<td>blank</td>
<td>blank</td>
<td>blank</td>
</tr>
<tr>
<td>Not titlepage, single-sided</td>
<td></td>
<td>title</td>
<td>folio</td>
</tr>
<tr>
<td>Not titlepage, double-sided, odd/first pages</td>
<td>title</td>
<td>folio</td>
<td>logo</td>
</tr>
<tr>
<td>Not titlepage, double-sided, even/blank pages</td>
<td>title</td>
<td>folio</td>
<td>logo</td>
</tr>
</tbody>
</table>

- “title” is the document’s title.
“folio” is the page number.

“logo” is the organization name defined in Section 12.2, “pdf.org.name: Organization’s name for the running footer” (p. 52).

“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.

```xml
<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="''"/>

  <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.)
  </fo:block>

  <xsl:choose>
    <xsl:when test="$pageclass = 'titlepage'">
      <!--no footer on title pages-->
    </xsl:when>
    <xsl:otherwise> <!--Not a title page-->
      <xsl:choose>
        <xsl:when test="$double.sided = 0">
          <!-- Single-sided -->
        </xsl:when>
        <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="$position = 'right'">
          <xsl:value-of select="$pdf.org.name"/>
        </xsl:when>
      </xsl:choose>
    </xsl:otherwise> <!-- Single-sided -->
  </xsl:choose>
</xsl:template>
```

The entire footer content is wrapped in a block container.
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.

```xml
<xsl:otherwise> <!--Double-sided-->
  <xsl:choose>
    <xsl:when test="$position = 'left'"> <!-- left/odd -->
      <xsl:when test="$sequence = 'even' or $sequence = 'blank'">
        <fo:page-number/>
      </xsl:when>
      <xsl:otherwise> <!-- left/odd -->
        <xsl:value-of select="$pdf.org.name" />
      </xsl:otherwise>
    </xsl:when>
    <xsl:when> <!-- left/odd -->
      <xsl:choose>
        <xsl:when test="$sequence = 'even' or $sequence = 'blank'">
          <fo:page-number/>
        </xsl:when>
        <xsl:otherwise> <!-- left/odd -->
          <xsl:value-of select="$pdf.org.name" />
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
  </xsl:choose>
</xsl:otherwise> <!-- Double-sided -->
</xsl:choose>
</xsl:template>
```

The center position always contains the running title.

```xml
<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.

