

XMLmind DITA Converter Manual

Hussein Shafie

Pixware

91, rue Gambetta

78120 Rambouillet

France

Phone: +33 (0)1 30 59 81 44

ditac-support+xmlmind.com

www.xmlmind.com/ditac/

February 18, 2013

Table of Contents

List of Figures	ii
List of Tables	iii
Introduction	iv
Part I. Using XMLmind DITA Converter	1
Chapter 1. Installing XMLmind DITA Converter	3
1. Contents of the installation directory	4
Chapter 2. Getting started	7
1. Using the ditac command-line utility	7
Chapter 3. The ditac command-line utility	13
Chapter 4. XSLT stylesheets parameters	19
Chapter 5. Syntax highlighting	41
Chapter 6. Rich media content	43
Part II. Customizing the output of XMLmind DITA Converter	49
Chapter 7. Simple customization	51
1. Customize the look of the HTML pages generated by ditac	51
2. Customizing the look of the PDF files generated by ditac	52
Chapter 8. Using ditac to convert documents conforming to a DITA specialization	55
Chapter 9. Extensive customization	59
Part III. Embedding XMLmind DITA Converter in a Java™ application	65
Chapter 10. High-level method: embedding com.xmlmind.ditac.convert.Converter	67
Chapter 11. Low-level method: embedding com.xmlmind.ditac.preprocess.PreProcessor	69
Appendix A. Translating the messages generated by ditac	73
Appendix B. Limitations and implementation specificities	75
Index	i

List of Figures

1. XMLmind XSL Utility main window	iv
2-1. XMLmind XSL Utility main window	7
4-1. Page areas	37
4-2. Layout of a header	38
6-1. My name is Hubble. I'm a 7-month old Golden Retriever.	47
9-1. The intermediate files generated by the ditac preprocessor	59

List of Tables

2-1. Supported filename extensions	8
2-2. Supported output formats	10

Introduction

XMLmind DITA Converter (*ditac* for short) allows to convert the most complex DITA 1.2 documents to production-quality XHTML 1.0, XHTML 1.1, HTML 4.01, Web Help, Java™ Help, HTML Help, Eclipse Help, EPUB, PDF, PostScript®, RTF (can be opened in Word 2000+), WordprocessingML (can be opened in Word 2003+), Office Open XML (.docx, can be opened in Word 2007+), OpenOffice (.odt, can be opened in OpenOffice/LibreOffice 2+).

The first part of this document explains how to install and use *ditac*. The target audience for this part is the DITA author.

The second part of this document explains how to customize the output of *ditac*. The target audience for this part is the DITA consultant.

The third part of this document explains how to embed *ditac* in a Java™ application. The target audience for this part is the Java™ programmer.

You'll find at the end of this document an appendix detailing the limitations and implementation specificities of *ditac*. Please refer to this appendix before posting support requests to the ditac-support+xmlmind.com, public, moderated, mailing list.

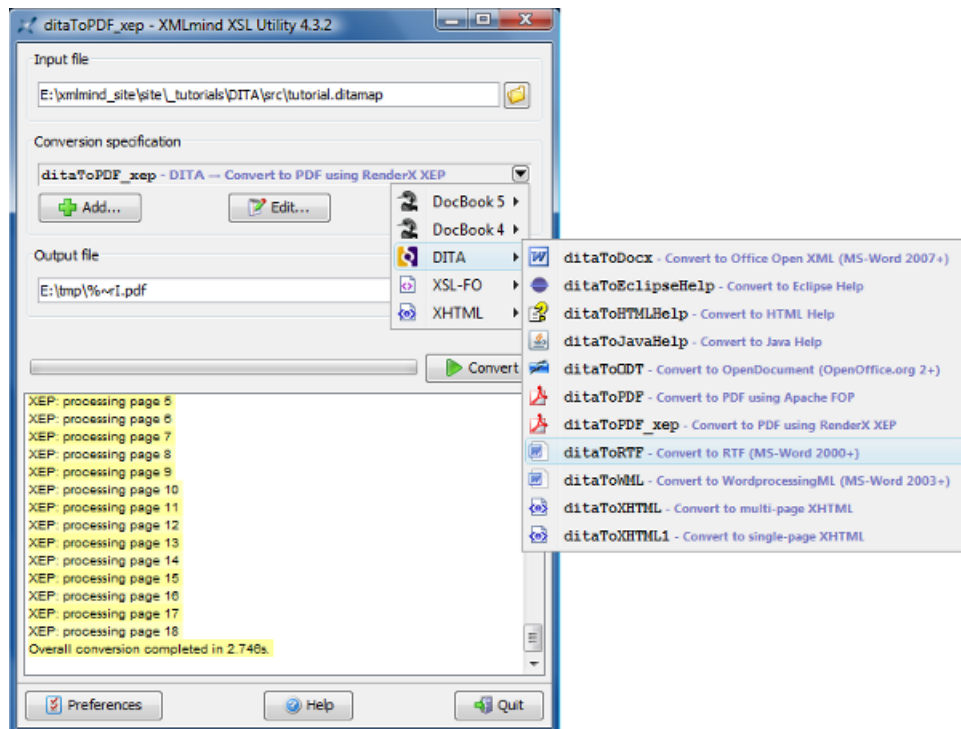


Tip

XMLmind DITA Converter has been integrated to [XMLmind XSL Utility](#), which is part of the [XMLmind XSL-FO Converter](#) commercial product.

Unlike *ditac*, which is a command-line utility, XMLmind XSL Utility is a graphical tool. It makes it easy parameterizing the DITA conversion process and then performing document conversions.

Figure 1. XMLmind XSL Utility main window



Moreover, this graphical tool comes in a Windows, auto-installable, self-contained, `setup.exe` distribution⁽¹⁾ which includes [Apache FOP](#), [XMLmind XSL-FO Converter](#) and *ditac*.

⁽¹⁾Of course, a .zip distribution is also available for platforms other than Windows.

If you just want to quickly and easily evaluate all the potential of ditac, you may want to download XMLmind XSL Utility Evaluation Edition from [XMLmind XSL-FO Converter web site](#). Do not be surprised because XMLmind XSL Utility Evaluation Edition generates output containing *random duplicate letters*. Of course, this does not happen with Professional Edition!

Part I. Using XMLmind DITA Converter

Chapter 1. Installing XMLmind DITA Converter

Before you begin

XMLmind DITA Converter (*ditac* for short) requires the Sun or Apple Java™ runtime 1.5 or above.

On Unix, make sure that the Java `bin/` directory is referenced in the `$PATH` and, at the same time, check that the Java runtime in the `$PATH` has the right version:

```
$ java -version
java version "1.6.0_16"
Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
Java HotSpot(TM) Server VM (build 14.2-b01, mixed mode)
```

On Windows and on the Mac, this verification is in principle not needed as the `java` executable is automatically found in the `$PATH` when Java has been properly installed.

Procedure

1. Unzip the distribution in any directory you want.

```
C:\> mkdir ditac
C:\> cd ditac
C:\ditac> unzip ditac-2_4_0.zip
C:\ditac> dir ditac-2_4_0
... <DIR> bin
... <DIR> doc
... <DIR> docsrc
...
```

XMLmind DITA Converter is intended to be used directly from the `ditac-2_4_0/` directory. That is, you can run the `ditac` command by simply executing (in a Command Prompt on windows, a terminal on Unix):

```
C:\ditac> ditac-2_4_0\bin\ditac
```

2. Depending the output formats you want to generate, you'll need to download and install third-party external tools.

- If you want to generate PDF or PostScript®, download and install [Apache FOP](#).

Alternatively, you may prefer to purchase [RenderX XEP](#) or [Antenna House Formatter](#). Note that [RenderX XEP Personal Edition](#) is free to use.



Note

If you have installed Apache FOP and your DITA document contains [MathML](#), you'll want to also install the [JEUclid FOP plug-in](#). This plug-in is needed to add MathML support to Apache FOP.

- If you want to generate RTF (can be opened in Word 2000+), WordprocessingML (can be opened in Word 2003+), Office Open XML (.docx, can be opened in Word 2007+) or OpenOffice (.odt, can be opened in OpenOffice/LibreOffice 2+), then you need to purchase [XMLmind XSL-FO Converter Professional Edition](#).

You can give XMLmind XSL-FO Converter a try by downloading Evaluation Edition from [XMLmind XSL-FO Converter web site](#). Do not be surprised because XMLmind XSL-FO Converter Evaluation Edition generates output containing *random duplicate letters*. Of course, this does not happen with Professional Edition!

- If you want to generate HTML Help, download and install the [HTML Help Workshop](#) (contains `hhc.exe`).

- If you want to generate Java Help, download and install [Java Help](#) (contains `jhindexer` and `jhindexer.bat`).
3. If you have installed any of the above external tools, you need now to instruct `ditac` where to find them. This can be done using the following command line options: `-fop`, `-xep`, `-ahf`, `-xfc`, `-jhindexer`, `-hhc`. However, it is much more convenient to specify these command-line options once for all in a `ditac.options` file.
- a. Create `ditac.options`, a plain text file encoded using the native encoding of the platform (e.g. `windows-1252` on a Western Windows PC), in the `ditac` user preferences directory.

The `ditac` user preferences directory is:

- `$HOME/.ditac/` on Linux.
- `$HOME/Library/Application Support/XMLmind/ditac/` on the Mac.
- `%APPDATA%\XMLmind\ditac\` on XP, Vista, 7, 8.

Example: `C:\Documents and Settings\john\Application Data/XMLmind\ditac\` on Windows XP. `C:\Users\john\AppData\Roaming/XMLmind\ditac\` on Windows Vista, 7, 8.

- b. Add the equivalent of a command-line option for each external tool installed in the preceding step. Use one or more newline characters to separate the options. More information in [The `ditac.options` file](#).

```
-fop E:\opt\fop-1.1\fop.bat

-xfc E:\opt\xfc_eval_java-4_8_0\bin\fo2rtf.bat

-jhindexer E:\opt\javahelp\javahelp\bin\jhindexer.bat

-hhc "C:\Program Files\HTML Help Workshop\hhc.exe"
```

1. Contents of the installation directory

bin/ditac, ditac.bat

Scripts used to run XMLmind DITA Converter (*ditac* for short). Use `ditac` on any Unix system. Use `ditac.bat` on Windows.

doc/index.html

Contains the documentation of `ditac`. *XMLmind DITA Converter Manual* is available in all the output formats supported by `ditac`. You'll also find there the reference manual of the API of `ditac` (generated by `javadoc`).

docsrc/manual/

Contains the DITA source of *XMLmind DITA Converter Manual*.

LEGAL/, LEGAL.txt

Contains legal information about `ditac` and about third-party components used in `ditac`.

lib/

All the (non-system) Java™ class libraries needed to run `ditac`:

ditac.jar

contains the code of XMLmind DITA Converter.

resolver.jar

is [Apache XML Commons Resolver](#) which implements catalog-based entity and URI resolution.

saxon9.jar

is Michael Kay's XSLT 2.0 engine. See <http://www.saxonica.com/>.

whcmin.jar

contains the code of [XMLmind Web Help Compiler](#).

xslthl.jar

contains the code of the [XSLT syntax highlighting](#) open source software component.

schema/

Contains the DTDs and the W3C XML Schemas of DITA 1.2, 1.1, 1.0.1. `schema/catalog.xml` is an XML catalog which points to these local copies.

src/

Contains the Java source code of ditac. `src/build.xml` is an [ant](#) build file which allows to rebuild `lib/ditac.jar`.

whc_template/

Contains the template directory of [XMLmind Web Help Compiler](#).

xsl/

Contains the [XSLT 2.0](#) stylesheets used to convert DITA documents to a variety of formats.

Chapter 2. Getting started

1. Using the `ditac` command-line utility

In this chapter, we'll explain how to run the `ditac` command-line utility by using examples. You'll find all the DITA input files used to run the following examples in the `ditac_install_dir/docsrc/manual/` directory.

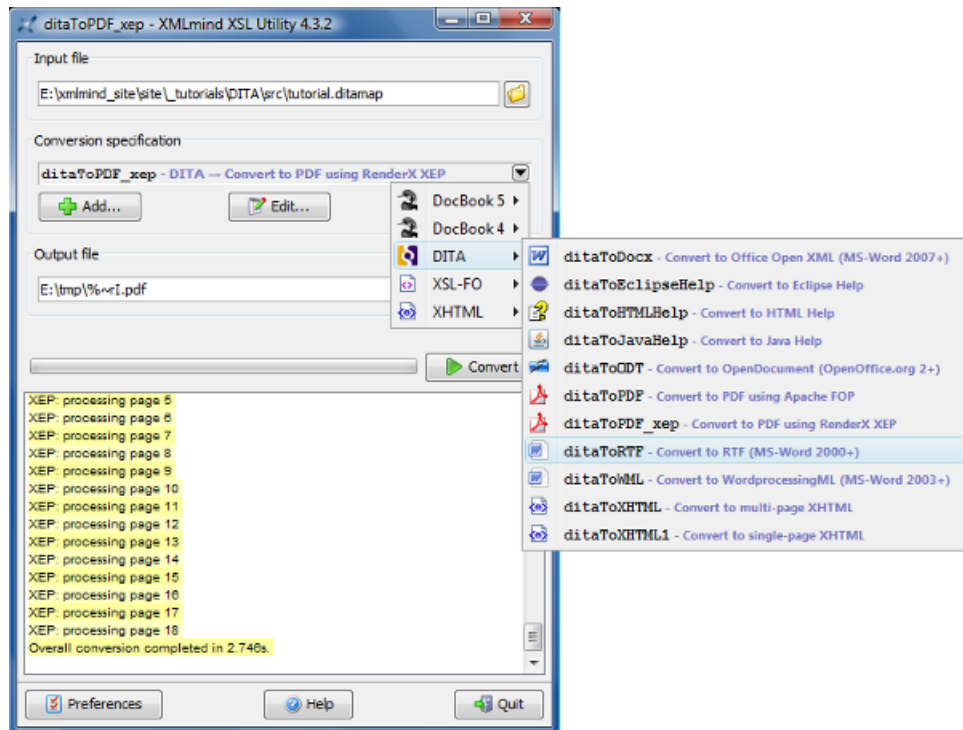


Tip

XMLmind DITA Converter has been integrated to XMLmind XSL Utility, which is part of the XMLmind XSL-FO Converter commercial product.

Unlike `ditac`, which is a command-line utility, XMLmind XSL Utility is a graphical tool. It makes it easy parameterizing the DITA conversion process and then performing document conversions.

Figure 2-1. XMLmind XSL Utility main window



Moreover, this graphical tool comes in a Windows, auto-installable, self-contained, `setup.exe` distribution⁽²⁾ which includes Apache FOP, XMLmind XSL-FO Converter and `ditac`.

If you just want to quickly and easily evaluate all the potential of `ditac`, you may want to download XMLmind XSL Utility Evaluation Edition from [XMLmind XSL-FO Converter web site](#). Do not be surprised because XMLmind XSL Utility Evaluation Edition generates output containing *random duplicate letters*. Of course, this does not happen with Professional Edition!

Converting a document to PDF

Converting a document to PDF is done by executing the following command:

```
$ ditac out/manual.pdf manual.ditamap
```

⁽²⁾Of course, a .zip distribution is also available for platforms other than Windows.

The output directory `out/` is automatically created if it does not already exist.

Unless you have specified in the `ditac.options` file which XSL-FO processor to use, you'll have to execute:

```
$ ditac -fop /opt/fop/fop out/manual.pdf manual.ditamap
```

or:

```
$ ditac -xep /opt/xep/xep out/manual.pdf manual.ditamap
```

or:

```
$ ditac -ahf "C:\AHFv6\AHFCmd.exe" out/manual.pdf manual.ditamap
```

The XSL-FO processors allowing to generate PDF also allows to generate PostScript®. Example:

```
$ ditac out/manual.ps manual.ditamap
```

Notice how the output format is determined by examining the filename extension of the output file.

Table 2-1. Supported filename extensions

Format	Extensions
XHTML 1.0	.html, .htm, .xhtml
EPUB 2	.epub
HTML Help	.chm
Java Help	.jar
PDF	.pdf
PostScript®	.ps
RTF (can be opened in Word 2000+)	.rtf, .doc
WordprocessingML(can be opened in Word 2003+)	.wml, .xml
Office Open XML (can be opened in Word 2007+)	.docx
OpenOffice (can be opened in OpenOffice/LibreOffice 2+)	.odt

Note that `ditac` also allows to convert one or more topic files rather than a single map or bookmap file:

```
$ ditac -toc \
  out/draft.pdf embed1.dita embed2.dita
```

Ditac does not generate a table of contents (TOC) by default. Unless the input file is a bookmap containing an empty `toc` element in its `frontmatter/booklists` descendant element, you'll have to explicitly use the `-toc` option. Using the `-toc` option when the input file already specifies a TOC is harmless, so you could as well add a `-toc` line to your `ditac.options` file.

Converting a document to a word processor format

Converting a document to a word processor format just requires the use of an XSL-FO processor different from the one which generates PDF or PostScript. Fortunately all this automatically handled by `ditac`.

Convert a document to RTF (can be opened in Word 2000+):

```
$ ditac out/manual.rtf manual.ditamap
```

Unless you have specified in the `ditac.options` file which XSL-FO processor to use, you'll have to execute:

```
$ ditac -xfc /opt/xfc/fo2rtf out/manual.rtf manual.ditamap
```

Suffice to specify the location of `fo2rtf` (`fo2rtf.bat` on Windows). Using this location, ditac infers the locations of `fo2wml`, `fo2docx` and `fo2odt`.

Convert a document to WordprocessingML (can be opened in Word 2003+):

```
$ ditac out/manual.xml manual.ditamap
```

Convert a document to Office Open XML (can be opened in Word 2007+):

```
$ ditac out/manual.docx manual.ditamap
```

Convert a document to OpenOffice (can be opened in OpenOffice.org 2+):

```
$ ditac -v -p number all \
  out/manual.odt manual.ditamap
```

Useful options

- `-v` instructs **ditac** to print progress messages on the console. Recommended when converting large documents.
- `"-p number all"` passes parameter "number" with value "all" to the XSLT stylesheets which generate the XSL-FO. The XSL-FO are then converted to OpenOffice format by the means of XMLmind XSL-FO Converter. The `number='all'` parameter instructs the XSLT stylesheets to number topics, tables and figures.

Converting a document to XHTML

Converting a document to multi-page XHTML 1.0 is done by executing the following command:

```
$ ditac -images img -p xsl-resources-directory res \
  out/manual/_ .html manual.ditamap
```

- All the files generated by **ditac** are created in the `out/manual/` directory.
- `"-images img"` instructs **ditac** to copy all the image files referenced by the input DITA document to `out/manual/img/`. Specifying the `-images` option when generating an output format based on XHTML/HTML is needed in almost all the use cases.
- `"-p xsl-resources-directory res"` instructs **ditac** to copy all the resources needed by the XSLT stylesheets (CSS stylesheet, navigation icons, etc) to `out/manual/res/`. Specifying a value for the `xsl-resources-directory` parameter when generating an output format based on XHTML/HTML is needed in almost all the use cases.
- Notice the strange name of the output file: `out/manual/_ .html`. In fact, this name is just used to specify the filename extension of the output files. The actual basenames of the output files are determined by examining the `chunk` and `copy-to` attributes possibly specified in the DITA map.

