XMLmind DITA Converter Manual

Hussein Shafie

XMLmind Software
35, rue Louis Leblanc
78120 Rambouillet
France
Phone: +33 (0)9 52 80 80 37
ditac-support@xmlmind.com
www.xmlmind.com/ditac/

July 22, 2020

Table of Contents

List of Figures

List of Tables

Introduction

XMLmind DITA Converter (*ditac* for short) allows to convert the most complex DITA 1.0, 1.1. 1.2 and **1.3** 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.

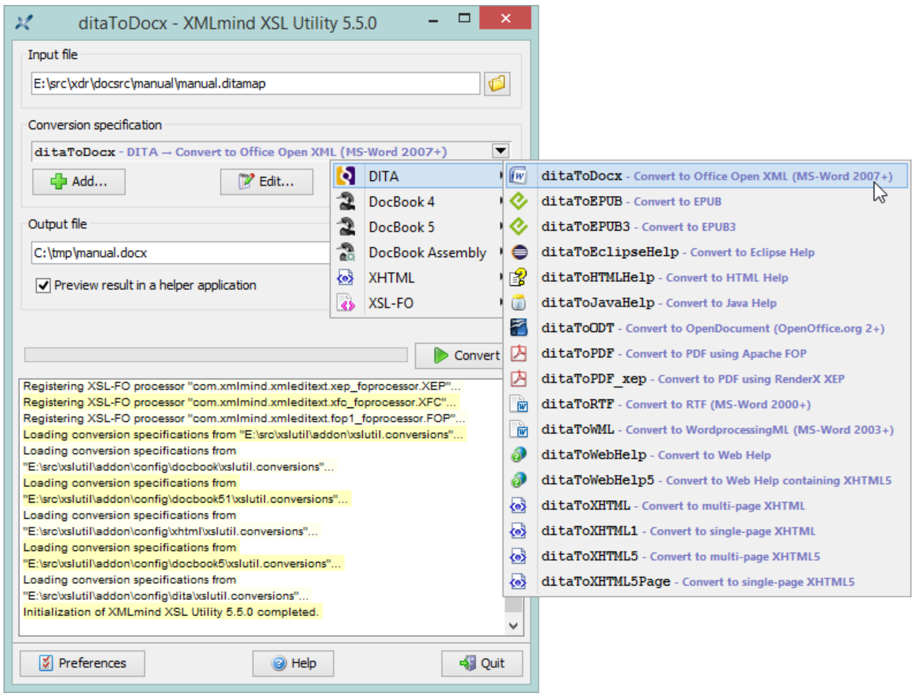
You'll find at the end of this document an appendix detailing the limitations and implementation specificities of ditac. Please refer to this appendix before posting support requests to the ditac-support@xmlmind.com, public, moderated, mailing list.

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



Input file: E:\src\xdr\docsrc\manual\manual.ditamap

Conversion specification: ditaToDocx - DITA → Convert to Office Open XML (MS-Word 2007+)

Output file: C:\tmp\manual.docx

[✓] Preview result in a helper application

```
Registering XSL-FO processor "com.xmlmind.xmleditext.xep_foprocessor.XEP"...
Registering XSL-FO processor "com.xmlmind.xmleditext.xfc_foprocessor.XFC"...
Registering XSL-FO processor "com.xmlmind.xmleditext.fop1_foprocessor.FOP"...
Loading conversion specifications from "E:\src\xslutil\addon\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\docbook\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\docbook51\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\xhtml\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\docbook5\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\dita\xslutil.conversions"...
Initialization of XMLmind XSL Utility 5.5.0 completed.
```

- ditaToDocx - Convert to Office Open XML (MS-Word 2007+)
- ditaToEPUB - Convert to EPUB
- ditaToEPUB3 - Convert to EPUB3
- ditaToEclipseHelp - Convert to Eclipse Help
- ditaToHTMLHelp - Convert to HTML Help
- ditaToJavaHelp - Convert to Java Help
- ditaToODT - Convert to OpenDocument (OpenOffice.org 2+)
- ditaToPDF - Convert to PDF using Apache FOP
- ditaToPDF_xep - Convert to PDF using RenderX XEP
- ditaToRTF - Convert to RTF (MS-Word 2000+)
- ditaToWML - Convert to WordprocessingML (MS-Word 2003+)
- ditaToWebHelp - Convert to Web Help
- ditaToWebHelp5 - Convert to Web Help containing XHTML5
- ditaToXHTML - Convert to multi-page XHTML
- ditaToXHTML1 - Convert to single-page XHTML
- ditaToXHTML5 - Convert to multi-page XHTML5
- ditaToXHTML5Page - Convert to single-page XHTML5

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.

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!

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

Part I. Using XMLmind DITA Converter

Chapter 1. Installing XMLmind DITA Converter

Before you begin

XMLmind DITA Converter (*ditac* for short) requires using a Java™ 1.8+ runtime.

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
openjdk version "14.0.1" 2020-04-14
OpenJDK Runtime Environment (build 14.0.1+7)
OpenJDK 64-Bit Server VM (build 14.0.1+7, 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-3_8_0.zip
   C:\ditac> dir ditac-3_8_0
   ... <DIR> bin
   ... <DIR> doc
   ... <DIR> docsrc
   ...
   ```

   XMLmind DITA Converter is intended to be used directly from the `ditac-3_8_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-3_8_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.

     WARNING

     Please install and use either Apache FOP 1.1 or FOP 2.1+. Please do *not* install and use Apache FOP 2.0 as we have found this version to have a severe bug (FOP-2461).

Note

If you have installed Apache FOP and your DITA document contain 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 Windows. Example: `C:\Users\john\AppData\Roaming\XMLmind\ditac\`.

   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-2.5\fop\fop.bat

-xfc E:\opt\xfc_eval_java-6_2_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.

  `relaxng.jar`
  : is Jing version 20030619, James Clark's RELAX NG validator, slightly modified for use in XMLmind XML Editor and XMLmind DITA Converter. The details of the modifications are found in `LEGAL/relaxng.README`.

  `saxon9.jar`
  : is Michael Kay's XSLT 2.0 engine. See http://www.saxonica.com/.

  `whcmin.jar`
  `snowball.jar`
  : contains the code needed to run XMLmind Web Help Compiler.

  `xslthl.jar`
  : contains the code of the XSLT syntax highlighting open source software component.

  `flexmark.jar`
  : flexmark-java is the software component used by ditac to parse Markdown and convert it to HTML.

`plus/`
: This directory is present only in the case of the `ditac-`*`N_N_N`*`-plus-fop.zip` distribution. It contains most recent Apache FOP (including hyphenation and MathML support). This XSL-FO processor is automatically declared and thus, ready to be used to generate PDF or PostScript.

`schema/`
: Contains the DTD, RELAX NG and W3C XML schemas of DITA 1.3 1.2, 1.1, 1.0.1. File `schema/catalog.xml` contains 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.

`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.

`xhtml/xhtml.xsl`
: Used to generate XHTML 1.0 pages.

`xhtml/xhtml1_1.xsl`
: Used to generate XHTML 1.1 pages.

`xhtml/html.xsl`
: Used to generate HTML 4.01 pages.

`xhtml/xhtml5.xsl`
: Used to generate XHTML 5 pages.

`webhelp/webhelp.xsl`
: Used to generate Web Help containing XHTML 1 pages, which are then compiled using XMLmind Web Help Compiler.

`webhelp/webhelp5.xsl`
: Used to generate Web Help containing XHTML 5 pages, which are then compiled using XMLmind Web Help Compiler.

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

`eclipsehelp/eclipsehelp.xsl`
: Used to generate Eclipse Help files.

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

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

`epub/epub3.xsl`
: Used to generate EPUB 3 files, which are then archived in a `.epub` file (Zip archive having a `.epub` extension).

`hdita/hdita.xsl`
: Translates XHTML5 to DITA according to the HDITA specification.

Chapter 2. Getting started

1. Using the `ditac` command-line utility

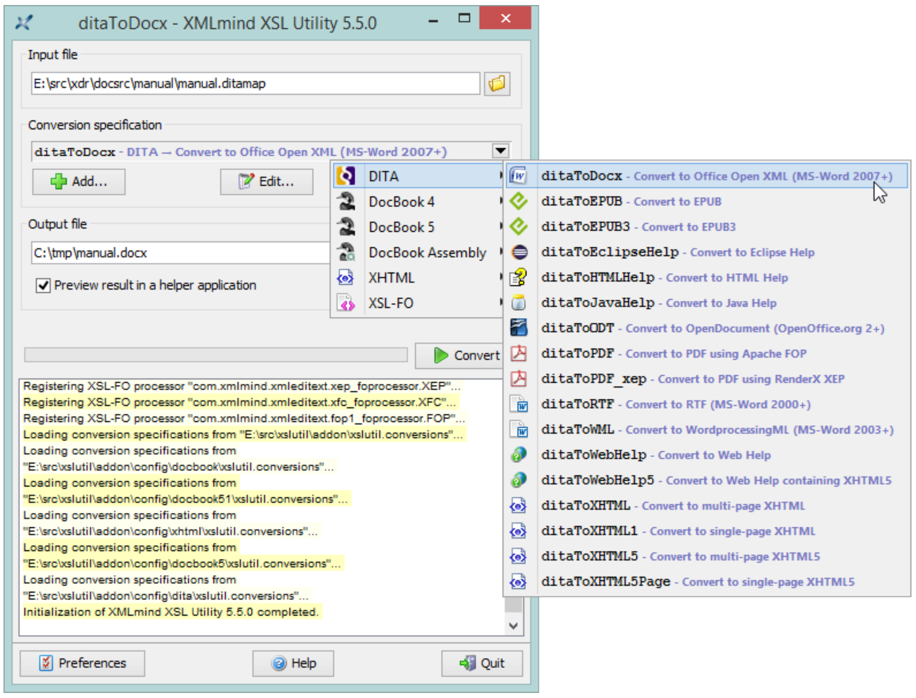
In this chapter, we'll explain how to run the **`ditac`** command-line utility by using examples. You'll find all the DITA input files used to run the following examples in the `ditac_install_dir/docsrc/manual/` directory.

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


ditaToDocx - XMLmind XSL Utility 5.5.0
Add...
Edit...
DITA
DocBook 4
DocBook 5
DocBook Assembly
XHTML
XSL-FO
ditaToDocx - Convert to Office Open XML (MS-Word 2007+)
ditaToEPUB - Convert to EPUB
ditaToEPUB3 - Convert to EPUB3
ditaToEclipseHelp - Convert to Eclipse Help
ditaToHTMLHelp - Convert to HTML Help
ditaToJavaHelp - Convert to Java Help
ditaToODT - Convert to OpenDocument (OpenOffice.org 2+)
ditaToPDF - Convert to PDF using Apache FOP
ditaToPDF_xep - Convert to PDF using RenderX XEP
ditaToRTF - Convert to RTF (MS-Word 2000+)
ditaToWML - Convert to WordprocessingML (MS-Word 2003+)
ditaToWebHelp - Convert to Web Help
ditaToWebHelp5 - Convert to Web Help containing XHTML5
ditaToXHTML - Convert to multi-page XHTML
ditaToXHTML1 - Convert to single-page XHTML
ditaToXHTML5 - Convert to multi-page XHTML5
ditaToXHTML5Page - Convert to single-page XHTML5
Convert
Preferences
Help
Quit

Input file: E:\src\xdr\docsrc\manual\manual.ditamap

Conversion specification: ditaToDocx - DITA → Convert to Office Open XML (MS-Word 2007+)

Output file: C:\tmp\manual.docx

[✓] Preview result in a helper application

```
Registering XSL-FO processor "com.xmlmind.xmleditext.xep_foprocessor.XEP"...
Registering XSL-FO processor "com.xmlmind.xmleditext.xfc_foprocessor.XFC"...
Registering XSL-FO processor "com.xmlmind.xmleditext.fop1_foprocessor.FOP"...
Loading conversion specifications from "E:\src\xslutil\addon\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\docbook\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\docbook51\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\xhtml\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\docbook5\xslutil.conversions"...
Loading conversion specifications from
"E:\src\xslutil\addon\config\dita\xslutil.conversions"...
Initialization of XMLmind XSL Utility 5.5.0 completed.
```

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:

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

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

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
```

Tip

No need to declare Apache FOP using the -fop command-line option if you have installed the distribution called `ditac-N_N_N-plus-fop.zip`. This distribution contains most recent Apache FOP (including hyphenation and MathML support). This XSL-FO processor is automatically declared and thus, ready to be used to generate PDF or PostScript.

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`.

WARNING

XMLmind XSL-FO Converter Evaluation Edition (download page) generates output containing *random duplicate letters*. This makes this edition useless for any purpose other than evaluating XMLmind XSL-FO Converter. Of course, this does not happen with XMLmind XSL-FO Converter Professional Edition!