```xml
<xsl:when test="$position = 'right'"> <!-- Double-sided -->
  <xsl:choose>
    <xsl:when test="$sequence = 'even' or $sequence = 'blank'">
      <xsl:value-of select="$pdf.org.name" />
    </xsl:when>
    <xsl:otherwise> <!-- Double-sided -->
      <xsl:choose>
        <xsl:when> <!-- Double-sided -->
          <xsl:apply-templates select="." mode="titleabbrev.markup"/>
        </xsl:when>
        <xsl:otherwise> <!-- Not a title page--> 
          <xsl:choose>
            <xsl:when> <!-- Not a title page--> 
              <xsl:apply-templates select="." mode="titleabbrev.markup"/>
            </xsl:when>
            <xsl:otherwise> <!-- Not a title page--> 
              <xsl:choose>
                <xsl:when> <!-- Not a title page--> 
                  <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                </xsl:when>
                <xsl:otherwise> <!-- Not a title page--> 
                  <xsl:choose>
                    <xsl:when> <!-- Not a title page--> 
                      <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                    </xsl:when>
                    <xsl:otherwise> <!-- Not a title page--> 
                      <xsl:choose>
                        <xsl:when> <!-- Not a title page--> 
                          <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                        </xsl:when>
                        <xsl:otherwise> <!-- Not a title page--> 
                          <xsl:choose>
                            <xsl:when> <!-- Not a title page--> 
                              <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                            </xsl:when>
                            <xsl:otherwise> <!-- Not a title page--> 
                              <xsl:choose>
                                <xsl:when> <!-- Not a title page--> 
                                  <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                </xsl:when>
                                <xsl:otherwise> <!-- Not a title page--> 
                                  <xsl:choose>
                                    <xsl:when> <!-- Not a title page--> 
                                      <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                    </xsl:when>
                                    <xsl:otherwise> <!-- Not a title page--> 
                                      <xsl:choose>
                                        <xsl:when> <!-- Not a title page--> 
                                          <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                        </xsl:when>
                                        <xsl:otherwise> <!-- Not a title page--> 
                                          <xsl:choose>
                                            <xsl:when> <!-- Not a title page--> 
                                              <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                            </xsl:when>
                                            <xsl:otherwise> <!-- Not a title page--> 
                                              <xsl:choose>
                                                <xsl:when> <!-- Not a title page--> 
                                                  <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                                </xsl:when>
                                                <xsl:otherwise> <!-- Not a title page--> 
                                                  <xsl:choose>
                                                    <xsl:when> <!-- Not a title page--> 
                                                      <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                                    </xsl:when>
                                                    <xsl:otherwise> <!-- Not a title page--> 
                                                      <xsl:choose>
                                                        <xsl:when> <!-- Not a title page--> 
                                                          <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                                        </xsl:when>
                                                        <xsl:otherwise> <!-- Not a title page--> 
                                                          <xsl:choose>
                                                            <xsl:when> <!-- Not a title page--> 
                                                              <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                                            </xsl:when>
                                                            <xsl:otherwise> <!-- Not a title page--> 
                                                              <xsl:choose>
                                                                <xsl:when> <!-- Not a title page--> 
                                                                  <xsl:apply-templates select="." mode="titleabbrev.markup"/>
                                                                </xsl:when>
                                                                <xsl:otherwise> <!-- Not a title page--> 
                                                                  <xsl:choose>
                                                                    <xsl:when> <!-- Not a title page--> 
                                                                      <xsl:apply-templates select="." mode="titleabbrev.markup"/>
11.3.1. 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.

11.4. PDF section and subsection headings

In stylesheet versions before 1.68.0, the body text was not considered to be indented. To make headings stand out, they were negatively indented by setting the title.margin.left parameter to a negative value.

However, in our transition from DocBook 4.2 to 4.3, we moved from stylesheet version 1.65.1 to 1.69.1. Currently, the body indentation is set by body.start.indent (see Section 11.1, “General page layout” (p. 34)), and titles are considered to be not indented. Therefore, we do not set the title.margin.left parameter anymore.

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

```xml
<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.

```xml
<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.

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

In the stock stylesheets, the space before level 1 and level 2 section titles was about the same as the space after those titles. This violates one of Robin Williams's style rules: titles should be closer to the sections that follow them than to the sections before, so that they have a visual association.

The stock values of `space-before` for section titles are 0.8, 1.0, and 1.2 ems, for minimum, optimum, and maximum, respectively. We'll roughly double these values. This template comes from `fo/param.xsl`.

11.5. 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 7.5, “HTML inline element customizations” (p. 19)), we define templates similar to the stock inline.italicseq template and others defined in fo/inline.xsl.

11.5.1. 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>
```

11.5.2. 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>
```

11.5.3. 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>
```

11.5.4. 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.
11.5.5. 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 programlisting 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 documentation\(^\text{27}\) 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.

```xml
<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.

```xml
<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.

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

11.5.6. firstterm

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

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

\(^\text{27}\) http://www.nmt.edu/tcc/help/pubs/docbook43/
11.5.7. **keysym**

The TCC Document Plan specifies caps-and-small-caps font for names of keyboard keys, but as discussed in Section 11.5.2, “The *inline.smallcaps* template” (p. 45), there is no suitable public domain font at this point. So we use italic sans-serif font instead.

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

11.5.8. **Inline math**

This rule italicizes letters in inline mathematical formulae. For the rationale and source, see Section 7.5.9, “HTML inline math” (p. 22).