Note that a command-line like:

```
$ ditac -images img -p xsl-resources-directory res \
  out/manual/foo.html manual.ditamap
```

works fine too. The only difference is that in such case, when a basename is needed and cannot be determined by examining the `chunk` and `copy-to` attributes specified in the DITA map, **ditac** will use "foo" as a basename and you *may* end up having some output files called `foo.html`, `foo-2.html`, `foo-3.html`, etc. When the basename is specified as `"_"`, it is the basename of the DITA map which is used. That is, you may have some output files called `manual.html`, `manual-2.html`, `manual-3.html`, etc.

What if you want to convert a document to HTML 4.01 or XHTML 1.1 or XHTML 5 rather than to XHTML 1.0? We have learned that there is no way to specify this using a [filename extension](#). The answer is: use the `-format` option (or `-f` in its short form). Example:

```
$ ditac -format html \
  -images img -p xsl-resources-directory res \
  out/manual/_ .html manual.ditamap
```

Table 2-2. Supported output formats

Format	Name
XHTML 1.0	xhtml
XHTML 1.1	xhtml1.1
HTML 4.01	html
XHTML 5	xhtml5. html5 is an alias for xhtml5.
Web Help containing XHTML 1 pages	webhelp
Web Help containing XHTML 5 pages	webhelp5
HTML Help	htmlhelp
Eclipse Help	eclipsehelp
EPUB 2	epub
EPUB 3	epub3
Java Help	javahelp
PDF	pdf
PostScript®	ps
RTF (can be opened in Word 2000+)	rtf
WordprocessingML(can be opened in Word 2003+)	wml
Office Open XML (can be opened in Word 2007+)	docx
OpenOffice (can be opened in OpenOffice.org 2+)	odt
XSL-FO	fo

Useful options

- "-p `chain-pages` both". This XSLT stylesheet parameter specifies that a header and a footer containing navigation icons should be generated in order to link together all the HTML pages.
- "-p `chain-topics` yes". This XSLT stylesheet parameter specifies that navigation icons should be generated in order to link together all the topics.
- "-p `default-table-width` 100%". Unless this XSLT stylesheet parameter is specified (or the `expanse="page"` attribute is specified for all tables), web browsers tend to layout the generated HTML tables in order to make them as narrow as possible.

A full-fledged command-line is thus:

```
$ ditac -images img -p xsl-resources-directory res \
  -p number all \
  -p chain-pages both \
  -p chain-topics yes \
  -p default-table-width 100% \
  out/manual/_html manual.ditamap
```

What if you want to generate a single XHTML page rather than multiple XHTML page? No need to create a new DITA map for that. Simply specify option "`-chunk single`" (or `-c` in its short form).

```
$ ditac -chunk single \
  -images img -p xsl-resources-directory res \
  out/manual.html manual.ditamap
```

Converting a document to Web Help

Converting a document to [Web Help](#) is similar to converting a document to multi-page XHTML 1. The main difference is that you need to explicitly specify `-format webhelp`:

```
$ ditac -format webhelp \
  -images img -p xsl-resources-directory res \
  webhelp/_html manual.ditamap
```

If you prefer to generate Web Help containing XHTML 5 pages rather than XHTML 1 pages, then specify `-format webhelp5`.



Remember

Do not specify any of the following command-line options when generating Web Help: `-toc`, `-index`.

Converting a document to HTML Help

Converting a document to HTML Help is done by executing the following command:

```
C:\> ditac -images img -p xsl-resources-directory res \
  out\manual.chm manual.ditamap
```

Unless you have specified in the `ditac.options` file the location of `hhc.exe`, you'll have to execute:

```
C:\> ditac -hhc "C:\Program Files\HTML Help Workshop\hhc.exe" \
  -images img -p xsl-resources-directory res \
  out\manual.chm manual.ditamap
```



Remember

Do not specify any of the following command-line options when generating HTML Help: `-toc`, `-index`.

Converting a document to Java™ Help

Converting a document to Java™ Help is done by executing the following command:

```
$ ditac -images img -p xsl-resources-directory res \
  out/manual.jar manual.ditamap
```

Unless you have specified in the `ditac.options` file the location of `jhindexer` (`jhindexer.bat` on Windows), you'll have to execute:

```
$ ditac -jhindexer /opt/jh2.0/javahelp/bin/jhindexer \
  -images img -p xsl-resources-directory res \
  out/manual.jar manual.ditamap
```



Remember

Do not specify any of the following command-line options when generating Java™ Help: `-toc`, `-index`.

Converting a document to Eclipse Help

Converting a document to [Eclipse](#) Help is similar to converting a document to multi-page XHTML. The main difference is that you need to explicitly specify `-format eclipsehelp`:

```
$ ditac -format eclipsehelp \
  -images img -p xsl-resources-directory res \
  out/com.acme.widget.userguide/_html manual.ditamap
```

In order to deploy the generated Eclipse Help, you need to copy the output directory as a whole (`com.acme.widget.userguide/` in the case of the above example) to the `plugins/` directory of Eclipse and then use a text or XML editor to modify the generated `output_directory/plugin.xml`:

```
<plugin name="EDIT HERE: title of this help"
  id="EDIT HERE: unique.id.of.this.plugin"
  provider-name="EDIT HERE: author, company or organization"
  version="1.0.0">
  <extension point="org.eclipse.help.toc">
    <toc file="toc.xml" primary="true"/>
  </extension>
  <extension point="org.eclipse.help.index">
    <index file="index.xml"/>
  </extension>
</plugin>
```

If you do not want to hand edit `plugin.xml`, suffice to pass extra XSLT stylesheet parameters to `ditac`:

```
$ ditac -format eclipsehelp \
  -p plugin-name "ACME Widget User's Guide" \
  -p plugin-id com.acme.widget.userguide \
  -p plugin-provider "ACME Corp." \
  -images img -p xsl-resources-directory res \
  out/com.acme.widget.documentation/_.html manual.ditamap
```

Parameter `plugin-id` is required to have the same value as the basename of the the output directory (`com.acme.widget.userguide/` in the case of the above example). Otherwise, you'll not be able to see your document by selecting **Help** → **Help Contents** in Eclipse.



Remember

Do not specify any of the following command-line options when generating Eclipse Help: `-toc`, `-index`.

Converting a document to EPUB

Converting a document to EPUB Help is done by executing the following command:

```
$ ditac -images img -p xsl-resources-directory res \
  out/manual.epub manual.ditamap
```

If you prefer to generate EPUB 3 rather than EPUB 2, then specify `-format epub3`.



Remember

Do not specify any of the following command-line options when generating EPUB: `-toc`. Note that you may specify option `-index`.

Related information

- Chapter 3. The `ditac` command-line utility

Chapter 3. The `ditac` command-line utility

ditac [*option*]* *output_file* [*in_dita_file*]+

Command-line usage

Converts specified DITA input files to specified output file.

The input files must comprise a single map or bookmap file or possibly several, possibly multi-topic, topic files.

Example: convert the `userguide.ditamap` map to multi-page XHTML:

```
C:\docsrc> ditac -p center "fig table" ..\doc\userguide.htm userguide.ditamap
```

Example: convert the `introduction.dita` and `quickstart.dita` topics to PDF:

```
C:\docsrc> ditac draft1.pdf introduction.dita quickstart.dita
```

An input file may be specified using its URL or its filename.

The output directory is created if it does not already exist.

In some case, there is no need to specify a real output filename: the output directory and the extension of the output files suffice. In such case, specify "_" as the basename of the output file.

Example: convert `foo.ditamap` to multi-page XHTML. The XHTML pages must be generated in the `bar/` subdirectory.

```
C:\docsrc> ditac bar\_ .html foo.ditamap
```

In the above case, the basenames of the generated XHTML pages will be taken from the `chunk` and `copy-to` attributes specified in `foo.ditamap` if any, and from the basename of the map ("`foo`" in the case of our example) otherwise.

Commonly used command-line options

Some options have both a short name and a long name. Example: `-p` is equivalent to `-param`.

-p *param_name param_value*

-param *param_name param_value*

Specifies a XSLT stylesheet parameter. See [Chapter 4](#).

-t *XSLT_stylesheet_URL_or_file*

-xslt *XSLT_stylesheet_URL_or_file*

Use the specified custom XSLT stylesheet rather than the stock one.

-c *none|single|auto*

-chunk *none|single|auto*

The "none" and "single" values may be used to force the generation of a single output file.

For example, "`-chunk single`" allows to reuse a map designed to output multiple HTML pages in order to generate a PDF file.

For example, "`-chunk none`" allows to reuse a map designed to output a PDF file in order to generate a single HTML page.

By default, the chunk mode is `auto` which means: generate a single output file (implicit "`-chunk none`") for formats such as `pdf`, `ps`, `rtf`, etc, and generate multiple output files for formats such as `html`, `xhtml`, `javahelp`, etc.

-f xhtml | xhtml1.1 | html | xhtml5 | html5 | webhelp | webhelp5 | epub | epub3 | javahelp | htmlhelp | ps | pdf | rtf | odt | wml | docx | fo
-format xhtml | xhtml1.1 | html | xhtml5 | html5 | webhelp | webhelp5 | epub | epub3 | javahelp | htmlhelp | ps | pdf | rtf | odt | wml | docx | fo

Explicitly specifies the output format. By default, the output format is determined using the extension of *output_file*.

Notes:

- A "htm" or "html" filename extension implicitly specifies an XHTML 1.0 output format, and not an HTML 4.01 output format. In order to generate HTML 4.01, explicitly specify "-f html". The same remark applies to xhtml1.1, xhtml5, webhelp, webhelp5.
- Option html5 is simply an alias for xhtml5.
- Option webhelp5 means Web Help containing XHTML 5 pages rather than XHTML 1 pages.
- Option epub specifies the EPUB 2 format.

-r resource_path

-resources resource_path

-i resource_path

-images resource_path

Copy the resource files, typically image files, referenced in the source topics to specified directory. If specified path is relative, it is relative to the output directory.

-resourcehandler class_name parameters

Pass the resource files, typically image files, referenced in the source topics to *class_name*, a Java™ class implementing interface `com.xmlmind.ditac.preprocess.ResourceHandler`. String *parameters* is used to configure the newly created `ResourceHandler`.

For example, "-r res" is equivalent to "-resourcehandler com.xmlmind.ditac.convert.ResourceCopier res".

-filter ditaval_URL_or_file

Apply specified conditional processing profile (.ditaval file) to the topics.

-toc

Equivalent to "-frontmatter toc".

Note that this option will *not* cause a **Table of Contents** to be generated when the map contains a single `topicref`⁽³⁾ having no `topicref` descendants.

-index

Equivalent to "-backmatter indexlist".

-frontmatter spec

Automatically generate specified sections: **Table of Contents**, **List of Tables**, etc, before the other pages.

The syntax of *spec* is:

```
spec -> same_page [ ',' same_page ]*
same_page -> section [ '+' section ]*
section -> 'toc' | 'figurelist' | 'tablelist' | 'examplelist' | 'indexlist'
```

Example: generate the **Table Of Contents** in its own page, followed by another page containing both the **List of Figures** and the **List of Tables**.

```
-frontmatter toc,figurelist+tablelist
```

⁽³⁾Not counting `topicrefs` contained in frontmatter and backmatter.

-backmatter *spec*

Automatically generate specified sections: **Table of Contents**, **List of Tables**, etc, after the other pages. See **-frontmatter** for more information.

-addindex

When an output file contains the **Table of Contents** (let's call this file `main.html`) and when no file called `index.html` has been generated, this option allows to copy `main.html` to `index.html`. Applies to formats: `xhtml`, `xhtml1.1`, `html`, `webhelp`.

-lang *language_code*

Specifies the main language of the document. Examples: `"fr"`, `"fr-CA"`. Needed to sort the index entries.

By default, this information is taken from the `xml:lang` attribute of the root element of the topic map (if any, `"en"` otherwise).

-v**-vv****-vvv**

Turn verbosity on. More Vs means more verbose.

-o *options_URL_or_file***-option *options_URL_or_file***

This option allows to specify a text file containing command-line arguments. This text file has the same format as [the ditac.options file](#).

Example:

```
$ ditac -v -o html.options foo.htm foo.ditamap
```

If `html.options` contains:

```
-format html
-p css http://www.acme.com/css/acme.css
```

then this is equivalent to running:

```
$ ditac -v -format html -p css http://www.acme.com/css/acme.css \
    foo.htm foo.ditamap
```

Command-line options used to configure ditac

-fop *executable_file*

Specifies the location of the `fop` shell script (`fop.bat` on Windows).

-xep *executable_file*

Specifies the location of the `xep` shell script (`xep.bat` on Windows).

-ahf *executable_file*

Specifies the location of `AHFCmd.exe` (`run.sh` on platforms other than Windows).

-xfc *executable_file*

Specifies the location of the `fo2rtf` shell script (`fo2rtf.bat` on Windows).

Suffice to specify the location of `fo2rtf`. Using this location, `ditac` infers the locations of `fo2wml`, `fo2docx` and `fo2odt`.

-foconverter *processor_name target_format command*

Register specified XSL-FO converter with `ditac`, a lower-level alternative to using **-xep**, **-fop**, **-ahf** or **-xfc**. Example:

```
-foconverter XFC rtf '/opt/xfc/bin/fo2rtf "%I" "%O"'
```

Note that this option can be specified several times with different values in the same command-line.

-jhindexer *executable_file*

Specifies the location of the `jhindexer` shell script (`jhindexer.bat` on Windows), the Java™ Help indexer.

-hhc *exe_file*

Specifies the location of `hhc.exe`, the HTML Help compiler.

-plugin *plugin_name*

Use the DTDs/schemas and the XSLT stylesheets found in the plug-in subdirectory having specified name preferably to those found in `ditac_install_dir/schema/` and in `ditac_install_dir/xsl/`. See [What is a plug-in?](#).

Command-line options used to debug ditac

-preprocess

Stop after preprocessing input files.

-automap *save_file*

Save the automatically generated topic map (if any) to specified file.

-keepfo

When generating PDF, RTF, etc, do not delete the temporary XSL-FO file.

-errout

Output all messages, including errors and warnings, to `stdout`.

-ignoreoptionsfile

Do not load the `ditac.options` options file. See below [The `ditac.options` file](#).

-dryrun

Use **ditac** as a validator. That is, do not generate any file; just report errors if any .

-version

Print version number and exit.

The `ditac.options` file

It is also possible to specify command-line options in the `ditac.options` options file. The content of this plain text file, encoded in the native encoding of the platform (e.g. Windows-1252 on a Western Windows PC), is automatically loaded by **ditac** each time this command is executed. The content of this file, command-line options separated by whitespace, is *prepended* to the options specified in the command-line.

Example: If `ditac.options` contains:

```
-v -p number all
```

Running:

```
~/docsrc$ ditac -p center "fig table" ../doc/userguide.htm userguide.ditamap
```

is equivalent to running:

```
~/docsrc$ ditac -v -p number all -p center "fig table" \
  ../doc/userguide.htm userguide.ditamap
```

The `ditac.options` options file is found in the `ditac` user preferences directory. This directory is:

- `$HOME/.ditac/` on Linux.
- `$HOME/Library/Application Support/XMLmind/ditac/` on the Mac.
- `%APPDATA%\XMLmind\ditac\` on Windows XP, Vista, 7, 8.

Example: `C:\Documents and Settings\john\Application Data/XMLmind\ditac\` on Windows XP. `C:\Users\john\AppData\Roaming/XMLmind\ditac\` on Windows Vista, 7 and 8.

The `ditac.options` options file is mainly useful to configure `ditac` once for all by specifying values for the **-fop**, **-xep**, **-xfc**, **-jhindexer**, **-hhc**, **-plugin** options.

Example:

```
-v
-xep E:\opt\xep\xep.bat
-fop E:\opt\fop-1.1\fop.bat
-xfc E:\opt\xfc_eval_java-4_8_0\bin\fo2rtf.bat
-jhindexer E:\opt\javahelp\javahelp\bin\jhindexer.bat
-hhc "C:\Program Files\HTML Help Workshop\hhc.exe"
```



Remember

- Relative filenames found in this file are relative to the current working directory, and not to the `ditac.options` options file. Therefore it is recommended to always specify absolute filenames.
- No comments (e.g. lines starting with '#') are allowed in `ditac.options`. Options must be separated by whitespace.
- In the above example, FOP is declared *after* XEP. This implies that it is FOP and not XEP, which will be used by `ditac` to generate PDF and PostScript®.
- An XSL-FO processor tend to consume a lot of memory. If the DITA conversion fails with an out-of-memory error, you need to edit the `xep` (`xep.bat`), `fop` (`fop.bat`), `fo2xxx` (`fo2xxx.bat`) scripts in order to increase the maximum amount of memory that the Java™ runtime may allocate. This is done by using the **-Xmx** option of the Java™ command-line. Example: `"java ... -Xmx512m ..."`.
- Starting from Java™ 1.6.0_23, converting XML documents to PDF using RenderX XEP randomly fails with false XSL-FO errors (e.g. attribute "space-before" may not be empty). This problem seems specific to the 64-bit runtime.

The workarounds for the above bug ("renderx #22766") are:

- Use a 32-bit Java™ runtime.
- OR Use a 64-bit Java™ runtime older than 1.6.0_23.
- OR Specify option **-valid** in the **xep** command-line.

What is a plug-in?

A plug-in is simply a subdirectory of `ditac_install_dir/plugin/`. For example, `ditac_install_dir/plugin/MyPlugin/`.