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`

Format	Name
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
```

Remember

If you want to see your document by selecting **Help** → **Help Contents** in Eclipse:

1. Do not specify any of the following command-line options when generating Eclipse Help: **-toc**, **-index**.
2. 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).

3. Copy this output directory to *eclipse_install_dir*/dropins/ and not *eclipse_install_dir*/plugins/.

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 conversion parameter, generally an XSLT stylesheet parameter. See Chapter 4.

A *`param_name`* starting with "`load.`*`doc_loader_name`*`.`" specifies an option which is passed to the alternate document loader called *`doc_loader_name`*. For example, `-p load.mdita.autolink true` turns on the `autolink` extension in the MDITA loader. See MDITA support.

`-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 the reuse of a map designed to output multiple HTML pages in order to generate a PDF file.

For example, "`-chunk none`" allows the reuse of 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.

`-attrvalues` *subject_scheme_map_URL_or_file*

Specify an external subject scheme map. The controlled attribute values found in this subject scheme map are prepended to those loaded from the subject scheme maps possibly referenced in the map or bookmap to be converted.

`-defaultattrvalues` *subject_scheme_map_URL_or_file*

Same as **`-attrvalues`** except that the controlled attribute values found in this subject scheme map will ***not*** be used if some controlled attribute values are loaded from the subject scheme maps referenced in the map or bookmap to be converted.

`-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.

(3) Not counting `<topicref>`s contained in `<frontmatter>` and `<backmatter>`.

`-index`
: Equivalent to "`-backmatter indexlist`".

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

When used on a `<bookmap>`, this option adds elements *after* any existing `<booklists>` elements.

The syntax of *spec* is:

```
spec -> same_page [ ',' same_page ]*

same_page -> section [ '+' section ]*

section -> 'toc'|'figurelist'|'tablelist'|'examplelist'|
           'equationlist'|'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
```

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

When used on a `<bookmap>`, this option adds elements *before* any existing `<booklists>` elements.

`-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 copies `main.html` to `index.html`. Applies to formats: `xhtml`, `xhtml1.1`, `html`, `webhelp`.

`-lang` ***language_code***
: Specifies the main language of the document

Shorthand for:

```
-foconverter  pdf "executable_file"
-foconverter  ps "executable_file"
```

. 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 lets the user 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).

Shorthand for:

```
-foconverter FOP pdf "executable_file" -q -r -fo "%I" -pdf "%O"
-foconverter FOP ps "executable_file" -q -r -fo "%I" -ps "%O"
```

`-xep` *executable_file*

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

Shorthand for:

```
-foconverter XEP pdf "executable_file" -quiet -valid -fo "%I" -pdf "%O"
-foconverter XEP ps "executable_file" -quiet -valid -fo "%I" -ps "%O"
```

`-ahf` *executable_file*

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

Shorthand for:

```
-foconverter AHF pdf "executable_file" -x 3 -p @PDF -d "%I\" -o "%O"
-foconverter AHF ps "executable_file" -x 3 -p @PS -d "%I" -o "%O"
```

`-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`.

Shorthand for:

```
-foconverter XFC rtf "fo2rtf_executable_file" "%I" "%O"
-foconverter XFC wml "fo2wml_executable_file" "%I" "%O"
-foconverter XFC docx "fo2docx_executable_file" "%I" "%O"
-foconverter XFC odf "fo2odt_executable_file" "%I" "%O"
```

WARNING

XMLmind XSL-FO Converter Evaluation Edition (download page) generates output containing *random duplicate letters*. This makes this edition useless for any purpose other than evaluating XMLmind XSL-FO Converter. Of course, this does not happen with XMLmind XSL-FO Converter Professional Edition!

`-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.

This low-level option may be used for example to specify a configuration file for Apache FOP:

```
-foconverter FOP pdf \
  '/opt/fop/fop -c /home/john/docs/fop.conf -q -r -fo "%I" -pdf "%O"'
```

`-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.

`-validate`

Validate *all* the XML files loaded by **`ditac`**. Any validation error will cause **`ditac`** to immediately stop running. Therefore the combination of the **`-validate`** and **`-dryrun`** options gives you a simple way to thoroughly check your DITA document.

Note that for the **`-validate`** option to work, *all* the XML files (maps, topics, even `.ditaval` filter files) loaded by **`ditac`** must start with the proper `<!DOCTYPE>` declaration.

This option is unrelated to attribute value validation validation by the means of subject scheme maps. When the map to be converted (or any of its submaps) references some subject scheme maps then the attribute value validation is automatic and cannot be turned off.

`-dryrun`
: Use **`ditac`** as a validator, and most notably check cross-references. 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. Example: `C:\Users\john\AppData\Roaming\XMLmind\ditac\`.

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-2.5\fop\fop.bat
-xfc E:\opt\xfc_eval_java-6_2_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`), `fo2`*`xxx`* (`fo2`*`xxx`*`.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. Note that this workaround is automatically used when you specify which RenderX XEP executable to use by the means of the **`-xep`** command-line option.

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
`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.
`xsl/xhtml/xhtml.xsl`	Used to generate XHTML 1.0 pages.
`xsl/xhtml/xhtml1_1.xsl`	Used to generate XHTML 1.1 pages.
`xsl/xhtml/html.xsl`	Used to generate HTML 4.01 pages.
`xsl/xhtml/xhtml5.xsl`	Used to generate XHTML 5 pages.

XSLT stylesheet	Description
`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 9, 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.
`cause-number-format`	Allowed values are: `'I'`, `'i'`, `'A'`, `'a'`, `'1'`. Default value: `'A'`.	In a `<troubleshooting>` topic, multiple `<remedy>` elements having no title are given numbers formatted using this format.
`center`	List of element names separated by whitespace. Example: `'fig equation-figure simpletable table'`. Default value: `''`.	Specifies which elements are to be centered horizontally on the page.
⚙`ditacListsURI`	URL[4]. Default value: *`output_dir`*`/ditac_lists.ditac_lists`.	The URL of file `ditac_lists.ditac_lists`.
`equation-number-after`	String. Default value: `')'`.	Text added after the contents of a `<equation-number>` element.

[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
`equation-number-before`	String. Default value: `'('`.	Text added before the contents of a `<equation-number>` element.
`extended-toc`	Allowed values are: `'frontmatter'`, `'backmatter'`, `'both'`, `'none'`. Default value: `'none'`.	Allows to add `<frontmatter>` and `<backmatter>` `<topicref>`s to the **Table of Contents** (TOC) of a document. Note that the `@toc`, `@navtitle`, `@locktitle`, etc, attributes are applied normally to `<frontmatter>` and `<backmatter>` `<topicref>`s 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 *external* `@scope` attribute and a *relative* `@href` attribute. Example of such `<xref>` elements: `<xref scope="external" format="java" href="src/Test.java">Test.java</xref>`. `''` Do not resolve the `@href` attribute. In this case, the external resource files are expected to be copied "by hand" to the output directory. **An URL ending with "/"** This URL is prepended to the value of the `@href` attribute. **`'#REMOVE'`** The `<xref>` or `<link>` element is processed as if it did not have an `@href` attribute.
`highlight-source`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Allows to turn off syntax highlighting in elements specializing `<pre>`. 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`.
`index-range-separator`	String. Default value: `'&#x2013;'` (EN DASH).	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='<-->'`: **C** `Cat 54, 87<-->90`
`link-auto-text`	List of values separated by whitespace. Allowed val-	This parameter specifies which text to generate for a `<link>` element, when this `<link>` element has no

Parameter	Value	Description
	ues are: `'number'` and `'text'`. Default value: `'number text'`.	`<linktext>` child element or when this `<linktext>` child element is empty. Similar to above parameter xref-auto-text but for `<link>` elements.
`note-icon-list`	List of `type` attribute values separated by whitespace. Default value: `'attention caution danger fastpath important note notes remember restriction tip'`.	Specifies the type (attribute `@type`) of the `<note>` elements for which icons should be used rather than text in order to represent note labels. Ignored unless `use-note-icon='yes'`.
`number`	List of values separated by whitespace. Allowed values are: `'topic'`, `'chapter-only'`, `'table'`, `'fig'`, `'example'`, `'equation-figure'`, `'all'`. Default value: `''` (number nothing).	Specifies which elements are to be numbered. `'all'` is a short form for `'topic table fig equation-figure'`. `'chapter-only'` means: number topics, but only those referenced in a bookmap as `<part>`, `<chapter>` and `<appendix>`. **Note** Please note that `'all'` does not include `'example'`. If you want to number all formal elements including examples, then you must specify `'all example'`.
`number-separator1`	String. Default value: `'.'`.	The string used to separate the hierarchical number of topics acting as sections.
`number-separator2`	String. Default value: `'-'`.	The string used to separate the hierarchical number of figures, tables, examples and equations. When possible, the number of figure, table, example or equation 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.
`mark-important-steps`	Allowed values are: `'yes'` and `'no'`. Default value: `'no'`.	Generates a "Required" (respectively "Optional") label for `<step>` and `<substep>` elements having an `@importance` attribute set to "`required`" (resp. "`optional`").

Parameter	Value	Description
`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 `prepend-chapter-to-section-number='yes'` 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.
`remedy-number-format`	Allowed values are: `'I'`, `'i'`, `'A'`, `'a'`, `'1'`. Default value: `'A'`.	In a `<troubleshooting>` topic, multiple `<remedy>` elements having no title are given numbers formatted using this format.
`show-draft-comments`	Allowed values are: `'yes'` and `'no'`. Default value: `'no'`.	Specifies whether `<draft-comments>` elements should be rendered.
`troubleSolution-number-format`	Allowed values are: `'I'`, `'i'`, `'A'`, `'a'`, `'1'`. Default value: `'1'`.	In a `<troubleshooting>` topic, multiple `<troubleSolution>` elements having no title are given numbers formatted using this format.
`title-after`	List of element names separated by whitespace. Example: `'fig equation-figure 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.

Parameter	Value	Description
		This custom title page is an XHTML file for XHTML-based formats (XHTML, HTML 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. • The child nodes of the `body` element of the custom XHTML title page are wrapped in a `div` contained in the XHTML/HTML file being generated by the XSLT stylesheet. Do not add a `<!DOCTYPE>` to such custom XHTML title page because otherwise, the XSLT stylesheet may fail loading it. See sample custom XHTML title page. • The child nodes of the first `fo:flow[@flow-name='xsl-region-body']` element of the custom XSL-FO title page are wrapped in a `fo:block` contained the XSL-FO file being generated by the XSLT stylesheet. See sample custom XSL-FO title page.
`title-prefix-separator1`	String. Default value: `'. '`.	The string used to separate the number of an formal object from its title.
`use-note-icon`	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 `<note>` element.
`watermark-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 a watermark in all the pages comprising the output document. See also parameter `watermark`. If you need this feature when generating RTF, WordprocessingML, Office Open XML (`.docx`), OpenDocument (`.odt`), please make sure to use XMLmind XSL-FO Converter v5.3+.
`xref-auto-text`	List of values separated by whitespace. Allowed values are: `'number'` and `'text'`.	This parameter specifies which text to generate for an `<xref>` element, when this `<xref>` element contains no text at all[5].

[5] This implies that the `xref-auto-text` parameter is ignored when an `<xref>` element contains some text.

Parameter	Value	Description

	Default value: `'number'`.	Let's suppose that an `<xref>` element containing no text at all points to a topic titled "`Installation`".
		
		Because the `<xref>` element points to an element having a `<title>` child element, ditac may use this title as a starting point for the generated text.
		
		Now let's suppose that topics are numbered and that the number of the "`Installation`" topic is "`Chapter 5`".
		
		The text generated for this `<xref>` element is thus:
		
		If `xref-auto-text='number'`
		`Chapter 5`
		
		If `xref-auto-text='text'`
		`Installation`
		
		If `xref-auto-text='number text'`
		`Chapter 5. Installation`
		
		Note that this specification is just a hint because ditac needs anyway to generate some text. For example, if topics are not numbered and `xref-auto-text='number'`, the generated text will be "`Installation`".

★`xsl-resources-directory`	URL. A relative URL is relative to the output directory. Default value: `'resources/'` resolved against the directory which contains the XSLT stylesheets.	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. If this directory does not exist, it is automatically created. If this directory does not already contain the resources needed by the XSLT stylesheets, such resources are automatically copied to this directory. The default value of this parameter is something like `file:/opt/ditac/xsl/xhtml/resources/` for the stylesheets generating XHTML. URL `file:/opt/ditac/xsl/xhtml/resources/` specifies an existing directory containing `basic.css`, `note.png`, `important.png`, etc. This means that by default, no directory is created and no resource is copied. 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.

Parameter	Value	Description
		Important • Explicitly specifying something like `xsl-resources-directory='res'` is almost *always required* when generating files having an XHTML/HTML based format (XHTML, HTML Help, etc). • Explicitly specifying something like `xsl-resources-directory='res'` is almost *never required* when generating files converted from XSL-FO (PDF, RTF, etc).

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
`add-index-toc`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Specifies whether an A-Z list should be added at the beginning of the back-of-the-book index.
`chain-pages`	Allowed values are: `'none'`, `'top'`, `'bottom'` or `'both'`. Default value: `'none'`.	Specifies whether a header and/or a footer containing navigation icons should be generated in order to link together all the HTML pages. **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'`.	Specifies whether navigation icons should be generated in order to link together all the topics.

Parameter	Value	Description
	Default value: `'no'`.	See also related parameter: `ignore-navigation-links`. **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: `''`.	Low-level parameter specifying which CSS stylesheet to use to style the generated (X)HTML pages. When neither `css` nor `custom-css` is specified, the default CSS stylesheet being used is *`xsl-resources-directory`*`/base.css`. **Restriction** Not supported by the stylesheets that generate EPUB.
`cssResourceName`	URL basename relative to the directory specified by parameter `xsl-resources-directory`. Default value: `'base.css'`.	Very low-level parameter specifying which CSS stylesheet to use. This CSS stylesheet is expected to be found in the resources directory. **Note** This parameter is not useful unless you develop a plug-in implementing a DITA specialization. More information in Part II, Chapter 10.
⭐`custom-css`	URL. Default value: `''`.	Specifies the custom CSS stylesheet used to style the generated (X)HTML pages. This high-level parameter has priority over low-level parameter `css`. This custom CSS stylesheet is copied to directory `xsl-resources-directory`. Therefore `custom-css` requires directory `xsl-resources-directory` to be specified as an URL which is relative to the output directory (e.g. `xsl-resources-directory='res'`).

Parameter	Value	Description
		How to use `custom-css` is explained in Part II, Chapter 9, Section 1.
`default-table-width`	A percentage, typically something like `'100%'` or `'90%'`. Default value: `''` (as narrow as possible).	The default width of `<table>` and `<simpletable>` elements.
`external-link-icon-height`	Length. A length may have a unit. Default is `px`. Default value: `'10'`.	The height of the "opens in new window" icon.
`external-link-icon-name`	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: `'10'`.	The width of the "opens in new window"" icon.
`format-to-type`	Zero or more DITA format/MIME type pairs. Example: `"txt text/plain xml application/xml html text/html"`. Default value: `''`, which means that DITA `xref/@format` is *not* converted to XHTML `a/@type`.	Allows to map DITA `xref/@format` to XHTML `a/@type`. Using default empty value, `<xref scope="external" format="txt" href="http://acme.com/info.xyz">` is converted to `<a href="http://acme.com/info.xyz" target="_blank">`. The fact that file extension "`.xyz`" is unknown may cause problems when attempting to navigate or download file "`info.xyz`" using a Web browser. If `-p format-to-type "txt text/plain"` is passed to **`ditac`** then `<xref scope="external" format="txt" href="http://acme.com/info.xyz">` is converted to `<a type="text/plain" href="http://acme.com/info.xyz" target="_blank">`, which is better.
`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 `<meta name="generator" content="`*`XXX`*`"/>` element added to your HTML pages.
`ignore-navigation-links`	Allowed values are: `'yes'`, 'no' and `'auto'`.	If `'yes'`, do not generate the navigation links corresponding to `topicref` attribute `@collection-type`.

Parameter	Value	Description
	Default value: `'auto'` for XHTML and its variants; `'yes'` for Web Help, Java Help, HTML Help, Eclipse Help and EPUB	If `'no'`, generate the navigation links corresponding to `topicref` attribute `@collection-type`. If `'auto'`, generate the navigation links corresponding to `topicref` attribute `@collection-type`, unless `chain-topics=yes`.

`javascripts`	String. List of URLs separated by whitespace.
	
	Default value: `''`.
		The URLs specified in this parameter must point to JavaScript files. These URLs are converted to `<script>` XHTML elements added to the `<html>/<head>` elements of the XHTML files generated by ditac.
		
		Note that an URL may end with `';async'`, `';defer'` or a combination of both flags. These flags are translated to the corresponding attributes of the `<script>` element. Example:
		
		`https://cdnjs.cloudflare.com/ajax/libs/mathjax/¬`
		`2.7.7/MathJax.js?config=MML_CHTML`**`;async`**
		
		is translated to:
		
		`<script type="text/javascript"` **`async="async"`**
		`src="https://cdnjs.cloudflare.com/ajax/libs/¬`
		`mathjax/2.7.7/MathJax.js?config=MML_CHTML">`
		`</script>`

`mathjax`	Allowed values are: `'yes'`, `'no'` and `'auto'`. Default value: `'no'`.	Very few web browsers (Firefox) can natively render MathML. Fortunately, there is MathJax. MathJax is a JavaScript display engine for mathematics that works in all browsers. **`'yes'`** Add a `<script>` XHTML element loading MathJax to the `<html>/<head>` elements of all XHTML files generated by ditac. **`'auto'`** Same as `'yes'`, but add `<script>` only to generated XHTML files containing MathML. Ignored by all XHTML-based formats but XHTML and Web Help.
`mathjax-url`	String. Default value: the URL pointing to the MathJax CDN, as recommended in the MathJax documentation.	The URL allowing to load the MathJax engine configured for rendering MathML. Ignored unless parameter `mathjax` is set to `'yes'`or `'auto'`.
`mark-external-links`	Allowed values are: `'yes'` and `'no'`.	Specifies whether an external link should be marked using a “opens in new window” icon.

Parameter	Value	Description
	Default value: `'no'`.	
`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 `note-icon-suffix='.svg'`, the default resources directory is expected to contain `first.svg`, `first_disabled.svg`, `last.svg`, etc. In principle, there is no need for an end-user to specify any of the `navigation-icon-suffix`, `navigation-icon-width` or `navigation-icon-height` parameters.
`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: see prose.	Low-level parameter. Do not change default value unless you know what you are doing. • Specify `'text/html'` to serve XHTML as HTML. This is the default value for all (X)HTML-based output formats except for EPUB 2 and (X)HTML5. • Specify `'application/xhtml+xml'` if you prefer to serve XHTML as XML. This is the default value for EPUB 2. • Specify an empty string if you prefer not to generate `<meta http-equiv="Content-Type">`. This is the default value for (X)HTML5 for which a `<meta charset="UTF-8">` is generated instead.

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'`.	If `'yes'`, add a pseudo TOC entry, bearing the title of the document, containing all the actual TOC entries. **Restriction** • Value `'no'` is not supported by the stylesheets that generate Eclipse Help. • Ignored by the stylesheets that generate Web Help and EPUB.
`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---`*`CSS_VAR_NAME`*	String. A valid CSS property value.
	
	No default.
		This kind of parameter may be used to override any of the default values of the CSS variables specified in any of the `NNtheme.css` template files (all found in *`ditac_install_dir`*`/whc_template/_wh/`).
		
		For example, the main `NNtheme.css` template file:
		`body {`
		`  ...`
		`  --navigation-width: 33%;`
		`  ...`
		`}`
		
		The `wh---navigation-width` CSS variable is used as follows in `NNcommon.css`, another CSS template file:
		`#wh-navigation {`
		`  ...`
		`  width: var(--navigation-width);`
		`  ...`
		`}`

Parameter	Value	Description
		Therefore parameter `wh---navigation-width` may be used to give the navigation side of the generated Web Help a different initial width. Example: `-p wh---navigation-width "25%"`. More examples in "*XMLmind Web Help Compiler Manual, Getting started*".
`wh-collapse-toc`	Allowed values are: `'yes'` and `'no'`. Default value: `'no'`.	Specifies whether the TOC should be initially collapsed.
`wh-index-numbers`	Allowed values are: `'yes'` and `'no'`. Default value: `'no'`.	Specifies whether words looking like numbers are to be indexed. Examples of such number-like words: `3.14`, `3,14`, `3times4equals12`, `+1`, `-1.0`, `3px`, `1,2cm`, `100%`, `1.0E+6`, `1,000.00$`.

`wh-inherit-font-and-colors`	Allowed values are: `'yes'` and `'no'`.
	
	Default value: `'yes'`.
		When `wh-inherit-font-and-colors` is set to `'no'`, the navigation pane of the generated Web Help uses fonts and colors of its own, which will generally differ from those used for the content of the Web Help.
		
		Setting `wh-inherit-font-and-colors` to `'yes'` lets you use for the navigation pane the same fonts and colors as those used for the content of the Web Help.
		
		So basically this parameter is a shorthand for:
		
		`-p wh---navigation-font-family inherit¬`
		`-p wh---navigation-font-size inherit¬`
		`-p wh---navigation-color inherit¬`
		`-p wh---navigation-background-color inherit`
		
		See above `wh---CSS_VAR_NAME` parameters.

`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:
		
		`https://code.jquery.com/¬`
		`jquery-3.4.1.slim.min.js`
		
		Specifying an "`https:`" URL is recommended when the generated Web Help is stored on an HTTPS server.

`wh-local-jquery`	Allowed values are: `'yes'` and `'no'`. Default value: `'no'`.	Specifies whether all jQuery files should be copied to `_wh/jquery/`, where `_wh/` is the directory containing the other Web Help files. By default, the jQuery files are accessed from the Web (typically from a CDN).

Parameter	Value	Description
		Note that this parameter is applied *after* jQuery has been possibly customized using parameter `wh-jquery`. For example, "`-p wh-jquery https://code.jquery.com/jquery-3.4.1.js`" copies a file downloaded from `https://code.jquery.com/ to _wh/jquery/`.
`wh-layout`	The name of a layout. Default value: classic.	Selects a layout for the generated Web Help. For now, only two layouts are supported: `classic` and `simple`.
`wh-responsive-ui`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Specifies whether the generated Web Help should be “responsive”, that is, whether it should adapt its layout to the size of the screen.
`wh-ui-language`	"`browser`" or "`document`" or a language code conforming RFC 3066. Examples: `de`, `fr-CA`. Default value: `'browser'`.	Specifies which language should be used for the messages (tab labels, button tool tips, etc) of the generated Web Help. Default value "`browser`" means that this language is the one used by the Web browser for its own messages. This language may often be specified in the user preferences of the Web browser. Value "`document`" means that the language of the document should be used. A language code such as `en`, `en-US`, `es`, `es-AR`, etc, may be used to explicitly specify which language should be used.
`wh-use-stemming`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Specifies whether *stemming*(6) should be used to implement the search facility. By default, stemming is used whenever possible, that is, 1. when the main language of the document can be determined; 2. when this main language is one of: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Russian, Spanish, Swedish, Romanian, Turkish. The main language of the document is specified by the `@xml:lang` attribute found on the root element of DITA map being converted; otherwise using the -lang command-line option; otherwise, it is assumed to be "`en`".

(6) In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form.

Parameter	Value	Description

☆`wh-user-css`	Filename or absolute URI of a CSS file. A relative filename 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 `output_directory/_wh/user/`.
		
		Sample user's CSS wh_resources/header_footer.css as used in the following example:
		
		`-p wh-user-header¬`
		`wh_resources/header.html`
		`-p wh-user-footer¬`
		`wh_resources/footer.html`
		`-p wh-user-css¬`
		`wh_resources/header_footer.css`
		`-p wh-user-resources¬`
		`wh_resources/header_footer_files`

☆`wh-user-footer`	Filename or absolute URI of an XHTML file. A relative filename is relative to the current working directory.	Specifies the user's footer which is to be added to each page of the Web Help.
		
		The content of the `<body>` element of `wh-user-footer` is inserted as is in the `<div id="wh-footer">` found in each page of the Web Help.
		
		Same remark as for parameter `wh-user-header` about the resources referenced by a user's footer.
		
		Sample user's footer wh_resources/footer.html as used in the following example:
		
		`-p wh-user-header¬`
		`wh_resources/header.html`
		`-p wh-user-footer¬`
		`wh_resources/footer.html`
		`-p wh-user-css¬`
		`wh_resources/header_footer.css`
		`-p wh-user-resources¬`
		`wh_resources/header_footer_files`
		
		More examples in "*XMLmind Web Help Compiler Manual, Getting started*".

☆`wh-user-header`	Filename or absolute URI of an XHTML file. A relative filename is relative to the current working directory.	Specifies the user's header which is to be added to each page of the Web Help. The content of the `<body>` element of `wh-user-header` is inserted as is in the `<div id="wh-header">` found in each page of the Web Help. 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 `wh-user-resources` must be specified.

Parameter	Value	Description
		Example: • The user's resource directory is called `header_footer_files/` and contains `header_footer_files/200x100.png`. • **`ditac`** is passed parameters: `-p wh-user-resources` *`PATH_TO/`*`header_footer_files` and `-p wh-user-header` *`PATH_TO/`*`header.html`. • `header.html` looks like this: `<html>` `...` `<body>` `...` `<img src="`**`_wh/user/header_footer_files/¬`** `logo200x100.png" />` `...` `</body>` `</html>` Notice the path used to reference `logo200x100.png`. Sample user's header wh_resources/header.html as used in the following example: `-p wh-user-header¬` `wh_resources/header.html` `-p wh-user-footer¬` `wh_resources/footer.html` `-p wh-user-css¬` `wh_resources/header_footer.css` `-p wh-user-resources¬` `wh_resources/header_footer_files` More examples in "*XMLmind Web Help Compiler Manual, Getting started*".
★`wh-user-resources`	Filename or absolute "`file:`" URI of a *directory*. URI schemes other than "`file`" (e.g. "`http`") are not supported for this parameter. A relative filename is relative to the current working directory.	Specifies a user's resource directory which is to be recursively copied to *`output_directory/`*`_wh/user/`. This directory typically contains image files referenced by the user's header, footer or CSS stylesheet. Sample user's resource directory wh_resources/header_footer_files/ as used in the following example: `-p wh-user-header¬` `wh_resources/header.html` `-p wh-user-footer¬` `wh_resources/footer.html`

Parameter	Value	Description
		`-p wh-user-css¬ wh_resources/header_footer.css -p wh-user-resources¬ wh_resources/header_footer_files` More examples in "*XMLmind Web Help Compiler Manual, Getting started*".
`whc-index-basename`	URL basename. Default value: `'whc_index.xml'`.	Basename of the Index XML input file of XMLmind Web Help Compiler. In principle, there is no need for an end-user to specify this parameter.
`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: `'JavaHelpSearch'`.	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: `'jhelptoc.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 *`%compiledFile%`*, *`%contentsFile%`*, *`%defaultTopic%`*, etc, which are substituted with their values.
`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'`. **Important** The subdirectory of `plugins/` containing the plug-in must have the same basename as the value of parameter `plugin-id`.
`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&apos;s Guide'`.

Parameter	Value	Description
`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.
`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).
`epub2-compatible`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Only applies to EPUB 3. By default, the EPUB 3 files generated by ditac are made compatible with EPUB 2 readers. Specify `'no'` if you don't need this compatibility.
`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 `epub:trigger` yet. When `generate-epub-trigger=no`, ditac generates an `@onclick` 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.

Tip

Inserting a `<?pagebreak?>` processing-instruction in the topic source between paragraphs, notes, tables, lists, etc, may be used to force a page break when generating any of the output formats which uses XSL-FO as an intermediate format (PDF, RTF, DOCX, etc).

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-start-indent`	Length. Default value: `'2pc'`.	Applies only to *alternate XSLT stylesheet* `ditac_install_dir/xsl/fo/fo_indent.xsl`. This stylesheet: • Indents all blocks but topic and section titles by the value of XSLT stylesheet parameter *`body-start-indent`*. By default *`body-start-indent`* is `2pc`. • Adds more vertical space after topic and section titles. • Only part, appendices, chapter and appendix titles are underlined. This stylesheet is invoked by passing option `-t ditac-xsl:fo/fo_indent.xsl` to **`ditac`**. Example of its output: `manual-fop.pdf`.
`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: `'&#x2022;' (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`.

Parameter	Value	Description
`equation-block-equation-width`	Length. Default value: `'90%'`.	In a numbered `<equation-block>` element, this parameter specifies the width of the column containing the equation.
`equation-block-number-width`	Length. Default value: `'10%'`.	In a numbered `<equation-block>` element, this parameter specifies the width of the column containing the `<equation-number>` element.
`external-href-after`	String. Default value: `']'`.	Appended after the external URL referenced by an `<xref>` or `<link>` element. Ignored unless `show-external-links='yes'`.
`external-href-before`	String. Default value: `' ['`.	Separates the text of an `<xref>` or `<link>` element from its referenced external URL. Ignored unless `show-external-links='yes'`.
`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.	Specifies the contents of the central part of a page footer. See Specifying a header or a footer.
		
		Supports a conditional specification.
		
		Default value:
		
		`two-sides even:: {{chapter-title}};;`
		`two-sides part||chapter||appendices||appendix odd::¬`
		`{{section1-title}};;`
		`one-side:: {{chapter-title}}`

`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.	Specifies the contents of the left part of a page footer. See Specifying a header or a footer. Supports a conditional specification. Default value: `two-sides even:: {{page-number}}`

Parameter	Value	Description
`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.	Specifies the contents of the right part of a page footer. See Specifying a header or a footer.
		
		Supports a conditional specification.
		
		Default value:
		
		`two-sides first||odd:: {{page-number}};;`
		`one-side:: {{page-number}}`

`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.
`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.

Parameter	Value	Description
`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.
`header-separator`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Specifies whether an horizontal rule should be drawn below the page header.
`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: `'2em'`.	The distance which separates columns in index pages.
`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: `'&#x2022;' (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: `'&#x2192;'` (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`.	The height of a note icon.

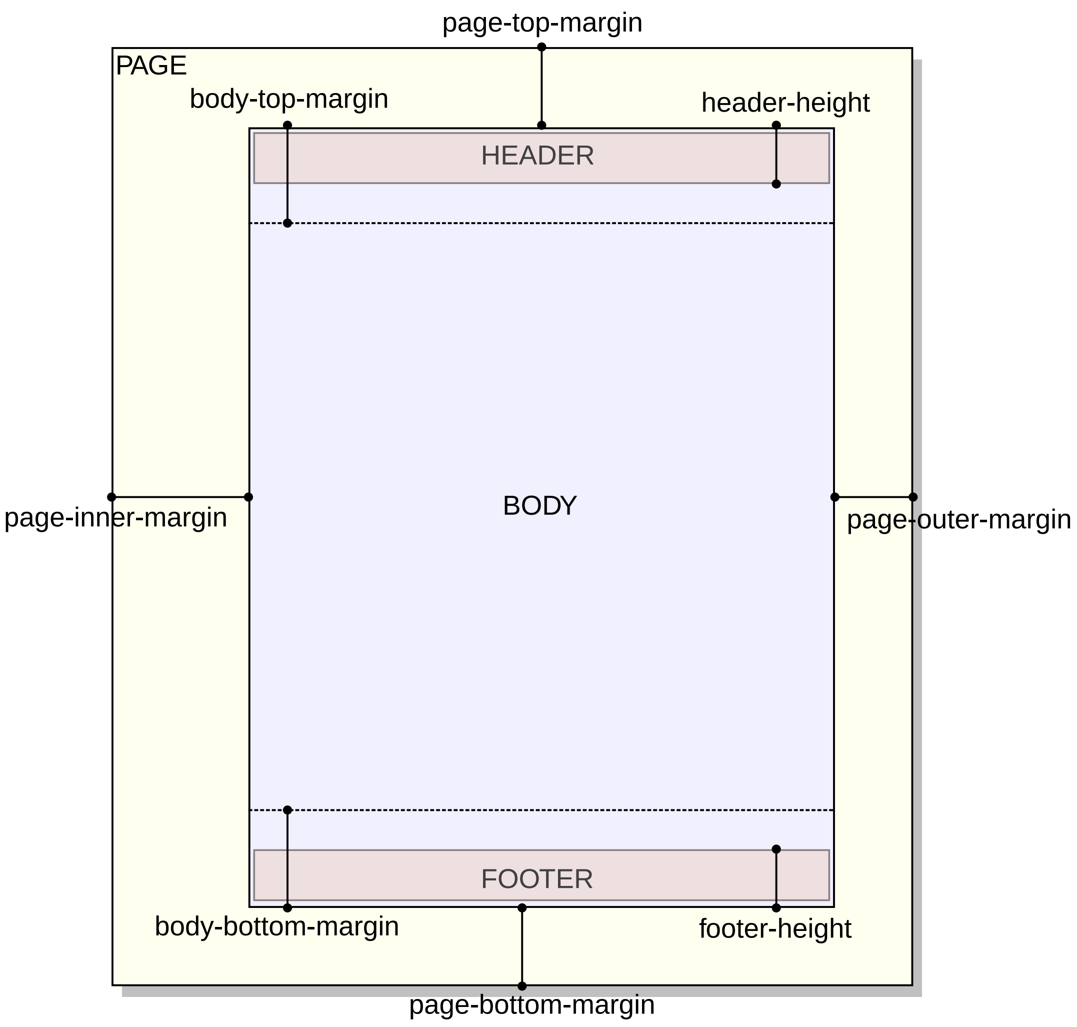
Parameter	Value	Description
	Default value: `'32'`. `'7mm'` for the XSLT stylesheets that generate XSL-FO.	
`note-icon-suffix`	Default value: `'.png'`.	The suffix of a note icon. The root name of a note icon should be identical to the value of the `@type` attribute it represents. For example, if `note-icon-suffix='.svg'`, 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.
`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.	See Figure 4-1 below.

Parameter	Value	Description
	Default value: if parameter `two-sided` is specified as `'yes'` then `'0.75in'` otherwise `'1in'`.	
`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'`, `'C3'`, `'C4'`, `'C5'`, `'C6'`, `'C7'`, `'C8'`, `'C9'`, `'C10'` (case-insensitive). Default value: `'A4'`.	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.
`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'`.

Parameter	Value	Description
`show-external-links`	Allowed values are: `'yes'` and `'no'`. Default value: `'no'`.	Specifies whether the *external URL* 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-imagemap-links`	Allowed values are: `'yes'` and `'no'`. Default value: `'yes'`.	Specifies whether a numbered list should be generated for an `<imagemap>` element, with one list item per `<area>` element. A list item contains the link specified by the `<area>` element. No list items are generated for “dead areas” (`<area>` elements specifying no link at all).
`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 *internal link target* 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: `Introduction [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.	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

Parameter	Value	Description
	Default value: `'&#x2022; &#x2013;'` (BULLET, EN DASH).	contained in a `<ul>`/`<li>` element and "+" will be used for `<ul>`/`<li>` elements nested in two `<ul>`/`<li>` elements. 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: `'&#x2022;' (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`.
`watermark`	Allowed values are one or more of `'blank'`, `'title'`, `'toc'`, `'booklist'`, `'frontmatter'`, `'body'`, `'backmatter'`, `'index'`, `'all'` separated by whitespace. Default value: `'all'`.	Specifies which pages in the output document are to be given a watermark. By default, all pages are given a watermark. If for example, parameter `watermark` is set to `'frontmatter body backmatter'`, then only the pages which are part of the front matter, body and back matter of the output document are given a watermark. The title page, TOC pages, etc, are not given a watermark. No effect unless parameter `watermark-image` is specified.
`xfc-render-as-table`	A string containing zero or more DITA element names separated by whitespace. Default value: `'note'`.	Specifies whether XMLmind XSL-FO Converter should render the `<fo:block>`s representing specified DITA elements as `<fo:table>`s. This parameter enables a workaround for a limitation of XMLmind XSL-FO Converter: a `<fo:block>` having a border and/or background color and containing several other blocks, lists or tables is very poorly rendered in RTF, WML, DOCX and ODT.

Figure 4-1. Page areas

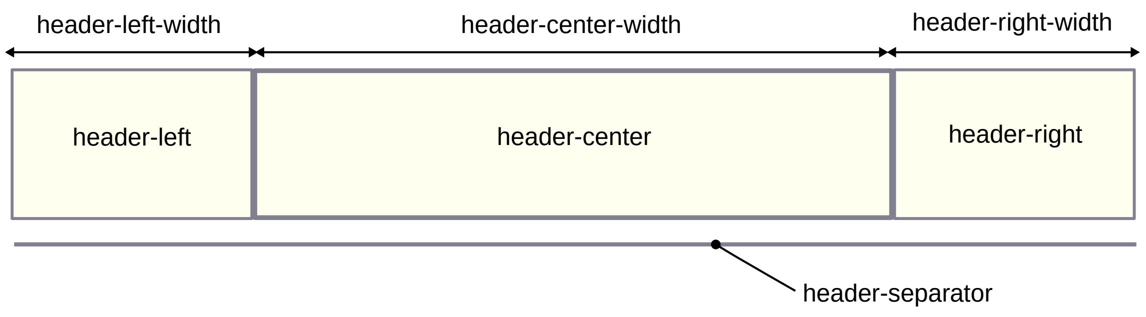


1. Page headers and footers

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.

: The value of the variable comes from the last `created` or `revised` element found in the `topicmeta/critdates` or `bookmeta/critdates` element of the map. More precisely, it comes the value of attribute `golive`, `modified` or `date`, considered in that order. The value of this attribute must be something like *YYYY-MM-DD*, because it is parsed and then formatted according to the `xml:lang` of the map. For example, if `golive="2014-02-23"`, with `xml:lang="en"`, it gives: "`February 02, 2014`" and with `xml:lang="fr"`, it gives: "`02 Février 2014`". If the map has no `critdates` element, then the current date is used. If the value of attribute `golive`, `modified` or `date` is not specified as *YYYY-MM-DD*, then this value is used as is.

`{{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}}`[7].

Restriction

If you use XMLmind XSL-FO Converter to convert your DITA document to RTF, WML, DOCX or ODT, then `{{section1-title}}` and `{{topic-title}}` won't work. (`{{chapter-title}}` works fine though.) The reason of this limitation is that, unlike other XSL-FO processors, XMLmind XSL-FO Converter does not implement `<fo:marker>` and `<fo:retrieve-marker>`.

[7] The `{{topic-title}}` of a `toc` is "**Table of Contents**", properly localized. The `{{topic-title}}` of a `indexlist` is "**Index**", properly localized.

`{{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).

`{{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, the only possible page side is `odd`. The other values, `first` and `even`, are not supported. For example, something like `'one-side chapter||appendix even:: {{chapter-title}};;'` cannot generate any text.

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. Controlling the numbering of ordered lists

This chapter explains how you can to control the numbering of ordered lists by the means of one or more directives specified in the `@outputclass` attribute of the `<ol>` element.

By default, the numbering of nested ordered lists automatically alternates between the "`1.`" and "`a.`" formats. If you want more control on the numbering of ordered lists, then you'll have to specify one or more of the following directives in the `@outputclass` attribute of the `<ol>` element.

`lower-alpha`
`upper-alpha`
`lower-roman`
`upper-roman`
`decimal`

Specifies the style of numbering.

`start(`*`positive_integer`*`)`

Numbering begins at specified *positive_integer*.

`continue`

Numbering begins where the preceding ordered list left off.

`inheritnum`

Numbering inherits from outer-level ordered lists. For example, using this feature (e.g. `<ol outputclass="upper-alpha inheritnum">`), the items of a list nested at level 2 are labelled "`1.A.`", "`1.B.`", "`1.C.`", etc.

Restriction

- When using XMLmind XSL-FO Converter v6.0+[8] to generate RTF, WML, DOCX or ODT, directive `inheritnum` is supported but with minor limitations. For example, when both `start(`*`positive_integer`*`)` and `inheritnum` are specified, `inheritnum` is not honored.
- Directive `inheritnum` is not honored when generating Java Help and HTML Help.

Example: `<ol outputclass="upper-roman start(10)">` specifies an ordered list which starts with an "`X.`".

Note that it is still possible to specify any class name you want in the `@outputclass` attribute of the `<ol>` element. Example: `<ol outputclass="continue fancy-list">`.

[8] v6.1+ strongly recommended.

Chapter 6. Giving a background color to table cells

This chapter explains how you can give a background color to table cells by adding a `bgcolor(`*`color`*`)` directive to the `@outputclass` attribute of most table elements.

It's possible to give a background color to table cells by adding a `bgcolor(`*`color`*`)` directive, where *color* is any CSS color value, to the `@outputclass` attribute of the following elements:

Inside a `<simpletable>` element
: `<simpletable>`, `<sthead>`, `<strow>`, `<stentry>`.

Inside a `<table>` element
: `<tgroup>`, `<thead>`, `<tbody>`, `<row>`, `<entry>`.

Example:

```
<table>
  <tgroup cols="2" outputclass="bgcolor(#F0FFFF)">
    <tbody>
      <row>
        <entry>C1,1</entry>
        <entry>C1,2</entry>
      </row>
      <row outputclass="bgcolor(#FFFFF0)">
        <entry>C2,1</entry>
        <entry>C2,2</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

Note that it is still possible to specify any class name you want in the `@outputclass` attribute of a table element. Example: `<simpletable outputclass="bgcolor(#FFFFF0) fancy-table">`.

Chapter 7. Fancy code blocks

This section explains how you can automatically add line numbers, “expand” tab characters and colorize the source code contained in `<pre>`, `<codeblock>` or any other element specializing `<pre>`.

Adding line numbers, “expanding” tab characters and colorizing the source code contained in `<pre>`, `<codeblock>` or any other element specializing `<pre>` is done by adding one or more of the following classes to the `@outputclass` attribute of this element:

`line-numbers`
`line-numbers-`*`N`* (where *N* is an integer > 0)
`show-line-numbers`

Give a number to the lines contained in the `<pre>` element.

By default, first line number is 1. This first line may be specified using the second form of the `line-numbers` class, for example, `line-numbers-100` specifies that lines are to be numbered and that first line number is 100.

`show-line-numbers`, an alias for `line-numbers-1`, is also accepted for compatibility with the DITA-OT.

`language-`*`L`* (where *L* is language name)

Colorize the source code contained in the `<pre>` element. *L*, a “programming language” such as `c`, `java`, `css`, `xml`, specifies how the source code should be colorized.

More information about this feature, commonly called *syntax highlighting*, in next section.

`tab-width-`*`W`* (*where W* is an integer >= 0)
`normalize-space`

Specifies whether tab characters should be expanded to a number of space characters. *W* is the maximum number of space characters for an expanded tab character, hence this value specifies the location of “tab stops”. Examples: `tab-width-4` means: expand tabs to up to 4 space characters; `tab-width-0` means: do not replace tabs by space characters.

In addition to replacing tab characters by a number of space characters, `tab-width-`*`W`* (where *W* > 0) also removes the space characters which are common to the beginning of all text lines. That is, it removes the superfluous “indentation” in the `<pre>` element, if any. See example below.

Moreover `tab-width-`*`W`* (where *W* > 0) also removes the (useless) space characters found just before newline characters.

