DITA Open Toolkit 2.0

This document describes the DITA Open Toolkit project—what the project is, and how to use the site.

What is the DITA Open Toolkit?

The DITA Open Toolkit, or DITA-OT for short, is a set of Java-based, open source tools that provide processing for DITA maps and topic content. You can download the OT and install it for free on your computer to get started with topic-based writing and publishing. The DITA-OT is licensed under the <u>CPL 1.0</u> and <u>Apache 2.0</u> open source licenses.

Note: While the DITA Standard itself is owned and developed by OASIS, the DITA Open Toolkit is an independent, open source implementation of the standard.

Key output formats for the toolkit include:

- XHTML
- PDF (formerly known as PDF2)
- ODT (Open Document Format)
- Eclipse Help
- TocJS (XHTML with a JavaScript frameset)
- HTML Help
- Java Help
- Eclipse Content (normalized DITA plus Eclipse project files)
- Word RTF (with some limitations)
- Docbook
- Troff

Toolkit documentation

There are two primary sources for documentation about the toolkit.

Stable documentation about toolkit usage, parameters, and project management can be found on this page, using the navigation panel on the left.

New information about the latest toolkit builds, plans for the next release, and other changing information can be found on the DITA-OT landing page at the dita.xml.org site (link below). That site also contains the release notes for all upcoming and previous releases.

Related concepts

Distribution packages

Related information

Main DITA-OT page at dita.xml.org Project News for DITA Open Toolkit

Shortcuts to important information

DITA-OT stable release DITA-OT latest development build

Getting Started with the DITA Open Toolkit

The *Getting Started Guide* is designed to provide a guided exploration of the DITA Open Toolkit. It is geared for an audience that has little or no knowledge of build scripts or DITA-OT parameters. It walks the novice user through installing the full-easy-install version of the toolkit and running a prompted build.

1. <u>Installing the full-easy-install package</u> For the simplest installation experience, install the full-easy-install package. This package can be installed on Linux, Mac OSX, and Windows. It contains everything that you need to run the DITA-OT except for Java.

2. Running the demo build

After you install the full-easy-install package, run the demo build to see the type of output that is produced by the DITA Open Toolkit.

3. <u>Building your own content using the demo build</u> You can use the demo build to generate output for your own DITA content.

Installing the full-easy-install package

For the simplest installation experience, install the full-easy-install package. This package can be installed on Linux, Mac OSX, and Windows. It contains everything that you need to run the DITA-OT except for Java.

Before you begin

- Ensure that you have Java JRE or JDK, version 7 or later installed.
- Ensure that you have HTML Help Workshop installed, if you want to generate HTML Help.

Procedure

1. Download the full-easy package from <u>SourceForge</u>.

Operating system	File name
Linux or Mac OSX	DITA-OT2.0.M3_full_easy_install_bin.tar.gz
Windows	<pre>DITA-OT2.0.M3_full_easy_install_bin.zip</pre>

- 2. Extract the contents of the package to the directory where you want to install the DITA-OT.
- 3. Run the startcmd file that is applicable for your operating system. This defines the necessary environment variables and opens a DOS prompt or terminal window in which you can invoke the toolkit. You can use the window to run as many builds as you want; if you close the window, you will need to run the applicable startcmd file again.

Parent topic: <u>Getting Started with the DITA Open Toolkit</u> Next topic: <u>Running the demo build</u>

Running the demo build

After you install the full-easy-install package, run the demo build to see the type of output that is produced by the DITA Open Toolkit.

Procedure

- 1. Run the startcmd file that is applicable for your operating system. The startcmd.bat and startcmd.sh files are in the directory where you installed the DITA-OT.
- 2. From the DITA-OT shell, enter the following command:

```
ant -f build_demo.xml
```

You receive the following prompt:

```
[echo] Please enter the filename for the DITA map that you
[echo] want to build including the directory path (if any).
[echo] The filename must have the .ditamap extension.
[echo] Note that relative paths that climb (..) are not support
[echo] To build the sample, press return without entering any
[input] The DITA map filename: [C:\DITA-OT2.0.M3\samples\hielder]
```

3. Press Enter.

You receive the following prompt:

```
[echo]
[echo] Please enter the name of the output directory or press
[echo] to accept the default.
[input] The output directory (out): [out]
```

4. Press Enter.

You receive the following prompt:

```
[echo] Please enter the type of output to generate.
[echo] Options include: eclipse, tocjs, htmlhelp, javahelp, |
[echo] Use lowercase letters.
[echo]
[input] The output type: (eclipse, tocjs, htmlhelp, javahelp)
```

5. Press **Enter** to accept the default transformation type: **web**. This will build XHTML files from the DITA source.

You receive the following prompt:

```
[echo] Ready to build C:\DITA-OT2.0.M3\samples\hierarchy.dit;
[echo] for web in out
[echo]
[input] Continue? (Y, [y], N, n)
```

6. Press **Y** or **y** to start the DITA-OT transformation.

The DITA-OT logs information to the command-prompt or terminal window. At the end, you see the following information:

```
prompt.output:
[echo]
[echo] output in the out directory
[echo]
[echo] Before rebuilding, please delete the output or the dir
BUILD SUCCESSFUL Total time: X minutes X seconds
```

7. Go to the out/ directory and open the toc.html file in a Web browser. Figure 1. XHTML output for the sample files

file:///C:/DITA-OT1.6.M5/out/toc.html Garage Tasks • Changing the oil in your car · Organizing the workbench and tools Shovelling snow Spray painting · Taking out the garbage Washing the car Garage Concepts Lawnmower o Oil o Paint o Shelving Snow shovel Tool box Tools Water hose

- Wheel barrow
- Workbench
- Windshield washer fluid

Results

The DITA-OT transformed the hierarchy.ditamap file (located in the samples directory) to XHTML; it wrote the output to the out/ directory.

Parent topic: <u>Getting Started with the DITA Open Toolkit</u> Previous topic: <u>Installing the full-easy-install package</u> Next topic: <u>Building your own content using the demo build</u>

Building your own content using the demo build

You can use the demo build to generate output for your own DITA content.

Procedure

1. If necessary, run the startcmd file that is applicable for your operating system.

You do not need to run the startcmd file if you already have a command-prompt or terminal window that was invoked by the startcmd file open.

2. From the DITA-OT shell, enter the following command:

ant -f build_demo.xml

- 3. When prompted, type the name of a map. You must specify the path for the DITA map. You either can specify a fully qualified file name, for example, C:\DITA-OT1.6.M5\doc\userguide.ditamap, or you can specify a relative path, for example, doc\userguide.ditamap
- 4. When prompted, type the name of the output directory.
- 5. When prompted, type the value for the transformation type.

Ouput format	Value	
Docbook	docbook	
Eclipse help	eclipse	
HTML help	htmlhelp	
PDF	pdf	
XHTML	web	
XHTML with a JavaScript frame set tocjs		

6. When prompted, press **Enter** to start the transformation.

Results

The DITA-OT generates output for the specified DITA content. It runs the transformation that you specified, and writes the output to the directory that you specified.

What to do next

Explore invoking the DITA-OT from either Ant or the command-line tool. This enables you to specify a wider array of parameters than those supported by the demo build.

Using Ant or the command-line tool, you can perform the following tasks (and more):

- Add a custom CSS file to the transformation
- Generate labels for the sections of task topics
- Specify that draft comments are included in the output
- Turn on "Related link" sections in a PDF file

Parent topic: <u>Getting Started with the DITA Open Toolkit</u> Previous topic: <u>Running the demo build</u>

DITA Open Toolkit User Guide

The *DITA Open Toolkit User Guide* is designed to provide basic information about the DITA-OT. It is geared for an audience that needs information about installing, running, and troubleshooting the toolkit. It contains documentation of the DITA-OT parameters; it also contains release notes and information about what components have been tested.

Overview of the DITA Open Toolkit The DITA Open Toolkit (DITA-OT) is an open-source implementation of the OASIS DITA specification, which is developed by the OASIS DITA Technical Committee. The DITA-OT is a set of Java-based, open-source tools and Ant scripts that transform DITA content (maps and topics) into deliverable formats, including Eclipse Help, HTML Help, JavaHelp, PDF, and XHTML.

Installing the DITA Open Toolkit

You can install the DITA Open Toolkit (DITA-OT) on Linux, Mac OSX, and Windows. The process for installing and setting up the DITA-OT depends on the type of distribution package that you select.

Publishing DITA content

You can use either Ant or the command-line tool to transform DITA content to the various output formats that are supported by the DITA Open Toolkit (DITA-OT).

Extending the DITA Open Toolkit

Plug-ins can be used to extend the functionality and configure the DITA Open Toolkit.

Globalizing DITA content

The DITA standard supports content that is written in or translated to any language. In general, the DITA Open Toolkit (DITA-OT) passes content through to the output format unchanged. The DITA-OT uses the values for the @xml:lang, @translate, and @dir attributes that are set in the source content to provides globalization support.

Error messages and troubleshooting

This section contains information about problems that you might encounter and how to resolve them.

Reference

This section is designed to help users to locate information easily and quickly. It includes documentation for the DITA Open Toolkit (DITA-OT) parameters and configuration properties.

Overview of the DITA Open Toolkit

The DITA Open Toolkit (DITA-OT) is an open-source implementation of the OASIS DITA specification, which is developed by the OASIS DITA Technical Committee. The DITA-OT is a set of Java-based, open-source tools and Ant scripts that transform DITA content (maps and topics) into deliverable formats, including Eclipse Help, HTML Help, JavaHelp, PDF, and XHTML.

While the DITA standard is owned and developed by OASIS, the DITA-OT project is governed separately; the DITA-OT is an independent, opensource implementation of the DITA standard. The DITA-OT is available without charge and is licensed under the CPL 1.0 and Apache 2.0 opensource licenses.

DITA Open Toolkit Release 2.0

DITA 1.2 Specification Support

DITA Open Toolkit 2.0 supports the DITA 1.2 specification. Initial support for this specification was added in version 1.5 of the toolkit; versions 1.5.1 and 1.5.2 contain minor modifications to keep up with the latest drafts. The specification itself was approved at approximately the same time as DITA-OT 1.5.2, which contained the final versions of the DTD and Schemas. DITA-OT 1.6 updated the DITA 1.2 XSDs to address minor errata in the standard; the DTDs remain up to date.

Tested platforms and tools

The DITA Open Toolkit (DITA-OT) has been tested against certain versions of Ant, ICU for Java, JDK, operating systems, XML parsers, and XSLT processors.

Parent topic: DITA Open Toolkit User Guide

Related information

Common Public License, version 1.0 Apache License, version 2.0

DITA Open Toolkit Release 2.0

Parent topic: Overview of the DITA Open Toolkit

General Enhancements and Changes

Migration from previous releases

Issues

The following items are included in DITA Open Toolkit Release 2.0. Issue numbers correspond to the tracking number in the <u>GitHub issues tracker</u>.