This subdirectory may contain an **XML catalog file**. This XML catalog file must be named `catalog.xml`. In the case of a DITA specialization, `catalog.xml` points to local copies of customized DTDs. Example: `ditac_install_dir/plugin/MyPlugin/catalog.xml`:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
  prefer="public">
  <public publicId="-//OASIS//DTD DITA Concept//EN"
    uri="dtd/concept.dtd"/>
  ...
</catalog>
```

This subdirectory may contain an `xsl/` subdirectory organized *exactly* like `ditac_install_dir/xsl/`. That is, this `xsl/` subdirectory may contain one or more of the following XSLT stylesheets:

XSLT stylesheet	Description
<code>xsl/fo/fo.xsl</code>	Used to generate an intermediate XSL-FO file. After that, the XSL-FO file is converted to PDF , PostScript , RTF , WordprocessingML , Office

XSLT stylesheet	Description
	Open XML (.docx) or OpenOffice/LibreOffice (.odt) by the means of an XSL-FO processor.
xsl/xhtml/xhtml1.xsl	Used to generate XHTML 1.0 pages.
xsl/xhtml/xhtml11_1.xsl	Used to generate XHTML 1.1 pages.
xsl/xhtml/html.xsl	Used to generate HTML 4.01 pages.
xsl/xhtml/xhtml15.xsl	Used to generate XHTML 5 pages.
xsl/webhelp/webhelp.xsl	Used to generate Web Help containing XHTML 1 pages, which are then compiled using XMLmind Web Help Compiler .
xsl/webhelp/webhelp5.xsl	Used to generate Web Help containing XHTML 5 pages, which are then compiled using XMLmind Web Help Compiler .
xsl/htmlhelp/htmlhelp.xsl	Used to generate HTML Help files, which are then compiled using hhc.exe.
xsl/eclipsehelp/eclipse-help.xsl	Used to generate Eclipse Help files.
xsl/javahelp/javahelp.xsl	Used to generate Java™ Help files, which are then archived in a .jar file.
xsl/epub/epub.xsl	Used to generate EPUB 2 files, which are then archived in a .epub file (Zip archive having a .epub extension).
xsl/epub/epub3.xsl	Used to generate EPUB 3 files, which are then archived in a .epub file (Zip archive having a .epub extension).

When **ditac** is passed command-line option **-plugin plugin_name**, it will use the DTDs/schemas and the XSLT stylesheets found in the plug-in subdirectory having specified name preferably to those found in *ditac_install_dir/schema/* and in *ditac_install_dir/xsl/*.



Tip

If you don't want your plug-ins to reside inside *ditac_install_dir/plugin/*, you may specify an alternate parent directory by the means of the `DITAC_PLUGIN_DIR` environment variable. Example:

- On Windows:

```
C:\>set DITAC_PLUGIN_DIR=C:\Users\john\ditac_plugins
```

- On Unix:

```
$ export DITAC_PLUGIN_DIR=/home/john/ditac_plugins
```

Related information



- [Chapter 4. XSLT stylesheets parameters](#)


Chapter 4. XSLT stylesheets parameters

Parameters common to all stylesheets



Note

- Parameters marked using this icon  are *system parameters*. They are automatically specified by the application executing the XSLT stylesheets. Such system parameters must not be specified by the end-user. Such system parameters are documented here only because the end-user may see them referenced in some configuration files.
- Parameters marked using this icon  are *pseudo-parameters*. They may or may not be passed to the XSLT stylesheets, but the important thing to remember is that they are also interpreted by ditac itself. By consequence, you cannot specify them in an XSLT stylesheet which customizes the stock ones (as explained in [Part II, Chapter 7, Section 2](#)).

Parameter	Value	Description
appendix-number-format	Allowed values are: 'I', 'i', 'A', 'a', '1'. Default value: 'A'.	The number format of topics referenced in a bookmap as appendix. By default, such topics are numbered as follows: Appendix A. Title of first appendix , Appendix B. Title of second appendix , etc.
center	List of element names separated by whitespace. Example: 'fig simpletable table'. Default value: ''.	Specifies which elements are to be centered horizontally on the page.
 ditacListsURI	URL ⁽⁴⁾ . Default value: <code>output_dir/ditac_lists.ditac_lists</code> .	The URL of file <code>ditac_lists.ditac_lists</code> .
extended-toc	Allowed values are: 'frontmatter', 'backmatter', 'both', 'none'. Default value: 'none'.	Allows to add frontmatter and backmatter topicrefs to the Table of Contents (TOC) of a document. Note that the <code>toc</code> , <code>navtitle</code> , <code>locktitle</code> , etc, attributes are applied normally to frontmatter and backmatter topicrefs when an extended TOC is generated.
external-resource-base	Allowed values are: '', an URL ending with "/" or '#REMOVE'. Default value: '#REMOVE' for EPUB 2 and EPUB 3, '' for all the other output formats.	Specifies how to resolve xref or link elements having an <i>external</i> scope attribute and a <i>relative</i> href attribute. Example of such xref elements: <code><xref scope="external" format="java" href="src/Test.java">Test.java</xref></code> . '' Do not resolve the href attribute. In this case, the external resource files are expected to be copied “by hand” to the output directory.


⁽⁴⁾ Unlike a filename, an URL must contain properly quoted characters. For example, do not specify 'Hello world.htm', instead specify 'Hello%20world.htm'.

Parameter	Value	Description
		<p>An URL ending with "/" This URL is prepended to the value of the href attribute.</p> <p>'#REMOVE' The xref or link element is processed as if it did not have an href attribute.</p>
highlight-source	<p>Allowed values are: 'yes' and 'no'.</p> <p>Default value: 'yes'.</p>	<p>Allows to turn off syntax highlighting in elements specializing pre.</p> <p>By default, syntax highlighting is turned on for all elements specializing pre and having an outputclass attribute equals to language-c, language-cpp, language-csharp, language-delphi, language-ini, language-java, language-javascript, language-m2, language-perl, language-php, language-python, language-ruby, language-tcl.</p>
index-range-separator	<p>String.</p> <p>Default value: '&#x2013;' (EN DASH).</p>	<p>The string used to separate the first page number from the last page number in a page range of an indexed term. Example: index-range-separator='<-->':</p> <pre>C Cat 54, 87<-->90</pre>
link-auto-text	<p>List of values separated by whitespace. Allowed values are: 'number' and 'text'.</p> <p>Default value: 'number text'.</p>	<p>This parameter specifies which text to generate for a link element, when this link element has no linktext child element or when this linktext child element is empty.</p> <p>Similar to above parameter xref-auto-text but for link elements.</p>
note-icon-list	<p>List of type attribute values separated by whitespace.</p> <p>Default value: 'attention caution danger fast-path important note notes remember restriction tip'.</p>	<p>Specifies the type (attribute type) of the note elements for which icons should be used rather than text in order to represent note labels.</p> <p>Ignored unless use-note-icon='yes'.</p>
number	<p>List of values separated by whitespace. Allowed values are: 'topic', 'chapter-only', 'table', 'fig', 'example', 'all'.</p> <p>Default value: '' (number nothing).</p>	<p>Specifies which elements are to be numbered.</p> <p>'all' is a short form for 'topic table fig'.</p> <p>'chapter-only' means: number topics, but only those referenced in a bookmap as part, chapter and appendix.</p>
number-separator1	<p>String.</p> <p>Default value: ' . '.</p>	<p>The string used to separate the hierarchical number of topics acting as sections.</p>

Parameter	Value	Description
number-separator2	String. Default value: '- '.	The string used to separate the hierarchical number of figures, tables and examples. When possible, the number of figure, table or example is made relative to the number of the ancestor chapter or appendix. This gives for example (for descendants of chapter 5): Figure 5-1. Title of first figure of chapter 5 , Figure 5-2. Title of second figure of chapter 5 , etc.
part-number-format	Allowed values are: 'I', 'i', 'A', 'a', '1'. Default value: 'I'.	The number format of topics referenced in a bookmap as part. By default, such topics are numbered as follows: Part I. Title of first part , Part II. Title of second part , etc.
prepend-chapter-to-section-number	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Normally topics which are descendants of chapters (that is, topics referenced in a bookmap as chapter) are numbered as follows: 1. Title of first section , 1.1. Title of first subsection , etc. Specifying <code>prepend-chapter-to-section-number='yes'</code> prepends the number of the chapter ancestor to the section number. This gives for example (for descendants of chapter 5): 5.1. Title of first section , 5.1.1. Title of first subsection , etc.
show-draft-comments	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether <code>draft-comments</code> elements should be rendered.
text-file-encoding	An encoding name such as UTF-8 or ISO-8859-1. Default value: ''.	Encoding of the text file referenced by <code>coderef/@href</code> . An empty string means: to be determined automatically.
title-after	List of element names separated by whitespace. Example: 'fig table'. Default value: ''.	Specifies which elements should have their titles displayed after their bodies.
title-page	Allowed values are: 'auto', 'none' or the URI of a custom title page. Default value: 'auto'.	Specifies the kind of "title page" (contains the title of the document, its author, etc) to be generated before the actual contents of the document. 'auto' Automatically generate a title page based on the title and metadata of the map. 'none' Do not generate a title page. URI of a custom title page Specifies the URI of a custom title page. If the URI is relative, it is relative to the current working directory of the user. This custom title page is an XHTML file for XHTML-based formats (XHTML, HTML

Parameter	Value	Description
		<p>Help, etc). This custom title page is an XSL-FO file for FO-based formats (PDF, RTF, etc). Such custom title pages are generally hand-written.</p> <ul style="list-style-type: none"> The child nodes of the <code>body</code> element of the custom XHTML title page are wrapped in a <code>div</code> contained in the XHTML/HTML file being generated by the XSLT stylesheet. <p>Do not add a <code><!DOCTYPE></code> to such custom XHTML title page because otherwise, the XSLT stylesheet may fail loading it.</p> <p>See sample custom XHTML title page.</p> <ul style="list-style-type: none"> The child nodes of the first <code>fo:flow[@flow-name='xsl-region-body']</code> element of the custom XSL-FO title page are wrapped in a <code>fo:block</code> contained the XSL-FO file being generated by the XSLT stylesheet. <p>See sample custom XSL-FO title page.</p>
<code>title-prefix-separator1</code>	String. Default value: ' . '.	The string used to separate the number of an formal object from its title.
<code>use-note-icon</code>	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether icons should be used rather than text in order to represent the label of a <code>note</code> element.
<code>xref-auto-text</code>	List of values separated by whitespace. Allowed values are: 'number' and 'text'. Default value: 'number'.	<p>This parameter specifies which text to generate for an <code>xref</code> element, when this <code>xref</code> element contains no text at all⁽⁵⁾.</p> <p>Let's suppose that an <code>xref</code> element containing no text at all points to a topic titled "Installation".</p> <p>Because the <code>xref</code> element points to an element having a <code>title</code> child element, ditac may use this title as a starting point for the generated text.</p> <p>Now let's suppose that topics are numbered and that the number of the "Installation" topic is "Chapter 5".</p> <p>The text generated for this <code>xref</code> element is thus:</p> <p>If <code>xref-auto-text='number'</code> Chapter 5</p> <p>If <code>xref-auto-text='text'</code> Installation</p>





⁽⁵⁾This implies that the `xref-auto-text` parameter is ignored when an `xref` element contains some text.

Parameter	Value	Description
		<p>If <code>xref-auto-text='number text'</code> Chapter 5. Installation</p> <p>Note that this specification is just a hint because ditac needs anyway to generate some text. For example, if topics are not numbered and <code>xref-auto-text='number'</code>, the generated text will be "Installation".</p>
★ <code>xsl-resources-directory</code>	<p>URL. A relative URL is relative to the output directory.</p> <p>Default value: <code>'resources/'</code> resolved against the directory which contains the XSLT stylesheets.</p>	<p>Most XSLT stylesheets generate files which reference resources such as icons or CSS stylesheets. This parameter specifies the target directory which is to contain such resources.</p> <p>If this directory does not exist, it is automatically created.</p> <p>If this directory does not already contain the resources needed by the XSLT stylesheets, such resources are automatically copied to this directory.</p> <p>The default value of this parameter is something like <code>file:/opt/ditac/xsl/xhtml/resources/</code> for the stylesheets generating XHTML. URL <code>file:/opt/ditac/xsl/xhtml/resources/</code> specifies an existing directory containing <code>basic.css</code>, <code>note.png</code>, <code>important.png</code>, etc. This means that by default, no directory is created and no resource is copied.</p> <p>If the value of this parameter is an absolute URI, then ditac assumes that no resource directory is to be created and no resource is to be copied because this has already been done by the user.</p> <div>  Important <ul style="list-style-type: none"> Explicitly specifying something like <code>xsl-resources-directory='res'</code> is almost <i>always required</i> when generating files having an XHTML/HTML based format (XHTML, HTML Help, etc). Explicitly specifying something like <code>xsl-resources-directory='res'</code> is almost <i>never required</i> when generating files converted from XSL-FO (PDF, RTF, etc). </div>


Parameters common to the stylesheets that basically generate XHTML or HTML

This applies to the stylesheets that generate XHTML, HTML, Web Help, Java™ Help, HTML Help, Eclipse Help, EPUB.


Parameter	Value	Description
<code>chain-pages</code>	<p>Allowed values are: <code>'none'</code>, <code>'top'</code>, <code>'bottom'</code> or <code>'both'</code>.</p>	<p>Specifies whether a header and/or a footer containing navigation icons should be generated in order to link together all the HTML pages.</p>

Parameter	Value	Description
	Default value: 'none'.	 Note There is no need to specify a value other than 'none' when generating Web Help, HTML Help, Eclipse Help, EPUB and Java™ Help.
chain-topics	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether navigation icons should be generated in order to link together all the topics.  Note There is no need to specify a value other than 'no' when generating Web Help, HTML Help, Eclipse Help, EPUB and Java™ Help.
css	URL. Default value: ''.	Specifies which CSS stylesheet to use. Has priority over the CSS stylesheet specified by the css-name parameter. An end-user wishing to use a custom CSS must typically: <ol style="list-style-type: none"> 1. Import the <code>basic.css</code> stock stylesheet in its own <code>custom.css</code> as follows: <pre>@import url(basic.css);</pre> 2. Invoke dita with the following parameters: <code>xsl-resources-directory='res'</code>, <code>css='res/custom.css'</code>. 3. Copy by hand <code>custom.css</code> to <code>output_dir/res/</code> after dita has finished its job.  Restriction Not supported by the stylesheets that generate EPUB.
css-name	URL basename relative to the directory specified by parameter xsl-resources-directory . Default value: 'basic.css'; 'java-help.css' when the output format is Java™ Help.	Specifies which CSS stylesheet to use. This CSS stylesheet is expected to be found in the resources directory.  Restriction Not supported by the stylesheets that generate EPUB.
default-table-width	A percentage, typically something like '100%' or '90%'. Default value: '' (as narrow as possible).	The default width of <code>table</code> and <code>simpletable</code> elements.
external-link-icon-height	Length. A length may have a unit. Default is px.	The height of the "opens in new window" icon.

Parameter	Value	Description
	Default value: '10'.	
external-link-icon-suffix	Basename. Default value: 'new_window.png'.	The basename of the "opens in new window" icon. This icon is found in the resources directory.
external-link-icon-width	Length. A length may have a unit. Default is px. Default value: '11'.	The width of the "opens in new window" icon.
generator-info	String Default value: 'XMLmind DITA Converter VERSION'.	The name of the software which has been used to create the HTML pages. Specify an empty string if you don't want to have a <code><meta name="generator" content="XXX"/></code> element added to your HTML pages.
ignore-navigation-links	Allowed values are: 'yes', 'no' and 'auto'. Default value: 'auto' for XHTML and its variants; 'yes' for Web Help, Java Help, HTML Help, Eclipse Help and EPUB	If 'yes', do not generate the navigation links corresponding to <code>topicref</code> attribute <code>collection-type</code> . If 'no', generate the navigation links corresponding to <code>topicref</code> attribute <code>collection-type</code> . If 'auto', generate the navigation links corresponding to <code>topicref</code> attribute <code>collection-type</code> , unless <code>chain-topics=yes</code> .
mark-external-links	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether an external link should be marked using a "opens in new window" icon.
navigation-icon-height	Length. A length may have a unit. Default is px. Default value: '16'.	The height of a navigation icon.
navigation-icon-suffix	String. Default value: '.png'.	The suffix of a navigation icon. The root names of navigation icons are fixed: first, first_disabled, last, last_disabled, next, next_disabled, previous, previous_disabled, parent, parent_disabled, child, child_disabled. For example, if <code>note-icon-suffix='.svg'</code> , the default resources directory is expected to contain <code>first.svg</code> , <code>first_disabled.svg</code> , <code>last.svg</code> , etc. In principle, there is no need for an end-user to specify any of the <code>navigation-icon-suffix</code> , <code>navigation-icon-width</code> or <code>navigation-icon-height</code> parameters.

Parameter	Value	Description
navigation-icon-width	Length. A length may have a unit. Default is px. Default value: '16'.	The width of a navigation icon.
screen-resolution	Positive integer. Default value: '96'.	The resolution of the screen in dot per inch (DPI). This resolution is used to convert image dimensions such as 3cm to pixels.
xhtml-mime-type	A MIME type without a parameter such as 'text/html', 'application/xhtml+xml', 'application/xml' or the empty string (''). Default value: 'text/html'.	<p>Applies to all the XHTML-based formats (XHTML, EPUB), not to the HTML-based formats (Web Help, Java™ Help, HTML Help, Eclipse Help).</p> <p>By default ('text/html'), serve XHTML as HTML.</p> <p>Specify 'application/xhtml+xml' if you prefer to serve XHTML as XML.</p> <p>Specify an empty string if you prefer not to generate <code><meta http-equiv="Content-Type" ...></code>.</p> <div>  Tip Web browsers such as Firefox or Opera will <i>not</i> render the MathML embedded in XHTML, if this XHTML is served as HTML. Therefore when your DITA document contains MathML equations, you'll have to generate ".xhtml" files(".html" files won't work) and also, preferably, to specify <code>xhtml-mime-type="application/xhtml+xml"</code> or <code>xhtml-mime-type=""</code>. </div>

Parameters common to the stylesheets that generate Web Help, Java™ Help, HTML Help, Eclipse Help and EPUB

Parameter	Value	Description
add-toc-root	Allowed values are: 'yes' and 'no'. Default value: 'yes'.	<p>If 'yes', add a pseudo TOC entry, bearing the title of the document, containing all the actual TOC entries.</p> <div>  Restriction <ul style="list-style-type: none"> Value 'no' is not supported by the stylesheets that generate Eclipse Help. Ignored by the stylesheets that generate Web Help and EPUB. </div>

Parameter	Value	Description
number-toc-entries	Allowed values are: 'yes' and 'no'. Default value: 'yes' for Web Help, 'no' for the other formats.	If 'yes', number the TOC entries. No effect unless the number parameter is used to specify that topics should be numbered.

Parameters specific to the stylesheets that generate Web Help

Parameter	Value	Description
★wh-collapse-toc	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether the TOC should be initially collapsed.
★wh-jquery	Relative or absolute URI. A relative URI is relative to the URI of a page of the Web Help. Default value: absolute URI of the corresponding file found on the Google CDN.	Specifies the location of the JavaScript file containing jQuery . Example: 'http://ajax.aspnetcdn.com/ajax/~jquery/jquery-1.9.1.min.js'.
★wh-jquery-css	Relative or absolute URI. A relative URI is relative to the URI of a page of the Web Help. Default value: absolute URI of the corresponding file found on the Google CDN.	Specifies the location of the CSS stylesheet of jQuery UI . Example: 'http://ajax.aspnetcdn.com/ajax/~jquery.ui/1.10.0/themes/redmond/jquery-ui.css'.
★wh-jquery-theme	The name of a theme. Examples: 'redmond', 'cupertino'. Default value: 'smoothness'.	Specifies the name of the jQuery UI theme used by the compiler. Ignored if parameter wh-jquery-css has been used to specify the CSS stylesheet of jQuery UI.
★wh-jquery-ui	Relative or absolute URI. A relative URI is relative to the URI of a page of the Web Help. Default value: absolute URI of the corresponding file found on the Google CDN.	Specifies the location of the JavaScript file containing jQuery UI . Example: 'http://ajax.aspnetcdn.com/ajax/~jquery.ui/1.10.0/jquery-ui.min.js'.
★wh-user-css	Relative or absolute URI of a CSS file. A relative URI is relative to the current working directory.	Specifies the user's CSS stylesheet which is to be added to each page of the Web Help. This file is copied to <code>output_directory/_wh/user/</code> . Sample user's CSS: wh_resources/header_footer.css .
★wh-user-footer	Relative or absolute URI of an XHTML file. A relative	Specifies the user's footer which is to be added to each page of the Web Help.

Parameter	Value	Description
	URI is relative to the current working directory.	<p>The content of the body element of <code>wh-user-footer</code> is inserted as is in the <code><div id="wh-footer"></code> found in a page of the Web Help.</p> <p>Same remark as for parameter <code>wh-user-header</code> about the resources referenced by a user's footer.</p> <p>Sample user's footer: wh_resources/footer.html.</p>
★ <code>wh-user-header</code>	Relative or absolute URI of an XHTML file. A relative URI is relative to the current working directory.	<p>Specifies the user's header which is to be added to each page of the Web Help.</p> <p>The content of the body element of <code>wh-user-header</code> is inserted as is in the <code><div id="wh-header"></code> found in a page of the Web Help.</p> <p>If a user's header references resources (e.g. image files), then these resources must either be referenced using absolute URLs or these resources must be found in a user's resource directory and parameter <code>wh-user-resources</code> must be specified.</p> <p>Example:</p> <ul style="list-style-type: none"> The user's resource directory is called <code>user2/</code> and contains <code>user2/logo100x50.png</code>. <code>ditac</code> is passed parameters: <code>-p user-resources user2</code> and <code>-p user-header header2.html</code>. <code>header2.html</code> looks like this: <pre><html> ... <body> </body> </html></pre> <p>Notice the path used to reference <code>logo100x50.png</code>.</p> <p>Sample user's header: wh_resources/header.html.</p>
★ <code>wh-user-resources</code>	<i>Filename</i> of a directory. A relative filename is relative to the current working directory.	<p>Specifies a user's resource directory which is to be recursively copied to <code>output_directory/_wh/user/</code>.</p> <p>This directory typically contains image files referenced by the user's header, footer or CSS stylesheet.</p> <p>Sample user's resource directory: wh_resources/header_footer_files/.</p>
<code>whc-index-basename</code>	<p>URL basename.</p> <p>Default value: <code>'whc_index.xml'</code>.</p>	<p>Basename of the Index XML input file of XMLmind Web Help Compiler.</p> <p>In principle, there is no need for an end-user to specify this parameter.</p>

Parameter	Value	Description
whc-toc-basename	URL basename. Default value: 'whc_toc.xml'.	Basename of the TOC XML input file of XMLmind Web Help Compiler. In principle, there is no need for an end-user to specify this parameter.


Parameters specific to the stylesheets that generate Java™ Help


In principle, there is no need for an end-user to specify any of the following parameters.

Parameter	Value	Description
helpset-basename	URL basename. Default value: 'jhelpset.hs'.	Basename of the Java™ Help HelpSet file.
index-basename	URL basename. Default value: 'jhelpidx.xml'.	Basename of the Java™ Help Index file.
indexer-directory-basename	URL basename. Default value: 'Java-HelpSearch'.	Basename of the directory which will contain the data generated by running jhindexer . A properly quoted relative URL, not a filename.
map-basename	URL basename. Default value: 'jhelpmap.jhm'.	Basename of the Java™ Help Map file.
toc-basename	URL basename. Default value: 'jhelp-toc.xml'.	Basename of the Java™ Help Contents file.


Parameters specific to the stylesheets that generate HTML Help

In principle, there is no need for an end-user to specify any of the following parameters.

Parameter	Value	Description
 chmBasename	URL basename. Default value: 'help.chm'.	Basename of the compiled HTML Help file.
hhc-basename	URL basename. Default value: 'toc.hhc'.	Basename of the HTML Help contents file.
hhp-template	URL basename. Default value: 'template.hhp' resolved against the directory which contains the XSLT stylesheets.	URL of the file containing the template of the HTML Help project file. This plain text file encoded in UTF-8 contains variables such as <i>%compiledFile%</i> , <i>%contentsFile%</i> , <i>%defaultTopic%</i> , etc, which are substituted with their values.

Parameter	Value	Description
 hhpBasename	URL basename. Default value: 'project.hhp'.	Basename of the HTML Help project file.
hhx-basename	URL basename. Default value: 'index.hhx'.	Basename of the HTML Help index file.

Parameters specific to the stylesheets that generate Eclipse Help

Parameter	Value	Description
plugin-id	String No default value.	An ID uniquely identifying the plug-in, typically a Java-like fully qualified name. Example: 'com.acme.widget.userguide'. <div>  Important The subdirectory of <code>plugins/</code> containing the plug-in must have the same basename as the value of parameter <code>plugin-id</code>. </div>
plugin-index-basename	URL basename. Default value: 'index.xml'.	Basename of the index file.
plugin-name	String No default value.	The name of the plug-in, typically the title of the document. Example: 'ACME Widget User's Guide'.
plugin-provider	String No default value.	The author, company or organization which has contributed the plug-in. Example: 'ACME Corp.'.
plugin-toc-basename	URL basename. Default value: 'toc.xml'.	Basename of the table of contents file.
plugin-version	String Default value: '1.0.0'.	The version of the plug-in.

Parameters specific to the stylesheets that generate EPUB

Parameter	Value	Description
cover-image	URI. If the URI is relative, it is relative to the current working directory of the user. No default value.	Specifies an image file which is to be used as the cover page of the EPUB file. This image must be a PNG or JPEG image. Its size must not exceed 1000x1000 pixels. In theory, EPUB 3 also accepts SVG 1.1 cover images.

Parameter	Value	Description
epub-identifier	String Default value: dynamically generated UUID URN.	A globally unique identifier for the generated EPUB document (typically the permanent URL of the EPUB document).
generate-epub-trigger	Allowed values are: 'yes' and 'no'. Default value: 'yes'.	Only applies to EPUB 3. Specify 'no' if your EPUB 3 reader does not support <code>epub:trigger</code> yet. When <code>generate-epub-trigger=no</code> , ditac generates an <code>onclick</code> attribute containing simple JavaScript code and declares the containing XHTML 5 page as being scripted.

Parameters specific to the stylesheets that generate XSL-FO

The XSL-FO file generated by the XSLT stylesheets is converted to PDF, PostScript®, RTF, WordprocessingML, Office Open XML (.docx), OpenOffice/LibreOffice (.odt) by the means of XSL-FO processors such as [Apache FOP](#), [RenderX XEP](#), [Antenna House XSL Formatter](#) or [XMLmind XSL-FO Converter](#).

Parameter	Value	Description
base-font-size	Default value: '10pt'.	The size of the ``main font" of the document. All the other font sizes are computed relatively to this font size
body-bottom-margin	Length. Default value: '0.5in'.	See Figure 4-1 below.
body-font-family	A string containing one or more font families separated by commas. Default value: 'serif'.	Specifies the family of the font used for the text of all elements except topic titles.
body-top-margin	Length. Default value: '0.5in'.	See Figure 4-1 below.
choice-bullets	A string containing one or more single characters separated by whitespace. Default value: '•' (BULLET).	Specify which bullet character to use for a choice element. Additional characters are used for nested choice elements. Changing the value of this parameter may imply changing the font-family attribute of the attribute-set choice-label.
external-href-after	String. Default value: '] '.	Appended after the external URL referenced by an <code>xref</code> or <code>link</code> element. Ignored unless <code>show-external-links='yes'</code> .
external-href-before	String. Default value: ' ['.	Separates the text of an <code>xref</code> or <code>link</code> element from its referenced external URL. Ignored unless <code>show-external-links='yes'</code> .

Parameter	Value	Description
 foProcessor	String. Examples: 'FOP', 'XEP', 'AHF', 'XFC'. Default value: ''.	The name of the XSL-FO processor used to convert the XSL-FO file generated by the XSLT stylesheets to the target output format.
footer-center	String. Default value:	Specifies the contents of the central part of a page footer. See Specifying a header or a footer . Supports a conditional specification . Default value: <pre>two-sides even:: {{chapter-title}};; two-sides part chapter appendices appendix odd::- {{section1-title}};; one-side even odd:: {{chapter-title}}</pre>
footer-center-width	String representing an integer larger than or equal to 1. Default value: '6'.	Specifies the proportional width of the central part of a page footer. See Specifying a header or a footer . Supports a conditional specification .
footer-height	Length. Default value: '0.4in'.	See Figure 4-1 below.
footer-left	String. Default value:	Specifies the contents of the left part of a page footer. See Specifying a header or a footer . Supports a conditional specification . Default value: <pre>two-sides even:: {{page-number}}</pre>
footer-left-width	String representing an integer larger than or equal to 1. Default value: '2'.	Specifies the proportional width of the left part of a page footer. See Specifying a header or a footer . Supports a conditional specification .
footer-right	String. Default value:	Specifies the contents of the right part of a page footer. See Specifying a header or a footer . Supports a conditional specification . Default value: <pre>two-sides first odd:: {{page-number}};; one-side:: {{page-number}}</pre>
footer-right-width	String representing an integer larger than or equal to 1. Default value: '2'.	Specifies the proportional width of the right part of a page footer. See Specifying a header or a footer . Supports a conditional specification .

Parameter	Value	Description
footer-separator	Allowed values are: 'yes' and 'no'. Default value: 'yes'.	Specifies whether an horizontal rule should be drawn above the page footer.
header-center	String. Default value: '{{document-title}}'.	Specifies the contents of the central part of a page header. See Specifying a header or a footer . Supports a conditional specification .
header-center-width	String representing an integer larger than or equal to 1. Default value: '6'.	Specifies the proportional width of the central part of a page header. See Specifying a header or a footer . Supports a conditional specification .
header-height	Length. Default value: '0.4in'.	See Figure 4-1 below.
header-left	String. Default value: ''.	Specifies the contents of the left part of a page header. See Specifying a header or a footer . Supports a conditional specification .
header-left-width	String representing an integer larger than or equal to 1. Default value: '2'.	Specifies the proportional width of the left part of a page header. See Specifying a header or a footer . Supports a conditional specification .
header-right	String. Default value: ''.	Specifies the contents of the right part of a page header. See Specifying a header or a footer . Supports a conditional specification .
header-right-width	String representing an integer larger than or equal to 1. Default value: '2'.	Specifies the proportional width of the right part of a page header. See Specifying a header or a footer . Supports a conditional specification .
hyphenate	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether words may be hyphenated.
index-column-count	Positive integer. Default value: '2'.	The number of columns of index pages.
index-column-gap	Length. Default value: '12pt'.	The distance which separates columns in index pages.

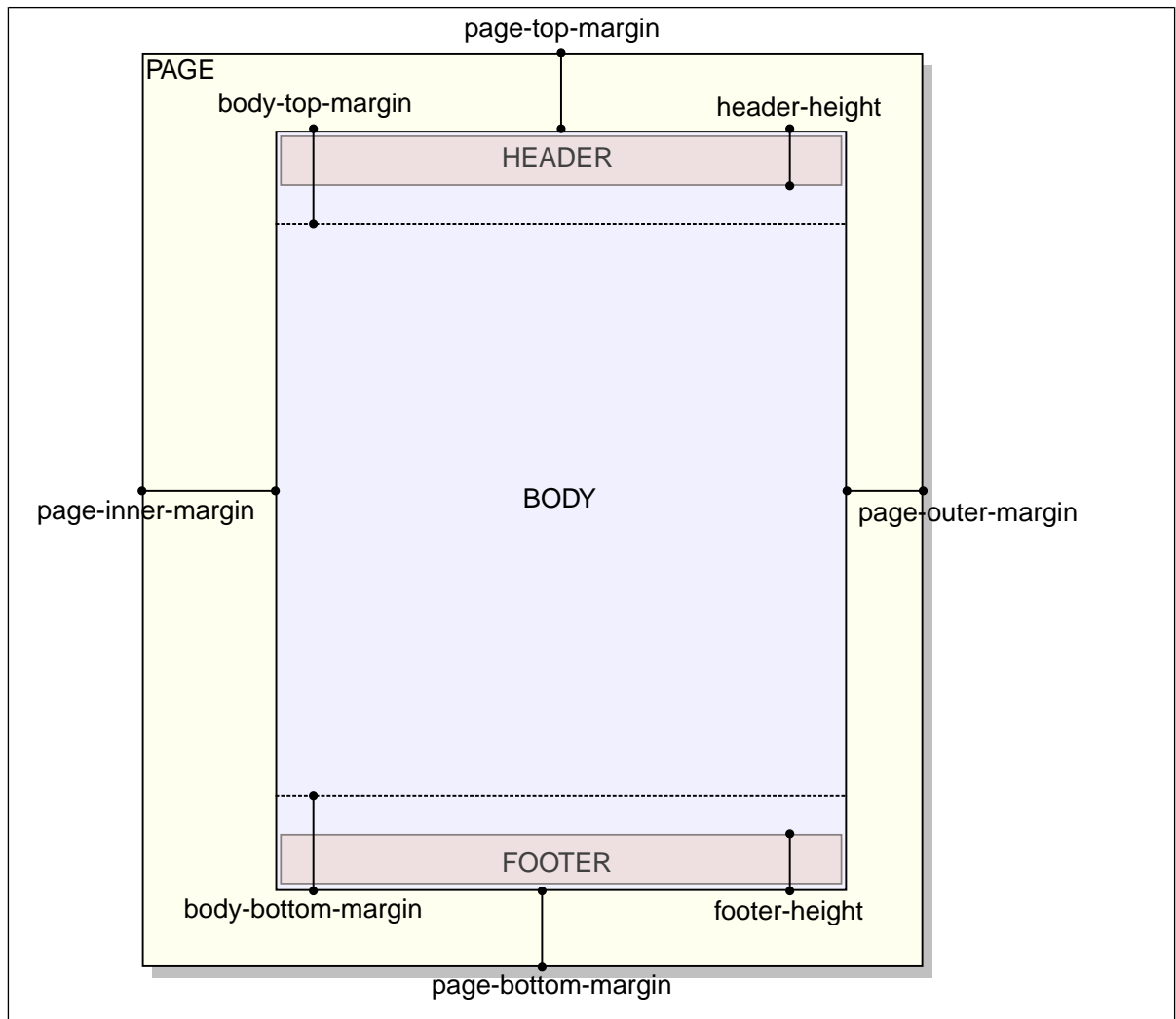
Parameter	Value	Description
justified	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether text (e.g. in paragraphs) should be justified (that is, flush left and right) or just left aligned (that is, flush left and ragged right).
link-bullet	A string containing a single character. Default value: '•' (BULLET).	Specify which character is inserted before the text of a link element. Changing the value of this parameter may imply changing the font-family attribute of the attribute-set link-bullet.
menucascade-separator	A string containing a single character. Default value: '→' (RIGHTWARDS ARROW).	Specify which character is used to separate the child elements of a menucascade element. Changing the value of this parameter may imply changing the font-family attribute of the attribute-set menucascade-separator.
note-icon-height	Length. A length may have a unit. Default is px. Default value: '32'. '7mm' for the XSLT stylesheets that generate XSL-FO.	The height of a note icon.
note-icon-suffix	Default value: '.png'.	The suffix of a note icon. The root name of a note icon should be identical to the value of the type attribute it represents. For example, if note-icon-suffix='.svg', the default resources directory is expected to contain note.svg, important.svg, caution.svg, etc. In principle, there is no need for an end-user to specify any of the note-icon-suffix, note-icon-width or note-icon-height parameters.
note-icon-width	Length. A length may have a unit. Default is px. Default value: '32'. '7mm' for the XSLT stylesheets that generate XSL-FO.	The width of a note icon.
page-bottom-margin	Length. Default value: '0.5in'.	See Figure 4-1 below.
page-height	Length. Example: '297mm'. Default value: depends on paper-type.	The height of the printed page.

Parameter	Value	Description
page-inner-margin	Length. Default value: if parameter two-sided is specified as 'yes' then '1.25in' otherwise '1in'.	See Figure 4-1 below.
page-orientation	Allowed values are: 'portrait' and 'landscape'. Default value: 'portrait'.	The orientation of the printed page.
page-outer-margin	Length. Default value: if parameter two-sided is specified as 'yes' then '0.75in' otherwise '1in'.	See Figure 4-1 below.
page-ref-after	String. Default value: ''.	Appended after the page number pointed to by an xref or link element. Ignored unless show-xref-page='yes' or show-link-page='yes'. When both page-ref-after and page-ref-before are specified as the empty string, in fact, this specifies that the generated string must be the localized equivalent of "on page".
page-ref-before	String. Default value: ''.	Separates the text of an xref or link element from the page number it points to. Ignored unless show-xref-page='yes' or show-link-page='yes'.
page-top-margin	Length. Default value: '0.5in'.	See Figure 4-1 below.
page-width	Length. Example: '8.5in'. Default value: depends on paper-type.	The width of the printed page.
paper-type	Allowed values are: 'Letter', 'Legal', 'Ledger', 'Tabloid', 'A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10', 'B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9', 'B10', 'C0', 'C1', 'C2',	A convenient way to specify the size of the printed page. It is also possible to specify a custom paper type by ignoring the paper-type parameter and directly specifying the page-width and page-height parameters.

Parameter	Value	Description
	'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10' (case-insensitive). Default value: 'A4'.	
pdf-outline	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether PDF bookmarks should be generated. Supported by the 'XEP', 'FOP' and 'AHF' XSL-FO processors. Not relevant, and thus ignored by 'XFC'.
show-external-links	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether the <i>external URL</i> referenced by an xref or link element should be displayed right after the text contained by this element. Example: show-external-links='yes' causes <xref href="http://www.oasis-open.org/">Oasis</xref> to be rendered as follows: Oasis [http://www.oasis-open.org/].
show-link-page	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Same as show-xref-page but for link elements.
show-xref-page	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether the page number corresponding to the <i>internal link target</i> referenced by an xref element should be displayed right after the text contained by this element. Example: show-xref-page='yes' causes <xref href="introduction.dita">Introduction</xref> to be rendered as follows: Oasis [3].
title-color	A string representing a color. Default value: 'black'.	Specifies the color used for the text of topic (of any kind) titles.
title-font-family	A string containing one or more font families separated by commas. Default value: 'sans-serif'.	Specifies the family of the font used for the text of topic (of any kind) titles.
two-sided	Allowed values are: 'yes' and 'no'. Default value: 'no'.	Specifies whether the document should be printed double sided.
ul-li-bullets	A string containing one or more single characters separated by whitespace. Default value: '•'	Specify which bullet character to use for an ul/li element. Additional characters are used for nested li elements. For example, if ul-li-bullets="* - +", "*" will be used for ul/li elements, "-" will be used for ul/li elements contained in a ul/li element and "+" will be used for ul/li elements nested in two ul/li elements.

Parameter	Value	Description
	– ' (BULLET, EN DASH).	Changing the value of this parameter may imply changing the font-family attribute of the attribute-set ul-li-label.
unordered-step-bullets	A string containing one or more single characters separated by whitespace. Default value: '•' (BULLET, EN DASH).	Specify which bullet character to use for a steps-unordered/step element. Additional characters are used for nested steps-unordered/step elements. Changing the value of this parameter may imply changing the font-family attribute of the attribute-set unordered-step-label.

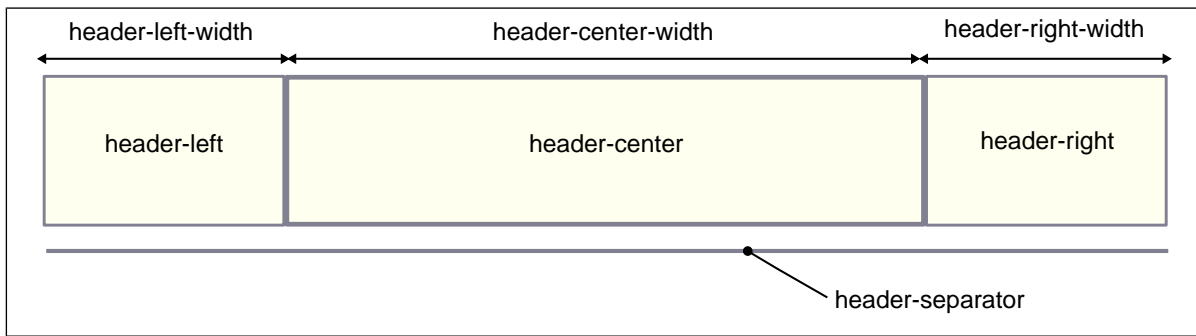
Figure 4-1. Page areas



Specifying a header or a footer

The header or the footer of a generated PDF, RTF, etc, page has 3 columns.

Figure 4-2. Layout of a header



The width of these columns may be specified using the `header-left-width`, `header-center-width`, `header-right-width` parameters for the header and the `footer-left-width`, `footer-center-width`, `footer-right-width` parameters for the footer.

The width of a column is specified as an integer which is larger than or equal to 1. This value is the *proportional width of the column*. For example, if the left column has a width equal to 2 and the right column has a width equal to 4, this simply means that the right column is twice ($4/2 = 2$) as wide as the left column.

The contents of these columns may be specified using the `header-left`, `header-center`, `header-right` parameters for the header and the `footer-left`, `footer-center`, `footer-right` parameters for the footer.

When `header-left`, `header-center`, `header-right` are all specified as the empty string, no header is generated. When `footer-left`, `footer-center`, `footer-right` are all specified as the empty string, no footer is generated.

The content of a column is basically a mix of text and variables. Example: "Page `{{page-number}}` of `{{page-count}}`".

Supported variables are:

`{{document-title}}`

The title of the document.

`{{document-date}}`

The publication date of the document.

`{{chapter-title}}`

The title of the current part, chapter, appendices or appendix. Empty if the map being converted is not a bookmap.

`{{section1-title}}`

The title of the current part, chapter, appendices or appendix or *section 1*. A section 1 is specified by a non-typed `topicref` (that is, not a part, chapter, preface, appendix, dedication, etc) which is a direct child of a map or bookmap.

`{{topic-title}}`

The title of the current topic. All topics are guaranteed to have a corresponding `{{topic-title}}`. Even automatically generated topics such as `toc` or `indexlist` have a corresponding `{{topic-title}}`⁽⁶⁾.

`{{page-number}}`

Current page number within the current document division (front matter, body matter or back matter).

`{{page-count}}`

Total number of pages of the current document division (front matter, body matter or back matter).

⁽⁶⁾ The `{{topic-title}}` of a `toc` is "Table of Contents", properly localized. The `{{topic-title}}` of a `indexlist` is "Index", properly localized.

`{{break}}`

A line break.

`{{image(URI)}}`

An image having specified URI. A relative URI is resolved against the current working directory. Example:
`"{{image(artwork/logo.svg) }}"`.

`{{page-sequence}}`

Not for production use. Inserts in the header/footer the name of the current page sequence . This allows to learn which name to use in a *conditional header or footer*. See [below](#).

Conditional headers and footers

The default value of `header-center` is `'{{document-title}}'`. This means that each page of the generated PDF, RTF, etc, file will have the document title centered on its top. But what if you want the pages containing the Table of Contents have a **"Contents"** header? Is there a way to specify: use **"Contents"** for the pages containing the Table of Contents and use the title of the document for any other page?

This is done by specifying the following conditional value for parameter `header-center`: `'toc:: Contents;; {{document-title}}'`.

A conditional value may contain one or more cases separated by `;;`. Each case is tested against the page being generated. The first case which matches the page being generated is the one which is selected.

*conditional_value --> case [";;" case]**

case --> [condition "::"] value*

*condition --> [test_page_sequence]?
 & [S test_page_layout]?
 & [S test_page_side]?*

Let's suppose you also want the the pages containing the Index have a **"Index"** header. Specifying `'toc:: Contents;; {{document-title}};; indexlist:: Index'` won't work as expected because the second case (having no condition at all) matches any page, including the Index pages. You need to specify: `'toc:: Contents;; indexlist:: Index;; {{document-title}}'`.

Let's remember that variable `{{topic-title}}` is substituted with the title of the current topic, including automatically generated topics such `toc` and `indexlist`.

Therefore our conditional value is better expressed as: `'toc:: indexlist:: {{topic-title}};; {{document-title}}'`. Notice how a case may have several conditions. Suffice for any of these conditions to match the page being generated for the case to be selected.

Even better, specify `'toc||indexlist:: {{topic-title}};; {{document-title}}'`. String `"||"` may be used to separate alternative values to be tested against the page being generated.

*test_page_sequence --> page_sequence ["||" page_sequence]**

*page_sequence --> "abbrevlist" | "amendments" | "appendices" | "appendix"
 | "backmattersection" | "bibliolist" | "bookabstract" | "booklist"
 | "chapter" | "colophon" | "dedication" | "draftintro"
 | "figurelist" | "glossarylist" | "indexlist" | "notices"
 | "part" | "preface" | "section1" | "tablelist"
 | "toc" | "trademarklist"*



Tip

It's not difficult to guess that the name of the page sequence corresponding to the Table of Contents is `toc` and that the name of the page sequence corresponding to the Index is `indexlist`. However the simplest way to learn what is the name of the page sequence being generated is to reference variable `{{page-sequence}}` in the specification of a header or a footer.

Now let's suppose that we want to suppress the document title on the first page of a part, chapter or appendix. This is specified as follows: `'first part|chapter|appendix:: ;; toc|indexlist:: {{topic-title}};; {{document-title}}'`.

For now, we have only described a condition about the page sequence being generated: TOC, Index, etc. In fact, a condition may test up to 3 facets of the page being generated:

- The page sequence to which belongs the page being generated.
- Whether the page being generated is part of a one-sided or a two-sided document.
- Whether the page being generated is the first page of its sequence. When the the page being generated is *not* the first page of its sequence, if the page being generated has an odd or an even page number.