`normalize-space`, an alias for `tab-width-8`, is also accepted for compatibility with the DITA-OT.

Remember

When the `<outputclass>` attribute of any element specializing `<pre>` contains class `line-numbers`/`line-numbers-`*`N`* and/or class `language-`*`L`* , then class `tab-width-8` is implicitly specified too, that is, whitespace normalization is automatically performed. If this is not what you want, please explicitly add class `tab-width-0` to `@outputclass`.

Example: a simple C program featuring line numbering and syntax highlighting

In the following C program, lines are indented using tab characters.

```
<pre class="language-c line-numbers tab-width-4">/* Hello World */
#include &lt;stdio.h&ght;

int main()
{
        printf("Hello World\n");
        return 0;
}</pre>
```

is rendered as:

```
/* Hello World */
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Example: superfluous indentation is removed by `tab-width-`*N* (where *N* > 0)

Attribute `@outputclass` implicitly also contains `tab-width-8`. First line "`    /tmp/`" starts with 4 space characters.

```
<pre outputclass="line-numbers">    /tmp/
    /usr/
        bin/
        lib/
        <b>local/</b>
                <b>bin/</b>
                <b>lib/</b>
                <b>src/</b>
        src/
    /var/
</pre>
```

is rendered as:

```
/tmp/
/usr/
    bin/
    lib/
    local/
            bin/
            lib/
            src/
    src/
/var/

```

1. Syntax highlighting

This section 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-bourne` (or `-shell` or `-sh`), `language-c`, `language-cmake` (or `-make` or `-makefile`), `language-cpp`, `language-csharp`, `language-css21` (or `-css`), `language-delphi`, `language-ini`, `language-java`, `language-javascript`, `language-lua`, `language-m2` (Modula 2), `language-perl`, `language-php`, `language-python`, `language-ruby`, `language-sql1999`, `language-sql2003`, `language-sql92` (or `-sql`), `language-tcl`, `language-upc` (Unified Parallel C), `language-html`, `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: customization of the syntax highlighting of a keyword for HTML-based output formats

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

How to use a custom CSS stylesheet is explained in Part II, Chapter 9, 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-set>`s: `hl-keyword`, `hl-string`, `hl-number`, `hl-comment`, `hl-doccomment`, `hl-directive`, `hl-annotation`, `hl-tag`, `hl-attribute`, `hl-value`, `hl-doctype`.

Example: customization of the syntax highlighting of a keyword for XSL-FO-based output formats

```
<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>
```

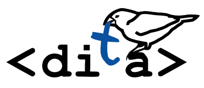
How to use a custom XSLT stylesheet generating XSL-FO is explained in Part II, Chapter 9, Section 2.

Chapter 8. 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 document either by reference or by inclusion. Use an `<svg-container>`/`<svgref>` element pointing to an SVG file to include it by reference. Example:



The XML source code corresponding to the above example is:

```
<p><svg-container><svgref href="media/graphic.svg"/></svg-container></p>
```

It's also possible to use an `<image>` element pointing to an SVG file to include it by reference. Example:

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

Embedding SVG graphics in a DITA document can be achieved using the same `<svg-container>` element. Example:

The XML source code corresponding to the above example is:

```
<p><svg-container>
  <svg:svg height="64.710144" version="1.1"
    viewBox="0 0 104.28986 51.768115" width="130.36232"
    xmlns:svg="http://www.w3.org/2000/svg">
  ...
  </svg:svg>
</svg-container></p>
```

Notes:

- It is still recommended to include SVG graphics by reference using the `<image>` element rather then `<svg-container>`/`<svgref>`. The `<image>` element has useful attributes (`@width`, `@height`, `@scale`, `@scale-fit`) allowing to adjust the dimension of the image. Moreover this elements permits on the fly conversion between image formats.
- It is not recommended to embed SVG graphics in a DITA document as this is likely to cause many validation problems.
- 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 (PDF, RTF, DOCX, etc) support SVG whatever the XSL-FO processor you may use.

MathML

It is possible to include math in a DITA document either by reference or by inclusion. Use an `<mathml>`/`<mathmlref>` element pointing to a MathML file to include it by reference. Example:

E =   0

The XML source code corresponding to the above example is:

```
<p><mathml><mathmlref href="media/math.mml"/></mathml></p>
```

Embedding MathML in a DITA document can be achieved using the same `<mathml>` element. Example:

{ ×E=- B t ×B=μ0 J+μ0 0   E t

The XML source code corresponding to the above example is:

```
<p><mathml>
  <m:math display="block"
    xmlns:m="http://www.w3.org/1998/Math/MathML">
    <m:row>
    ...
    </m:mrow>
  </m:math>
</mathml></p>
```

Notes:

- For clarity, it is recommended to wrap `<mathml>` into the following equation elements: `<equation-inline>`, `<equation-block>`, `<equation-figure>`.
- There is an option to number `<equation-figure>` elements having a `<title>`. Example:

  Equation 8-1. Gauss's law in its differential form

  E= 0

  `<equation-block>` elements containing a empty `<equation-number>` are automatically numbered. Example:

  E= 0 (8-1)

  The counter used to number to `<equation-figure>` elements having a `<title>` and the counter used to number `<equation-block>` elements containing an empty `<equation-number>` are different. Therefore mixing numbered `<equation-figure>`s and numbered `<equation-block>`s in the same DITA document may result in a hard to understand equation numbering.
- 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(PDF, RTF, DOCX, etc) 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" value="media/audio.ogg"
            valuetype="ref" type="audio/ogg"/>

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

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

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

Notes:

- 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 `<param>` child elements to the `<object>` element. Such `<param>` elements must have a `name="source"` attribute, a `valuetype="ref"` attribute, a `@value` attribute referencing an audio file and preferably, a `@type` attribute specifying the media type of the 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.
- Lightweight DITA has an `<audio>` element, so there is no need to use an `<object>` element. The equivalent of the above `<object>` example would be:

  ```
  <audio>
    <media-controls value="true"/>

    <media-source value="media/audio.mp3"/>
    <media-source value="media/audio.ogg"/>
    <media-source value="media/audio.m4a"/>
    <media-source value="media/audio.wav"/>
  </audio>
  ```

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" value="media/video.ogv"
            valuetype="ref" type='video/ogg; codecs="theora, vorbis"'/>

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

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

Notes:

- 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 `<param>` child elements to the `<object>` element. Such `<param>` elements must have a `name="source"` attribute, a `valuetype="ref"` attribute, a `@value` attribute referencing a video file and preferably, a `@type` attribute specifying the media type of the 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.
- Lightweight DITA has an `<video>` element, so there is no need to use an `<object>` element. The equivalent of the above `<object>` example would be:

  ```
  <video width="320">
    <video-poster value="media/video_poster.jpg"/>

    <media-controls value="true"/>
  ```

```
  <media-source value="media/video.mp4"/>
  <media-source value="media/video.ogv"/>
  <media-source value="media/video.webm"/>
</video>
```

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"
           valuetype="ref" type="application/x-shockwave-flash"/>

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

Notes:

- 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.

Other uses of the `<object>` element

We have seen in previous sections how the `<object>` DITA element may be used to add audio, video and Adobe® Flash® animations to your DITA topics. In any case other than those described in previous sections, the `<object>` DITA element is converted to the equivalent `<object>` XHTML element. For example, if you want to add a YouTube video to your DITA topics, simply do it in DITA as you would do it in XHTML using the `<object>` element.

Watch this test video on YouTube.

The XML source code corresponding to the above example is:

```
<p><object data="https://www.youtube.com/embed/C0DPdy98e4c"
           width="640" height="360">
     <desc><image href="media/youtube_icon.png"/> Watch this <xref format="html"
     href="https://youtu.be/C0DPdy98e4c" scope="external">test video</xref> on
     YouTube.</desc>
</object></p>
```

Notes:

- If the `<object>` element has a `<desc>` child element, then this `<desc>` element is used to generate fallback content in case the media object 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 media 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 8-1. 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 9. Simple customization

1. Customize the look of the (X)HTML pages generated by ditac

We'll explain how to customize the look of the (X)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.

About this task

The easiest way to customize the look of the (X)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/base.css**
   : Used for the XHTML 1.0, XHTML 1.1, HTML 4.01 and XHTML 5 output formats.

   ***ditac_install_dir*/xsl/webhelp/resources/base.css**
   : Used for the Web Help output format.

   ***ditac_install_dir*/xsl/htmlhelp/resources/base.css**
   : Used for the HTML Help output format.

   ***ditac_install_dir*/xsl/eclipsehelp/resources/base.css**
   : Used for the Eclipse Help output format.

   ***ditac_install_dir*/xsl/javahelp/resources/base.css**
   : Used for the Java™ Help output format.

   ***ditac_install_dir*/xsl/epub/resources/base.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(base.css);
   ```

   Restriction

   Microsoft HTML Help viewer `hh.exe` does not support `@import`. Therefore you must *copy* `base.css` into your custom CSS stylesheet if you generate HTML Help.

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

   The XSLT stylesheets generating (X)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 (X)HTML pages generated by ditac.

```
@import url(base.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: 2px solid #403480;
}
```

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

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

   The above command gives the expected results because:

   1. "`-p xsl-resources-directory res`" copies all stock resources, including `base.css`, to subdirectory `out/manual/res/`.
   2. "`-p custom-css customize/custom.css`" copies `custom.css` to subdirectory `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.

   This stock XSLT stylesheet is found in `*ditac_install_dir*/xsl/fo/fo.xsl`. It is 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.

   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.xsl`" option when running ditac.

```
$ ditac -t customize/custom_fo.xsl \
    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 10. 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 `<time>` and `<kbd>` elements (similar to `<time>` and `<kbd>` HTML5 elements) topic contents. These elements are modeled as follows (see sample_plugin/dtd/sampleDomain.mod)

```
<!ENTITY % time "time">

<!ELEMENT time (#PCDATA | %text;)*>

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

<!ATTLIST time %global-atts;
               class CDATA "+ topic/ph sample-d/time ">

<!ATTLIST time %univ-atts;
               datetime CDATA #IMPLIED>

<!ENTITY % kbd "kbd">

<!ELEMENT kbd (#PCDATA | %text;)*>

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

<!ATTLIST kbd %global-atts;
               class CDATA "+ topic/ph sample-d/kbd ">
```

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 9, Section 1 and Chapter 9, 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/ditabase.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 9, 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:param name="cssResourceName" select="'xhtml.css'"/>

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

  <xsl:template match="*[contains(@class,' sample-d/time ')]">
    <span>
      <xsl:call-template name="commonAttributes"/>
      <xsl:apply-templates/>
    </span>
  </xsl:template>

</xsl:stylesheet>
```

   Note that the XSLT template called `commonAttributes` adds a `class="kbd"` attribute to the generated `<tt>` element. Similarly, it adds a `class="time"` attribute to the `<span>` element generated for the `<time>` element. So how to style the generated `<tt class="kbd">` and `<span class="time">`?

   a. Copy `ditac_install_dir/xsl/xhtml/resources/` and `ditac_install_dir/xsl/xhtml/resources.list` to `your_plugin_dir/xsl/xhtml/`.

   b. Copy the following `xhtml.css` CSS stylesheet to `your_plugin_dir/xsl/xhtml/resources/`.

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

.kbd {
    font-family: monospace;
    font-size: 90%;
    border: 1px solid #C0C0C0;
    border-radius: 3px;
    -moz-border-radius: 3px;
    -webkit-border-radius: 3px;
    padding: 2px;
    background-color: #F0F0F0;
}

.time {
    background-color: #FFFFCC;
    padding: 2px;
}
```

c. Append the following line to *your_plugin_dir*/xsl/xhtml/resources.list.

```
resources/xhtml.css
```

d. Declare that the default CSS stylesheet is xhtml.css and not stock base.css. This is done by using XSLT stylesheet parameter cssResourceName.

```
<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:param name="cssResourceName" select="'xhtml.css'"/>
  ...
```

3. Create a customization of *ditac_install_dir*/xsl/fo/fo.xsl as explained in Chapter 9, 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="kbd" use-attribute-sets="monospace-style">
    <xsl:attribute name="border">1px solid #C0C0C0</xsl:attribute>
    <xsl:attribute name="background-color">#F0F0F0</xsl:attribute>
    <xsl:attribute name="padding">0.25em</xsl:attribute>
  </xsl:attribute-set>

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

  ...
```

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

4. Pass command-line option **-plugin** *plugin_name* to **ditac** in order to use the DTDs (or 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 `<time>` and `<kbd>` elements 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 11. 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.

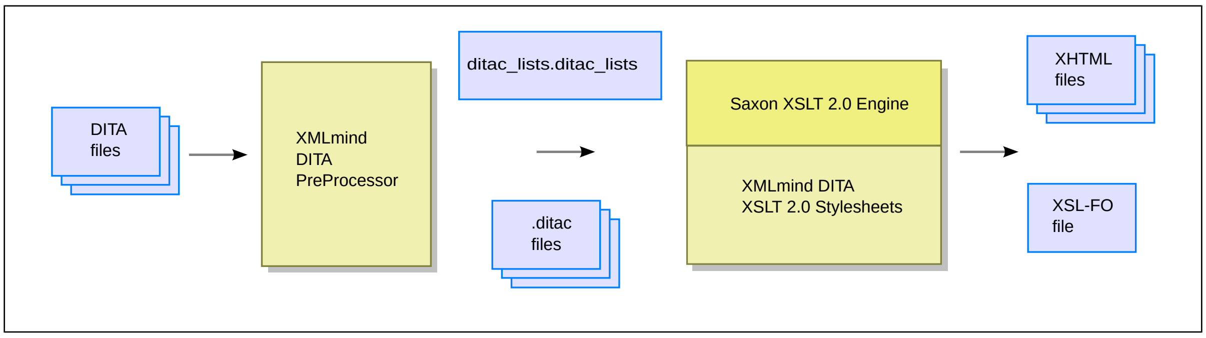
The only difference is that this time, you need to know exactly what is the format of the files you are going to transform. The bad news first: the ditac XSLT 2.0 stylesheets do not transform plain DITA files. They transform `.ditac` files, which are *fully preprocessed DITA files*. Now, the good news: `.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[9] out of the source DITA files.