Feature requests

- #1192 Empty kills FOP (milestone 1)
- #1267 PDF support for figurelink/tablelink.style (milestone 1)
- #1347 Issue with "xsl/dita2xhtml.xsl"? (milestone 1)
- #1506 New command line tool (milestone 1)
- #1507 Convert XSLT to version 2.0 (milestone 1)
- #1511 Use Ivy for dependency management (milestone 1)
- #1522 Add HTML5 transtype (milestone 1)
- #1523 Reduce HTML/XHTML code duplication (milestone 1)
- #1524 Use DITAVAL for print filtering (milestone 1)
- #1548 Support RFC 5147 in coderef (milestone 1)
- #1561 Combine chunkedtopic lists into fullditatopic list (milestone 1)
- #1569 Index capability with FOP (milestone 1)
- #1601 Change output for rendering a single topic to PDF (milestone 1)
- #1602 Add plug-in installation operation to integration (milestone 1)
- #1608 img should be used instead of embed in XHTML output for all image types (milestone 1)
- #1610 Filtered content leads to total fail of PDF rendering duplicate (milestone 1)

- #1612 Update Ant to 1.9.2 (milestone 1)
- #1614 ant build should not bail out when fop directory has no lib folder (milestone 1)
- #1616 Remove legacy PDF from default distributions (milestone 1)
- #1509 Remove deprecated code (in progress)
- #1649 Support DITA 1.3 link syntax (milestone 2)
- #1644 Add more HTML entities to file entities.properties in dost.jar (milestone 2)
- #1643 Integrator task constructs property constructs with platformdependent path separators (milestone 2)
- #1636 Support DITA 1.3 cascade attribute (milestone 2)
- #1635 Implement DITA 1.3 profiling (milestone 2)
- #1631 Process keyrefs before conref (milestone 2)
- #1673 Remove support for generate.copy.outer=2 (milestone 3)
- #1671 Add support for generating back cover on PDF (milestone 3)
- #1667 Remove FrameMaker index syntax support (milestone 3)
- #1512 Alternative preprocessing plug-in (in progress)
- #1544 Use URI and File instead of String (in progress)
- #1654 Add DITA 1.3 div element (in progress)
- #1652 Add DITA 1.3 markup and xml domain support (in progress)
- #1651 Add new DITA 1.3 highlighting elements (in progress)

Bugs

• #1095 Empty tags result in invalid HTML (milestone 1)

- #1239 Essentially needs XercesImpl (milestone 1)
- #1247 Enable specifying MathML file as image/@href (milestone 1)
- #1272 chapter numbering confused when chapters and parts in bookmap (milestone 1)
- #1407 "Ambiguous rule match" in PDF processing XSLT when <imagemap> is used in topic (milestone 1)
- #1421 Warnings when converting to PDF with XEP (milestone 1)
- #1462 Support for rowheader in PDF output (milestone 1)
- #1486 PDF transform does not apply @rowsep correctly (milestone 1)
- #1607 Keyref not processed for pushed conrefs (milestone 1)
- #1620 ODT plugin fails due to missing getMax() (milestone 2)
- #1619 Add validation for attribute generalization (milestone 2)
- #1103 DITA-OT doesn't rewite lq/@href (milestone 2)
- #1668 Fix table cell @id and @headers attributes #1596 (milestone 3)
- #1634 XML Catalog which has a DOCTYPE declaration is not properly parsed by Integrator task (milestone 3)
- #1596 xref to table cell of DITA simple table does not result in correct target in <a> element (XHTML output) (milestone 3)
- #1539 TopicMerge generates multiple topic entries in the merged middle file. (milestone 3)
- #1231 <screen> and <codebock> do not support all frame attributes (milestone 3)
- #1086 metadata in ancestor map not inherited by topics in childmap (milestone 3)

#1600 Indirect reference to keys erases one file, causes failures (in progress)

DITA 1.2 Specification Support

DITA Open Toolkit 2.0 supports the DITA 1.2 specification. Initial support for this specification was added in version 1.5 of the toolkit; versions 1.5.1 and 1.5.2 contain minor modifications to keep up with the latest drafts. The specification itself was approved at approximately the same time as DITA-OT 1.5.2, which contained the final versions of the DTD and Schemas. DITA-OT 1.6 updated the DITA 1.2 XSDs to address minor errata in the standard; the DTDs remain up to date.

Earlier versions of the DITA Open Toolkit contained a subset of the specification material, including descriptions of each DITA element. This material was shipped in source, CHM and PDF format. This was possible in part because versions 1.0 and 1.1 of the DITA Specification contained two separate specification documents: one for the architectural specification, and one for the language specification.

In DITA 1.2, each of these has been considerably expanded, and the two have been combined into a single document. The overall document is much larger, and including the same set of material would double the size of the DITA-OT package. Rather than include that material in the package, we've provided the links below to the latest specification material.

Highlights of DITA 1.2 support in the toolkit include:

- Processing support for all new elements and attributes
- Link redirection and text replacement using keyref
- New processing-role attribute in maps to allow references to topics that will not produce output artifacts
- New conref extensions, including the ability to reference a range of elements, to push content into another topic, and to use keys for resolving a conref attribute.
- The ability to filter content with controlled values and taxonomies,

using the new Subject Scheme Map

- Processing support for both default versions of task (original, limited task, and the general task with fewer constraints on element order)
- Acronym and abbreviation support with the new <abbreviated-form> element
- New link grouping abilities available with headers in relationship tables
- OASIS Subcommittee specializations from the learning and machine industry domains (note that the core toolkit contains only basic processing support for these, but can be extended to produce related artifacts such as SCORM modules)

To find detailed information about any of these features, see the specification documents at OASIS. The DITA Adoption Technical Committee has also produced several papers to describe individual new features. In general, the white papers are geared more towards DITA users and authors, while the specification is geared more towards tool implementors, though both may be useful for either audience. The DITA Adoption papers can be found from that TC's main web page.

Parent topic: Overview of the DITA Open Toolkit

Related information

DITA 1.2 Specification (XHTML) DITA 1.2 Specification (PDF) DITA 1.2 Specification (zip of the DITA source) DITA 1.2 Specification (zip of the HTML Help) DITA Adoption Technical Committee Building subsets of the specification

Tested platforms and tools

The DITA Open Toolkit (DITA-OT) has been tested against certain versions of Ant, ICU for Java, JDK, operating systems, XML parsers, and XSLT processors.

Application	Tested version
	Ant 1.7.1
Ant	Ant 1.8.2—1.8.4, 1.9.2
	ICU4J 3.4.4
ICU for Java	ICU4J 49.1
	IBM 1.6
JDK	OpenJDK 1.7
	Oracle 1.6
Operating evotors	Mac OS X 10.6—10.9
	SLES 10
Operating system	Windows XP
	Windows 7
XML parcor	Xerces 2.9.0
XML parser	Xerces 2.11.0
	Saxon 6.5
XSLT processor	Saxon 9

Saxon-PE/EE 9.3

Parent topic: Overview of the DITA Open Toolkit

Installing the DITA Open Toolkit

You can install the DITA Open Toolkit (DITA-OT) on Linux, Mac OSX, and Windows. The process for installing and setting up the DITA-OT depends on the type of distribution package that you select.

Distribution packages The DITA Open Toolkit is distributed in four packages: minimal, standard, client, and full-easy-install. The source code is available both as a Git repository and a ZIP file.

Prerequisite software

The prerequisite software that the DITA-OT requires depends on the type of distribution package that you intend to install and the types of transformations that you want to use.

Installing the client package

For the simplest installation experience, install the client package. This package can be installed on Linux, Mac OSX, and Windows. It contains everything that you need to run the DITA-OT except for Java.

Installing the full-easy-install package

For the simplest installation experience, install the full-easy-install package. This package can be installed on Linux, Mac OSX, and Windows. It contains everything that you need to run the DITA-OT except for Java.

Installing the minimal or standard package on Linux or Mac OSX

If you already have locally-installed copies of Ant and the other required tools, install either the minimal or standard package.

Installing the minimal or standard package on Windows

If you already have locally-installed copies of Ant and the other required tools, install either the minimal or standard package.

Parent topic: DITA Open Toolkit User Guide

Distribution packages

The DITA Open Toolkit is distributed in four packages: minimal, standard, client, and full-easy-install. The source code is available both as a Git repository and a ZIP file.

Minimal package

This package is designed for vendors that embed the toolkit within a product. It contains all of the core processing code: CSS and XSLT files, Ant build scripts, Java code (dost.jar), resource files, and the OASIS DITA DTDs and Schemas. Users need to have their own versions of Ant and other libraries; they also need to set up environment variables for each library. The only external files that are included are the DTDs and Schemas, along with the following open-source libraries:

- Apache Catalog Resolver, version 1.1
- Apache Commons Codec, version 1.8
- Apache Xerces, version 2.11.0

The minimal package has the following file names:

- DITA-OT2.0.M3_minimal_bin.zip
- DITA-OT2.0.M3_minimal_bin.tar.gz

Standard package

This package is designed for people who want the core toolkit functionality, but who already have locally-installed copies of Ant and other required tools. It contains everything in the minimal package, plus documentation, demo code (for example, legacy support for the old bookmap), sample Ant scripts, and sample DITA files. The standard package includes the following open-source libraries:

- Apache Catalog Resolver, version 1.1
- Apache Commons Codec, version 1.8
- Apache Xerces, version 2.11.0

The standard package has the following file names:

- DITA-OT2.0.M3_standard_bin.zip
- DITA-0T2.0.M3_standard_bin.tar.gz

Client package

This package is designed for users who want the simplest installation experience. In addition to the core DITA-OT code and the external libraries that are in the minimal and standard packages, it contains Apache Ant and FOP. The client package also contains scripts for a guided demo of the DITA-OT. The client package includes the following external libraries:

- Apache Ant, version 1.9.2
- Apache Catalog Resolver, version 1.1
- Apache Commons Codec, version 1.8
- Apache FOP, version 1.1
- ICU for Java, version 52.1
- Apache Xerces, version 2.11.0
- Saxon, version 9.1.0.8

The client package has the following file names:

- DITA-OT2.0.M3_client_bin.zip
- DITA-0T2.0.M3_client_bin.tar.gz

Full-easy-install package

This package is designed for users who want the simplest installation experience. In addition to the core DITA-OT code and the external libraries that are in the minimal and standard packages, it contains Apache Ant and FOP. The full-easy-install package also contains batch files designed to set up a build environment using those tools, as well as a scripts for a guided demo of the DITA-OT. The full-easy-install package includes the following external libraries:

- Apache Ant, version 1.9.2
- Apache Catalog Resolver, version 1.1
- Apache Commons Codec, version 1.8
- Apache FOP, version 1.1
- ICU for Java, version 52.1
- Apache Xerces, version 2.11.0
- Saxon, version 9.1.0.8

The full-easy-install package has the following file names:

- DITA-0T2.0.M3_full_easy_install_bin.zip
- DITA-OT2.0.M3_full_easy_install_bin.tar.gz

Parent topic: Installing the DITA Open Toolkit

Related information

DITA Open Toolkit distibution package downloads DITA Open Toolkit source

Prerequisite software

The prerequisite software that the DITA-OT requires depends on the type of distribution package that you intend to install and the types of transformations that you want to use.

Software required for core DITA-OT processing

The DITA-OT requires the following software applications:

JRE or JDK, version 7 or later

Provides the basic environment for the DITA-OT. You can download the Oracle JRE or JDK from <u>http://www.oracle.com/technetwork/java/javase/downloads/index</u> If you opt to use the full-easy-install package, this is the *only* prerequisite software that you need to install.

Ant, version 1.7.1 or later

Provides the standard setup and sequencing of processing steps. You can download Ant from <u>http://ant.apache.org/</u>.

XSLT processor

Provides the main transformation services. It must be compliant with XSLT 2.0. The DITA-OT is tested with Saxon. You can download Saxon, version 9.1.0.8 from http://saxon.sourceforge.net/.

Software required for specific transformations

Depending on the type of output that you want to generate, you might need the following applications:

ICU for Java

ICU for Java is a cross-platform, Unicode-based, globalization library. It includes support for comparing locale-sensitive strings; formatting dates, times, numbers, currencies, and messages; detecting text boundaries; and converting character sets. You can download ICU for Java from <u>http://www.icu-</u> <u>project.org/download/</u>.

Microsoft Help Workshop

Required for generating HTML help. You can download the Help Workshop from <u>http://msdn.microsoft.com/en-</u> <u>us/library/windows/desktop/ms669985%28v=vs.85%29.aspx</u>.

XSL-FO processor

Required for generating PDF output. You can download FOP from <u>http://xmlgraphics.apache.org/fop/download.html</u>; you also can use Antenna House Formatter or RenderX.

See <u>Tested platforms and tools</u> for detailed information about versions of the prerequisite applications that have been tested with the current DITA-OT release.

Parent topic: Installing the DITA Open Toolkit

Installing the client package

For the simplest installation experience, install the client package. This package can be installed on Linux, Mac OSX, and Windows. It contains everything that you need to run the DITA-OT except for Java.

Before you begin

- Ensure that you have Java JRE or JDK, version 7 or later installed.
- Ensure that you have HTML Help Workshop installed, if you want to generate HTML Help.

Procedure

1. Download the client package from <u>SourceForge</u>.

Operating system	File name
Linux or Mac OSX	DITA-OT2.0.M3_client_bin.tar.gz
Windows	DITA-OT2.0.M3_client_bin.zip

- 2. Extract the contents of the package to the directory where you want to install the DITA-OT.
- 3. **Optional:** Add bin directory to *PATH* system variable. This defines the necessary environment variable to run the dita command from the command-line.

Parent topic: Installing the DITA Open Toolkit

Installing the full-easy-install package

For the simplest installation experience, install the full-easy-install package. This package can be installed on Linux, Mac OSX, and Windows. It contains everything that you need to run the DITA-OT except for Java.

Before you begin

- Ensure that you have Java JRE or JDK, version 7 or later installed.
- Ensure that you have HTML Help Workshop installed, if you want to generate HTML Help.

Procedure

1. Download the full-easy package from <u>SourceForge</u>.

Operating system	File name
Linux or Mac OSX	DITA-OT2.0.M3_full_easy_install_bin.tar.gz
Windows	<pre>DITA-OT2.0.M3_full_easy_install_bin.zip</pre>

- 2. Extract the contents of the package to the directory where you want to install the DITA-OT.
- 3. Run the startcmd file that is applicable for your operating system. This defines the necessary environment variables and opens a DOS prompt or terminal window in which you can invoke the toolkit. You can use the window to run as many builds as you want; if you close the window, you will need to run the applicable startcmd file again.