```xml
<xsl:template match="phrase[@role='math']">
    <xsl:call-template name="inline.italicseq"/>
</xsl:template>
```

11.5.9. **ulink**

When translating a *ulink* element, the default PDF rendering is to place the URL in square brackets following the link text. This is not only ugly and intrusive, but tends to lead to ugly line breaks and paragraph formatting, too.

A better approach is to set the *ulink.footnotes* parameter to 1. This moves such URLs to footnotes.

```xml
<xsl:param name="ulink.footnotes">1</xsl:param>
```

11.5.10. **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.

```xml
<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>
```
11.6. PDF block element customizations

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

11.6.1. 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.

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

  <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>
```

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.

11.6.2. 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.

```xml
<xsl:param name="shade.verbatim" select="1"/>
```
Program listings will look best when lines are limited to 75 characters. However, in those cases where someone has a longer line, we don’t want to chop it off.

Chapter 26 of Stayton, in the chapter on program listings, has a section entitled “Breaking long lines.” Assuming that the line has some spaces in it somewhere, the following change to the monospace.verbatim.properties attribute set will wrap long lines on a space and insert a right angle quote (») character at the end of all but the last.

We also use a tiny margin (half a point) around all verbatim displays. If we don’t do this, programlisting elements inside table entry elements will overlap adjacent cells.

```xml
<xsl:attribute-set name="monospace.verbatim.properties">
  <xsl:attribute name="wrap-option">wrap</xsl:attribute>
  <xsl:attribute name="hyphenation-character">&amp;#x00bb;</xsl:attribute>
  <xsl:attribute name="margin">0.5pt</xsl:attribute>
</xsl:attribute-set>
```

For the display of literate code fragments, we define an additional attribute set named lit.shading.style that specifies a pale green background.

```xml
<xsl:attribute-set name="lit.shading.style">
  <xsl:attribute name="background-color">#eef8e8</xsl:attribute>
</xsl:attribute-set>
```

The template below changes the formatting of the programlisting element when it is a literate block, that is, when it has a role attribute that starts with "outFile:"

```xml
<xsl:template match="programlisting">
  <xsl:choose>
    <xsl:when test="@role and starts-with(@role, 'outFile:')">
      <xsl:variable name='fileName' select="substring(@role,9)"/>
      <fo:block keep-with-next.within-page="always" text-align="right" margin-top="3pt" margin-bottom="0pt" line-height="6pt" font-family="monospace" font-size="6pt">
        <xsl:value-of select="$fileName"/>
      </fo:block>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="fo-lit-block">
        <xsl:with-param name="file-name" select="$fileName"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:when>
</xsl:template>
```

First we extract the output file name from the role attribute and store it in variable fileName.

```xml
<xsl:variable name='fileName' select="substring(@role,9)"/>
```

This fo:block displays the file name, right-justified, in tiny type, just above the literate block.

```
<fo:block keep-with-next.within-page="always" text-align="right" margin-top="3pt" margin-bottom="0pt" line-height="6pt" font-family="monospace" font-size="6pt">
  <xsl:value-of select="$fileName"/>
</fo:block>
```

Formatting of the contents of the literate block are handled by Section 11.6.3, “The fo-lit-block template: Format a literate block” (p. 50).

```xml
<xsl:call-template name="fo-lit-block">
  <xsl:with-param name="file-name" select="$fileName"/>
</xsl:call-template>
```

If the programlisting element is not a literate block, the apply-imports step punts to the default rendering.
11.6.3. The **fo-lit-block** template: Format a literate block

This template formats a `programlisting` element that contains a literate code fragment. It has one required parameter named `file-name`, which must contain the name of the file to be displayed.

```xml
<xsl:template name="fo-lit-block">
  <xsl:param name="file-name">file-name???</xsl:param>
  <xsl:choose>
    <xsl:when test="$suppress-numbers = '0'
               and @linenumbering = 'numbered'
               and $use.extensions != '0'
               and $linenumbering.extension != '0'">
      <xsl:call-template name="number.rtf.lines">
        <xsl:with-param name="rtf">
          <xsl:apply-templates/>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

This template is a slightly modified version of the `programlisting` template from the stock `html/verbatim.xsl` file. The first few lines, here, are unaltered from the stock template.