```
test_page_layout --> page_layout [ "|" page_layout ]*
```

```
page_layout --> "two-sides" | "one-side"
```

```
test_page_side --> page_side [ "|" page_side ]*
```

```
page_side --> "first" | "odd" | "even"
```



Remember

When the document has one side, there is no difference between even and odd. That is, even, odd, even|odd all simply mean: other than first.

The order of the tests is not significant. For example, `'first part|chapter|appendix'` is equivalent to `'part|chapter|appendix first'`.

Therefore `'first part|chapter|appendix:: ;; toc|indexlist:: {{topic-title}};; {{document-title}}'` reads as follows:

1. Use the empty string for the first page of a part, chapter or appendix.
2. Use the topic title for the pages containing the Table of Contents. This title is "**Table of Contents**", but localized according to the main language of the DITA document being converted.
3. Use the topic title for the pages containing the Index. This title is "**Index**", but localized according to the main language of the DITA document being converted.
4. For any other page, use the title of the DITA document.



Note

Everything explained in this section applies not only to the contents of a column of a header or footer, but also to the proportional width of a column of a header or footer. Example: `-p footer-right-width "first|odd:: 4;; even:: 1"`.

Chapter 5. Syntax highlighting

This chapter explains how you can automatically colorize the source code contained in `pre`, `codeblock` or any other element specializing `pre`.

You can automatically colorize the source code contained in `pre`, `codeblock` or any other element specializing `pre`. This feature, commonly called *syntax highlighting*, has been implemented using an open source software component called "[XSLT syntax highlighting](#)".

If you want to turn on syntax highlighting in a DITA document, suffice to add attribute `outputclass` to a `pre`, `codeblock` or any other element specializing `pre`. The value of attribute `outputclass` must be any of: `language-c`, `language-cpp`, `language-csharp`, `language-delphi`, `language-ini`, `language-java`, `language-javascript`, `language-m2` (Modula 2), `language-perl`, `language-php`, `language-python`, `language-ruby`, `language-tcl`, `language-xml`.

If you want to customize syntax highlighting for an HTML-based output format (XHTML, EPUB, etc), then redefine any of the following CSS styles:

- `.hl-keyword` (keywords of a programming language),
- `.hl-string` (string literal),
- `.hl-number` (number literal),
- `.hl-comment` (any type of comment),
- `.hl-doccomment` (comments used as documentation, i.e. javadoc, or xmldoc),
- `.hl-directive` (preprocessor directive or in XML, a processing-instruction),
- `.hl-annotation` (annotations or "attributes" as they are called in .NET),
- `.hl-tag` (XML tag, i.e. element name),
- `.hl-attribute` (XML attribute name),
- `.hl-value` (XML attribute value),
- `.hl-doctype` (`<!DOCTYPE>` and all its content).

Example:

```
.hl-keyword {
    font-weight: bold;
    color: #602060;
}
```

How to use a custom CSS stylesheet is explained in [Part II, Chapter 7, Section 1](#).

If you want to customize syntax highlighting for an XSL-FO-based output format (PDF, RTF, etc), then redefine any of the following attribute-sets: `hl-keyword`, `hl-string`, `hl-number`, `hl-comment`, `hl-doccomment`, `hl-directive`, `hl-annotation`, `hl-tag`, `hl-attribute`, `hl-value`, `hl-doctype`.

Example:

```
<xsl:attribute-set name="hl-keyword" use-attribute-sets="hl-style">
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="color">#602060</xsl:attribute>
</xsl:attribute-set>
```

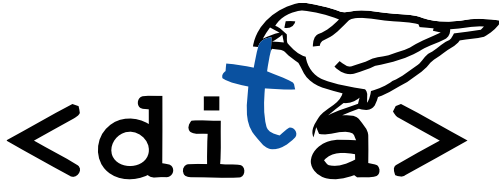
How to use a custom XSLT stylesheet generating XSL-FO is explained in [Part II, Chapter 7, Section 2](#).

Chapter 6. Rich media content

This chapter explains how to add SVG, MathML, audio, video and Flash animations to your DITA topics and how **ditac** processes this rich media content in the case where the output format supports rich media (e.g. XHTML 5, EPUB 3) and also in the case where the output format does not support rich media (e.g. XHTML 1, PDF, RTF).

SVG

It is possible to include SVG graphics in a DITA topic either by reference or by inclusion. Use an `image` element pointing to an SVG file to include it by reference. Example:



The XML source code corresponding to the above example is:

```
<p><image href="media/graphic.svg" /></p>
```

Embed the `svg:svg` element in a `foreign` element (or any element specializing `foreign`) to include it by inclusion. Example:



The XML source code corresponding to the above example is:

```
<p><foreign><svg:svg height="140" id="svg2" version="1.1"
  viewBox="0 0 264 120" width="320"
  xmlns:svg="http://www.w3.org/2000/svg">
  <svg:defs id="defs39"/>
  ...
  </svg:g>
</svg:svg></foreign></p>
```

- It is recommended to include SVG graphics by reference as the `image` element has useful attributes (`width`, `height`, `scale`, `scalefit`) allowing to adjust the dimension of the image.
- Only the following screen formats may contain SVG: XHTML 5, XHTML 5 Web Help and EPUB 3. Note that only modern web browsers support XHTML 5 and XHTML 5 Web Help. Very few EPUB readers (e.g. iBooks) support EPUB 3.
- All XSL-FO based formats support SVG whatever the XSL-FO processor you may use.

MathML

Embed the `mml:math` element in a `foreign` element (or any element specializing `foreign`) to add math to your DITA topics. Example:

$$\{ \times E = - B \quad t \times B = \mu_0 \quad J + \mu_0 \quad 0 \quad E \quad t$$


The XML source code corresponding to the above example is:

```
<p><foreign><mml:math display="block"
xmlns:mml="http://www.w3.org/1998/Math/MathML">
<mml:mrow>
<mml:mo>{</mml:mo>
<mml:mtable>
<mml:mtr>
<mml:mtd>
...
</mml:mtd>
</mml:mtr>
</mml:mtable>
</mml:mrow>
</mml:math></foreign></p>
```

- Only the following screen formats may contain MathML: XHTML 5, XHTML 5 Web Help and EPUB 3. Most modern web browsers (Firefox, Chrome) support XHTML 5 and XHTML 5 Web Help containing MathML. Very few EPUB readers (e.g. iBooks) support EPUB 3.
- XSL-FO based formats support MathML depending on the XSL-FO processor you use:
 - Apache FOP requires you to download and install the the [JEuclid FOP plug-in](#).
 - RenderX XEP does not support MathML.
 - Antenna House Formatter supports MathML as an option.
 - XMLmind XSL-FO Converter supports MathML out of the box.

Audio

Use the object DITA element to add audio to your DITA topics. Example:

 [audio.mp3 \(audio/mpeg\)](#)

The XML source code corresponding to the above example is:

```
<p><object data="media/audio.mp3" type="audio/mpeg">
<param name="source.src" value="media/audio.ogg"/>
<param name="source.type" value="audio/ogg"/>

<param name="source.src" value="media/audio.m4a"/>
<param name="source.type" value="audio/mp4"/>

<param name="source.src" value="media/audio.wav"/>
<param name="source.type" value="audio/wav"/>

<param name="controls" value="true"/>
</object></p>
```

- The data and type attributes are required. The value of the type attribute must start with "audio/".
- It is strongly recommended to specify *alternate audio files* as modern web browsers, while all supporting the HTML 5 audio element, vary in their support of audio formats. This is done by adding pairs of param child elements to the object element. The first param of a pair must have name="source.src" attribute and its value attribute must reference an audio file. The second param of a pair must have name="source.type" attribute and its value attribute must contain the media type of the preceding audio file.
- It is possible to add param elements corresponding to the attributes supported by the HTML 5 audio element (crossorigin, preload, autoplay, mediagroup, loop, muted, controls). In the above example, we have added a param element corresponding to the controls HTML 5 attribute. Note that in the case of HTML 5 *boolean* attributes (autoplay, loop, muted, controls), the value attribute of a param is not significant. For example, in the case of the above example, you could have specified "yes", "on", "1", etc, instead of "true".
- If the object element has a desc child element, then this desc element is used to generate fallback content in case audio is not supported. If the object element has no desc child element, then a simple fallback content

is automatically generated by ditac. This automatic fallback content basically consists in a link allowing to download the audio file.

- When ditac is used to generate an XSL-FO based format (PDF, RTF, etc), only the fallback content appears in the output file.

Video

Use the `object` DITA element to add video to your DITA topics. Example:



[video.mp4 \(video/mp4\)](#)

The XML source code corresponding to the above example is:

```
<p><object data="media/video.mp4" type="video/mp4">
  <param name="source.src" value="media/video.ogv"/>
  <param name="source.type" value='video/ogg; codecs="theora, vorbis"'/>

  <param name="source.src" value="media/video.webm"/>
  <param name="source.type" value="video/webm"/>

  <param name="width" value="320"/>
  <param name="controls" value="yes"/>
  <param name="poster" value="media/video_poster.jpg"/>
</object></p>
```

- The `data` and `type` attributes are required. The value of the `type` attribute must start with "video/".
- It is strongly recommended to specify *alternate video files* as modern web browsers, while all supporting the HTML 5 `video` element, vary in their support of video formats. This is done by adding pairs of `param` child elements to the `object` element. The first `param` of a pair must have `name="source.src"` attribute and its `value` attribute must reference an video file. The second `param` of a pair must have `name="source.type"` attribute and its `value` attribute must contain the media type of the preceding video file.
- It is possible to add `param` elements corresponding to the attributes supported by the HTML 5 `video` element (`crossorigin`, `poster`, `preload`, `autoplay`, `mediagroup`, `loop`, `muted`, `controls`, `width`, `height`). In the above example, we have added a `param` element corresponding to the `width`, `controls` and `poster` HTML 5 attributes. Note that in the case of HTML 5 *boolean* attributes (`autoplay`, `loop`, `muted`, `controls`), the `value` attribute of a `param` is not significant. For example, in the case of the above example, you could have specified "true", "on", "1", etc, instead of "yes".
- If the `object` element has a `desc` child element, then this `desc` element is used to generate fallback content in case video is not supported. If the `object` element has no `desc` child element, then a simple fallback content is automatically generated by ditac. This automatic fallback content basically consists in a link allowing to download the video file. The `param` element corresponding to the `poster` HTML 5 attribute, if present, is used to generate a nicer automatic fallback content.
- When ditac is used to generate an XSL-FO based format (PDF, RTF, etc), only the fallback content appears in the output file.

Flash animation

Use the `object` DITA element to add Adobe® Flash® animations to your DITA topics. Example:

 [animation.swf \(application/x-shockwave-flash\)](#)

(You may have to right-click on the above screenshot and select **Play** from the Flash popup menu to replay the animation.)

The XML source code corresponding to the above example is:

```
<p><object data="animation.swf"
         type="application/x-shockwave-flash"
         width="431" height="123">
  <param name="movie" value="animation.swf" />

  <param name="menu" value="true" />
  <param name="quality" value="low" />
</object></p>
```

- The data, type, width and height attributes are required. The param name=movie child element having the same value as attribute data is required too.
- You may add any other param child element supported by the Flash object. In the above example, you'll find menu and quality in addition to required movie.
- If the object element has a desc child element, then this desc element is used to generate fallback content in case Flash is not supported. If the object element has no desc child element, then a simple fallback content is automatically generated by ditac. This automatic fallback content basically consists in a link allowing to download the .swf file.
- When ditac is used to generate an XSL-FO based format (PDF, RTF, etc), only the fallback content appears in the output file.

Actions

Unless you add param name="controls" (see [above](#)), you'll not be able to play audio or video. Even worse, without the controls param, an audio object is not rendered on screen (that is, it is invisible).

A simple solution for this problem is to insert a `<?onclick?>` processing-instruction in a DITA element (typically an *inline* element such as xref or ph). The `<?onclick?>` processing-instruction allows to specify an number of actions:

play

Play the associated resource from the beginning. Only applicable to video or audio targets.

pause

Pause playing . Only applicable to video or audio targets.

resume

Resume playing . Only applicable to video or audio targets.

mute

Mute sound . Only applicable to video or audio targets.

unmute

Unmute sound . Only applicable to video or audio targets.

show

Set the visibility property of the target element to visible.

hide

Set the visibility property of the target element to hidden.

The above actions are exactly those supported by EPUB 3's `epub:trigger`.


The `<?onclick?>` processing-instruction is processed by **ditac** for the following output formats: XHTML 5, XHTML 5 Web Help and EPUB 3. It is discarded for any other output format.

The syntax for the content of `<?onclick?>` is:

```
onclick_data -> action (S action)*
action -> op '(' target_id? ')'
op -> 'play' | 'pause' | 'resume' | 'mute' | 'unmute'
      'show' | 'hide'
```

When *target_id* is not specified, it is taken from the href attribute of the element containing the `<?onclick?>` processing-instruction. For example, `<xref href="#media/target_audio"><?onclick play()?>` is equivalent to: `<xref href="#media/target_audio"><?onclick play(media/target_audio)?>`.

Example 1: Say: "*Viens Hubble!*", which, in French, means: "Come here Hubble!".

 No audio. Say: "*Viens Hubble!*", which, in French, means: "Come here Hubble!".

The XML source code corresponding to the above example is:

```
<p>Example 1: <xref href="#media/target_audio"><?onclick play()?>
Say "<ph xml:lang="fr">Viens Hubble!</ph>"</xref>
...
<object data="media/audio.wav" id="audio_sample" type="audio/wav">
  <desc> ... </desc>
</object></p>
```

Example 2: Hide Hubble. Show Hubble.

Figure 6-1. My name is Hubble. I'm a 7-month old Golden Retriever.



The XML source code corresponding to the above example is:

```
<p>Example 2:
<xref href="#media/target_image"><?onclick hide()?>Hide Hubble</xref>.
<xref href="#media/target_image"><?onclick show()?>Show Hubble</xref>.</p>
```


Part II. Customizing the output of XMLmind DITA Converter

Chapter 7. Simple customization

1. Customize the look of the HTML pages generated by ditac

We'll explain how to customize the look of the HTML pages generated by ditac by using an example. Let's suppose we want to render topic titles in a nice dark blue color rather than in black.



Restriction

The customization method described below will not work for formats for which the generated HTML pages are automatically compiled or archived. This is the case for HTML Help, Java™ Help and EPUB.

About this task

The easiest way to customize the look of the HTML pages generated by ditac is to use a custom CSS stylesheet rather than the stock one.

Procedure

1. Create a custom CSS stylesheet importing the stock CSS stylesheet.

The stock CSS stylesheet is found in:

`ditac_install_dir/xsl/xhtml/resources/basic.css`

Used for the XHTML 1.0, XHTML 1.1, HTML 4.01 and XHTML 5 output formats.

`ditac_install_dir/xsl/webhelp/resources/webhelp.css`

Used for the Web Help output format.

`ditac_install_dir/xsl/htmlhelp/resources/basic.css`

Used for the HTML Help output format.

`ditac_install_dir/xsl/eclipsehelp/resources/basic.css`

Used for the Eclipse Help output format.

`ditac_install_dir/xsl/javahelp/resources/javahelp.css`

Used for the Java™ Help output format.

`ditac_install_dir/xsl/epub/resources/basic.css`

Used for the EPUB output format.

Initial contents of the custom CSS stylesheet (a copy of this file is found in [customize/custom.css](#)).