Parent topic: Installing the DITA Open Toolkit

Installing the minimal or standard package on Linux or Mac OSX

If you already have locally-installed copies of Ant and the other required tools, install either the minimal or standard package.

Before you begin

Ensure that you have the following prerequisite software installed:

- Ant, version 1.7.1 or later
- Java runtime environment or development kit, version 7 or later
- XSLT 2.0 processor. You can use Saxon, version 9.1.0.8 or later.

In addition, determine the specific DITA-OT transformations that you intend to support and ensure that you have the prerequisite software installed for them.

For more information, see <u>Prerequisite software</u> and <u>Tested platforms</u> <u>and tools</u>.

Procedure

1. Download the minimal or standard package from <u>SourceForge</u>.

Package File nameMinimal DITA-OT2.0.M3_minimal_bin.tar.gzStandard DITA-OT2.0.M3_standard_bin.tar.gz

For production use, we recommend that you use the latest stable release.

2. Extract the contents of the package into an installation directory.

Note: You can extract the files either to your private home directory for your exclusive use or to the /usr/local/share/ directory, if you want to share the DITA-OT with other users.

3. Verify that the JAVA_HOME environment variable is set.

```
export JAVA_HOME=<JRE_dir>
```

4. Verify that the ANT_HOME environment variable is been set.

```
export ANT_HOME=<Ant_dir>
```

5. Verify that the *PATH* environment variable includes the Java and Ant executable files.

```
export PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$PATH
```

6. Set the *DITA_HOME* environment variable to point to the DITA-OT installation directory.

```
export DITA_HOME=<DITA-OT_dir>
```

7. Set up the CLASSPATH environment variable.

```
export CLASSPATH=$DITA_HOME/lib/dost.jar:$CLASSPATH
export CLASSPATH=$DITA_HOME/lib:$CLASSPATH
```

export CLASSPATH=\$DITA_HOME/lib/resolver.jar:\$CLASSPATH export CLASSPATH=\$DITA_HOME/lib/commons-codec-1.8.jar:\$CLASS

8. **Optional:** If you use Ant, version 1.8 or later, set up the *CLASSPATH* environment variable to include Apache Xerces.

export CLASSPATH=<xerces_dir>/xercesImpl.jar:<xerces_dir>/xm.

9. Set up the XSLT processor:

Set up the *CLASSPATH* environment variable to include the Saxon JAR files, for example:

export CLASSPATH=<saxon_dir>/saxon9.jar:<saxon_dir>/saxon9-d

Set up the ANT_OPTS environment variable, for example:

export ANT_OPTS=\$ANT_OPTS -Djavax.xml.transform.TransformerFa

10. Optional: For index processing, set up ICU for Java.

export CLASSPATH=<icu4j_dir>/icu4j.jar:\$CLASSPATH

11. **Optional:** For JavaHelp, set the *JHHOME* environment variable.

export JHHOME=<javahelp_dir>

12. **Optional:** For PDF output, set up the XSL-FO processor:

Processor Action

FOP	Add the FOP installation directory to the <pre>local.properties file as the fop.home property, for example: fop.home=/usr/share/java/fop</pre>
RenderX	Add the RenderX installation directory to the local.properties file as the xep.dir property, for example:
	xep.dir=/usr/share/java/xep

Antenna House	Add the AH Formatter installation directory to the local.properties file as the axf.path property, for example:
	axf.path=/usr/share/java/AHFormatterV6

13. Test the DITA-OT installation by transforming the sample files. The samples\ant_sample directory contains Ant scripts designed to build various output formats.

```
/usr/local/share/DITA-OT2.0.M3$ ant -f samples/ant_sample/sam
```

The generated output is written to the *DITA-dir*\out\samples directory. The following output formats are generated:

- Docbook
- Eclipse help
- HTML Help
- ODT
- JavaHelp
- PDF
- TocJS
- TROFF
- XHTML

Parent topic: Installing the DITA Open Toolkit

Installing the minimal or standard package on Windows

If you already have locally-installed copies of Ant and the other required tools, install either the minimal or standard package.

Before you begin

Ensure that you have the following prerequisite software installed:

- Ant, version 1.7.1 or later
- Java runtime environment or development kit, version 7 or later
- XSLT 2.0 processor. You can use Saxon, version 9.1.0.8 or later.

In addition, determine the specific DITA-OT transformations that you intend to support and ensure that you have the prerequisite software installed for them.

For more information, see <u>Prerequisite software</u> and <u>Tested platforms</u> <u>and tools</u>.

Procedure

1. Download the minimal or standard package from <u>SourceForge</u>.

Package File nameMinimal DITA-0T2.0.M3_minimal_bin.zipStandard DITA-0T2.0.M3_standard_bin.zip

For production use, we recommend that you use the latest stable release.

- 2. Extract the contents of the package into an installation directory. For example, C:\pkg\DITA-0T2.0.M3.
- 3. Verify that the JAVA_HOME environment variable is set.

set JAVA_HOME=<JRE_dir>

4. Verify that the ANT_HOME environment variable is set.

set ANT_HOME=<Ant_dir>

5. Verify that the *PATH* environment variable includes the Java and Ant executable files.

```
set PATH=%JAVA_HOME%\bin;%ANT_HOME%\bin;%PATH%
```

6. Set the *DITA_HOME* environment variable to point to the DITA-OT installation directory.

set DITA_HOME=<DITA-OT_dir>

- 7. Set up the CLASSPATH environment variable.
 - set CLASSPATH=%DITA_HOME%\lib\dost.jar;%CLASSPATH%
 - set CLASSPATH=%DITA_HOME%\lib;%CLASSPATH%
 - set CLASSPATH=%DITA_HOME%\lib\resolver.jar;%CLASSPATH%
 - set CLASSPATH=%DITA_HOME%\lib\commons-codec-1.8.jar;%CLASSPA[·]

8. **Optional:** If you use Ant, version 1.8 or later, set up the *CLASSPATH* environment variable to include Apache Xerces.

```
set CLASSPATH=<xerces_dir>\xercesImpl.jar;<xerces_dir>\xml-a
```

9. Set up the XSLT processor:

Set up the *CLASSPATH* environment variable to include the Saxon JAR files, for example:

```
set CLASSPATH=<saxon_dir>\saxon9.jar;<saxon_dir>\saxon9.dom.;
```

Set up the ANT_OPTS environment variable, for example:

set ANT_OPTS=%ANT_OPTS% -Djavax.xml.transform.TransformerFac

10. **Optional:** For index processing, set up ICU for Java.

set CLASSPATH=<icu4j_dir>\icu4j.jar;%CLASSPATH%

11. **Optional:** For JavaHelp, set the *JHHOME* environment variable.

set JHHOME=<javahelp_dir>

12. **Optional:** For HTML Help, add the installation directory for the HTML Help Workshop to the local.properties file as the **hhc.dir** property.

```
hhc.dir=C:\\Program Files (x86)\\HTML Help Workshop
```

13. **Optional:** For PDF output, set up the XSL-FO processor:

ProcessorAction	
FOP	Add the FOP installation directory to the local.properties file as the fop.home property, for example:
	<pre>fop.home=C:\\Program Files\\fop</pre>
	Add the RenderX installation directory to the

RenderX	local.properties file as the xep.dir property, for example:
	xep.dir=C:\\Program Files\\xep
Antenna House	Add the AH Formatter installation directory to the local.properties file as the axf.path property, for example:
	axf.path=C:\\Program Files\\AHFormatterV6

14. Test the DITA-OT installation by transforming the sample files. The samples\ant_sample directory contains Ant scripts designed to build various output formats.

```
C:\DITA-OT2.0.M3>ant -f samples\ant_sample\sample_all.xml
```

The generated output is written to the *DITA-dir*\out\samples directory. The following output formats are generated:

- Docbook
- Eclipse help
- HTML Help
- ODT
- JavaHelp
- PDF
- TocJS
- TROFF
- XHTML

Parent topic: Installing the DITA Open Toolkit

Publishing DITA content

You can use either Ant or the command-line tool to transform DITA content to the various output formats that are supported by the DITA Open Toolkit (DITA-OT).

DITA-OT transformations The DITA Open Toolkit (DITA-OT) ships with several core transformations. Each core transformation represents an implementation of all processing that is defined by OASIS in the DITA specification.

Publishing DITA content with dita command

DITA-OT includes a dita command-line tool. You can invoke the DITA-OT from the command-line tool and generate output.

Publishing DITA content from Ant

You can use Ant to invoke the DITA Open Toolkit (DITA-OT) and generate output. This is the most robust method of transforming DITA content; you can use the complete set of parameters that are supported by the toolkit.

Publishing DITA content from the command-line tool

The DITA Open Toolkit (DITA-OT) includes a command-line tool designed for users who are unfamiliar with Ant. You can invoke the DITA-OT from the command-line tool and generate output. This method is less robust than Ant.

Parent topic: DITA Open Toolkit User Guide

DITA-OT transformations

The DITA Open Toolkit (DITA-OT) ships with several core transformations. Each core transformation represents an implementation of all processing that is defined by OASIS in the DITA specification.

DITA to Docbook The **docbook** transformation converts DITA maps and topics into a Docbook output file. Complex DITA markup might not be supported, but the transformation supports most common DITA structures.

DITA to Eclipse Content

The **eclipsecontent** transformation generates normalized DITA files and Eclipse control files. It originally was designed for an Eclipse plug-in that dynamically rendered DITA content, but the output from the transformation can be used by other applications that work with DITA.

DITA to Eclipse help

The **eclipsehelp** transformation generates XHTML output, CSS files, and the control files that are needed for Eclipse help.

DITA to HTML5

The **html5** transformation generates HTML5 output and a table of contents (TOC) file.

DITA to HTML Help (CHM)

The **htmlhelp** transformation generates HTML output, CSS files, and the control files that are needed to produce a Microsoft HTML Help file.

DITA to Open Document Type

The **odt** transformation produces output files that use the Open Document format, which is used by tools such as Open Office.

DITA to PDF (PDF2)

The **pdf** (or **pdf2**) transformation generates PDF output.

DITA to Rich Text Format

The **wordrtf** transformation produces an RTF file for use by Microsoft Word.

DITA to TocJS

The **tocjs** transformation generates HTML5 output, a frameset, and a JavaScript-based table of contents with expandable and collapsible entries. The transformation was originally created by Shawn McKenzie as a plug-in and was added to the default distribution in DITA OT, release 1.5.4.

DITA to Troff

The **troff** transformation produces output for use with the Troff viewer on Unix-style platforms, particularly for programs such as the Man page viewer.

DITA to XHTML

The **xhtml** transformation generates XHTML output and a table of contents (TOC) file. This was the first transformation created for the DITA Open Toolkit, and it is the basis for all the HTML-based transformations.

Parent topic: Publishing DITA content

DITA to Docbook

The **docbook** transformation converts DITA maps and topics into a Docbook output file. Complex DITA markup might not be supported, but the transformation supports most common DITA structures.

DITA to Eclipse Content

The **eclipsecontent** transformation generates normalized DITA files and Eclipse control files. It originally was designed for an Eclipse plug-in that dynamically rendered DITA content, but the output from the transformation can be used by other applications that work with DITA.

Normalized DITA files have been through the DITA Open Toolkit preprocessing operation. In comparison to the source DITA files, the normalized DITA file are modified in the following ways:

- Map-based links, such as those generated by map hierarchy and relationship tables, are added to the topics.
- Link text is resolved.
- Any DTD or Schema reference is removed.
- Class attributes that are defaulted in the DTD or Schema are made explicit in the topics.
- Map attributes that cascade are made explicit on child elements.

The normalized DITA files have an extension of .xml.

Parent topic: DITA-OT transformations

Related reference

Ant parameters: Eclipse content transformation Command-line tool parameters: Eclipse content transformation

DITA to Eclipse help

The **eclipsehelp** transformation generates XHTML output, CSS files, and the control files that are needed for Eclipse help.

In addition to the XHTML output and CSS files, this transformation returns the following files, where *mapname* is the name of the master DITA map.

File name	Description	
plugin.xml	Control file for the Eclipse plug-in	
<i>mapname</i> .xml	Table of contents	
index.xml	Index file	
plugin.properties		
META-INF/MANIFEST.MF		

Parent topic: DITA-OT transformations

Related reference

Ant parameters: Eclipse help transformation Command-line tool parameters: Eclipse help transformation

Related information

Official Eclipse Web site

DITA to HTML5

The **html5** transformation generates HTML5 output and a table of contents (TOC) file.

The HTML5 output is always associated with the default DITA-OT CSS file (commonltr.css or commonrtl.css for right-to-left languages). You can use toolkit parameters to add a custom style sheets to override the default styles.

To run the HTML5 transformation, set the **transtype** parameter to **html5**.