```xml
<xsl:param name="suppress-numbers" select="'0'"/>
<xsl:variable name="id">
  <xsl:call-template name="object.id"/>
</xsl:variable>
<xsl:variable name="content">
  <xsl:choose>
    <xsl:when test="$hyphenate.verbatim != 0 and
                   function-available('exsl:node-set')">
      <fo:block id="{$id}" margin-top="-3pt" white-space-collapse='false'
                white-space-treatment='preserve' linefeed-treatment='preserve'
                xsl:use-attribute-sets="monospace.verbatim.properties
                                         lit.shading.style">
        <xsl:choose>
          <xsl:when test="$hydrate.verbatim != 0 and
                         function-available('exsl:node-set')">
            ...<xsl:apply-templates/>
          </xsl:when>
          <xsl:otherwise>
            ...<xsl:apply-templates/>
          </xsl:otherwise>
        </xsl:choose>
      </fo:block>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

The stock template uses the `shade.verbatim.style` attribute set. Our version substitutes the `lit.shading.style` attribute set defined above. We also omit the code for the case where `$shade.verbatim` was false. Everything else here is from the stock template.
11.6.4. table

By default, formal table elements (as opposed to informaltable elements) cannot be split across pages. For publications with many long tables (especially our local Tkinter reference) this makes lots of huge blank spots. The default value for keep-together.within-column is always; the value auto allows these tables to be split anywhere (except between the caption and table).

```
<xsl:attribute-set name="table.properties">
  <xsl:attribute name="keep-together.within-column">auto</xsl:attribute>
</xsl:attribute-set>
```

11.6.5. 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"/>
```

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>
```

11.7. PDF epilogue

Here's the end tag for xsl:stylesheet.

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

12. The fo_params.xsl file: PDF branding

The fo_params.xsl file presented here defines various text strings, URLs, and other branding information specific to the TCC. Non-TCC users may override these parameters to produce a document with different branding.
The root element is `xsl:stylesheet`, with the usual namespace declarations for XSLT.

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!--File fo_params.xsl: Branding parameters for PDF.
  ! Do not edit this file directly. It is mechanically extracted
  ! from the documentation:
  ! http://www.nmt.edu/tcc/doc/docbook43/ims/
  !-->

  The parameters are defined here in alphabetical order.

12.1. pdf.logo: Image source for the PDF organizational logo

```xml
  <!--pdf.logo: Image source for the PDF organizational logo-->
  <xsl:param name="pdf.logo">
    url(http://www.nmt.edu/tcc/images/logo.pdf)</xsl:param>
```

12.2. pdf.org.name: Organization's name for the running footer

```xml
  <!--pdf.org.name: Organization name in the running footer-->
  <xsl:param name="pdf.org.name">
    New Mexico Tech Computer Center</xsl:param>
```

12.3. fo_params.xsl: Epilogue

Here is the closing tag that ends the `fo_params.xsl` file.

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

13. 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.

```xml
<!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"> 
]>
```
Next comes the root element, which contains declarations for the various namespaces.

```xml
<!--File fo-titlepage.xml. For documentation, see:
! Do not edit this file directly. It is extracted automatically
! from the documentation:
! http://www.nmt.edu/tcc/doc/docbook43/ims/
!-->
<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 9, “html-
titlepage.xml: HTML title page template” (p. 26)). 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.

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

	: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`.

```xml
	: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.

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

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

```xml
	: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 11.2, “Templates for title elements” (p. 36). The stock
template calls the uncustomized `component.title` here.
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">
```

This is an experimental section to specify the title page layout for the index.

As in Section 9, "html-titlepage.xml: HTML title page template" (p. 26), 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.