Figure 11-1. 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
```

[9] 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:equationList>`
: This empty placeholder element means: generate a **List of Equations** 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-block>`
`<ditac:flags-inline>`
: Wrapper elements used to implement *flagging* in ditac. Flagging is specified by the means of a conditional processing profile (a `.ditaval` file). See also the **`-filter`** command-line option.

  - A DITA block element which supports all flagging features(10) is wrapped in a `<ditac:flags-block>` element having flagging attributes such as `@color`, `@text-decoration`, `@change-bar-placement`, etc.
  - A DITA inline element which supports all flagging features(11) is wrapped in a `<ditac:flags-inline>` element having flagging attributes such as `@color`, `@text-decoration`, `@startImage`, etc.
  - Any other DITA element is considered not to support all flagging features and as such, is given *flagging attributes* like `@ditac:flags-color`, `@ditac:flags-background-color`, `@ditac:flags-font-weight`, etc, rather than being wrapped in a `<ditac:flags-block>` or `<ditac:flags-inline>` element.

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

Example:

(10) That is, `<topic>`, `<p>`, `<lq>`, `<note>`, `<dl>`, `<ul>`, `<ol>`, `<sl>`, `<pre>`, `<lines>`, `<fig>`, `<object>`, `<table>`, `<simpletable>`, `<section>`, `<example>` and their specializations.

(11) That is, `<ph>`, `<term>`, `<xref>`, `<cite>`, `<q>`, `<boolean>`, `<state>`, `<keyword>`, `<tm>`, `<image>`, `<foreign>` and their specializations.

```
<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: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>`, `<svgref>`, `<mathmlref>` 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:frontmatterTOC>`
`<ditac:toc>`
`<ditac:backmatterTOC>`

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:equationList>`

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 `<equation-figure>` elements having a `<title>` child element.

Example:

```
<ditac:equationList>
  <ditac:equation file="part222.html" id="part222__first_equation"
                 number="part.2 equation.1" title="First equation">
    <ditac:description>This is a short description of the first equation.
    It should be displayed in the <i class="+ topic/ph hi-d/i ">
    <b class="+ topic/ph hi-d/b ">List of
    Equations</b></i>.</ditac:description>
  </ditac:equation>
  .
  .
  .
  <ditac:equation file="trademarks.html" id="trademarks__I_qa9vmk_"
                 number="equation.4" title="Second equation"/>
</ditac:equationList>
```

`<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_hd1wr_">
      <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_f1jh_" 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**, **List of Equations** 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 12. 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 `Converter`s.

     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("/opt/fop/fop")) {
       return 1;
   }
   ```

   There are several methods which may be used 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.

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, errors messages are displayed on the `Console`.

Environment required for running this kind of embedding

Aside ".jar" files like ditac.jar, resolver.jar, saxon9.jar, etc, which are all listed in *ditac_install_dir*/doc/manual/embed/build.xml (see below), this kind of embedding also needs to access:

- The DITA DTD, schemas and XML catalogs normally found in *ditac_install_dir*/schema/.
- The XSL stylesheets normally found in *ditac_install_dir*/xsl/.

Therefore the requirements for running this kind of embedding are:

1. Use system property *xml.catalog.files* to point to *ditac_install_dir*/schema/catalog.xml or to an equivalent of this XML catalog.

2. Stock *ditac_install_dir*/schema/catalog.xml contains the following entry:

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

   This <rewriteURI> entry is needed to find the location of the directory containing the XSL stylesheets. Make sure that this entry exists in your XML catalogs and that it points to the actual location of the directory containing the XSL stylesheets.

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” like "/opt/fop/fop". This means that, without modifications, 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 13. Low-level method: embedding com.xmlmind.ditac.preprocess.PreProcessor

Chapter 13. 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.a. 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`.
   - Specifiying `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.

   1.b. 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.

   2.a. 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.b. 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 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.

2.c. 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 = AppUtil.getXSLResourceFile("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">
    <java 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" />
  </java>
</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`.
- AppUtil.getXSLResourceFile is a utility function used to locate files found in the XSL directory (normally *`ditac_install_dir`*`/xsl/`).

2.d. 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 = AppUtil.getXSLResourceFile("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);
   ```

Environment required for running this kind of embedding

Aside "`.jar`" files like `ditac.jar`, `resolver.jar`, `saxon9.jar`, etc, which are all listed in `ditac_install_dir/doc/manual/embed/build.xml` (see below), this kind of embedding also needs to access:

- The DITA DTD, schemas and XML catalogs normally found in `ditac_install_dir/schema/`.
- The XSL stylesheets normally found in `ditac_install_dir/xsl/`.

Therefore the requirements for running this kind of embedding are:

1. Use system property `xml.catalog.files` to point to `ditac_install_dir/schema/catalog.xml` or to an equivalent of this XML catalog.

2. Stock `ditac_install_dir/schema/catalog.xml` contains the following entry:

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

   This `<rewriteURI>` entry is needed to find the location of the directory containing the XSL stylesheets. Make sure that this entry exists in your XML catalogs and that it points to the actual location of the directory containing the XSL stylesheets.

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`.

Related information

• Part II, Chapter 11. Extensive customization
• Chapter 12. High-level method: embedding `com.xmlmind.ditac.convert.Converter`

Appendix A. About DITA support in XMLmind DITA Converter

DITA 1.3 support

As of version 3.0, XMLmind DITA Converter (ditac for short) fully supports DITA 1.3 and as such, allows to convert DITA documents conforming to the DITA 1.3 DTD, W3C XML Schema or *RELAX NG schema*. However, there are still limitations, deemed minor, and implementation specificities which are documented in Appendix C.

In fact, when ditac v2.6+ is used, DITA 1.2 documents are automatically “upgraded” to DITA 1.3. This is caused by the fact that the following `<!DOCTYPE>` means "use latest version of the DITA DTD":

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
<topic id="MyTopic">
...
</topic>
```

This should not be a problem as DITA 1.3 is a superset of DITA 1.2.

Technical content only

Ditac only supports "Technical content elements". However Classification elements (e.g. subject scheme maps) are still not supported.

DITA 1.3 RELAX NG schema

Ditac has no problem processing a DITA document pointing to a RELAX NG schema, rather than to a DTD or W3C XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:topic.rng"?>
<topic id="MyTopic">
...
</topic>
```

The `<?xml-model?>` processing-instruction used in the above example is the standard way to associate a document to a RELAX NG schema. See "*Associating Schemas with XML documents 1.0*".

The `DTDToSchema` facility

The **`DTDToSchema`** facility can be used to “upgrade” your documents conforming to a DITA 1.3 DTD to the equivalent DITA 1.3 W3C XML Schema or RELAX NG schema. Command-line example showing how to invoke the **`DTDToSchema`** facility:

```
$ java -cp ditac_install_dir/lib/ditac.jar com.xmlmind.ditac.tool.DTDToSchema¬
-rng MyTopic.dita
```

Before invoking the **`DTDToSchema`** facility, `MyTopic.dita` contained:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
<topic id="MyTopic">
...
</topic>
```

After invoking the **`DTDToSchema`** facility, `MyTopic.dita` contains:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:topic.rng"?>
<topic id="MyTopic">
...
</topic>
```

The **`DTDToSchema`** facility is auto-documented:

```
$ java -cp ditac_install_dir/lib/ditac.jar com.xmlmind.ditac.tool.DTDToSchema

Usage: java -cp ditac.jar com.xmlmind.ditac.tool.DTDToSchema
  -rng|-xsd [ in_dita_file|in_dir_containing_dita_files ]+

"Upgrades" specified DITA documents conforming to a standard
DITA 1.3 DTD to the corresponding W3C XML schema or
RELAX NG schema.

Processes files or directories. Files are modified in place.
Directories are recursively processed. All the '.ditamap', '.dita'
and '.ditaval' files found in specified directories are processed.

Options:
-rng Upgrade to RELAX NG schema.
-xsd Upgrade to W3C XML schema.
```

Related information

- Appendix C. Limitations and implementation specificities

Appendix B. Lightweight DITA support

XMLmind DITA Converter fully supports Lightweight DITA (AKA LwDITA) support, whether XDITA (very small subset of DITA XML, plus new `<audio>` and `<video>` elements), HDITA (topics and maps written in HTML5) or MDITA Extended Profile (topics and maps written in Markdown).

XMLmind DITA Converter can of course process DITA documents comprising a mix of XDITA, HDITA, MDITA and (full) DITA topics and maps.

You'll find in the following sections some templates to start writing topics and maps in XDITA, HDITA and MDITA. You'll also find the list of HDITA and MDITA implementation specificities and limitations.

1. XDITA support

XMLmind DITA Converter fully supports XDITA, which is basically a very small subset of DITA XML, plus new audio and video elements.

Template of an XDITA topic (lwdita_templates/xdita_topic.dita):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Topic//EN"
"lw-topic.dtd">
<topic id="???">
  <title></title>
  <shortdesc></shortdesc>
  <body>
    <p></p>
  </body>
</topic>
```

Template of an XDITA map (lwdita_templates/xdita_map.dita):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD LIGHTWEIGHT DITA Map//EN"
"lw-map.dtd">
<map>
  <topicmeta>
    <navtitle></navtitle>
  </topicmeta>
  <topicref href="???">
    <topicref href="???"/>
  </topicref>
  <topicref href="???"/>
</map>
```

2. HDITA support

XMLmind DITA Converter fully supports HDITA, which specifies how to write DITA topics and maps in HTML5.

Important

Only *XHTML5*, that is, the XML syntax of HTML5, is supported. Plain HTML5 is *not* supported.

In practice, this means that all tags must be closed (e.g. `<img/>` and not `<img>`), all attributes must have a quoted value (e.g. `controls=""` and not `controls`) and that elements like `head` and `body` may not be omitted.

Template of an HDITA topic (`lwdita_templates/hdita_topic.html`):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="UTF-8"/>
    <title>Topic title here</title>
  </head>
  <body>
    <article id="???">
      <h1>Topic title here</h1>
      <p>Short description here.</p>
      <p>Topic body starts here.</p>
    </article>
  </body>
</html>
```

Template of an HDITA map (`lwdita_templates/hdita_map.html`):

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="UTF-8"/>
    <title>Map title here</title>
  </head>
  <body>
    <nav>
      <h1>Map title here</h1>
      <ul>
       <li><p><a href="???"></a></p>
         <ul>
           <li><p><a href="???"></a></p></li>
           <li><p><a href="???"></a></p></li>
         </ul>
       </li>
       <li><p><a href="???"></a></p></li>
     </ul>
    </nav>
  </body>
</html>
```

Implementation specificities

- Adding attribute `data-class="concept"` to the top-level `article` element may be used to generate a DITA `concept` rather than a DITA `topic`.
- A footnote may be represented by `<div data-class="fn">`, in addition to `<span data-class="fn">`.

A footnote reference may be represented by `<a href="#FOOTNOTE_ID"></a>`.

- An internal link may be specified as `<a href="#TARGET_ID">`. Notice that there is no need to specify, like in DITA XML, `<a href="#./TARGET_ID">` or `<a href="#TOPIC_ID/TARGET_ID">`.
- Only the `meta` elements having the following names are translated to their DITA equivalent (that is, elements contained in the `prolog` of a DITA `topic` or the `topicmeta` of a DITA `map`):
  - `audience`
  - `author`
  - `category`
  - `created` or `dcterms.created` (maps to `<critdates>/<created>`)
  - `keyword` (maps to `<keywords>/<keyword>`)
  - `permissions`
  - `publisher` or `dcterms.publisher`
  - `resourceid`
  - `revised` or `dcterms.modified` (maps to `<critdates>/<revised>`)
  - `source`

  A `meta` element having any other name is translated to DITA element `data`.
- While most HTML5 elements are faithfully translated to their DITA equivalent,
  - some elements (`br`, `hr`, `iframe`, `script`, etc) are simply ignored;
  - some other elements are translated to a DITA `ph` or `div` having an `outputclass` attribute reflecting their HTML5 origin.

    Example 1: element `small` is translated to DITA `<ph outputclass="role-small">`.

    Example 2: an `h1` element other than the very first one (specifying the title of the topic) is translated to DITA `<div outputclass="role-h1">`.

    Example 3: nested sections are translated to DITA `<div outputclass="role-section">`.

Limitations

- Only *XHTML5*, that is, the XML syntax of HTML5, is supported. Plain HTML5 is *not* supported.
- Using attribute `rowspan` in `td` or `th` elements will generally cause an incorrect DITA table to be generated.

3. MDITA support

XMLmind DITA Converter fully supports MDITA, which specifies how to write DITA topics and maps in Markdown.

Template of an MDITA topic lwdita_templates/mdita_topic.md:

```
---
id: ???
---

# Topic title here

```

```
Short description here.

Topic body starts here.
```

Template of an MDITA map lwdita_templates/mdita_map.md:

```
# Map title here {.map}

- [???](???)
  - [???](???)
  - [???](???)
- [???](???)
```

Notice the `{.map}` `class` attribute added to the title of the map. Without it, the above template would be translated to a DITA `topic`.

Implementation specificities

- The encoding of an MDITA file is, by default, the system encoding (e.g. `window-1252` on a Western PC).

  If you want to explicitly specify the encoding of an MDITA file, please save your file with a `UTF-8` or `UTF-16` BOM (Byte Order Mark) or add an *encoding directive* inside a comment anywhere at the beginning of your file. Example:

  ```
  <!-- -*- coding: iso-8859-1 -*- -->

  Heading
  =======

  ## Sub-heading

  Paragraphs are separated
  by a blank line.
  ```

  The above example should work fine because ditac understands the GNU Emacs file variable called `coding`.

- Adding a `{.concept}` `class` attribute to the title of an MDITA topic may be used to generate a DITA concept rather than a DITA topic.

- Out of the box, ditac supports the so-called *Extended Profile*.

  This Extended Profile may be customized by the means of `-p load.mdita.XXX` parameters. These `load.mdita.XXX` parameters are documented below.

Limitations

- Without a `{.map}` `class` attribute added to the title of an MDITA map, this map is confused with a topic.