Parent topic: DITA-OT transformations

Related reference

<u>Ant parameters: HTML5 and XHTML transformation</u> <u>Command-line tool parameters: HTML5 and XHTML transformation</u>

DITA to HTML Help (CHM)

The **htmlhelp** transformation generates HTML output, CSS files, and the control files that are needed to produce a Microsoft HTML Help file.

In addition to the HTML output and CSS files, this transformation returns the following files, where *mapname* is the name of the master DITA map.

File name	Description
<i>mapname</i> .hhc	Table of contents
<i>mapname</i> .hhk	Sorted index
<i>mapname</i> .hhp	HTML Help project file
	Compiled HTML Help
<i>mapname</i> .chm	Note: This file is generated only if the HTML Help Workshop is installed on the build system.

Parent topic: DITA-OT transformations

Related reference

<u>Ant parameters: HTML Help transformation</u> <u>Command-line tool parameters: HTML help transformation</u>

DITA to Open Document Type

The **odt** transformation produces output files that use the Open Document format, which is used by tools such as Open Office.

This transform returns an ODT document, which is a zip file that contains the ODF XML file (content.xml), referenced images, and default styling (in the file styles.xml).

Parent topic: DITA-OT transformations

Related reference Ant parameters: ODT transformation Command-line tool parameters: ODT transformation

DITA to PDF (PDF2)

The **pdf** (or **pdf2**) transformation generates PDF output.

This transformation was originally created as a plug-in and maintained outside of the main toolkit code. It was created as a more robust alternative to the demo PDF transformation in the original toolkit, and thus was known as PDF2. The plug-in was bundled into the default toolkit distribution with release 1.4.3.

Parent topic: DITA-OT transformations

Related reference Ant parameters: PDF transformation Command-line tool parameters: PDF transformation

DITA to Rich Text Format

The **wordrtf** transformation produces an RTF file for use by Microsoft Word.

The structure of the generated RTF file is the same as the navigation structure in the DITA map. To avoid losing files in the final output, make sure the DITA map contains all topics that are referenced from any individual topics.

The **wordrtf** transformation has the following limitations:

- Flagging, filtering, and revision bars are not supported.
- Style attributes for tables are not supported.
- Tables within list items are not supported.
- Output styles supported by other DITA-OT transformations, for example, X and Y, are not supported.

DITA to TocJS

The **tocjs** transformation generates HTML5 output, a frameset, and a JavaScript-based table of contents with expandable and collapsible entries. The transformation was originally created by Shawn McKenzie as a plug-in and was added to the default distribution in DITA OT, release 1.5.4.

The **tocjs** transformation was updated so that it produces HTML5 output and uses a default frameset. This transformation also was added to the build_demo.xml script as a transformation-type option.

DITA to Troff

The **troff** transformation produces output for use with the Troff viewer on Unix-style platforms, particularly for programs such as the Man page viewer.

Each DITA topic generally produces one troff output file. The **troff** transformation supports most common DITA structures, but it does not support or <simpletable> elements. Most testing of troff output was performed using the Cygwin Linux emulator.

DITA to XHTML

The **xhtml** transformation generates XHTML output and a table of contents (TOC) file. This was the first transformation created for the DITA Open Toolkit, and it is the basis for all the HTML-based transformations.

The XHTML output is always associated with the default DITA-OT CSS file (commonltr.css or commonrtl.css for right-to-left languages). You can use toolkit parameters to add a custom style sheets to override the default styles.

To run the XHTML transformation, set the **transtype** parameter to **xhtml**. If you are running the demo build, specify **web** rather than **xhtml**.

Parent topic: DITA-OT transformations

Related reference

Ant parameters: HTML5 and XHTML transformation Command-line tool parameters: HTML5 and XHTML transformation

Publishing DITA content with dita command

DITA-OT includes a dita command-line tool. You can invoke the DITA-OT from the command-line tool and generate output.

Building output using the dita command You can invoke DITA-OT and build output using the dita command.

Parent topic: Publishing DITA content

Building output using the dita command

You can invoke DITA-OT and build output using the dita command.

Procedure

From the command prompt, issue the following command:

installation-directory\bin\dita -i input-file -f transformation-u

where:

- *input-file* is the DITA map or DITA file that you want to process.
- *transformation-type* is the transformation type.
- *parameter-name* is the name of an optional parameter.
- *value* is an applicable value for the optional parameter.

If you add *installation-directory*\bin directory to *PATH* system variable, you can invoke the dita command without the absolute path.

If you do not specify an output directory, by default, the DITA-OT writes the output to out subdirectory of current directory.

Example

The following command generates HTML5 output for the sequence.ditamap file and specifies the output is written to the test directory

```
dita -i samples/sequence.ditamap -o test -f html5
```

Parent topic: Publishing DITA content with dita command

Publishing DITA content from Ant

You can use Ant to invoke the DITA Open Toolkit (DITA-OT) and generate output. This is the most robust method of transforming DITA content; you can use the complete set of parameters that are supported by the toolkit.

Ant Ant is a Java-based, open-source tool that is provided by the Apache Foundation. It can be used to declare a sequence of build actions. It is well suited for both development and document builds. The full-easy-install version of the toolkit ships with a copy of Ant.

Building output using Ant

You can build output by running the ant command and specifying the DITA-OT parameters at the command prompt. You also can use an Ant build script to provide the DITA-OT parameters

Creating an Ant build script

Instead of typing the DITA-OT parameters at the command prompt, you might want to create an Ant build script that contains all of the parameters.

Parent topic: Publishing DITA content

Related concepts Ant

Related reference Ant parameters

Ant

Ant is a Java-based, open-source tool that is provided by the Apache Foundation. It can be used to declare a sequence of build actions. It is well suited for both development and document builds. The full-easyinstall version of the toolkit ships with a copy of Ant.

The DITA-OT uses Ant to manage the XSLT scripts that are used to perform the various transformation; it also uses Ant to manage intermediate steps that are written in Java.

The most important Ant script is the build.xml file. This script defines and combines common pre-processing and output transformation routines; it also defines the DITA-OT extension points.

Parent topic: Publishing DITA content from Ant

Related tasks

Building output using Ant Creating an Ant build script Publishing DITA content from Ant

Related reference Ant parameters Apache Ant documentation

Building output using Ant

You can build output by running the ant command and specifying the DITA-OT parameters at the command prompt. You also can use an Ant build script to provide the DITA-OT parameters

Procedure

- 1. Run the startcmd file that is applicable for your operating system. The startcmd.bat and startcmd.sh files are in the directory where you installed the DITA-OT.
- 2. To provide the DITA-OT parameters from the command prompt, issue the following command:

```
ant -Dargs.input=input-file -Dtranstype=transformation-type
```

where:

- input-file is the DITA map or DITA file that you want to process.
- *transformation-type* is the transformation type.
- *parameter-name* is the name of an optional parameter.
- value is an applicable value for the optional parameter.

If you do not specify an output directory, by default, the DITA-OT writes the output to the *installation-directory*\out directory.

3. If you use a build script, issue the following command:

```
ant -f build-script target
```

where:

- *build-script* is name of the Ant build script.
- *target* is an optional switch that specifies the name of the Ant target that you want to run. If you do not specify a target, the value of the @default attribute for the Ant project is used.

Parent topic: Publishing DITA content from Ant

Related concepts

Related tasks

Creating an Ant build script

Related reference Ant parameters Apache Ant documentation

Creating an Ant build script

Instead of typing the DITA-OT parameters at the command prompt, you might want to create an Ant build script that contains all of the parameters.

Procedure

1. Create an XML file that contains the following content:

```
<project name="@project-name@" default="@default-target@" bas
<property name="dita.dir" location="@path-to-DITA-OT@"/>
<target name="@target-name@">
<ant antfile="${dita.dir}${file.separator}build.xml">
<property name="args.input" value="@DITA-input@"/>
<property name="args.input" value="@DITA-input@"/>
<property name="transtype" value="html5"/>
</ant>
</project>
```

You will replace the placeholder content (indicated by the @ signs) with content applicable to your environment.

- 2. Specify project information:
 - 1. Set the value of the @name attribute to X.
 - Set the value of the @default attribute to the name of a target in the build script.
 If the build script is invoked without specifying a target, this target will be run.
- 3. Set the value of the **dita.dir** property to the location of the DITA-OT. This can be a fully qualified path, or you can specify it relative to the location of the Ant build script that you are writing.
- 4. Create the Ant target:
 - 1. Set the value of the @name attribute.

- 2. Specify the value for the **args.input** property.
- 3. Specify the value of the **transtype** property.
- 5. Save the build script.

Example

The following Ant build script generates CHM and PDF output for the userguide.ditamap file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="Toolkit-documentation" default="all" basedir=".">
    <property name="dita.dir" location="C:\DITA-OT1.6.M5"/>
    <target name="all" description="build CHM and PDF" depends="(
    <target name="chm" description="build CHM">
        <ant antfile="${dita.dir}\build.xml">
            <property name="args.input" value="C:\dita-ot\src\ma:</pre>
            <property name="args.gen.task.lbl" value="YES"/>
            <property name="output.dir" value="C:\kje\temp\out"/>
            <property name="transtype" value="htmlhelp"/>
        </ant>
    </target>
    <target name="pdf" description="build PDF">
        <ant antfile="${dita.dir}\build.xml">
            <property name="args.input" value="C:\dita-ot\src\ma:</pre>
            <property name="args.gen.task.lbl" value="YES"/>
            <property name="args.rellinks" value="nofamily"/>
            <property name="output.dir" value="C:\kje\temp\out"/>
            <property name="transtype" value="pdf"/>
        </ant>
    </target>
</project>
```

In addition to the mandatory parameters (**args.input** and **transtype**), the chm and pdf targets each specify some optional parameters:

- The **args.gen.task.lbl** property is set to YES, which ensures that headings are automatically generated for the sections of task topics.
- The **output.dir** property specifies where the DITA OT writes the output of the transformations.

The pdf target also specifies that related links should be generated in the PDF, but only those links that are created by relationship tables and <link> elements.

Finally, the all target simply specifies that both the chm and pdf target should be run.

What to do next

Another resource for learning about Ant scripts are the files in the samples/ant_samples directory. This directory contains the Ant build files used by the demo build, as well as templates that you can use to create Ant scripts.

Parent topic: Publishing DITA content from Ant

Related concepts Ant

Related tasks Building output using Ant

Related reference Ant parameters Apache Ant documentation

Publishing DITA content from the command-line tool

The DITA Open Toolkit (DITA-OT) includes a command-line tool designed for users who are unfamiliar with Ant. You can invoke the DITA-OT from the command-line tool and generate output. This method is less robust than Ant.

Command-line tool The DITA Open Toolkit (DITA-OT) provides a command-line tool for users with little knowledge of Ant. The command-line tool supports a subset of the parameters that are available to the Ant builds.

Building output using the command-line tool

If you are unfamiliar with Ant, you can invoke the DITA Open Toolkit (DITA-OT) and build output from the command-line tool.

Parent topic: Publishing DITA content

Related concepts Command-line tool

Related reference

Command-line tool parameters

Command-line tool

The DITA Open Toolkit (DITA-OT) provides a command-line tool for users with little knowledge of Ant. The command-line tool supports a subset of the parameters that are available to the Ant builds.

The command-line tool is a wrapper around the Ant interface; it takes the simplified parameters as input, converts them to Ant parameters, and then runs an Ant build. The Ant parameters that are passed to the DITA-OT are saved to the property.temp file that is written to the output directory.

For individual builds, the additional Java overhead is minimal, but for repeated or server-based builds, the extra memory usage might become an issue. Applications that embed the toolkit should invoke Ant directly.

Parent topic: Publishing DITA content from the command-line tool

Related tasks

Building output using the command-line tool Publishing DITA content from the command-line tool

Related reference

Command-line tool parameters

Building output using the command-line tool

If you are unfamiliar with Ant, you can invoke the DITA Open Toolkit (DITA-OT) and build output from the command-line tool.

Procedure

- 1. Run the startcmd file that is applicable for your operating system. The startcmd.bat and startcmd.sh files are in the directory where you installed the DITA-OT.
- 2. From the command prompt, issue the following command:

```
java -jar lib/dost.jar /i:input-file /transtype:transfor
```

where:

- *input-file* is the DITA map or DITA file that you want to process.
- *transformation-type* is the transformation type.
- *parameter-name* is the name of an optional parameter.
- *value* is an applicable value for the optional parameter.

If you do not specify an output directory, by default, the DITA-OT writes the output to the *installation-directory*\out directory.

Example

The following command generates HTML5 output for the sequence.ditamap file and specifies the output is written to the test directory

java -jar lib/dost.jar /i:samples/sequence.ditamap /outdir:test /

Parent topic: Publishing DITA content from the command-line tool

Related concepts Command-line tool

Related reference Command-line tool parameters

Extending the DITA Open Toolkit

Plug-ins can be used to extend the functionality and configure the DITA Open Toolkit.

Installing plug-ins Plug-ins are distributed as zip files and can be installed using either the command line tool or Ant.

Removing plug-ins Plug-ins can be installed by running uninstallation process.

Parent topic: DITA Open Toolkit User Guide

Installing plug-ins

Plug-ins are distributed as zip files and can be installed using either the command line tool or Ant.

Procedure

Run plug-in installation process.

- Using the dita command line tool, run the installation command: dita -install *plug-in-zip*
- Using Ant, from the toolkit directory run the installation target:

```
ant -f integrator.xml install -Dplugin.file=plug-in-zip
```

The *plug-in-zip* can be either a local file path or a URL.

Parent topic: Extending the DITA Open Toolkit

Removing plug-ins

Plug-ins can be installed by running uninstallation process.

Procedure

Run plug-in uninstallation process.

- Using the dita command line tool, run the uninstallation command: dita -uninstall *plug-in-id*
- Using Ant, from the toolkit directory run the uninstallation target:

```
ant -f integrator.xml uninstall -Dplugin.id=plug-in-id
```

Parent topic: Extending the DITA Open Toolkit

Globalizing DITA content

The DITA standard supports content that is written in or translated to any language. In general, the DITA Open Toolkit (DITA-OT) passes content through to the output format unchanged. The DITA-OT uses the values for the @xml:lang, @translate, and @dir attributes that are set in the source content to provides globalization support.

<u>Globalization support offered by the DITA-OT</u> The DITA Open Toolkit (DITA-OT) offers globalization support in the following areas: Generated text, index sorting, and bi-directional text.

Supported languages: HTML-based transformations

The DITA Open Toolkit (DITA-OT) supports over 50 languages and language variants for the HTML-based transformations, for example, Eclipse Help, HTML Help, and TocJS.

Supported languages: PDF transformations

The DITA Open Toolkit (DITA-OT) supports a smaller set of languages for the PDF (pdf2) transformation. This transformation was donated to the DITA-OT project after the project inception, and it uses a different and larger set of generated text than the HTMLbased transformations.

Parent topic: DITA Open Toolkit User Guide

Related reference

Localization overview in the OASIS DITA standard

Globalization support offered by the DITA-OT

The DITA Open Toolkit (DITA-OT) offers globalization support in the following areas: Generated text, index sorting, and bi-directional text.

Generated text

Generated text is text that is rendered automatically in the output that is generated by the DITA-OT; this text is not located in the DITA source files. The following are examples of generated text:

- The word "Chapter in a PDF file.
- The phrases "Related concepts," "Related tasks," and "Related reference" in XHTML output.

Index sorting

The DITA-OT can use only a single language to sort indexes.

Bi-directional text

The DITA-OT contains style sheets (CSS files) that support both left-to-right (LTR) and right-to-left (RTL) languages.

When the DITA-OT generates output, it takes the first value for the @xml:lang attribute that it encounters, and then it uses that value to create generated text, perform index sorting, and determine which default CSS file is used. If no value for the @xml:lang attribute is found, the toolkit defaults to US English.

Parent topic: Globalizing DITA content

Supported languages: HTML-based transformations

The DITA Open Toolkit (DITA-OT) supports over 50 languages and language variants for the HTML-based transformations, for example, Eclipse Help, HTML Help, and TocJS.

• •	nguages: HTML-based rmations
Language	Language code
Arabic	ar or ar-eg
Belarusian	be or be-by
Brazilian Portuguese	pt-br
Bulgarian	bg or bg-bg
Catalan	ca-es
Chinese (simplified)	zh-cn or zh-hans
Chinese (traditional)	zh-tw or zh-hant
Croatian	hr or hr-hr
Czech	CS OF CS-CZ
Danish	da or da-dk
Dutch	nl or nl-nl
Dutch (Belgian)	nl-be
English (US)	en or en-us
English (British)	en-gb
English (Canadian)	en-ca
Estonian	et or et-ee
Finnish	fi or fi-fi
French	fr or fr-fr
French (Belgian)	fr-be
French (Canadian)	fr-ca
French (Swiss)	fr-ch
German	de or de-de
German (Swiss)	de-ch
Greek	el or el-gr
Hebrew	he or he-il

Hindi	hi or hi-hi
Hungarian	hu or hu-hu
Icelandic	is or is-is
Indonesian	id or id-id
Italian	it or it-it
Italian(Swiss)	it-ch
Japanese	ja or ja-jp
Kazakh	kk or kk-kz
Korean	ko or ko-kr
Latvian	lv or lv-lv
Lithuanian	lt or lt-lt
Macedonian	mk or mk-mk
Malay	ms or ms-my
Norwegian	no or no-no
Polish	pl or pl-pl
Portuguese	pt or pt-pt
Romanian	ro or ro-ro
Russian	ru or ru-ru
Serbian (Cyrillic script)	sr, sr-rs, or sr-sp
Serbian (Latin script)	sr-latn-rs
Slovak	sk or sk-sk
Slovenian	sl or sl-si
Spanish	es or es-es
Spanish (Latin American)	es-419
Swedish	sv or sv-se
Thai	th or th-th
Turkish	tr or tr-tr
Ukrainian	uk or uk-ua
Urdu	ur or ur-pk

Parent topic: Globalizing DITA content

Related reference

How to add support for new languages in XHTML

Supported languages: PDF transformations

The DITA Open Toolkit (DITA-OT) supports a smaller set of languages for the PDF (pdf2) transformation. This transformation was donated to the DITA-OT project after the project inception, and it uses a different and larger set of generated text than the HTML-based transformations.

transfo	rmation
Language	Language code
Chinese (simplified)	zh-cn or zh-hans
Dutch	nl or nl-nl
English (US)	en or en-us
Finnish	fi or fi-fi
French	fr or fr-fr
German	de or de-de
Hebrew	he or he-il
Italian	it or it-it
Japanese	ja or ja-jp
Romanian	ro or ro-ro
Russian	ru or ru-ru
Slovenian	sl or sl-Sl
Spanish	es or es-es
Swedish	sv or sv-se

Table 1. Supported languages: PDF

Parent topic: Globalizing DITA content

Error messages and troubleshooting

This section contains information about problems that you might encounter and how to resolve them.

DITA-OT error messages The error messages generated by the DITA Open Toolkit contain a message ID, severity information, and message text. This topic lists each error message generated by the toolkit and provides additional information that might be helpful in understanding and resolving the error condition.

Other error messages

In addition to error messages generated by the DITA Open Toolkit, you might also encounter error messages generated by Java or other tools.

Log files

When you run the DITA-OT, key information is logged on the screen. This information can also be written to a log file. If you encounter a problem, you can analyze this information to determine the source of the problem and then take action to resolve it.

Accessing help from the command-line tool

You can access a list of supported parameters for the command-line tool by issuing the **-help** parameter.

Determing the version of the DITA Open Toolkit

You can use the command-line tool to determine the version of the DITA OT.

Enabling debug mode

When the debug mode is enabled, additional diagnostic information is written to the log file. This information, which includes environment variables and stack trace data, can help you determine the root cause of a problem.

Increasing Java memory allocation

If you are working with large documents with extensive metadata or key references, you will need to increase the memory allocation for the Java process. You can do this from the command-line prompt for a specific session, or you can increase the value of the ANT_OPTS environmental variable.

Parent topic: DITA Open Toolkit User Guide

DITA-OT error messages

The error messages generated by the DITA Open Toolkit contain a message ID, severity information, and message text. This topic lists each error message generated by the toolkit and provides additional information that might be helpful in understanding and resolving the error condition.

Each message ID is composed of a message prefix, a message number, and a letter that indicates the severity (I, W, E, or F). The toolkit uses the following severity scale:

Informational (I)

The toolkit encountered a condition of which you should be aware. For example, draft comments are enabled and will be rendered in the output.

Warning (W)

The toolkit encountered a problem that should be corrected. Processing will continue, but the output might not be as expected.

Error (E)

The toolkit encountered a more severe problem, and the output is affected. For example, some content is missing or invalid, or the content is not rendered in the output

Fatal (F)

The toolkit encountered a severe condition, processing stopped, and no output is generated.

Message ID	Severity	Message text	Add
		"%1" is not a recognized transformation	Defa ship xhtm

DOTA001F	Fatal	type. Supported transformation types are docbook, eclipsecontent, eclipsehelp, html5, htmlhelp, javahelp, net.sourceforge.dita-ot.html, odt, pdf, pdf2, tocjs, troff, wordrtf, xhtml.	tocjs eclip and trans avail insta
DOTA002F	Fatal	Input file is not specified, or is specified using the wrong parameter.	The spec DITA Ensu prop argu Com you a the ii
DOTA003F	Fatal	Cannot find the user specified XSLT stylesheet '%1'.	An a spec defa but t load para style
DOTA004F	Fatal	Invalid DITA topic extension '%1'. Supported values are '.dita' and '.xml'.	This to se topic temr Only ".xm
DOTA006W	Warning	Absolute paths on the local file system are not supported for the CSSPATH parameter. Please use a relative path or full URI instead.	If the absc that the f syste http Absc syste cont syste

DOTA007E	Error	Cannot find the running-footer file "%1". Please double check the value to ensure it is specified correctly.	The conta the k outp or re by a value that
DOTA008E	Error	Cannot find the running-header file "%1". Please double check the value to ensure it is specified correctly.	The conta the ta topic This in tha shou valua as a
DOTA009E	Error	Cannot find the specified heading file "%1". Please double check the value to ensure it is specified correctly.	The conta the < XHT locat caus para also spec
DOTA011W	Warning	Argument "%1" is deprecated. This argument is no longer supported in the toolkit.	
DOTA012W	Warning	Argument "%1" is deprecated. Please use the argument "%2" instead.	
DOTA066F	Fatal	Cannot find the user specified XSLT stylesheet '%1'.	An a spec defa but t load para

			style
DOTA067W	Warning	Ignoring index-see '%1' inside parent index entry '%2' because the parent indexterm contains indexterm children. According to the DITA Specification, the index-see element should be ignored if the parent indexterm contains other indexterm children.	This instru Stan
DOTA068W	Warning	Ignoring index-see-also '%1' inside parent index entry '%2' because the parent indexterm contains indexterm children. According to the DITA Specification, the index-see-also element should be ignored if the parent indexterm contains other indexterm children.	This instru Stan
DOTA069F	Fatal	Input file '%1' cannot be located or read. Ensure that file was specified properly and that you have permission to access it.	Plea path corre
DOTA069W	Warning	Target "%1" is deprecated. Remove references to this target from your custom XSLT or plug-ins.	
DOTJ001F	Fatal	Invalid command line syntax for the parameter '%1'. Parameters to the command line tool should use the syntax /parameter:value.	See <mark>para</mark> para
DOTJ002F	Fatal	Unsupported parameter '%1'. Please refer to the DITA-OT User Guide for supported parameters.	See <mark>para</mark> para
DOTJ003F	Fatal	Parameter '%1' was specified without a value. Parameters to the command line tool should use the syntax /parameter:value.	See <mark>para</mark> para
			The crea direc by a prev direc

DOTJ004F	Fatal	Cannot create temporary processing directory '%1'. Please ensure that you have permission to create the directory '%1'.	locat have defa alter locat argu Com for d temp
DOTJ005F	Fatal	Failed to create new instance for '%1'. Please ensure that '%1' exists and that you have permission to access it.	
DOTJ006F	Fatal	An Ant build file used the following illegal syntax when calling AntInvoker: extparam='%1'. Please correct the call to AntInvoker when directly calling DITA-OT Java code from Ant; for example, extparam="maplinks=XXXX;other=YYYY".	This Ant k pipel of us mod para your extpa "nam
DOTJ007E	Error	Duplicate condition in filter file for rule '%1'. The first encountered condition will be used.	If a c than audio rese first o
DOTJ009E	Error	Cannot overwrite file '%1' with file '%2'. The modified result may not be consumed by the following steps in the transform pipeline. Check to see whether the file is locked by some other application during the transformation process.	
			This inval accic file a map

DOTJ012F	Fatal	Failed to parse the input file '%1'. The XML parser reported the following error:	elerr are i has as X usin docu exte pars issue shou infor caus
DOTJ013E	Error	Failed to parse the referenced file '%1'. The XML parser reported the following error:	This refer as a PDF was uses allow file th be p also docu pars issue shou infor caus
DOTJ014W	Warning	Found an indexterm element with no content. Setting the term to ***.	An e was the i shou sour