### 14. Configuration of XEP

xep is the vital link in the toolchain that converts the XSL-FO that comes out of the stylesheets into a PDF file. For vendor information, see the RenderX homepage.\(^\text{28}\)

Local configuration information is in the TCC wiki under DocBook/FontCustomization\(^\text{29}\). That page also discusses how to download and install the various fonts.

---

\(^\text{28}\) [http://www.renderx.com/](http://www.renderx.com/)

\(^\text{29}\) [https://docs.nmt.edu/tccwiki/DocBook/FontCustomization](https://docs.nmt.edu/tccwiki/DocBook/FontCustomization)
However, the current document is where we manage the customization of the \texttt{xep.xml} file, which for version 4.2 of \texttt{xep} is installed as \texttt{/usr/local/xep-4.2/xep.xml} in all TCC Linux boxen.

The starting point for all customization is RenderX's stock \texttt{xep.xml} file. There is a local copy, \texttt{xep.xml.orig}, online as part of the current document.

- Section 14.1, “\texttt{xep.xml}: Stock prologue” (p. 56): This is unchanged since \texttt{xep.xml.orig}.
- Section 14.2, “\texttt{xep.xml}: TCC modifications” (p. 58).
- Section 14.3, “\texttt{xep.xml}: Stock epilogue” (p. 61): Also unchanged since \texttt{xep.xml.orig}.

14.1. \texttt{xep.xml}: Stock prologue

Here is the part of the stock \texttt{xep.xml} file that we don't change locally.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- ==================================================================== -->
<!-- RenderX XEP Configuration -->
<!-- ==================================================================== -->
<config xmlns="http://www.renderx.com/XEP/config">
  <!-- Formatter options -->
  <!-- ==================================================================== -->
  <options>
    <option name="LICENSE" value="license.xml"/>
    <option name="TMPDIR" value="none"/>
    <option name="BROKENIMAGE" value="images/404.gif"/>
  </options>
  <!-- Backend options -->
  <generator-options format="PDF">
    <!-- <option name="COMPRESS" value="false"/> -->
    <!-- <option name="PDF_VERSION" value="1.3"/> -->
  </generator-options>
  <generator-options format="PostScript">
    <!-- <option name="LANGUAGE_LEVEL" value="2"/> -->
    <!-- <option name="CLONE_EPS" value="true"/> -->
  </generator-options>
</config>
```

\begin{footnote}
  \url{http://www.nmt.edu/tcc/doc/docbook43/ims/xep.xml.orig}
\end{footnote}
The following section has been removed from the unmodified xep.xml file, because it sets up the Windows fonts, and this configuration is for Linux.

```
<font-family name="Courier">
<font-data afm="Courier.afm"/>
<font style="oblique">
<font-data afm="Courier-Oblique.afm"/>
<font weight="bold">
<font-data afm="Courier-Bold.afm"/>
<font weight="bold" style="oblique">
<font-data afm="Courier-BoldOblique.afm"/>
</font-family>

<font-family name="Helvetica">
<font-data afm="Helvetica.afm"/>
<font style="oblique">
<font-data afm="Helvetica-Oblique.afm"/>
<font weight="bold">
<font-data afm="Helvetica-Bold.afm"/>
<font weight="bold" style="oblique">
<font-data afm="Helvetica-BoldOblique.afm"/>
</font-family>

<font-family name="Times" ligatures="&FB01; &FB02;">
<font-data afm="Times-Roman.afm"/>
<font style="italic">
<font-data afm="Times-Italic.afm"/>
<font weight="bold">
<font-data afm="Times-Bold.afm"/>
<font weight="bold" style="italic">
<font-data afm="Times-BoldItalic.afm"/>
</font-family>

<font-family name="Symbol">
<font-data afm="Symbol.afm"/>
</font-family>
<font-family name="ZapfDingbats">
<font-data afm="ZapfDingbats.afm"/>
</font-family>
</font-group>