`load.mdita.XXX` parameters

Parameter `-p load.mdita.extended-profile true` is implicitely passed to `ditac`. This parameter is simply a shorthand for:

```
-p load.mdita.abbreviation true
-p load.mdita.admonition true
-p load.mdita.attributes true
```

```
-p load.mdita.definition true
-p load.mdita.footnotes true
-p load.mdita.gfm-strikethrough true
-p load.mdita.ins true
-p load.mdita.superscript true
-p load.mdita.tables true
-p load.mdita.typographic true
-p load.mdita.yaml-front-matter true
```

where abbreviation, admonition, attributes, etc, are all *Markdown extensions*, documented in Markdown extensions.

If for example, you don't like the stock Extended Profile and prefer to use a simpler one, plus the `autolink` Markdown extension(12), then pass:

```
-p load.mdita.core-profile true
-p load.mdita.autolink true
```

to `ditac`.

Parameter `-p load.mdita.core-profile true` is simply a shorthand for:

```
-p load.mdita.gfm-strikethrough true
-p load.mdita.superscript true
-p load.mdita.tables true
-p load.mdita.yaml-front-matter true
```

3.1. Markdown extensions

Abbreviations

Converts plain text abbreviations (e.g. IBM) to `<abbr>` elements.

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.abbreviation false` to `ditac`.

Example:

```
The HTML specification is maintained by the W3C.

*[HTML]: Hyper Text Markup Language
*[W3C]: World Wide Web Consortium
```

is converted to:

```
<p>The <keyword>HTML</keyword> specification is maintained
by  the <keyword>W3C</keyword>.</p>
```

which is rendered as:

The HTML specification is maintained by the W3C.

Admonitions

Syntax for creating admonitions such as notes, tips, warnings, etc.

(12) Turns plain text URLs and email addresses into `<xref href="...">` elements.

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.admonition false` to `ditac`.

After the "`!!!`" tag, the admonition type must be one of "`note`", "`attention`","`caution`", "`danger`", "`fastpath`", "`important`", "`notice`", "`remember`", "`restriction`", "`tip`","`trouble`", "`warning`".

A note example not having a title:

```
!!! note ""
    Support is limited to bug reports.
```

is converted to:

```
<note>
  <p>Support is limited to bug reports.</p>
</note>
```

which is rendered as:

Note

Support is limited to bug reports.

A tip example having a title:

```
!!! tip "How do you do a hard reboot on an iPad?"
    Press and hold both the **Home** and **Power** buttons
    until your iPad&reg; reboots.

    You can release both buttons when you see Apple&reg; logo.
```

is converted to:

```
<note type="tip">
  <div outputclass="note-title role-h4">How do you
  do a hard reboot on an iPad?</div>

  <p>Press and hold both the <b>Home</b> and <b>Power</b>
  buttons until your iPad® reboots.</p>

  <p>You can release both buttons when you see
  Apple® logo.</p>
</note>
```

which is rendered as:

Tip

How do you do a hard reboot on an iPad?

Press and hold both the **Home** and **Power** buttons until your iPad® reboots.

You can release both buttons when you see Apple® logo.

Attributes

Syntax for adding attributes to the generated HTML elements:

```
attributes -> '{' attribute_spec ( S attribute_spec)* '}'

attribute_spec ->   name=value
                  | name='value'
                  | name="value"
                  | #id
                  |.class
```

Remember

An easy rule to remember

If an `{...}` specification is separated by space characters from some plain text (e.g. `some plain text {...}`) then the attributes are added to the parent element of the text.

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.attributes false` to `ditac`.

Example:

```
The *circumference { .first-term }* is the length of one circuit along the
circle, or the distance around the circle. {#circumference}
```

is converted to:

```
<p id="circumference">The <i outputclass="first-term">circumference</i>
is the length of one circuit along the circle, or the distance around
the circle.</p>
```

which is rendered as:

The *circumference* is the length of one circuit along the circle, or the distance around the circle.

Attention

Pitfall

By default, heading IDs are not “rendered” in HTML (which is somewhat counterintuitive). You must pass `-p load.mdita.renderer.RENDER_HEADER_ID true` to `ditac` get them “rendered”.

Automatic links

Turns plain text URLs and email addresses into `<a href="...">` elements.

This Markdown syntax extension is disabled by default. In order to enable it, pass parameter `-p load.mdita.autolink true` to `ditac`.

Example:

```
Please send your bug reports to support@xmlmind.com, a public,
moderated, mailing list. More information in https://xmlmind.com/.
```

is converted to:

```
<p>Please send your bug reports to <xref
href="mailto:support@xmlmind.com">support@xmlmind.com</xref>,
a public, moderated, mailing list. More information in <xref
href="https://xmlmind.com/">https://xmlmind.com/</xref>.</p>
```

which is rendered as:

Please send your bug reports to support@xmlmind.com, a public, moderated, mailing list. More information in https://www.xmlmind.com/.

Definition lists

Syntax for creating definition lists, that is `<dl>`, `<dt>` and `<dd>` elements.

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.definition false` to `ditac`.

Example:

```
Glossary:

LED
: Light emitting diode.

ABS
: Antilock braking system.

ESC
ESP
: Electronic stability control, also known as Electronic Stability Program.

: On motorcycles, ESC/ESP is called *Traction Control*.

  > Ducati was one of the first to introduce a true competition-level
  > traction control system (**DTC**) on a production motorcycle.

EBA
: Emergency brake assist.
```

is converted to:

```
<p>Glossary:</p>
<dl>
   <dlentry>
     <dt>LED</dt>
     <dd>
       <p>Light emitting diode.</p>
     </dd>
   </dlentry>
   <dlentry>
```

```
      <dt>ABS</dt>
      <dd>
        <p>Antilock braking system.</p>
      </dd>
    </dlentry>
    <dlentry>
      <dt>ESC</dt>
      <dt>ESP</dt>
      <dd>
        <p>Electronic stability control, also known as
        Electronic Stability Program.</p>
      </dd>
      <dd>
        <p>On motorcycles, ESC/ESP is called <i>Traction Control</i>.</p>
        <lq>
          <p>Ducati was one of the first to introduce a
          true competition-level traction control system
          (<b>DTC</b>) on a production motorcycle.</p>
        </lq>
      </dd>
    </dlentry>
    <dlentry>
      <dt>EBA</dt>
      <dd>
        <p>Emergency brake assist.</p>
      </dd>
    </dlentry>
  </dl>
```

which is rendered as:

Glossary:

LED

Light emitting diode.

ABS

Antilock braking system.

ESC
ESP

Electronic stability control, also known as Electronic Stability Program.

On motorcycles, ESC/ESP is called *Traction Control*.

> Ducati was one of the first to introduce a true competition-level traction control system (**DTC**) on a production motorcycle.

EBA

Emergency brake assist.

Remember

Remember that:

- The leading ":" character of a definition must be followed by one or more space characters.
- Terms must be separated from the previous definition by a blank line.
- A blank line is not allowed between two consecutive terms.
- A blank line is allowed before a definition.

Footnotes

Syntax for creating footnotes and footnote references.

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.footnotes false` to `ditac`.

Example:

```
The differences between the programming languages C++[^1] and Java can be
traced to their heritage.

[^1]: The C++ Programming Language by Bjarne Stroustrup.

C++[^1] was designed for systems and applications programming, extending the
procedural programming language C[^2].

[^2]: The C Programming Language by Brian Kernighan and Dennis Ritchie.

      Originally published in 1978.
```

is converted to:

```
<p>The differences between the programming languages
C++<xref href="#./__FN1" type="fn"/> and Java can
be traced to their heritage.</p>

<div>
  <fn id="__FN1">The C++ Programming Language by
  Bjarne Stroustrup.</fn>
</div>

<p>C++<xref href="#./__FN1" type="fn"/> was designed
for systems and applications programming, extending
the procedural programming
language C<xref href="#./__FN2" type="fn"/>.</p>

<div>
  <fn id="__FN2">The C Programming Language by
  Brian Kernighan and Dennis Ritchie.
  <p>Originally published in 1978.</p> </fn>
</div>
```

which is rendered as:

The differences between the programming languages C++[13] and Java can be traced to their heritage.

C++[13] was designed for systems and applications programming, extending the procedural programming language C[14].

Strikethrough and subscript

Converts

- tagged text "`~~something deleted~~`" to `<del>something deleted</del>`, which is rendered as: something deleted
- tagged text "`~a subscript~`" to `<sub>a subscript<sub/>`, which is rendered as: $_{\text{a subscript}}$

This Markdown syntax extension, which is part of the MDITA Core Profile and the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.gfm-strikethrough false` to `ditac`.

Ins

Converts tagged text "`++something new++`" to `<ins>something new</ins>`, which is rendered as: something new

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.ins false` to `ditac`.

Superscript

Converts tagged text "`^a superscript^`" to `<sup>a superscript</sup>`, which is rendered as: $^{\text{a superscript}}$

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.superscript false` to `ditac`.

Media tags

Converts prefixed links to audio and video HTML5 elements.

- `!A[Text](links)` - audio. *Links* is one or more links separated by character “`|`”.
- `!V[Text](links)` - video.

Audio example:

```
Audio example: !A[Sample audio](media/audio.mp3|media/audio.wav).
```

is converted to:

```
<p>Audio example: <audio>
  <desc>Sample audio</desc>
  <media-controls value="true"/>
  <media-source value="media/audio.mp3"/>
```

[13] The C++ Programming Language by Bjarne Stroustrup.

[14] The C Programming Language by Brian Kernighan and Dennis Ritchie.

Originally published in 1978.

```
  <media-source value="media/audio.wav"/>
</audio>.</p>
```

Video example:

```
Video example: !V[Sample video](media/video.mp4).
```

is converted to:

```
<p>Video example: <video>
  <desc>Sample video</desc>
  <media-controls value="true"/>
  <media-source value="media/video.mp4"/>
</video>.</p>
```

This Markdown syntax extension is disabled by default. In order to enable it, pass parameter `-p load.mdita.media-tags true` to `ditac`.

Restriction

This extension cannot be used if you are authoring a DITA document. It will only work if you are authoring an ***LwDITA*** document. Only LwDITA supports the `<audio>` and `<video>` elements.

Tables

Converts pipe "`|`" delimited text to `<table>` elements.

This Markdown syntax extension, which is part of the MDITA Core Profile and the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.tables false` to `ditac`.

Simple table example:

```
| Header 1 | Header 2 | Header 3 |
| -------- | -------- | -------- |
| Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,1 | Cell 2,2 | Cell 2,3 |
```

is converted to:

```
<table >
  <tgroup cols="3">
    <thead>
      <row valign="middle">
        <entry align="center">Header 1</entry>
        <entry align="center">Header 2</entry>
        <entry align="center">Header 3</entry>
      </row>
    </thead>
    <tbody>
      <row valign="middle">
        <entry>Cell 1,1</entry>
        <entry>Cell 1,2</entry>
        <entry>Cell 1,3</entry>
      </row>
      <row valign="middle">
        <entry>Cell 2,1</entry>
        <entry>Cell 2,2</entry>
        <entry>Cell 2,3</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

which is rendered as:

Header 1	Header 2	Header 3
Cell 1,1	Cell 1,2	Cell 1,3
Cell 2,1	Cell 2,2	Cell 2,3

Table example having centered and right-aligned columns:

```
| Header 1 | Header 2         | Table Header 3 |
| -------- | :--------------: | -------------: |
| Cell 1,1 | Table cell 1,2   | Cell 1,3       |
| Cell 2,1 | Cell 2,2         | Cell 2,3       |
```

is converted to:

```
<table>
  <tgroup cols="3">
    <thead>
      <row valign="middle">
        <entry align="center">Header 1</entry>
        <entry align="center">Header 2</entry>
        <entry align="right">Table Header
        3</entry>
      </row>
    </thead>
    <tbody>
      <row valign="middle">
        <entry>Cell 1,1</entry>
        <entry align="center">Table cell
        1,2</entry>
        <entry align="right">Cell 1,3</entry>
      </row>
      <row valign="middle">
        <entry>Cell 2,1</entry>
        <entry align="center">Cell 2,2</entry>
        <entry align="right">Cell 2,3</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

which is rendered as:

Header 1	Header 2	Table Header 3
Cell 1,1	Table cell 1,2	Cell 1,3
Cell 2,1	Cell 2,2	Cell 2,3

Table example having cells spanning several columns and a caption:

```
| Header 1 | Header 2 | Header 3 |
| -------- | -------- | -------- |
| Cell 1,1 + 1,2     || Cell 1,3 |
| Cell 2,1 + 2,2 + 2,3         |||
| Cell 3,1 | Cell 3,2 | Cell 3,3 |
[Table caption here]
```

is converted to:

```
<table>
  <title>Table caption here</title>
  <tgroup cols="3">
    <colspec colname="c1" rowheader="headers"/>
    <colspec colname="c2" rowheader="headers"/>
    <colspec colname="c3" rowheader="headers"/>
    <thead>
      <row valign="middle">
        <entry align="center">Header 1</entry>
        <entry align="center">Header 2</entry>
        <entry align="center">Header 3</entry>
      </row>
    </thead>
    <tbody>
      <row valign="middle">
        <entry nameend="c2" namest="c1">Cell 1,1
        + 1,2</entry>
        <entry>Cell 1,3</entry>
      </row>
      <row valign="middle">
        <entry nameend="c3" namest="c1">Cell 2,1
        + 2,2 + 2,3</entry>
      </row>
      <row valign="middle">
        <entry>Cell 3,1</entry>
        <entry>Cell 3,2</entry>
        <entry>Cell 3,3</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

which is rendered as:

Table B-1. Table caption here

Header 1	Header 2	Header 3
Cell 1,1 + 1,2		Cell 1,3
Cell 2,1 + 2,2 + 2,3		
Cell 3,1	Cell 3,2	Cell 3,3

Typographic characters

Converts

- `"'"` to apostrophe `&rsquo;`, which is rendered as in word: "don’t"
- `"..."` and `". . ."` to ellipsis `&hellip;`, which are both rendered as: …
- `"--"` to en dash `&ndash;`, which is rendered as: –
- `"---"` to em dash `&mdash;`, which is rendered as: —
- single quoted `'some text'` to `&lsquo;some text&rsquo;`, which is rendered as: ‘some text’
- double quoted `"some text"` to `&ldquo;some text&rdquo;`, which is rendered as: “some text”
- double angle quoted `<<some text>>` to `&laquo;some text&raquo;`, which is rendered as: «some text»

This Markdown syntax extension, which is part of the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.typographic false` to `ditac`.

If you don’t want some of the above plain text sequences to be processed, specify:

`-p load.mdita.typographic.ENABLE_QUOTES false`
: Do not process single quotes, double quotes, double angle quotes.

`-p load.mdita.typographic.ENABLE_SMARTS false`
: Do not process `"'"`, `"..."`, `". . ."`, `"--"`, `"---"`.

YAML front matter

Syntax for adding metadata to the generated DITA topic or map, that is, for populating the `<prolog>` element of a DITA topic and the `<topicmeta>` element of a DITA map.

These metadata are specified by key/value pairs written using a subset of the YAML (see also http://yaml.org/) syntax.

Supported metadata are:

- `audience`
- `author`
- `category`
- `created` (maps to `<critdates>/<created>`)
- `keyword` (maps to `<keywords>/<keyword>`)
- `permissions`
- `publisher`
- `resourceid`
- `revised` (maps to `<critdates>/<revised>`)

- `source`

Any other metadata is translated to DITA element `<data>`.

This Markdown syntax extension, which is part of the MDITA Core Profile and the MDITA Extended Profile, is enabled by default. In order to disable it, pass parameter `-p load.mdita.yaml-front-matter false` to `ditac`.

Example:

```
---
author:
  - Brian W. Kernighan
  - Dennis Ritchie
publisher: Prentice Hall
created: 1978/01/01
revised: 1988/01/01
---
```

is converted to:

```
<prolog>
  <author>Brian W. Kernighan</author>
  <author>Dennis Ritchie</author>
  <publisher>Prentice Hall</publisher>
  <critdates>
    <created date="1978/01/01"/>
    <revised modified="1988/01/01"/>
  </critdates>
</prolog>
```

Other extensions

The following Markdown syntax extensions are also supported:

- anchorlink
- aside
- emoji
- enumerated-reference
- gfm-issues
- gfm-tasklist
- gfm-users
- toc
- wikilink
- youtube-embedded

All the above extensions are disabled by default. In order to enable an extension, pass parameter `-p load.mdita.EXTENSION_NAME true` to `ditac`. For example: `-p load.mdita.emoji true`

Any extension listed in this section may be parameterized by passing parameter `-p load.mdita.EXTENSION_NAME.PARAMETER_NAME PARAMETER_VALUE`[15] to `ditac`. Examples:

- `-p load.mdita.emoji.ATTR_IMAGE_SIZE 16`
- `-p load.mdita.emoji.ATTR_ALIGN ""`
- `-p load.mdita.emoji.USE_IMAGE_TYPE IMAGE_ONLY`
- `-p load.mdita.emoji.ROOT_IMAGE_PATH https://www.webpagefx.com/tools/emoji-cheat-sheet/graphics/emojis/`

More generally, the Markdown parser (pseudo *EXTENSION_NAME* is "`parser`") and the HTML renderer (pseudo *EXTENSION_NAME* is "`renderer`") may also be parameterized this way. For example, automatically generate an ID for all headings not already having an ID **and** “render” all heading IDs in HTML[16]:

```
-p load.mdita.renderer.GENERATE_HEADER_ID true
-p load.mdita.renderer.RENDER_HEADER_ID true
```

More information about extensions and their parameters in Extensions (flexmark-java is the software component used by ditac to parse Markdown and convert it to HTML).

[15] The only types supported for *PARAMETER_VALUE* are: string, boolean (`true` or `false`), integer and any enumerated type.

[16] By default, heading IDs are not “rendered” in HTML, which is somewhat counterintuitive.

Appendix C. Limitations and implementation specificities

Conversion to XHTML and XSL-FO

- The following elements are ignored:
  - the `<syntaxdiagram>` element and all its descendant elements;
  - `<ux-window>`;
  - `<sort-as>`.
- The `<titlealts>`/`<navtitle>` element of topic is ignored.
- Layout of `<simpletable>` elements:
  - Attribute `@frame` is ignored.
  - Conversion to XHTML:
    - Attribute `@expanse` is partially supported. Its value is considered to always be 100%.
  - Conversion to XSL-FO:
    - Attribute `@expanse` 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:
  - Attribute `<entry>/@rotate` is not supported.
  - Conversion to XHTML:
    - Attribute `<table>/@orient="land"` (landscape table) is not supported.
    - 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 `<table>/@orient="land"` (landscape table) *is* supported. However, except when the XSL-FO processor being invoked by ditac is XMLmind XSL-FO Converter ***v6.2+***, the landscape table must have few enough rows to fit onto one page. When this is not the case, the last rows of the landscape table will simply not appear in the output.
    - 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.
- Ditac supports `<examplelist>` and `<equationlist>` in addition to `<toc>`, `<tablelist>`, `<figurelist>`, `<indexlist>`.
- Contents corresponding to the following empty `<bookmap>` elements: `<trademarklist>`, `<abbrevlist>`, `<bibliolist>`, `<glossarylist>` *cannot* be automatically generated by ditac.
- Entries automatically generated by ditac for `<toc>`, `<tablelist>`, `<figurelist>`, `<examplelist>`, `<equationlist>` and `<indexlist>` only contain plain text. For example, if a topic title is "`<title>The Java<sup>TM</sup> <b>Spring</b> framework</title>`", then the corresponding TOC entry contains "`The JavaTM Spring framework`".
- 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

- 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.
- Key-based, cross-deliverable addressing is not implemented.
- Topics which are not directly or indirectly referenced by the root map are automatically added to the root key scope. Such topics typically contain common content which is included by other topics using `@conref`.

  If you don't want this to happen, please explicitly reference such common content topics in your maps and mark these references as being resource-only. Example:

```
<topigroup keyscope="MyKeycope">
  <topicref href="commonContent.dita" processing-role="resource-only"/>
```

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.
- By default, the character encoding of the text file included using a `<coderef>` element is automatically determined by ditac (e.g. by examining the BOM or `<?xml encoding="XXX"?>`). You may specify this character encoding explicitly by adding a `format="text; charset=XXX"` attribute to the `<coderef>` element. Example: `<coderef format="text; charset=US-ASCII" href="../src/sieve.cpp"/>`.

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 the referenced `<topic>` element.
- Unless `topicref/topicmeta/@lockmeta=no`, `topicref/topicmeta/searchtitle` supplements or overrides `topic/titlealts/searchtitle`.
- 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`.

Subject scheme maps

- XMLmind DITA Converter supports all the features documented in "*2.2.3 Subject scheme maps and their usage*" when these features are related to attribute values.
- Only the following useful subset of the grammar of subject scheme maps is supported by ditac. Any other subject scheme element is silently ignored:

```
<subjectScheme>
  Content: [ subjectdef | enumerationdef | schemeref ]*
</subjectScheme>

<subjectdef
  keys = name of a set of attribute values OR an attribute value
  OR
  keyref = name of a set of attribute values OR an attribute value
  navtitle = description of this subjectdef
>
```

```
  Content: [ <topicmeta>
               <navtitle>description of this subjectdef</navtitle>
             </topicmeta> ]?
           [ subjectdef ]*
</subjectdef>

<enumerationdef>
  Content: [ elementdef ]*
           attributedef
           <subjectdef
             keyref = name of a set of attribute values
                      (keyref absent means:
                       don't use this attribute)
           />
</enumerationdef>

<elementdef
  name = element qualified name
/>

<attributedef
  name = attribute qualified name
/>

<schemeref
  href = location of another subject scheme map
/>
```

- All the subject scheme maps referenced in the map[17] to be converted and in all its submaps are loaded in turn and their contents are merged as if a single subject scheme map was specified at the very beginning of the main map.
- It's also possible to specify which subject scheme map to use by the means of the **`-attrvalues`** and **`-defaultattrvalues`** command-line options.
- Attribute groups are fully supported both for attribute value validation and when filtering and flagging. Example, some of the values declared for attribute `@platform`:

  ```
  <subjectdef keys="macos"/>
  <subjectdef keys="linux">
    <subjectdef keys="redhat"/>
    <subjectdef keys="ubuntu"/>
  </subjectdef>
  ```

  A validation error will be reported for attribute `platform="linux(redhat debian)"` because `"debian"` has not been declared. A validation error will be reported for attribute `platform="macos(redhat)"` because `"redhat"` is not a kind of `"macos"`.

[17] Typically using `<mapref type="scheme" href="my_subject_scheme_map.ditamap"/>`.

Conditional processing

- 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`, `@deliveryTarget` and attributes specialized from `@props`) may be used to filter or flag an element. For example, the `@status` attribute may be used to highlight changes. See below.
- Subject scheme maps, which should be used to validate attribute values and also to implement smarter conditional processing, are currently ignored.
- If a map directly contains multiple `<ditavalref>` elements, all `<ditavalref>` elements but the first one are ignored. When this is the case, a warning is reported, though.
- The externally specified `DITAVAL` file is combined with the `<ditavalref>` element, if any, which is a direct child of a map.
- `<ditavalref>` error conditions are not detected.
- In a DITAVAL file, `action="passthrough"` is not supported.

Flagging contents

- Only the following elements (and, of course, their specializations) can be flagged *without restrictions*: `<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>`.

  Any other element (`<li>`, `<dlentry>`, `<step>`, `<stentry>`, etc) is just given *some* of the colors and font styles, if any, specified by the flagging elements and attributes found in the `.ditaval` file.
- In a `.ditaval` file, attribute `style="double-underline"` is processed as if it were `underline`.
- In a `.ditaval` file, attribute `style="line-through"` is supported in addition to `underline` and `overline`.
- The `@status` attribute may be used to highlight changes. Example:

  ```
  $ ditac -filter status.ditaval doc.pdf doc.ditamap
  ```

  where file `status.ditaval` contains:

  ```
  <val>
    <prop action="flag" att="status" backcolor="#FFFF99" style="underline"
          val="new"/>

    <prop action="flag" att="status" backcolor="#99FF99" val="changed"/>

    <prop action="flag" att="status" backcolor="#FF7F7F" style="line-through"
          val="deleted"/>
  </val>
  ```

  and where `doc.ditamap` references a topic containing for example:

  ```
  <p>A paragraph containing <ph status="new">new text</ph>,
  <ph status="changed">changed text</ph>,
  <ph status="deleted">deleted text</ph>.</p>
  ...
  <p status="new">New paragraph.</p>
  ```

```
...
<ul status="changed">
  <li>First item in changed <tt>ul</tt>.</li>
  <li><p>Second item.</p>
  <p status="deleted">Deleted paragraph.</p></li>
  <li>Third item.</li>
</ul>
```

- In a `.ditaval` file, the value of the `@changebar` attribute of the `<revprop>` element, has the following syntax:

  changebar -> *prop* [S ';' S *prop*]+

  prop -> *prop_name* ':' S *prop_value*

  The style properties supported there are:

Name	Value	Default	Description
`color`	*<color>*	The value of the `color` property.	See XSL 1.1 property `change-bar-color`.
`offset`	*<length>*	`6pt`	See XSL 1.1 property `change-bar-offset`.
`placement`	`start` | `end` | `left` | `right` | `inside` | `outside` | `alternate`	`start`	See XSL 1.1 property `change-bar-placement`.
`style`	*<border-style>*	`solid`	See XSL 1.1 property `change-bar-style`.
`width`	*<border-width>*	`medium`	See XSL 1.1 property `change-bar-width`.

  Example:

```
$ ditac -filter changebar.ditaval doc.pdf doc.ditamap
```

  where file `changebar.ditaval` contains:

```
<val>
  <revprop action="flag" val="2.1"
    changebar="style: double; width: 3px; placement: start;" ></revprop>
</val>
```

  and where `doc.ditamap` references a topic containing for example:

```
...
<fig rev="2.1">
    <title>The logo of ACME corp.</title>
    <image href="acme_logo.png"/>
</fig>
...
```

- Change bars are implemented by the following processors: Apache FOP, RenderX XEP and Antenna House Formatter. For any other XSL-FO processor (e.g. XMLmind XSL-FO Converter) and also for all XHTML-based output formats (e.g. EPUB, Web Help), change bars are emulated using left or right borders. This emulation may give poor results when a change bar is added to a table.

Generating links

- Attribute `@collection-type`, whatever its value, is ignored inside the `<reltable>` element.
- Ditac 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

- `<topicref>` elements found inside a `<reltable>` do not “pull” the corresponding topics. In other words, a `<reltable>` cannot be used to add some content to a deliverable. With ditac, a `<reltable>` is just used to create links between topics which are already part of the deliverable.
- 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 `<topicref>`s, 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>`, `<examplelist>`, `<equationlist>`, `<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*.
- Convenience element `<glossref>` cannot be used with ditac without setting some of its attributes. Example:

```
<glossref href="ONE.dita" keys="key_ONE"/>
```

is strictly equivalent to:

```
<topicref href="ONE.dita" keys="key_ONE" linking="none" print="no"
          toc="no" search="no"/>
```

Notice default attribute `print="no"`. Therefore, when generating PDF, such `<glossref>` is discarded at a very early stage by ditac. The consequence is that each occurrence of `<abbreviated-form keyref="key_ONE"/>` will cause ditac to report a "cannot resolve keyref" warning. The workaround is to simply avoid using `<glossref>` and to stick to `<topicref>` with a `@keys` attribute.

Related information

- Appendix A. About DITA support in XMLmind DITA Converter

Appendix D. 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>
     ...
   ```

   For some languages, like CJK languages, you'll have to insert variable `%{N}` in the localized text corresponding to numbered elements (Chapter, Appendix, Table, Figure, etc). This variable is replaced by the number of the element.

   Japanese example: excerpts of a possible *`ditac_install_dir`*`/xsl/common/messages/ja.xml`:

   ```
   <message name="chapter"> %{N} </message>
   ```

   For the first chapter of the document, this gives " 1 ", which means "The 1 Chapter".

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.

Index