DOTJ015F	Fatal	Log directory cannot be null. Please specify a valid directory for the build log.	
DOTJ016F	Fatal	Failed to create log directory '%1'. Please specify a valid directory for the build log.	

DOTJ017F	Fatal	No input file was specified; failed to initialize log name based on input file.	The the in log fi nam could
DOTJ018I	Informational	Log file '%1' was generated successfully in directory '%2'. Any messages from the transformation process are available in the log file; additional details about each message are available in the DITA-OT user guide.	
DOTJ020W	Warning	At least one plug-in in '%1' is required by plug-in '%2'. Plug-in '%2' cannot be loaded. Check and see whether all prerequisite plug-ins are installed in toolkit.	This insta anot corre in is plug
DOTJ021W	Warning	File '%1' will not generate output since it is invalid or all of its content has been filtered out by the ditaval file. Please check the file '%1' and the ditaval file to see if this is the intended result.	This conc of a to be mess filter to th build
DOTJ022F	Fatal	Failed to parse the input file '%1' because all of its content has been filtered out. This will happen if the input file has filter conditions on the root element, and a ditaval excludes all content based on those conditions.	Eithe file s your all co
DOTJ023E	Error	Failed to get the specified image file '%1', so it will not be included with your output.	Cheo in tho exist
			This appe •

DOTJ025E	Error	The input to the "topic merge" transform process could not be found. Correct any earlier transform errors and try the build again, or see the DITA-OT User Guide for additional causes.	•
DOTJ026E	Error	The "topic merge" did not generate any output. Correct any earlier transform errors and try the build again, or see the DITA-OT User Guide for additional causes.	This appe is dir topic error some some corre prog repo deve bug
DOTJ028E	Error	No format attribute was found on a reference to file '%1', which does not appear to be a DITA file. If this is not a	Whe file, t indic refer HTM

		DITA file, set the format attribute to an appropriate value, otherwise set the format attribute to "dita".	files may refe topic
DOTJ029I	Informational	No 'domains' attribute was found for element '<%1>'. This generally indicates that your DTD or Schema was not developed properly according to the DITA specification.	The spec help elen mes spec prop
DOTJ030I	Informational	No 'class' attribute for was found for element '<%1>'. This generally indicates that your DTD or Schema was not developed properly according to the DITA specification.	All s mus prov This spec not c
DOTJ031I	Informational	No specified rule for '%1' was found in the ditaval file. This value will use the default action, or a parent prop action if specified. To remove this message, you can specify a rule for '%1' in the ditaval file.	This inter cond spec is co
DOTJ033E	Error	No valid content is found in topicref '%1' during chunk processing. Please specify an existing and valid topic for the topicref.	
DOTJ034F	Fatal	Failed to parse the input file '%1' (the content of the file is not valid). If the input file '%1' does not have a DOCTYPE declaration, please make sure that all class attributes are present in the file.	DITA class elen defa Sche is us mus map
		The file "%1" referenced by "%2" is	This outs for e map

DOTJ035F	Fatal	outside the scope of the input dita/map directory. If you want to lower the severity level, please use the Ant parameter 'outer.control', and set the value to "warn" or "quiet". Otherwise, move the referenced file "%1" into the input dita/map directory.	dire resu to be outp DITA (oute gene OT (argu /gen
DOTJ036W	Warning	The file "%1" referenced by "%2" is outside the scope of the input dita/map directory. If you do not want to see the warning message, please use the Ant parameter 'outer.control', and set the value to "quiet". Otherwise, move the referenced file "%1" into the input dita/map directory.	This outsi for e map dire resu to be outp DITA (oute gene OT (argu /gen
DOTJ037W	Warning	The XML schema and DTD validation function of the parser is turned off. Please make sure the input is normalized DITA with class attributes included, otherwise it will not be processed correctly.	DITA class elem defa Sche DTD the c expli topic
DOTJ038E	Error	The tag "%1" is specialized from unrecognized metadata. Please make sure that tag "%1" is specialized from an existing metadata tag in the core DITA vocabulary.	This in cru elem docu com

			DITA
DOTJ039E	Error	There is no target specified for conref push action "pushafter". Found in file="%1", element="%2". Please add <elementname <br="" conref="pushtarget">conaction="mark"> before current element.</elementname>	Plea <u>Pusł</u> for d for tł
DOTJ040E	Error	An element uses the attribute conaction="replace", but a conref attribute is not found in the expected location. Found in file="%1", element="%2".	Plea <u>Pusł</u> for d for tł
DOTJ041E	Error	The attribute conref="%1" uses invalid syntax. The value should contain '#' followed by a topic or map ID, optionally followed by '/elemID' for a sub-topic element.	The URI elem on <u>U</u> DITA the c
DOTJ042E	Error	Two elements both use conref push to replace the target "%1". Please delete one of the duplicate "replace" actions.	The usec with Only repla push DITA infor push
DOTJ043W	Warning	The conref push function is trying to replace an element that does not exist (element "%1" in file "%2").	The actio maki corre exist base spec expe corre targe build
		There is a redundant conref action	Plea

DOTJ044W	Warning	"pushbefore". Found in file="%1", element="%2". Please make sure that "mark" and "pushbefore" occur in pairs.	Pus for c for t
DOTJ045I	Informational	The key "%1" is defined more than once in the same map file. The reference href="%2" is ignored.	No r keys this catc
DOTJ046E	Error	Conkeyref="%1" can not be resolved because it does not contain a key or the key is not defined. The build will use the conref attribute for fallback, if one exists.	See deta usa(
DOTJ047I	Informational	Unable to find key definition for keyref="%1", href may be used as fallback if it exists.	This help spec spec may
DOTJ049W	Warning	The attribute value %1="%3" on element "%2" does not comply with the specified subject scheme. According to the subject scheme map, the following values are valid for the %1 attribute: %4	A DI was avai attrii attrii the a
DOTJ050W	Warning	Found an <index-see> or <index-see- also> reference to the term '%1', but that term is not defined in the index.</index-see- </index-see>	The valu othe not o refer this Eclip defin term
			The elen exte not l verif corre

DOTJ051E	Error	Unable to load target for coderef "%1".	Note reference direct defa reference any the se that shou work para reference argun gene OT (argun /gene
DOTJ052E	Error	Code reference charset "%1" not supported. See the DITA-OT User guide for supported charset values on the format attribute.	The synt spec targe <u>func</u> expe
DOTJ053W	Warning	Input file '%1' is not valid DITA file name. Please check '%1' to see if it is correct. The extensions ".dita" or ".xml" are supported for DITA topics.	By d supp and man Spec your exte toolk exte
DOTJ054E	Error	Unable to parse invalid %1 attribute value "%2"	
DOTJ055E	Error	Invalid key name "%1".	

DOTJ056E	Error	Invalid xml:lang "%1".	
DOTJ057E	Error	The id attribute value "%1" is not unique within the topic that contains it.	
DOTJ058E	Error	Both <i>%1</i> and <i>%2</i> attributes defined. A single element may not contain both generalized and specialized values for the same attribute.	
DOTJ059E	Error	Invalid key scope name "%1".	
DOTJ060W	Warning	Key "%1" was used in conkeyref but is not bound to a DITA topic or map. Cannot resolve conkeyref value "%2" as a valid conref reference.	
DOTX001W	Warning	No string named '%1' was found for language '%2'. Using the default language '%3'. Add a mapping between default language and desired language for the string '%1'.	This such infor abov toolk string appe This toolk upda lang settii
DOTX002W	Warning	The title element or attribute in the ditamap is required for Eclipse output.	The requ gene Plea map outp
DOTX003I	Informational	The anchorref attribute should either reference another dita map or an Eclipse XML TOC file. The value '%1' does not appear to reference either.	Eclip to cc For t gene anch refer Eclip

			anot pres an E
DOTX004I	Informational	Found a navref element that does not reference anything. The navref element should either reference another dita map or an Eclipse XML file.	Eclip <nav Eclip foun does file;</nav
DOTX005E	Error	Unable to find navigation title for reference to '%1'. The build will use '%1' as the title in the Eclipse Table of Contents.	To re prov refer topic refer targe
DOTX006E	Error	Unknown file extension in href="%1". References to non-DITA resources should set the format attribute to match the resource (for example, 'txt', 'pdf', or 'html').	Set f iden the r docu docu exte exte
DOTX007I	Informational	Only DITA topics, HTML files, and images may be included in your compiled CHM file. The reference to "%1" will be ignored. To remove this message, you can set the toc="no" or processing-role="resource- only" attribute on your topicref.	The only infor CHN will r
DOTX008E	Error	File '%1' does not exist or cannot be loaded.	Ensi can of th have stan ('.dit origi file; the t than

DOTX008W	Warning	File '%1' cannot be loaded, and no navigation title is specified for the table of contents.	To fix spec map refer be a nam may stan ('.dita origii file; i the t than
DOTX009W	Warning	Could not retrieve a title from '%1'. Using '%2' instead.	No ti spec cont fallba
DOTX010E	Error	Unable to find target for conref="%1".	The URI DITA topic in the deta Note this I char topic '.xml exter may temp to the If the your sure build will r

			conr
DOTX011W	Warning	There is more than one possible target for the reference conref="%1". Only the first will be used. Remove the duplicate id in the referenced file.	Whe conr pull 1 the t the r the r mes: to us exter inste usec inclu direc origii
DOTX012W	Warning	When you conref another topic or an item in another topic, the domains attribute of the target topic must be equal to or a subset of the current topic's domains attribute. Put your target under an appropriate domain. You can see the messages guide for more help.	This shou logs.
DOTX013E	Error	A element with attribute conref="%1" indirectly includes itself, which results in an infinite loop.	This you refer that refer will r toolk conre corre sour of th have stand ('.dita origi

			file; i the t than
DOTX014E	Error	The attribute conref="%1" uses invalid syntax. Conref references to a map element should contain '#' followed by an ID, such as mymap.ditamap#mytopicrefid.	The URI elerr on <u>U</u> DITA the e
DOTX015E	Error	The attribute conref="%1" uses invalid syntax. The value should contain '#' followed by a topic or map ID, optionally followed by '/elemID' for a sub-topic element.	The URI elem on U DITA the e the r mesa to us exte inste usec inclu direc origi
DOTX016W	Warning	A reference to "%2" appears to reference a DITA document, but the format attribute has inherited a value of "%1". The document will not be processed as DITA.	This catcl DITA une> DITA the t To re form indic the r mes to us exte inste

			usec inclu direc origi
DOTX017E	Error	Found a link or cross reference with an empty href attribute (href=""). Remove the empty href attribute or provide a value.	Four href= emp servi caus tools remo
DOTX018I		The type attribute on a topicref was set to '%1', but the topicref references a more specific '%2' topic. Note that the type attribute cascades in maps, so the value '%1' may come from an ancestor topicref.	The inter the t refer may Gene optic will a value case more actua error unex
DOTX019W	Warning	The type attribute on a topicref was set to '%1', but the topicref references a '%2' topic. This may cause your links to sort incorrectly in the output. Note that the type attribute cascades in maps, so the value '%1' may come from an ancestor topicref.	The inter the t refer may Gene optic will a value case does

			may inap
DOTX020E	Error	Missing navtitle attribute or element for peer topic "%1". References must provide a local navigation title when the target is not a local DITA resource.	The dyna the t exte
DOTX021E	Error	Missing navtitle attribute or element for non-DITA resource "%1". References must provide a local navigation title when the target is not a local DITA resource.	The dyna the t reso