<font-group label="Assorted Type 1" embed="true">
<font-family name="OmegaSerifIPA">
<font-data afm="IPA.afm" pfa="IPA.pfa" glyph-list="IPA.glyphs"/>
</font-family>
<font-alias name="OmegaIPA" value="OmegaSerifIPA"/>
<font-alias name="Phonetic" value="OmegaSerifIPA"/>
</font-group>
```

`<!-- Sample configuration for Windows TrueType fonts. -->`

```xml
<font-group xml:base="file://C:/Windows/Fonts/"
label="Windows TrueType" embed="true" subset="true">
```

New Mexico Tech Computer Center    TCC DocBook Stylesheets  57
14.2. xep.xml: TCC modifications

Here we begin the additional font declarations local to the TCC Linux configuration.
All the TCC local fonts are part of a `font-group` element.

The main body face for our documents is Palladio, a public-domain version of Palatino.

Our principal monospaced font is Vera Sans Mono.

To increase our coverage of Unicode code points, we include a number of public-domain fonts. First, FreeFont, from the GNU Foundation:
Next, the DejaVu family:

```
<font-family name="DejaVuMonoSans"
  xml:base="/u/www/docs/tcc/help/fonts/ttf/">
  <font data ttf='DejaVuMonoSans.ttf'/></font>
  <font data ttf='DejaVuMonoSansBold.ttf'/></font>
  <font data ttf='DejaVuMonoSansOblique.ttf'/></font>
  <font data ttf='DejaVuMonoSansBoldOblique.ttf'/></font>
</font-family>

<font-family name="DejaVuSerif"
  xml:base="/u/www/docs/tcc/help/fonts/ttf/">
  <font data ttf='DejaVuSerif.ttf'/></font>
  <font data ttf='DejaVuSerifBold.ttf'/></font>
  <font data ttf='DejaVuSerifItalic.ttf'/></font>
  <font data ttf='DejaVuSerifBoldItalic.ttf'/></font>
</font-family>

<font-family name="DejaVuSans"
  xml:base="/u/www/docs/tcc/help/fonts/ttf/">
  <font data ttf='DejaVuSans.ttf'/></font>
  <font data ttf='DejaVuSansBold.ttf'/></font>
  <font data ttf='DejaVuSansOblique.ttf'/></font>
  <font data ttf='DejaVuSansBoldOblique.ttf'/></font>
</font-family>
```

Following the `font-group` element is a sequence of `font-alias` elements that connect the five major font families with the specific fonts defined above. For each alias, the `value` attribute is a comma-separated list of the families, in descending order by preference.
14.3. xep.xml: Stock epilogue

The rest of the file is unaltered from the original xep.xml file.

<!-- Required aliases -->
<font-alias name="monospace"
    value="VeraSansMono,FreeMono,DejaVuMonoSans"/>
<font-alias name="sans-serif" value="FreeSans,DejaVuSans"/>
<font-alias name="serif" value="Palladio,FreeSerif,DejaVuSerif"/>
<font-alias name="cursive" value="FreeSerif"/>
<font-alias name="fantasy" value="FreeSerif"/>
</fonts>

<languages default-language="en-US" xml:base="hyphen/">
    <language name="English (US)" codes="none en-US eng-US">
        <hyphenation pattern="hyphen.tex"/>
    </language>

    <language name="English (UK)" codes="en-GB eng-GB en eng">
        <hyphenation pattern="ukhyphen.tex"/>
    </language>

    <language name="Russian" codes="ru rus">
        <hyphenation pattern="ruhyphal.tex" encoding="koi8-r"/>
    </language>

    <language name="French" codes="fr fra fre">
        <hyphenation pattern="frhyph_rx.tex"/>
    </language>

    <language name="German" codes="de deu ger">
        <hyphenation pattern="dehyph_rx.tex"/>
    </language>

    <language name="Spanish" codes="es esl spa">
        <hyphenation pattern="eshyph_rx.tex"/>
    </language>

    <language name="Polish" codes="pl pol">
        <hyphenation pattern="plhyph_rx.tex"/>
    </language>

</languages>
</config>