```
@import url(basic.css);
```

2. Add one or more rules to the custom CSS stylesheet.

The XSLT stylesheets generating XHTML/HTML pages make extensive use of the class attribute. Generally the XHTML element generated for a DITA element has a `class` attribute bearing the name of the DITA element. Example: a DITA `<p>` is converted to a XHTML `<div class="p">`.

For more information, you'll have to refer to the stock CSS stylesheet or even to the HTML pages generated by ditac.

```
@import url(basic.css);
```

```
.part-title,  
.chapter-title,  
.appendix-title,
```

```
.section1-title,
.section2-title,
.section3-title,
.section4-title,
.section5-title,
.section6-title,
.section7-title,
.section8-title,
.section9-title,
.topic-title {
    color: #403480;
    border-bottom: 1px solid #403480;
}
```

3. Specify the "-p `css` `res/custom.css`" option when running `ditac`.

```
$ ditac -images img -p xsl-resources-directory res \
  -p css res/custom.css \
  out/manual/_html manual.ditamap
```

4. Last but not least, do not forget to copy `custom.css` to the resources subdirectory of the output directory (`out/manual/res/` in the case of the above example). The `ditac` command-line cannot do that automatically for you.

```
$ cp customize/custom.css out/manual/res
```

2. Customizing the look of the PDF files generated by ditac

We'll explain how to customize the look of the PDF files generated by `ditac` by using an example. Let's suppose we want to render topic titles in a nice dark blue color rather than in black.

About this task

A PDF file is created by converting the XSL-FO file generated by the `ditac` XSLT 2.0 stylesheet by the means of an XSL-FO processor such as Apache FOP, RenderX XEP or Antenna House Formatter. Therefore we need to generate a custom XSL-FO file. This is done by creating a very simple variant of the stock XSLT stylesheet which generates XSL-FO.

Procedure

1. Create a custom XSLT stylesheet importing the stock one.

The stock (topmost) XSLT stylesheets are:

`ditac_install_dir/xsl/fo/fo.xsl`

Used to generate an intermediate XSL-FO file. After that, the XSL-FO file is converted to PDF, PostScript, RTF, WordprocessingML, Office Open XML (.docx) or OpenOffice/LibreOffice (.odt) by the means of an XSL-FO processor.

`ditac_install_dir/xsl/xhtml/xhtml.xsl`

Used to generate XHTML 1.0 pages.

`ditac_install_dir/xsl/xhtml/xhtml1_1.xsl`

Used to generate XHTML 1.1 pages.

`ditac_install_dir/xsl/xhtml/html.xsl`

Used to generate HTML 4.01 pages.

`ditac_install_dir/xsl/xhtml/xhtml15.xsl`

Used to generate HTML 4.01 pages.

`ditac_install_dir/xsl/webhelp/webhelp.xsl`

Used to generate Web Help containing XHTML 1 pages, which are then compiled using [XMLmind Web Help Compiler](#).

`ditac_install_dir/xsl/webhelp/webhelp5.xsl`

Used to generate Web Help containing XHTML 5 pages, which are then compiled using [XMLmind Web Help Compiler](#).

`ditac_install_dir/xsl/htmlhelp/htmlhelp.xsl`

Used to generate HTML Help files, which are then compiled using `hhc.exe`.

`ditac_install_dir/xsl/eclipsehelp/eclipsehelp.xsl`

Used to generate Eclipse Help files.

`ditac_install_dir/xsl/javaahelp/javaahelp.xsl`

Used to generate Java™ Help files, which are then archived in a `.jar` file.

`ditac_install_dir/xsl/epub/epub.xsl`

Used to generate EPUB 2 files, which are then archived in a `.epub` file (Zip archive having a `.epub` extension).

`ditac_install_dir/xsl/epub/epub3.xsl`

Used to generate EPUB 3 files, which are then archived in a `.epub` file (Zip archive having a `.epub` extension).

Initial contents of the custom XSLT stylesheet (a copy of this file is found in [customize/custom_fo.xsl](#)).

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="2.0">

  <xsl:import href="ditac-xsl:fo/fo.xsl"/>

</xsl:stylesheet>
```

Notice the funny looking URI `"ditac-xsl:fo/fo.xsl"`. `"ditac-xsl:"` is an easy way to refer to `ditac_install_dir/xsl/`. This works because the XML catalog used by the `ditac` command-line utility (found in `ditac_install_dir/schema/catalog.xml`) contains:

```
<rewriteURI uriStartString="ditac-xsl:" rewritePrefix="../xsl/" />
```

2. Redefine one or more named `xsl:attribute-sets` in your custom XSLT stylesheet.

Named `xsl:attribute-sets` are not documented yet. For more information, you'll have to refer to the XSLT stylesheets found in `ditac_install_dir/xsl/fo/`.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="2.0">

  <xsl:import href="ditac-xsl:fo/fo.xsl"/>

  <xsl:attribute-set name="topic-title" use-attribute-sets="topic-title">
    <xsl:attribute name="color">#403480</xsl:attribute>
    <xsl:attribute name="font-size">160%</xsl:attribute>
    <xsl:attribute name="padding-bottom">0.05em</xsl:attribute>
    <xsl:attribute name="border-bottom">0.5pt solid #403480</xsl:attribute>
    <xsl:attribute name="space-before.optimum">1.5em</xsl:attribute>
    <xsl:attribute name="space-before.minimum">1.2em</xsl:attribute>
    <xsl:attribute name="space-before.maximum">1.8em</xsl:attribute>
  </xsl:attribute-set>
```

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

3. Specify the "-t customize/custom_fo.xml" option when running ditac.

```
$ ditac -t customize/custom_fo.xml \  
    out/manual.pdf manual.ditamap
```

Alternatively, package your custom XSLT stylesheet as a [plug-in](#) and then specify the name of this plug-in using the [-plugin](#) command-line option. By doing this, your custom XSLT stylesheet will be automatically used whatever the output format which uses XSL-FO as its intermediate format (PDF, RTF, .odt, .docx, etc).

Chapter 8. Using ditac to convert documents conforming to a DITA specialization

We'll explain by example how to use ditac to convert documents conforming to a DITA specialization. Let's suppose we have a DITA specialization which adds a `tag` element (similar to [DocBook 5's tag element](#)) to topic contents. This tag element is modelled as follows (see [sample_plugin/dtd/technicalDomain.mod](#))

```
<!ELEMENT tag (#PCDATA)*>

<!ATTLIST tag %univ-atts;
            outputclass CDATA #IMPLIED>

<!ATTLIST tag %global-atts;
            class CDATA "+ topic/keyword tech-d/tag ">

<!ATTLIST tag %univ-atts;
            kind (attribute|attvalue|element|emptytag|endtag|
                genentity|localname|namespace|numcharref|
                paramentity|pi|prefix|comment|starttag) #REQUIRED>
```

All the example files of this tutorial have been packaged as a **plug-in** called "sample_plugin". They are found in directory `sample_plugin/`. In order to give this plug-in a try, you'll have to copy directory `sample_plugin/` to `ditac_install_dir/plugin/`.

About this task

Using ditac to convert documents conforming to a DITA specialization basically requires customizing the output of the tool using the same techniques as those explained in [Chapter 7, Section 1](#) and [Chapter 7, Section 2](#).

Procedure

1. Create an XML catalog pointing to a local copy of your custom DTD. This file must be named `catalog.xml` and must be found in your plug-in directory.

File `sample_plugin/catalog.xml`:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
        prefer="public">

  <public publicId="-//OASIS//DTD DITA Concept//EN"
          uri="dtd/concept.dtd"/>

  <public publicId="-//OASIS//DTD DITA Composite//EN"
          uri="dtd/database.dtd"/>

  <public publicId="-//OASIS//DTD DITA General Task//EN"
          uri="dtd/generalTask.dtd"/>

  ...

</catalog>
```

2. Create a customization of `ditac_install_dir/xsl/xhtml/xhtml.xsl` as explained in [Chapter 7, Section 2](#). This file must be found in `your_plugin_dir/xsl/xhtml/xhtml.xsl` in order to be used by ditac.

File `sample_plugin/xsl/xhtml/xhtml.xsl`:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml"
                version="2.0">

  <xsl:import href="ditac-xsl/xhtml/xhtml.xsl"/>
```

```

<xsl:template match="*[contains(@class,' tech-d/tag ')]">
  <tt>
    <xsl:call-template name="commonAttributes"/>

    <xsl:choose>
      <xsl:when test="@kind = 'attvalue'">
        <xsl:text>&quot;;</xsl:text>
      </xsl:when>
      .
      .
      .
      <xsl:when test="@kind = 'starttag'">
        <xsl:text>&lt;;</xsl:text>
      </xsl:when>
    </xsl:choose>

    <xsl:apply-templates/>

    <xsl:choose>
      <xsl:when test="@kind = 'attvalue'">
        <xsl:text>&quot;;</xsl:text>
      </xsl:when>
      .
      .
      .
      <xsl:when test="@kind = 'starttag'">
        <xsl:text>&gt;;</xsl:text>
      </xsl:when>
    </xsl:choose>
  </tt>
</xsl:template>
</xsl:stylesheet>

```

Note that the XSLT template called `commonAttributes` adds a `class="tag"` attribute to the generated `tt` element. This makes it easy styling the generated `tt` element using the technique explained in [Chapter 7, Section 1](#).

3. Create a customization of `ditac_install_dir/xsl/fo/fo.xsl` as explained in [Chapter 7, Section 2](#). This file must be found in `your_plugin_dir/xsl/fo/fo.xsl` in order to be used by `ditac`.

File `sample_plugin/xsl/fo/fo.xsl`:

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  version="2.0">
  <xsl:import href="ditac-xsl:fo/fo.xsl"/>

  <xsl:attribute-set name="tag" use-attribute-sets="monospace-style">
  </xsl:attribute-set>

  <xsl:template match="*[contains(@class,' tech-d/tag ')]">
    <fo:inline xsl:use-attribute-sets="tag">
      <xsl:call-template name="commonAttributes"/>

      <xsl:choose>
        <xsl:when test="@kind = 'attvalue'">
          <xsl:text>&quot;;</xsl:text>
        </xsl:when>
        .
        .
        .
        <xsl:when test="@kind = 'starttag'">

```



```

        <xsl:text>&lt;/xsl:text>
      </xsl:when>
    </xsl:choose>

    <xsl:apply-templates/>

    <xsl:choose>
      <xsl:when test="@kind = 'attvalue'">
        <xsl:text>&quot;/xsl:text>
      </xsl:when>
      .
      .
      .
      <xsl:when test="@kind = 'starttag'">
        <xsl:text>&gt;/xsl:text>
      </xsl:when>
    </xsl:choose>
  </fo:inline>
</xsl:template>
</xsl:stylesheet>

```

4. Pass command-line option **-plugin *plugin_name*** to **ditac** in order to use the DTDs/schemas and the XSLT stylesheets found in your plug-in subdirectory, preferably to those found in *ditac_install_dir/schema/* and in *ditac_install_dir/xsl/*.

You'll find a sample DITA document making use of the custom `tag` element in [sample_plugin/sample/sample.ditamap](#). You can convert this sample document to single-page XHTML and to PDF by running `sample_plugin/sample/run.sh` (`sample_plugin\sample\run.bat` on Windows):

```

../../../../bin/ditac -plugin sample_plugin \
  out/sample.pdf sample.ditamap

```


Chapter 9. Extensive customization

In order to extensively customize the output of ditac, you need to learn how it works.

Basically, this means that you'll have to understand the contents of the `ditac_lists.ditac_list` file and the `.ditac` files, which are generated by the *ditac preprocessor*.

An extensive customization works exactly like a simple one:

1. Create a custom XSLT 2.0 stylesheet which imports the stock one.
2. Redefine one or more attribute sets and/or one or more templates in the custom XSLT 2.0 stylesheet.

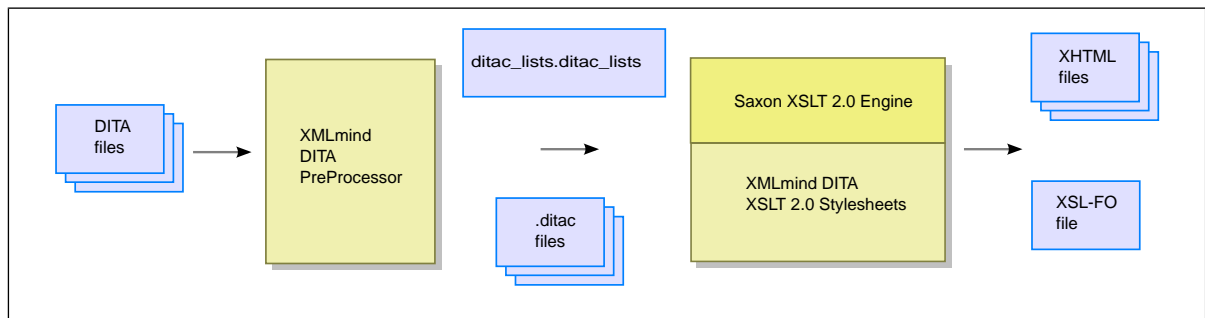
The only difference is that this time, you need to know exactly what is the format of the files you are going to transform. The bad news first: the ditac XSLT 2.0 stylesheets do not transform plain DITA files. They transform `.ditac` files, which are *fully preprocessed DITA files*. Now, the good news: `.ditac` files mainly contains DITA elements and because the ditac preprocessor performs all the grunt work beforehand, `.ditac` files are really straightforward to transform.

In fact, transforming `.ditac` files rather than plain DITA files allows to concentrate on creating great-looking output.

How it works

The ditac preprocessor generates a single `ditac_lists.ditac_list` file and one or more `.ditac` files⁽⁷⁾ out of the source DITA files.

Figure 9-1. The intermediate files generated by the ditac preprocessor



Then, each `.ditac` file, which mainly contains fully preprocessed DITA topics, is transformed in turn by the ditac XSLT 2.0 stylesheets.

The `ditac_lists.ditac_list` file, which contains useful information about the overall DITA document being converted, is not directly transformed by the ditac XSLT 2.0 stylesheets. Instead, when needed to, the ditac XSLT 2.0 stylesheets *query* the `ditac_lists.ditac_list` file in order to generate optional items. Example: number topics, tables, figures, etc, when parameter `number='XXX'` has been specified.

It is possible to examine the contents of the `ditac_lists.ditac_list` file and those of the `.ditac` files by specifying the `-preprocess` command-line option. Example:

```
$ ditac -preprocess \
-v -chunk single \
-images img -p xsl-resources-directory res \
out/manual.html manual.ditamap
```

⁽⁷⁾ A single `.ditac` file for a print output; one or more `.ditac` files for a screen output.

Contents of a .ditac file

The root element of a .ditac file is `ditac:chunk`. A `ditac:chunk` element may have the following child elements (in any order and in any number):

`ditac:titlePage`

This empty placeholder element means: generate a "title page" section here.

`ditac:toc`

This empty placeholder element means: generate a **Table of Contents** section here.

`ditac:figureList`

This empty placeholder element means: generate a **List of Figures** section here.

`ditac:tableList`

This empty placeholder element means: generate a **List of Tables** section here.

`ditac:exampleList`

This empty placeholder element means: generate a **List of Examples** section here.

`ditac:indexList`

This empty placeholder element means: generate an **Index** section here.

A DITA topic of any kind

A fully preprocessed topic. This topic is guaranteed not to contain nested topics.

`ditac:flags`

A wrapper element for a DITA topic of any kind. A `ditac:flags` element is used to wrap any DITA element which has been flagged by the means of a conditional processing profile (a .ditaval file). See [the `-filter` command-line option](#).

More formally, the content model of `ditac:chunk` is specified by the [schema/ditac.rnc](#) RELAX NG grammar.

Example:

```
<ditac:chunk xmlns:ditac="http://www.xmlmind.com/ditac/schema/ditac"
             xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/">
  <ditac:titlePage/>
  <ditac:toc/>
  <topic class="- topic/topic "
        domains="(topic ui-d) (topic hi-d) (topic pr-d) (topic sw-d)
                 (topic ut-d) (topic indexing-d)"
        id="introduction" ditaarch:DITAArchVersion="1.1">
    <title class="- topic/title ">Introduction</title>
    .
    .
    .
  </topic>
  <topic class="- topic/topic " id="I_2yl4p_">
    <title class="- topic/title ">Using XMLmind DITA Converter</title>
  </topic>
  <task class="- topic/topic task/task "
        domains="(topic ui-d) (topic hi-d) (topic pr-d) (topic sw-d)
                 (topic ut-d) (topic indexing-d)"
        id="install" ditaarch:DITAArchVersion="1.1">
    <title class="- topic/title ">Installing XMLmind DITA Converter</title>
    .
    .
    .
  </task>
  .
  .
  .
</ditac:chunk>
```

```
<ditac:indexList/>
</ditac:chunk>
```



Important

The DITA topics contained in a .ditac file are fully preprocessed. What does this mean? Basically that they are ready to be transformed without further efforts:

- Conref inclusions have been processed.
- Unspecified attributes having default values have been added to the elements. Example: a <p> element becomes <p class="- topic/p ">.
- Elements now have a ``flat'', globally unique, ID. Example: the id attribute of this title element <topic id="introduction"><title id="start"> becomes id="introduction__start".
- The href attribute of xref, link, image elements now point to the (future) output files. Example: the href attribute of this xref element <xref href="intro.dita#introduction/start"> becomes href="userguide-1.html#introduction__start".
- Some text may have been added to empty xref and link elements.
- The reltable elements of the DITA map have been converted to related-links sections or to extra link elements.
- Filtered elements have been removed. Flagged elements have been wrapped in a ditac:flags element.

Contents of the ditac_lists.ditac_list file

The root element of the ditac_lists.ditac_list file is ditac:lists. A ditac:lists element may have the following child elements (in this exact order and in this exact number):

ditac:chunkList

A ditac:chunkList contains a ditac:chunk element for each .ditac file. A ditac:chunk element may be seen as the *manifest* of a .ditac file.

Example:

```
<ditac:chunkList>
  <ditac:chunk file="manual.html">
    <ditac:titlePage/>
    <ditac:toc/>
    <ditac:topic id="introduction" number="bookabstract.1"
      role="bookabstract" title="Introduction"/>
    .
    .
    .
    <ditac:topic id="limitations" number="appendix.1" role="appendix"
      title="Limitations and implementation specificities"/>
    <ditac:indexList/>
  </ditac:chunk>
</ditac:chunkList>
```

ditac:titlePage

Contains all the DITA elements needed to generate the ``title page'' section of a document. This element exists but is empty if the DITA document being converted has no title and no metadata (e.g. topicmeta/autor, bookmeta/publisherinformation, etc).

Example:

```
<ditac:titlePage>
  <title class="- topic/title ">XMLmind DITA Converter Manual</title>
  <bookmeta class="- map/topicmeta bookmap/bookmeta ">
```

```

    <authorinformation class="+ topic/author xnal-d/authorinformation ">
    .
    .
    .
    <critdates class="- topic/critdates ">
      <created class="- topic/created " date="September 17, 2009"/>
    </critdates>
  </bookmeta>
</ditac:titlePage>

```

ditac:toc

Contains all the information needed to generate **Table of Contents** section of a document.

Example:

```

<ditac:toc>
  <ditac:tocEntry file="manual.html" id="I_2yl4p_" number="part.1"
    role="part" title="Using XMLmind DITA Converter">
    <ditac:tocEntry file="manual.html" id="install"
      number="part.1 chapter.1" role="chapter"
      title="Installing XMLmind DITA Converter">
      .
      .
      .
    </ditac:tocEntry>
  </ditac:tocEntry>
  <ditac:tocEntry file="manual.html" id="limitations" number="appendix.1"
    role="appendix"
    title="Limitations and implementation specificities"/>
</ditac:toc>

```

ditac:figureList

Contains all the information needed to generate the **List of Figures** section of a document. This element exists but is empty if the DITA document being converted contains no `fig` elements having a `title` child element.

Example:

```

<ditac:figureList>
  <ditac:figure file="manual.html" id="xsltParams__page_areas"
    number="part.1 chapter.4 figure.1" title="Page areas"/>
  <ditac:figure file="manual.html" id="howItWorks__I_6gb2s_"
    number="part.2 chapter.3 figure.1"
    title="The intermediate files generated by
      the ditac preprocessor"/>
</ditac:figureList>

```

ditac:tableList

Contains all the information needed to generate the **List of Tables** section of a document. This element exists but is empty if the DITA document being converted contains no `table` elements having a `title` child element.

Example:

```

<ditac:tableList>
  <ditac:table file="manual.html"
    id="quickStart__supported_filename_extensions"
    number="part.1 chapter.2 table.1"
    title="Supported filename extensions"/>
  <ditac:table file="manual.html"
    id="quickStart__supported_output_formats"
    number="part.1 chapter.2 table.2"
    title="Supported output formats"/>
</ditac:tableList>

```

ditac:exampleList

Contains all the information needed to generate the **List of Examples** section of a document. This element exists but is empty if the DITA document being converted contains no `example` elements having a `title` child element.

Example:

```
<ditac:exampleList>
  <ditac:example file="topicAAA.htm" id="topicAAA__I_6wcr3_"
    number="part.1 example.1" title="Example AAA.1"/>
  .
  .
  .
  <ditac:example file="topicHHH.htm" id="topicHHH__exampleHHH.2"
    number="appendix.2 example.2" title="Example HHH.2"/>
</ditac:exampleList>
```

ditac:indexList

Contains all the information needed to generate the **Index** section of a document. This element exists but is empty if the DITA document being converted contains no `indexterm` elements.

Example:

```
<ditac:indexList>
  <ditac:div title="A">
    <ditac:indexEntry term="appendix-number-format, parameter"
      xml:id="I_hdlwr_">
      <ditac:indexAnchor file="manual.html"
        id="xsltParams__I_8bona_"
        number="1"/>
    </ditac:indexEntry>
    <ditac:indexEntry sortAs="automap" term="-automap, option"
      xml:id="I_2gud9_">
      <ditac:indexAnchor file="manual.html"
        id="commandLine__I_5x8va_"
        number="1"/>
    </ditac:indexEntry>
  </ditac:div>
  .
  .
  .
  <ditac:div title="X">
    .
    .
    .
    <ditac:indexEntry sortAs="xslt" term="-xslt, option"
      xml:id="I_atn9k_">
      <ditac:indexAnchor file="manual.html"
        id="commandLine__I_cu3ew_"
        number="1"/>
      <ditac:indexAnchor file="manual.html"
        id="customAttributeSet__I_gis5b_" number="2"/>
      <ditac:indexAnchor file="manual.html"
        id="specialize__I_11514_"
        number="3"/>
      <ditac:indexSeeAlso ref="I_bhy05_" term="-t, option"/>
      <ditac:indexSeeAlso ref="I_fljh_" term="-xslt, option"/>
    </ditac:indexEntry>
  </ditac:div>
</ditac:indexList>
```

More formally, the content model of `ditac:lists` is specified by the `schema/ditac_lists.rnc` RELAX NG grammar.

Currently the `ditac_lists.ditac_list` file is used to generate:

- the ``title page" section of a document;
- the **Table of Contents** section of a document;
- the **List of Figures**, **List of Tables**, **List of Examples** sections of a document;
- the **Index** section of a document;
- the navigation icons in a multi-page HTML document;
- all the files (`project.hhp`, `toc.hhc`, etc) required by the HTML Help system;
- all the files (`jhelpset.hs`, `jhelpmap.jhm`, etc) required by the Java™ Help system.

Part III. Embedding XMLmind DITA Converter in a Java™ application

Chapter 10. High-level method: embedding

`com.xmlmind.ditac.convert.Converter`

Quick and easy embedding: embed `com.xmlmind.ditac.convert.Converter`, the Java™ class which is used to implement the **ditac** command-line utility.

Converter is the object which is at the core of the **ditac** command-line utility. Its `run` method accepts the same string arguments as the **ditac** command-line utility.

The full source code of the `Embed1` sample is found in `Embed1.java`.

1. Create the Converter.

```
StyleSheetCache cache = new StyleSheetCache();

Console console = new Console() {
    public void showMessage(String message, MessageType messageType) {
        System.err.println(message);
    }
};

Converter converter = new Converter(cache, console);
```

- **StyleSheetCache** is a simple cache for the ditac XSLT 2.0 stylesheets. It is a thread-safe object which is intended to be shared by several Converters.

Unlike **StyleSheetCache**, **Converter** is not thread-safe. Each thread must own its **Converter**. However, the `run` method of a **Converter** may be invoked several times.

- **Console** is a very simple interface. Implementing this interface allows to do whatever you want with the messages reported by a **Converter**.

2. Configure the Converter.

```
if (!converter.registerFOP(path("/opt/fop/fop"))) {
    return 1;
}

File xslDir = new File(path("../..../xsl"));
System.setProperty("DITAC_XSL_DIR", xslDir.getAbsolutePath());
```

- There are several methods which allows to register an XSL-FO processor with a **Converter**. From high-level ones to low-level ones, these methods are: `registerFOP`, `registerXEP`, `registerAHF`, `registerXFC`, `registerExternalFOConverter`, `registerFOConverter`.
- A **Converter** can automatically determine where to find the directory containing all the ditac XSLT 2.0 stylesheets. It assumes that `ditac.jar` is in the class path of Java™. It also assumes that `ditac.jar` is contained in a `lib/` directory. The `xsl/` directory is assumed to be a sibling of the `lib/` directory.

However, the situation may be very different in applications other than the **ditac** command-line. That's why you may need to set the `DITAC_XSL_DIR` system property to the absolute path of the `xsl/` directory containing the XSLT stylesheets.

3. Invoke the run method.

```
String[] args = {
    "-v",
    "-p", "number", "all",
    outFile.getPath(),
    inFile.getPath(),
};
```

```
return converter.run(args);
```

The `run` method returns 0 if the conversion is successful and an integer greater than 0 otherwise. When the conversion fails, error messages are displayed on the Console.

Compiling and executing the `Embed1` sample

Compile the `Embed1` sample by running **ant** in `ditac_install_dir/doc/manual/embed/`.

Execute the `Embed1` sample by running **ant** `embed1` in `ditac_install_dir/doc/manual/embed/`. This will convert `ditac_install_dir/docsrc/manual/manual.ditamap` to `ditac_install_dir/doc/manual/embed/manual.pdf`, using Apache FOP.

Note that `Embed1.java` contains "hardwired filenames". This means that this sample cannot be run from elsewhere than `ditac_install_dir/doc/manual/embed/` and that you'll almost certainly need to modify the source code in order to specify the actual location of the `fop` (`fop.bat`) script.

Related information

- [Chapter 11. Low-level method: embedding `com.xmlmind.ditac.preprocess.PreProcessor`](#)

Chapter 11. Low-level method: embedding

`com.xmlmind.ditac.preprocess.PreProcessor`

Advanced embedding method: first invoke a preprocessor which will generate intermediate `.ditac` files, then invoke the XSLT 2.0 engine in order to transform all these `.ditac` files.

This method consists in first invoking the `PreProcessor` in order to pre-process the DITA source files into a `ditac_lists.ditac_lists` file and one or more `.ditac` files; then invoking the `Saxon` XSLT 2.0 engine in order to transform all the `.ditac` files.

For some output formats, PDF, RTF, etc, the final third step consists in invoking an XSL-FO processor such as `Apache FOP` in order to convert the XSL-FO generated by the XSLT stylesheets to the desired output format.

The full source code of the `Embed2` sample is found in `Embed2.java`.

1. Invoke the `ditac PreProcessor` to pre-process the DITA source files into a `ditac_lists.ditac_lists` file and one or more `.ditac` files.

1. Create and configure the `PreProcessor`.

```
Console console = new Console() {
    public void showMessage(String message, MessageType messageType) {
        System.err.println(message);
    }
};

PreProcessor preProc = new PreProcessor(console);
preProc.setChunking(Chunking.SINGLE);
preProc.setMedia(Media.SCREEN);

ResourceCopier resourceCopier = new ResourceCopier();
resourceCopier.parseParameters("img");
preProc.setResourceHandler(resourceCopier);
```

- `Console` is a very simple interface. Implementing this interface allows to do whatever you want with the messages reported by a `PreProcessor`.
- Specifying `preProc.setChunking(Chunking.SINGLE)` allows to generate a single HTML page using a DITA map designed to generate multiple HTML pages.
- A `PreProcessor` is not concerned about the *exact* output format. However its behaves differently depending on the target `Media`.
- A `PreProcessor` handles to an `ResourceHandler` all the resource files, typically image files, referenced in the DITA source using relative URLs. An `ResourceHandler` is registered with a `PreProcessor` using method `setResourceHandler`.

In the case of the `Embed2` sample, we use the simplest possible `ResourceHandler` which is `ResourceCopier`.

2. Pre-process the DITA source files.

```
URL inFileURL = null;
try {
    inFileURL = inFile.toURI().toURL();
} catch (MalformedURLException cannotHappen) {}

File[] preProcFiles = null;
try {
    preProcFiles = preProc.process(new URL[] { inFileURL }, outFile);
} catch (IOException e) {
    console.showMessage(e.toString(), Console.MessageType.ERROR);
}
```

```

}
if (preProcFiles == null) {
    return false;
}

```

The `process` method of a `PreProcessor` returns null if an error other than an `IOException` has caused the pre-processing to fail. When this is the case, errors messages are displayed on the Console.

Note that a `PreProcessor` is not thread-safe. Each thread must own its `PreProcessor`. However, the `process` method of a `PreProcessor` may be invoked several times.

2. Invoke the Saxon XSLT 2.0 engine, in order to transform all the `.ditac` files. Note that this is done using the standard JAXP API.

1. Pass *required system parameters* to the XSLT stylesheets, in addition to the normal, user, parameters.

```

String ditacListsURI = "";

int count = preProcFiles.length;
for (int i = 0; i < count; ++i) {
    File ditacFile = preProcFiles[i];

    if (ditacFile.getPath().endsWith(".ditac_lists")) {
        ditacListsURI = ditacFile.toURI().toASCIIString();
        break;
    }
}

String[] params = {
    "ditacListsURI", ditacListsURI,
    "xsl-resources-directory", "res",
    "use-note-icon", "yes",
    "default-table-width", "100%"
};

```

These required system parameters are:

- `ditacListsURI`, always required.
- `foProcessor`, required by the XSLT stylesheets that generate XSL-FO.
- `chmBasename`, `hhpBasename`, required by the XSLT stylesheets that generate HTML Help.

2. Use the Saxon XSLT 2.0 engine to create a `TransformerFactory`, then configure this `TransformerFactory`.

```

private static
TransformerFactory createTransformerFactory(URIResolver uriResolver,
                                           ErrorListener errorListener)
    throws Exception {
    Class<?> cls = Class.forName("net.sf.saxon.TransformerFactoryImpl");
    TransformerFactory transformerFactory =
        (TransformerFactory) cls.newInstance();

    ExtensionFunctions.registerAll(transformerFactory);

    transformerFactory.setURIResolver(uriResolver);
    transformerFactory.setErrorListener(errorListener);

    return transformerFactory;
}

```

- Creating an instance of Saxon 9.2+ is absolutely needed. XMLmind DITA Converter is not designed to work with any other XSLT engine (e.g. the Xalan XSLT 1.0 engine, which is part of the Java™ runtime).
- The ditac XSLT 2.0 stylesheets make use of a few XSLT extension functions written in Java™. These extension functions must be registered with Saxon. This is done using [ExtensionFunctions.registerAll](#).

3. Create and configure a Transformer.

```
private static Transformer createTransformer(String[] params,
                                           Console console)
    throws Exception {
    URIResolver uriResolver = Resolve.createURIResolver();
    ErrorListener errorListener = new ConsoleErrorListener(console);

    TransformerFactory factory = createTransformerFactory(uriResolver,
                                                         errorListener);

    File xslFile = new File(path("../.../xsl/xhtml/html.xsl"));
    Transformer transformer =
        factory.newTransformer(new StreamSource(xslFile));

    transformer.setURIResolver(uriResolver);
    transformer.setErrorListener(errorListener);

    for (int i = 0; i < params.length; i += 2) {
        transformer.setParameter(params[i], params[i+1]);
    }

    return transformer;
}
```

- [Resolve](#) is a helper class making it easy to use the services of [XML Catalog resolvers](#).

By default, [Resolve](#) automatically loads all the XML catalogs specified using the `xml.catalog.files` Java™ system property. Excerpts of the `ant build.xml` file:

```
<target name="embed2" depends="compile,clean_embed2">
  <javac classpathref="cp" fork="yes" classname="Embed2">
    <sysproperty key="xml.catalog.files"
      value="${ditac.dir}/schema/catalog.xml" />
    <arg value="${ditac.dir}/docsrc/manual/manual.ditamap" />
    <arg value="manual.html" />
  </javac>
</target>
```

However, static method [setResolverFactory](#) allows to configure this thread-safe utility class (used by ditac in many places) differently.

- [ConsoleErrorListener](#) is an implementation of [ErrorListener](#) which displays its messages on a Console.

4. Invoke the Transformer to transform each .ditac file.

```
for (int i = 0; i < count; ++i) {
    File ditacFile = preProcFiles[i];

    String ditacFilePath = ditacFile.getPath();
    if (ditacFilePath.endsWith(".ditac")) {
        File transformedFile = new File(
            ditacFilePath.substring(0, ditacFilePath.length()-5) +
            ".html");
    }
}
```

```

        try {
            transformer.transform(new StreamSource(ditacFile),
                                new StreamResult(transformedFile));
        } catch (Exception e) {
            console.showMessage(e.toString(),
                               Console.MessageType.ERROR);
            cleanUp(preProcFiles);
            return false;
        }
    }
}

```

In the case of `Embed2`, the above loop is not strictly needed. We specified `preProc.setChunking(Chunking.SINGLE)` and therefore the `PreProcessor` generates a single `.ditac` file.

3. Copy the resources of the XSLT stylesheets (CSS stylesheets, icons, etc) to output subdirectory `res/`. Note that the images referenced in the DITA source, if any, have already been copied to output subdirectory `img/` by the `ImageCopier`.

```

File dstDir = new File("res");
if (!dstDir.exists()) {
    File srcDir = new File(path("../../xsl/xhtml/resources"));
    try {
        FileUtil.copyDir(srcDir, dstDir, false);
    } catch (IOException e) {
        console.showMessage(e.toString(), Console.MessageType.ERROR);
        cleanUp(preProcFiles);
        return false;
    }
}

```

4. Delete the `ditac_lists.ditac_lists` and `.ditac` files.

```
cleanUp(preProcFiles);
```

Compiling and executing the `Embed2` sample

Compile the `Embed2` sample by running **ant** in `ditac_install_dir/doc/manual/embed/`.

Execute the `Embed2` sample by running **ant** `embed2` in `ditac_install_dir/doc/manual/embed/`. This will convert `ditac_install_dir/docsrc/manual/manual.ditamap` to single HTML 4.01 page `ditac_install_dir/doc/manual/embed/manual.html`.

Note that `Embed2.java` contains ``hardwired filenames". This means that this sample cannot be run from elsewhere than `ditac_install_dir/doc/manual/embed/`.

Related information

- [Part II, Chapter 9. Extensive customization](#)
- [Chapter 10. High-level method: embedding `com.xmlmind.ditac.convert.Converter`](#)

Appendix A. Translating the messages generated by ditac

About this task

The messages generated by ditac (**Table of Contents**, **List of Figures**, **Chapter**, **Appendix**, etc) are available in English (*en*), French (*fr*), German (*de*), Spanish (*es*), etc. Now let's suppose that you are routinely authoring Portuguese documents and that you want ditac to also support this language.

Procedure

1. Go to the `ditac_install_dir/xsl/common/messages/` directory.

```
~$ cd /opt/ditac/xsl/common/messages/
```

2. Copy `en.xml` to `pt.xml`.

Note that "pt" is the [ISO 639-1](#) two-letter code of the Portuguese language.

Country variants of a language are supported too. Example: `pt-BR` (Brazilian Portuguese). However, when this is the case, make sure that the name of the file containing your messages use lower-case characters. Example: `pt-br.xml` should be fine, while `pt-BR.xml` or `pt_br.xml` would not work.

```
/opt/ditac/xsl/common/messages$ cp en.xml pt.xml
```

3. Open `pt.xml` in a text or XML editor and translate to Portuguese all the messages found in this file.

```
<?xml version="1.0" encoding="UTF-8"?>
<messages xml:lang="pt">
  <!-- Task sections -->
  <message name="prereq">Pré-requisito</message>
  ...
```

4. Open `ditac_install_dir/xsl/common/commonUtil.xsl` in a text or XML editor and add string 'pt' at the end of the `messageFileNames` list:

```
<!-- localize ===== -->

<xsl:param name="messageFileNames" select="'en', 'fr', 'de', 'es', 'pt'"/>
```

5. When done, please send us (ditac-support@xmlmind.com) your translation (e.g. `pt.xml`) so we can add to the distribution of XMLmind DITA Converter.

Appendix B. Limitations and implementation specificities

Conversion to XHTML and XSL-FO

- XMLmind DITA Converter (ditac) has been tested only against "Technical content elements". Other vocabularies, "Learning and training elements", "Classification elements", "Task requirements domain", etc, are not yet officially supported.
- The `syntaxdiagram` element and all its descendant elements are not processed at all. (We plan to automatically convert this element to SVG graphics.)
- The `titlealts/navtitle` element of topic is ignored.
- It is not possible to flag (i.e. using a `ditaval` specification) *all* elements.
 - It is possible to flag the following inline elements (and of course, as always, all their specializations): `ph`, `term`, `xref`, `cite`, `q`, `boolean`, `state`, `keyword`, `tm`, `image`, `foreign`.
 - It is possible to flag the following block elements: `topic`, `p`, `lq`, `note`, `dl`, `ul`, `ol`, `sl`, `pre`, `lines`, `fig`, `object`, `table`, `simpletable`, `section`, `example`.
- Layout of `simpletable` elements:
 - Attribute `frame` is ignored.
 - Conversion to XHTML:
 - Attribute `expanses` is partially supported. Its value is considered to always be 100%.
 - Conversion to XSL-FO:
 - Attribute `expanses` is ignored. The width of a `simpletable` is always 100% and thus, you cannot center a `simpletable` using the `center` parameter.
- Layout of (CALS) `table` element:
 - Conversion to XHTML:
 - Attribute `pgwide` is partially supported. Its value is considered to always be 100%.
 - Something like `colwidth="2*+3pt"` is treated as if it were `colwidth="2*"`. Moreover, because no Web browser seems to support relative lengths, a relative length is approximated to a percentage.
 - Conversion to XSL-FO:
 - Attribute `pgwide` is ignored. The width of a `table` is always 100% and thus, you cannot center a `table` using the `center` parameter.
- The qualified ID of a descendant element of a topic is transformed as follows: `topicID/descendantID` becomes `topicID__descendantID` in the generated content. (The separator string being used comprises *two* underscore characters.)

Example: let's suppose a topic having "parameters" as its `id` attribute, containing a table having "default_values" as its `id` attribute, has been converted to HTML. The generated HTML file which contains the topic is called `userguide.html`.

 - URL `"userguide.html#parameters"` allows to address the topic.
 - URL `"userguide.html#parameters__default_values"` allows to address the table.

Booklists

- Contents corresponding to the following empty bookmap elements: `toc`, `tablelist`, `figurelist`, `indexlist` can be automatically generated by ditac.
- Contents corresponding to the following empty bookmap elements: `trademarklist`, `abbrevlist`, `bibliolist`, `glossarylist` *cannot* be automatically generated by ditac.
- About the automatically generated `indexlist`:
 - Specifying an `indexterm` element in the `topicmeta/keywords` element of a `topicref` element is equivalent to specifying it in the `prolog/metadata/keywords` element of the corresponding topic. Any other `indexterm` element found in a map is ignored.
 - In a topic, the implicit end of an index range is always after the last child of the topic, not including nested topics.
 - Overlapping index ranges are not supported.
 - The markup possibly contained in an `indexterm` (`option`, `parmname`, `apiname`, etc) is ignored.
 - Because we consider this feature to be truly useful, we'll generate page references and "see also" redirections even for non-leaf index terms. No warnings will be reported in this case. If you don't like this specificity, simply do not author such `indexterm` elements.
 - Unless specified using the `-lang` command-line option, the language of the document is taken from the `xml:lang` attribute of the root element of the topic map. If there is no such attribute, the language defaults to "en". Knowing the language of the document is required to be able to generate localized text (e.g. "Kapitel") and to sort and group the index entries.

Keyref processing

- The effective value of a `topicref` element having a `keyref` attribute is almost certainly not conforming as we have not really understood the spec.
- Matching element content taken from a key definition is limited to the following cases:
 - A `link` element gets its `linktext` child from `key_definition/topicmeta/linktext` and its `desc` child from `key_definition/topicmeta/shortdesc`.
 - An `xref` element gets its contents from `key_definition/topicmeta/linktext`.
 - Elements `ph`, `cite`, `keyword`, `dt` and `term` all get their content from `key_definition/topicmeta/keywords/keyword`, if any. Otherwise the contents of `key_definition/topicmeta/linktext` is used as a fallback.

Transclusion

- During a `conref` transclusion, ditac does not check the compatibility of the domains of the referencing document with the domains of the referenced document. This can be changed by defining system property `DITAC_CHECK_DOMAINS` (that is, adding `-DDITAC_CHECK_DOMAINS=1` to the `bin/ditac` shell script or to `bin/ditac.bat`). However, the verifications performed by ditac are almost certainly not conforming as we have not really understood the spec.
- Transclusion does not implement automatic generalization. For example, transcluding `<li conref="foo.dita#foo/item3"/>` will report a fatal error if "foo/item3" is a `step` element.

A `step` element is a specialization of a `li` element. Some DITA processors are capable of automatically converting a `step` element to an `li` element. This is not the case of ditac.

Cascading of attributes and metadata

- Filtering and flagging may be performed using any attribute. However only the following attributes: `audience`, `platform`, `product`, `otherprops`, `props`, specializations of attributes `props` and `rev` properly cascade

with a map, within the `related-links` element of a topic and from a `topicref` element to the referenced `topic` element.

- Both attribute (e.g. `audience`) and element (e.g. `audience`) metadata are copied from a `topicref` element to its referenced `topic` element.

By default `topicref/topicmeta/@lockmeta=yes`.

If `topicref/topicmeta/@lockmeta=no`, then conflicts (non-additive metadata attributes or elements found in both the `topicref/topicmeta` element and the `topic/prolog` element) are resolved in favor of the topic.

`Topicref/topicmeta/searchtitle` and `topic/titlealts/searchtitle` conflicts are also resolved using `topicref/topicmeta/@lockmeta`.

- In the following case, `<topicref href="foo.dita"/>`, the `topicref` metadata is copied only to the first topic found in `foo.dita`. An alternative would be to copy metadata to all topics found in `foo.dita`.

Conditional processing

- `double-underline` is processed as if it were `underline`.
- `line-through` is supported in addition to `underline` and `overline`.
- The value of the `changebar` attribute is ignored. That is, a `changebar` cannot be styled.
- Only the following elements (and, of course, their specializations) can be flagged: `topic`, `p`, `lq`, `note`, `dl`, `ul`, `ol`, `sl`, `pre`, `lines`, `fig`, `object`, `table`, `simpletable`, `section`, `example`, `ph`, `term`, `xref`, `cite`, `q`, `boolean`, `state`, `keyword`, `tm`, `image`, `foreign`.
- Conditional processing is also applied to the information (e.g. `title`, `metadata`) contained in a map. However, only the `exclude` action will work. The `flag` action does not work in this context.
- Any attribute (that is, not only `audience`, `platform`, `product`, `rev`, `otherprops` and attributes specialized from `props`) may be used to filter or flag an element.
- Attribute `otherprops` is assumed to contain simple, non-structured, strings.

Generating links

- Attribute `collection-type`, whatever its value, is ignored inside the `reltable` element.
- Dita cannot generate "smart labels" for related links. The label is always "Related Links". It could have been "Related Concepts", "Related Reference" or even something determined using what is specified in the `title` child element of a `relcolspec` element.

Chunking

- The "to-navigation" chunk value is ignored.
- When the `copy-to` attribute is used to specify an URI, the parent path part (e.g. "foo" in "foo/bar.htm") and the extension part (e.g. ".htm" in "foo/bar.htm") are ignored. Only the "root name" (e.g. "bar" in "foo/bar.htm") is taken into account during the processing of the map.
- The default chunking policy is `by-document`.
- When the deliverable targets a print media, all chunk specifications are removed and a `chunk="to-content"` attribute is added to the root element of the map.

Other limitations and specificities

- There are several limitations and inconsistencies when working with files containing multiple topics and/or nested topics.

For example, let's suppose a map contains the following `topicrefs`, where `multi.dita` contains multiple topics (first topic being `t1`, second topic being `t2`), each topic possibly containing nested topics.

```
<topicref href="multi.dita"/>
<topicref href="multi.dita"/>
<topicref href="multi.dita#t1"/>
<topicref href="multi.dita#t2"/>
```

- As expected, the first `topicref` pulls all the topics, including nested ones, contained in `multi.dita`. However parent, child, sibling, etc, related links will *not* be automatically generated for these topics.
 - The second `topicref` pulls a copy of all the topics, including nested ones, contained in `multi.dita`. The third `topicref` pulls a copy of topic `t1` (excluding nested topics). The fourth `topicref` is *not* detected as pulling a copy of topic `t2`. Therefore the fourth `topicref` does nothing at all, as topic `t2` has already being pulled into the deliverable (by the first `topicref`).
- The following `topicref` elements are not treated differently from the others: `topicset`, `topicsetref`.
 - The following bookmap elements: `abbrevlist`, `amendments`, `appendices`, `appendix`, `bibliolist`, `bookabstract`, `booklist`, `chapter`, `colophon`, `dedication`, `draftintro`, `figurelist`, `glossarylist`, `indexlist`, `notices`, `part`, `preface`, `tablelist`, `toc`, `trademarklist`, are considered to have an *implicit title* when
 - they have no `href` attribute,
 - and they have no explicit title,
 - and they contain one or more `topicref` (of any type) child elements.

For example:

```
<glossarylist>
  <topicref href="term1.dita"/>
  <topicref href="term2.dita"/>
  <topicref href="term3.dita"/>
</glossarylist>
```

is processed as if it was:

```
<glossarylist navtitle="Glossary">
  <topicref href="term1.dita"/>
  <topicref href="term2.dita"/>
  <topicref href="term3.dita"/>
</glossarylist>
```

- All attributes and elements — `map/@anchorref`, `anchorref`, `anchor`, `navref` — related to runtime integration of maps are ignored.
- Ditac reports a "*topicB*, href points outside processed topics" warning when *topicA* references *topicB* and *topicB* is not referenced in the map. In order to suppress this warning, add to the map a `topicref` having attribute `toc="no"` and pointing to *topicB*.

Index

A

-addindex, option, 15
 add-toc-root, parameter, 26
 -ahf, option, 15
 AHF, XSL-FO processor, 8, 15
 animation, 45
 Antenna House Formatter. *See* AHF, XSL-FO processor
 Apache FOP. *See* FOP, XSL-FO processor
 appendix-number-format, parameter, 19
 audio, 44
 -automap, option, 16

B

-backmatter, option, 15
 base-font-size, parameter, 31
 body-bottom-margin, parameter, 31
 body-font-family, parameter, 31
 body-top-margin, parameter, 31
 {{break}}, page header/footer variable, 39

C

-c, option, 13
 center, parameter, 19
 chain-pages, parameter, 23
 chain-topics, parameter, 24
 {{chapter-title}}, page header/footer variable, 38
 .chm, filename extension, 8. *See also* HTML Help, output format
 chmBaseline, parameter, 29
 choice-bullets, parameter, 31
 -chunk, option, 13
 cover-image, parameter, 30
 CSS, custom, 51
 css, parameter, 24, 52
 css-name, parameter, 24

D

default-table-width, parameter, 24
 ditac.options, options file, 4, 16
 ditacListsURI, parameter, 19
 DITA specialization, 55
 .doc, filename extension, 8. *See also* RTF, output format
 {{document-date}}, page header/footer variable, 38
 {{document-title}}, page header/footer variable, 38
 .docx, filename extension, 8. *See also* Office Open XML, output format
 docx, output format name. *See* Office Open XML, output format
 -dryrun, option, 16

E

Eclipse Help, output format, 10, 11, 18, 51, 53

eclipsehelp, output format name. *See* Eclipse Help, output format
 .epub, filename extension, 8. *See also* EPUB 2, output format
 epub, output format name. *See* EPUB 2, output format
 epub:trigger, 46
 EPUB 2, output format, 8, 10, 12, 18, 19, 51, 53
 EPUB 3, output format, 10, 12, 18, 19, 30, 31, 43, 44, 46, 51, 53
 epub3, output format name. *See* EPUB 3, output format
 epub-identifier, parameter, 31
 -errout, option, 16
 extended-toc, parameter, 19
 external-href-after, parameter, 31
 external-href-before, parameter, 31
 external-link-icon-height, parameter, 24
 external-link-icon-suffix, parameter, 25
 external-link-icon-width, parameter, 25
 external-resource-base, parameter, 19

F

-f, option, 14
 -filter, option, 14
 Flash, 45
 fo, output format name. *See* XSL-FO, output format
 -foconverter, option, 15
 footer-center, parameter, 32
 footer-center-width, parameter, 32
 footer-height, parameter, 32
 footer-left, parameter, 32
 footer-left-width, parameter, 32
 footer-right, parameter, 32
 footer-right-width, parameter, 32
 footer-separator, parameter, 33
 -fop, option, 15
 FOP, XSL-FO processor, 8, 15
 foProcessor, parameter, 32
 -format, option, 14
 -frontmatter, option, 14

G

Generated messages. *See* Translation
 generate-epub-trigger, parameter, 31
 generator-info, parameter, 25

H

header-center, parameter, 33
 header-center-width, parameter, 33
 header-height, parameter, 33
 header-left, parameter, 33
 header-left-width, parameter, 33
 header-right, parameter, 33
 header-right-width, parameter, 33
 helpset-basename, parameter, 29
 -hhc, option, 16
 hhc-basename, parameter, 29
 hhpBaseline, parameter, 30

hhp-template, parameter, 29
 hhx-basename, parameter, 30
 highlight-source, parameter, 20
 .htm, filename extension, 8. *See also* XHTML 1.0, output format
 .html, filename extension, 8. *See also* XHTML 1.0, output format
 html, output format name. *See* HTML 4.01, output format
 HTML 4.01, output format, 9, 10, 18, 51, 52
 HTML Help, output format, 8, 10, 11, 18, 51, 53
 htmlhelp, output format name. *See* HTML Help, output format
 hyphenate, parameter, 33

I

-i, option, 14
 ignore-navigation-links, parameter, 25
 -ignoreoptionsfile, option, 16
 {{image(URI)}}, page header/footer variable, 39
 -images, option, 14
 -index, option, 14
 index-basename, parameter, 29
 index-column-gap, parameter, 33
 indexer-directory-basename, parameter, 29
 index-range-separator, parameter, 20

J

Java Help, output format, 8, 10, 11, 18, 51, 53
 javahelp, output format name. *See* Java Help, output format
 jhindexer, 29
 -jhindexer, option, 16
 justified, parameter, 34

K

-keepfo, option, 16

L

-lang, option, 15
 link-auto-text, parameter, 20
 link-bullet, parameter, 34
 Localization. *See* Translation

M

map-basename, parameter, 29
 mark-external-links, parameter, 25
 MathML, 43
 menucascade-separator, parameter, 34

N

navigation-icon-height, parameter, 25
 navigation-icon-suffix, parameter, 25
 navigation-icon-width, parameter, 26
 note-icon-height, parameter, 34
 note-icon-list, parameter, 20

note-icon-suffix, parameter, 34
 note-icon-width, parameter, 34
 number, parameter, 9, 20
 number-separator1, parameter, 20
 number-separator2, parameter, 21
 number-toc-entries, parameter, 27

O

-o, option, 15
 .odt, filename extension, 8. *See also* OpenOffice, output format
 odt, output format name. *See* OpenOffice, output format
 Office Open XML, output format, 8, 9, 10, 18, 52
 onclick, processing-instruction, 46
 OpenOffice, output format, 8, 9, 10, 18, 52
 -options, option, 15
 out of memory error, 17
 outputclass, attribute, 20

P

-p, option, 13
 page-bottom-margin, parameter, 34
 {page-count}, page header/footer variable, 38
 page-height, parameter, 34
 page-inner-margin, parameter, 35
 {page-number}, page header/footer variable, 38
 page-orientation, parameter, 35
 page-outer-margin, parameter, 35
 page-ref-after, parameter, 35
 page-ref-before, parameter, 35
 {page-sequence}, page header/footer variable, 39
 page-top-margin, parameter, 35
 page-width, parameter, 35
 paper-type, parameter, 35
 -param, option, 13
 Parameters. *See* XSLT stylesheets parameters
 part-number-format, parameter, 21
 .pdf, filename extension, 8. *See also* Java Help, output format; PDF, output format
 PDF, output format, 7, 8, 10, 17, 52
 pdf, output format name. *See* PDF, output format
 pdf-outline, parameter, 36
 -plugin, option, 16
 plugin-id, parameter, 30
 plugin-index-basename, 30
 plugin-name, parameter, 30
 plugin-provider, parameter, 30
 plugin-toc-basename, parameter, 30
 plugin-version, parameter, 30
 PostScript, output format, 8, 10, 17, 52
 prepend-chapter-to-section-number, parameter, 21
 -preprocess, option, 16, 59
 .ps, filename extension, 8. *See also* PostScript, output format
 ps, output format name. *See* PostScript, output format

R

-r, option, 14
 RenderX XEP. *See* XEP, XSL-FO processor
 -resourcehandler, option, 14
 -resources, option, 14
 .rtf, filename extension, 8. *See also* RTF, output format
 RTF, output format, 8, 10, 17, 52
 rtf, output format name. *See* RTF, output format

S

screen-resolution, parameter, 26
 {{section1-title}}, page header/footer variable, 38
 show-draft-comments, parameter, 21
 show-external-links, parameter, 36
 show-link-page, parameter, 36
 show-xref-page, parameter, 36
 SVG, 43
 .swf, 45
 syntax highlighting, 20, 41

T

-t, option, 13, 54
 text-file-encoding, parameter, 21
 title-after, parameter, 21
 title-color, parameter, 36
 title-font-family, parameter, 36
 title-page, parameter, 21
 title-prefix-separator1, parameter, 22
 -toc, option, 8, 14
 toc-basename, parameter, 29
 {{topic-title}}, page header/footer variable, 38, 39
 Translation, 73
 two-sided, parameter, 36

U

ul-li-bullets, parameter, 36
 unordered-step-bullets, parameter, 37
 use-note-icon, parameter, 22

V

-v, option, 9, 15
 -version, option, 16
 video, 45
 -vv, option, 15
 -vvv, option, 15

W

Web Help, output format, 10, 18, 51, 53
 webhelp, output format name. *See* Web Help, output format
 Web Help 5, output format, 10, 11, 43, 44, 46
 webhelp5, output format name. *See* Web Help 5, output format
 whc-index-basename, parameter, 28
 wh-collapse-toc, parameter, 27
 whc-toc-basename, parameter, 29

wh-jquery, parameter, 27
 wh-jquery-css, parameter, 27
 wh-jquery-theme, parameter, 27
 wh-jquery-ui, parameter, 27
 wh-user-css, parameter, 27
 wh-user-footer, parameter, 27
 wh-user-header, parameter, 28
 wh-user-resources, parameter, 28
 .wml, filename extension, 8. *See also*
 WordprocessingML, output format
 wml, output format name. *See* WordprocessingML, output format
 WordprocessingML, output format, 8, 9, 10, 17, 52

X

-xep, option, 15
 XEP, XSL-FO processor, 8, 15
 -xfc, option, 15
 XFC, XSL-FO Converter, 9
 XFC, XSL-FO processor, 8, 15
 .xhtml, filename extension, 8. *See also* XHTML 1.0, output format
 xhtml, output format name. *See* XHTML 1.0, output format
 XHTML 1.0, output format, 8, 9, 10, 18, 51, 52
 XHTML 1.1, output format, 9, 10, 18, 51, 52
 xhtml1.1, output format name. *See* XHTML 1.1, output format
 XHTML 5, output format, 9, 10, 18, 43, 44, 46, 51, 53
 xhtml5, output format name. *See* XHTML 5, output format
 xhtml-mime-type, parameter, 26
 .xml, filename extension, 8. *See also*
 WordprocessingML, output format
 XML catalog, 55
 XMLmind XSL-FO Converter. *See* XFC, XSL-FO processor
 -Xmx, Java option, 17
 xref-auto-text, parameter, 22
 XSL-FO, output format, 10
 xsl-resources-directory, parameter, 23
 -xslt, option, 13
 XSLT stylesheets parameters, 19