DOTX022W	Warning	Unable to retrieve navtitle from target: '%1'. Using linktext (specified in topicmeta) as the navigation title.	The title 1 inste crea <link <top< td=""></top<></link
DOTX023W	Warning	Unable to retrieve navtitle from target: '%1'.	If the ensu and Othe title, form appr
DOTX024E	Error	Missing linktext and navtitle for peer topic "%1". References must provide a local navigation title when the target is not a local DITA resource.	The dyna link t local reso
DOTX025E	Error	Missing linktext and navtitle for non-DITA resource "%1". References must provide a local navigation title when the target is not a local DITA resource.	The dyna the t reso
DOTX026W	Warning	Unable to retrieve linktext from target: '%1'. Using navigation title as fallback.	The did r gene navi(fallba The

DOTX027W	Warning	Unable to retrieve linktext from target: '%1'.	spec gene no fa loca from inco
DOTX028E	Error	Link or cross reference must contain a valid href or keyref attribute; no link target is specified.	The no ta gene
DOTX029I	Informational	The type attribute on a %1 element was set to %3, but the reference is to a more specific %4 %2. This may cause your links to sort incorrectly in the output.	The inter the t refer may Gen optic will a valu case more actu erro unex this
DOTX030W	Warning	The type attribute on a %1 element was set to %3, but the reference is to a %4 %2. This may cause your links to sort incorrectly in the output.	The inter the t refer may Gen optic will a valu case does may inap

DOTX031E	Error	The file %1 is not available to resolve link information.	the s retrive desc be for poss usec conte awar above your
DOTX032E	Error	Unable to retrieve link text from target: '%1'. If the target is not accessible at build time, or does not have a title, provide the link text inside the reference.	Whe does will a title f targe set th appr does when sure the c
DOTX033E	Error	Unable to generate link text for a cross reference to a list item: '%1'	An < type: to a num to us spec refer are r item
DOTX034E	Error	Unable to generate link text for a cross reference to an undered list item: '%1'	The list it proc gene list it prov refer

DOTX035E	Error	Unable to generate the correct number for a cross reference to a footnote: '%1'	type: to a num to us spec refer are r footr
DOTX036E	Error	Unable to generate link text for a cross reference to a dlentry (the dlentry or term could not be found): '%1'	An < type: link t the t use a link t ensu an a
DOTX037W	Warning	No title found for this document; using "***" in XHTML title bar.	No ti docu file v This the t brow
DOTX038I	Informational	The longdescref attribute on tag '%1' will be ignored. Accessibility for object elements needs to be handled another way.	The does long mak may after be a <par obje</par
DOTX039W	Warning	Required cleanup area found. To remove this message and hide the content, build your content without using the DRAFT parameter.	This wher orde topic up; t appe stylir

			The the
DOTX040I	Informational	Draft comment area found. To remove this message and hide the comments, build your content without using the DRAFT parameter.	This whe
DOTX041W	Warning	Found more than one title element in a section. Using the first one for the section's title.	Bec DIT not sec edit corr spe title Whe only the
DOTX042I	Informational	supported for inline phrases in XHTML; ignoring flag value on '%1' attribute.	If it of ir on t con war the ima doc
DOTX043I	Informational	once in '%2'	The dup Hov san attri pos toge san role

DOTX044E	Error	The area element in an image map does not specify a link target. Please add an xref element with a link target to the area element.	spec both a rel gene DITA attrik The map for th add child it spe
DOTX045W	Warning	The area element in an image map should specify link text for greater accessibility. Link text should be specified directly when the target is not a local DITA resource.	Cros <are prov read diffei</are
DOTX046W	Warning	Area shape should be: default, rect, circle, poly, or blank (no value). The value '%1' is not recognized.	
DOTX047W	Warning	Area coordinates are blank. Coordinate points for the shape need to be specified.	The defir map spec regic
DOTX048I	Informational	In order to include peer or external topic '%1' in your help file, you may need to recompile the CHM file after making the	The exter your be ir refer that canr proje

		file available.	toc <i>a</i> topic chec was was in the othe
DOTX049I		References to non-dita files will be ignored by the PDF, ODT, and RTF output transforms.	The proc conv DITA the r refer
DOTX050W	Warning	Default id "org.sample.help.doc" is used for Eclipse plug-in. If you want to use your own plug-in id, please specify it using the id attribute on your map.	Eclip spec Eclip expe root
DOTX052W	Warning	No string named '%1' was found when creating generated text; using the value '%1' in your output file.	The gene string appe requ foun outp string that string error over requ neec the s you
		A element that references another map indirectly includes itself, which results in	This refer then anot

DOTX053E	Error	an infinite loop. The original map reference is to '%1'.	refer resu map of m circu
DOTX054W	Warning	Conflict text style is applied on the current element based on DITAVAL flagging rules. Please check ditaval and dita source to make sure there is no style conflict on the element which needs to be flagged.	This file c rules elem
DOTX055W	Warning	Customized stylesheet uses deprecated template "flagit". Conditional processing is no longer supported using this template. Please update your stylesheet to use template "start-flagit" instead of deprecated template "flagit".	The depr 1.4, form The DITA usec upda
DOTX056W	Warning	The file '%1' is not available to resolve link information.	The the s retriv desc be fc poss usec conte poss locat the r was onlyt /only spec path matc
		The link or cross reference target '%1' cannot be found, which may cause errors	The synta elerr

DOTX057W	Warning	creating links or cross references in your output file.	canr that not r DITA
DOTX058W	Warning	No glossary entry was found associated with key '%1' on %2 element. The build will try to determine the best display text and hover text for terms and abbreviations.	Proc or al asso elem glos This key asso topic the b (usu refer
DOTX060W	Warning	Key '%1' was used in an abbreviated-form element, but the key is not associated with a glossary entry. Abbreviated-form should ONLY be used to reference to a glossary entry.	Proc elem from attrik (glos mes was asso topic supp asso the e any refer elem
DOTX061W	Warning	ID '%1' was used in topicref tag but did not reference a topic element. The href attribute on a topicref element should only	Acco Spea map Map DITA belo

		reference topic level elements.	(usir of a href OAS the <u>t</u>
DOTX062I	Informational	It appears that this document uses constraints, but the conref processor cannot validate that the target of a conref is valid. To enable constraint checking, please upgrade to an XSLT 2.0 processor.	
DOTX063W	Warning	The dita document '%1' is linked to from your content, but is not referenced by a topicref tag in the ditamap file. Include the topic in your map to avoid a broken link.	This gene that topic not r will r shou map build
DOTX064W	Warning	The copy-to attribute [copy-to="%1"] uses the name of a file that already exists, so this attribute is ignored.	The copy that sure use copy cont
DOTX065W	Warning	Two unique source files each specify copy-to="%2", which results in a collision. The value associated with href="%1" is ignored.	Two to th to; a woul first i value corre attrik
		Template "%1" is deprecated. Remove	This your rely remc

DOTX066W	Warning	references to this template from your custom XSLT or plug-ins.	relea whei beer temr depr upda
DOTX067E	Error	No string named '%1' was found for language '%2'. Add a mapping for the string '%1'.	This text, "Rela gene grou to loo spec will <i>a</i> langu indic neec your langu
DOTX068W	Warning	A topicref element that references a map contains child topicref elements. Child topicref elements are ignored.	
PDFJ001E	Error	Index entry '%1' is dropped, because corresponding group is not found.	
PDFJ002E	Error	Build stopped. Problems occured during Index preprocess task. Please check the messages above.	
PDFX001W	Warning	There is no index entry found which closing range for ID="%1".	
PDFX002W	Warning	There are multiple index entry found which is opening range for ID="%1" but there is only one which close it or ranges are overlapping.	
PDFX003W	Warning	There are multiple index entry found which closing range for ID="%1".	
PDFX004F	Fatal	Empty href was specified for some topic reference. Please correct your ditamap or bookmap file.	

PDFX005F	Fatal	Topic reference (href : %1) not found. Reference may be incorrect. Please correct your ditamap or bookmap file.	
PDFX006E	Error	Number of columns must be specified.	
PDFX007W	Warning	There is no index entry found which opening range for ID="%1".	
PDFX008W	Warning	Font definition not found for the logical name or alias '%1'.	
PDFX009E	Error	Attribute set reflection can't handle XSLT element %1.	
PDFX010W	Warning	Index generation is not supported in FOP.	
PDFX011E	Error	Both index-see and %1 defined for index entry '%2'. Recovering by treating the index-see as an index-see-also.	
PDFX012F	Fatal	Unrecognized PDF formatter '%1'. Use "fop" for FOP, "xep" for RenderX XEP, or "ah" for AH Formatter.	
XEPJ001W	Warning	%1	
XEPJ002E	Error	%1	
XEPJ003E	Error	%1	

Other error messages

In addition to error messages generated by the DITA Open Toolkit, you might also encounter error messages generated by Java or other tools.

Out of Memory error

In some cases, you might receive a message stating the build has failed due to an out of Memory error. Try the following approaches to resolve the problem:

- 1. Increase the memory available to Java; see <u>Increasing Java memory</u> <u>allocation</u>.
- Reduce memory consumption by setting the generate-debugattributes option to false. This option is set in the lib/configuration.properties file. This will disable debug attribute generation (used to trace DITA-OT error messages back to source files) and will reduce memory consumption.
- 3. Set dita.preprocess.reloadstylesheet Ant property to true. This will allow the XSLT processor to release memory when converting multiple files.
- 4. Run the transformation again.

java.io.IOException: Can't store Document

After running a JavaHelp transformation, you may receive a java.io.IOException: Can't store Document Message.

This problem occurs when HTML files unrelated to the current transformation are found in the output directory. Delete the content of the output directory and run the transformation again.

Stack Overflow error

If you receive an error about a stack memory overflow, increase the JVM and run the transformation again. See <u>Increasing Java memory</u> <u>allocation</u>.

Log files

When you run the DITA-OT, key information is logged on the screen. This information can also be written to a log file. If you encounter a problem, you can analyze this information to determine the source of the problem and then take action to resolve it.

The logging behavior varies depending on whether you use the dita command, DITA-OT command-line tool, or Ant to invoke a toolkit build.

dita command

By default, only warning and error messages are written to the screen. If you use the **-v** option, logging will be more verbose and informative messages are also written out. The **-I** option can be used to write the log messages into a file.

Ant

By default, status information is written to the screen. If you issue the **-I** parameter, the build runs silently and the information is written to a log file with the name and location that you specified. (You also can use other Ant loggers; see the Ant documentation for more information.)

Command-line tool

Status information is written to the screen and the log file. The log file name contains the input file name and transformation type; by default, it is located in the output directory. If you issue the *llogdir* parameter, you can specify a different location for where the log file is written.

Accessing help from the command-line tool

You can access a list of supported parameters for the command-line tool by issuing the **-help** parameter.

Procedure

- 1. Run the startcmd file that is applicable for your operating system. The startcmd.bat and startcmd.sh files are in the directory where you installed the DITA-OT.
- 2. From the command prompt, issue the following command:

```
java -jar lib/dost.jar -help
```

Results

You can see the brief description of the supported parameters in the command-line window.

Determing the version of the DITA Open Toolkit

You can use the command-line tool to determine the version of the DITA OT.

Procedure

- 1. Run the startcmd file that is applicable for your operating system. The startcmd.bat and startcmd.sh files are in the directory where you installed the DITA-OT.
- 2. From the command prompt, issue the following command:

```
java -jar lib/dost.jar -version
```

Enabling debug mode

When the debug mode is enabled, additional diagnostic information is written to the log file. This information, which includes environment variables and stack trace data, can help you determine the root cause of a problem.

Procedure

From the command prompt, add the following parameters:

Application	Parameters
dita command	-d or -debug
Ant	-v -Dargs.debug=yes
Command-line too	/d Or -debug

You also can add a <property> element to an Ant target in your build file, for example:

```
<property name="args.debug" value="yes"/>
```

Increasing Java memory allocation

If you are working with large documents with extensive metadata or key references, you will need to increase the memory allocation for the Java process. You can do this from the command-line prompt for a specific session, or you can increase the value of the ANT_OPTS environmental variable.

Procedure

• To change the value for an specific session, from the command prompt, issue the following command:

Platform	Command
Windows	set ANT_OPTS=%ANT_OPTS% -Xmx <i>1024</i> M
Linux/OS	X export ANT_OPTS=\$ANT_OPTS -Xmx1024M

This increases the JVM memory allocation to 1024 megabytes. The amount of memory which can be allocated is limited by available system memory and the operating system.

• To persistently change the value, change the value allocated to the ANT_OPTS environment variable on your system. If you use the startcmd file from the Full Easy Install to set up a toolkit session, edit that file to change the value.

Parent topic: Error messages and troubleshooting

Reference

This section is designed to help users to locate information easily and quickly. It includes documentation for the DITA Open Toolkit (DITA-OT) parameters and configuration properties.

Ant parameters Certain parameters apply to all DITA-OT transformations. Other parameters are common to the HTML-based transformations. Finally, some parameters apply only to the specific transformation types.

dita command arguments and options

The dita command takes mandatory arguments to process DITA, manage plug-in, or print information about the command. Options can be used modify the command behaviour or provide additional configuration.

Command-line tool parameters

Certain parameters apply to all DITA-OT transformations. Other parameters are common to the HTML-based transformations. Finally, some parameters apply only to the specific transformation types.

lib/configuration.properties file

The lib/configuration.properties file controls certain common properties, as well as some properties that control PDF processing.

Parent topic: DITA Open Toolkit User Guide

Ant parameters

Certain parameters apply to all DITA-OT transformations. Other parameters are common to the HTML-based transformations. Finally, some parameters apply only to the specific transformation types.

<u>Ant parameters: All transformations</u> Certain parameters apply to all transformations that are supported by the DITA Open Toolkit.

Ant parameters: Common HTML-based transformations

Certain parameters apply to all the HTML-based transformation types: Eclipse help, HTML Help, JavaHelp, TocJS, HTML5, and XHTML.

Ant parameters: Eclipse content transformation

Certain parameters are specific to the Eclipse content transformation.

Ant parameters: Eclipse help transformation

Certain parameters are specific to the Eclipse help transformation.

Ant parameters: HTML Help transformation

Certain parameters are specific to the HTML Help transformation.

Ant parameters: JavaHelp transformation

Certain parameters are specific to the JavaHelp transformation.

Ant parameters: ODT transformation

Certain parameters are specific to the ODT transformation.

Ant parameters: Other

These Ant parameters enable you to reload style sheets that the DITA-OT uses for specific pre-processing stages.

Ant parameters: PDF transformation Certain parameters are specific to the PDF2 transformation.

Ant parameters: HTML5 and XHTML transformation

Certain parameters are specific to the HTML5 and XHTML transformation.

Parent topic: Reference

Related concepts

Related tasks Publishing DITA content from Ant

Ant parameters: All transformations

Certain parameters apply to all transformations that are supported by the DITA Open Toolkit.

Figure 1. Ant parameters: All transformations

args.debug

Specifies whether debugging information is included in the log. The allowed values are **yes** and **no**; the default value is **no**.

args.draft

Specifies whether the content of <draft-comment> and <required-cleanup> elements is included in the output. The allowed values are **yes** and **no**; the default value is **no**. Corresponds to XSLT parameter **DRAFT** in most XSLT modules.

Tip: For PDF output, setting the **args.draft** parameter to **yes** causes the contents of the <titlealts> element to be rendered below the title.

args.figurelink.style

Specifies how cross references to figures are styled in output. The allowed values are **NUMBER** and **TITLE**. Specifying **NUMBER** results in "Figure 5"; specifying **TITLE** results in the title of the figure. Corresponds to the XSLT parameter **FIGURELINK**.

Note: Support for PDF was added in DITA-OT 2.0. By default PDF uses the value **NUMTITLE**, which is not supported for other transform types; this results in "Figure 5. Title".

args.filter

Specifies a filter file to be used to include, exclude, or flag

content.

args.grammar.cache

Specifies whether the grammar-caching feature of the XML parser is used. The allowed values are **yes** and **no**; the default value is **no**.

Note: This option dramatically speeds up processing time. However, there is a known problem with using this feature for documents that use XML entities. If your build fails with parser errors about entity resolution, set this parameter to **no**.

args.input

Specifies the master file for your documentation project. Typically this is a DITA map, however it also can be a DITA topic if you want to transform a single DITA file. The path can be absolute, relative to **args.input.dir**, or relative to the directory where your project's ant build script resides if **args.input.dir** is not defined.

args.input.dir

Specifies the base directory for your documentation project. The default value is the parent directory of the file specified by **args.input**.

args.logdir

Specifies the location where the DITA-OT places log files for your project.

args.rellinks

Specifies which links to include in the output. The following values are supported:

• **none** – No links are included.

- **all** All links are included.
- **nofamily** Parent, child, next, and previous links are not included.

Default value depends on the transformation type.

args.tablelink.style

Specifies how cross references to tables are styled. Specifying **NUMBER** results in "Table 5"; specifying **TITLE** results in the title of the table. Corresponds to the XSLT parameter **TABLELINK**.

Note: Support for PDF was added in DITA-OT 2.0. By default PDF uses the value **NUMTITLE**, which is not supported for other transform types; this results in "Table 5. Title".

clean.temp

Specifies whether the DITA-OT deletes the files in the temporary directory after it finishes a build. The allowed values are **yes** and **no**; the default value is **yes**.

dita.dir

Specifies where the DITA-OT is installed.

dita.temp.dir

Specifies the location of the temporary directory. The temporary directory is where the DITA-OT writes temporary files that are generated during the transformation process.

dita.input.valfile

Specifies a filter file to be used to include, exclude, or flag content. Filter files must have a .ditaval or .DITAVAL extension.

Note: Deprecated in favor of the args.filter parameter.

generate.copy.outer

Specifies whether to generate output files for content that is not located in or beneath the directory containing the DITA map file. The following values are supported:

- **1** (default) Do not generate output for content that is located outside the DITA map directory.
- **3** Shift the output directory so that it contains all output for the publication.

See <u>generate.outer.copy parameter</u> for more information.

onlytopic.in.map

Specifies whether files that are linked to, or referenced with a @conref attribute, generate output. If set to **yes**, only files that are referenced directly from the map will generate output.

outer.control

Specifies how the DITA OT handles content files that are located in or below the directory containing the master DITA map. The following values are supported:

- **fail** Fail quickly if files are going to be generated or copied outside of the directory
- warn (default) Complete the operation if files will be generated or copied outside of the directory, but log a warning
- **quiet** Quietly finish with only those files; do not generate warnings or errors.

The **gen-list-without-flagging** Ant task generates a harmless warning for content that is located outside the map directory; you can suppress these warnings by setting the property to **quiet**.

Warning: Microsoft HTML Help Compiler cannot produce HTML Help for documentation projects that use outer content. The content files must reside in or below the directory containing the master DITA map file, and the map file cannot specify ".." at the start of the @href attributes for <topicref> elements.

output.dir

Specifies the name and location of the output directory. By default, the output is written to *DITA-dir*\out.

transtype

Specifies the output format. You can create plug-ins to add new values for this parameter; by default, the following values are available:

- docbook
- eclipsehelp
- eclipsecontent
- html5
- htmlhelp
- javahelp
- odt
- pdf
- wordrtf
- troff
- xhtml

validate

Specifies whether the DITA-OT validates the content. The allowed values are **true** and **false**; the default value is **true**.

Parent topic: Ant parameters

Ant parameters: Common HTML-based transformations

Certain parameters apply to all the HTML-based transformation types: Eclipse help, HTML Help, JavaHelp, TocJS, HTML5, and XHTML.

Figure 1. Ant parameters: HTML-based transformations

args.artlbl

Specifies whether to generate a label for each image; the label will contain the image file name. The allowed values are **yes** and **no**; the default value is **no**.

args.breadcrumbs

Specifies whether to generate breadcrumb links. The allowed values are **yes** and **no**; the default value is **no**. Corresponds to the XSLT parameter **BREADCRUMBS**.

args.copycss

Specifies whether to copy the custom .css file to the output directory.

args.css

Specifies the name of a custom .css file.

args.csspath

Specifies the location of a copied .css file relative to the output directory. Corresponds to XSLT parameter **CSSPATH**.

args.cssroot

Specifies the directory that contains the custom .css file. DITA-OT will copy the file from this location.

args.dita.locale

Specifies the language locale file to use for sorting index entries.

Note: This parameter is not available for the XHTML transformation.

args.ftr

Specifies an XML file that contains content for a running footer. Corresponds to XSLT parameter **FTR**.

Note: The XML file must contain valid XML. A common practice is to place all content into a <div> element.

args.gen.default.meta

Specifies whether to generate extra metadata that targets parental control scanners, meta elements with name="security" and name="Robots". The allowed values are **yes** and **no**; the default value is **no**. Corresponds to the XSLT parameter **genDefMeta**.

args.gen.task.lbl

Specifies whether to generate headings for sections within task topics. The allowed values are **YES** and **NO**; the default value is **NO**. Corresponds to the XSLT parameter **GENERATE-TASK-LABELS**.

args.hdf

Specifies an XML file that contains content to be placed in the document head.

args.hdr

Specifies an XML file that contains content for a running header. Corresponds to the XSLT parameter **HDR**.

Note: The XML file must contain valid XML. A common practice is to place all content into a <div> element.

args.hide.parent.link

Specifies whether to hide links to parent topics in the HTML or XHTML output. The allowed values are **yes** and **no**; the default value is **no**. Corresponds to the XSLT parameter **NOPARENTLINK**.

Note: This parameter is deprecated in favor of the **args.rellinks** parameter.

args.indexshow

Specifies whether the content of <indexterm> elements are rendered in the output. The allowed values are **yes** and **no**; the default value is **no**.

args.outext

Specifies the file extension for HTML or XHTML output. The default value is **html**. Corresponds to XSLT parameter **OUTEXT**.

args.xhtml.classattr

Specifies whether to include the DITA class ancestry inside the XHTML elements .For example, the <prereq> element (which is specialized from section) would generate class="section prereq. The allowed values are **yes** and **no**; the default value is **yes**. Corresponds to the XSLT parameter **PRESERVE-DITA-CLASS**.

Note: Beginning with DITA OT release 1.5.2, the default value is **yes**. For release 1.5 and 1.5.1, the default value was **no**.

args.xsl

Specifies a custom XSL file to be used instead of the default XSL transformation (xsl\dita2xhtml.xsl). The parameter must specify a fully qualified file name.

generate.outer.copy parameter Elaboration on how the generate.outer.copy parameter functions.

Parent topic: Ant parameters

Related reference

Ant parameters: Eclipse content transformation Ant parameters: Eclipse help transformation Ant parameters: HTML Help transformation Ant parameters: JavaHelp transformation Ant parameters: HTML5 and XHTML transformation

Ant parameters: Eclipse content transformation

Certain parameters are specific to the Eclipse content transformation.

Figure 1. Ant parameters: Eclipse content transformation

args.eclipsecontent.toc

Specifies the name of the TOC file

Parent topic: Ant parameters

Related concepts

Eclipse content transformation

Related reference

Ant parameters: All transformations Ant parameters: Common HTML-based transformations

Ant parameters: Eclipse help transformation

Certain parameters are specific to the Eclipse help transformation.

Figure 1. Ant parameters: Eclipse help transformation

args.eclipsehelp.toc

Specifies the name of the TOC file.

args.eclipse.country

Specifies the region for the language that is specified by the **args.eclipse.language** parameter. For example, **us**, **ca**, and **gb** would clarify a value of **en** set for the **args.eclipse.language** parameter. The content will be moved into the appropriate directory structure for an Eclipse fragment.

args.eclipse.language

Specifies the base language for translated content, such as **en** for English. This parameter is a prerequisite for the **args.eclipse.country** parameter. The content will be moved into the appropriate directory structure for an Eclipse fragment.

args.eclipse.provider

Specifies the name of the person or organization that provides the Eclipse help. The default value is **DITA**.

Tip: The toolkit ignores the value of this parameter when it processes an Eclipse map.

args.eclipse.version

Specifies the version number to include in the output. The default value is **0.0.0**.

Tip: The toolkit ignores the value of this parameter when it processes an Eclipse map.

args.eclipse.symbolic.name

Specifies the symbolic name (aka plugin ID) in the output for an Eclipse Help project. The @id value from the DITA map or the Eclipse map collection (Eclipse help specialization) is the symbolic name for the plugin in Eclipse. The default value is **org.sample.help.doc**.

Tip: The toolkit ignores the value of this parameter when it processes an Eclipse map.

Parent topic: Ant parameters

Related concepts Eclipse help transformation

Related reference

Ant parameters: All transformations Ant parameters: Common HTML-based transformations

Ant parameters: HTML Help transformation

Certain parameters are specific to the HTML Help transformation.

Figure 1. Ant parameters: HTML Help transformation

args.htmlhelp.includefile

Specifies the name of a file that you want included in the HTML Help.

Parent topic: Ant parameters

Related concepts HTML help transformation

Related reference Ant parameters: All transformations Ant parameters: Common HTML-based transformations

Ant parameters: JavaHelp transformation

Certain parameters are specific to the JavaHelp transformation.

Figure 1. Ant parameters: JavaHelp transformation

args.javahelp.map

Specifies the name of the ditamap file for a JavaHelp project.

args.javahelp.toc

Specifies the name of the file containing the TOC in your JavaHelp output. Default value is the name of the ditamap file for your project.

Parent topic: Ant parameters

Related concepts JavaHelp transformation

Related reference

Ant parameters: All transformations Ant parameters: Common HTML-based transformations

Ant parameters: ODT transformation

Certain parameters are specific to the ODT transformation.

Figure 1. Ant parameters: ODT transformation

args.odt.img.embed

Determines whether images are embedded as binary objects within the ODT file.

Parent topic: Ant parameters

Related concepts ODT transformation

Related reference Ant parameters: All transformations

Ant parameters: Other

These Ant parameters enable you to reload style sheets that the DITA-OT uses for specific pre-processing stages.

Figure 1. Ant parameters: Other

dita.preprocess.reloadstylesheet dita.preprocess.reloadstylesheet.conref dita.preprocess.reloadstylesheet.mapref dita.preprocess.reloadstylesheet.mappull dita.preprocess.reloadstylesheet.maplink dita.preprocess.reloadstylesheet.topicpull

Specifies whether the DITA-OT reloads the XSL style sheets that are used for the transformation. The allowed values are **true** and **false**; the default value is **false**.

Tip: Set the parameter to **true** if you want to use more than one set of style sheets to process a collection of topics. The parameter also is useful for large projects that generate Java out-of-memory errors during transformation. Alternatively, you can adjust the size of your Java memory heap if setting dita.preprocess.reloadstylesheet for this reason.

Parent topic: Ant parameters

Ant parameters: PDF transformation

Certain parameters are specific to the PDF2 transformation.

Figure 1. Ant parameters: PDF transformation

args.bookmap-order

Specifies if the frontmatter and backmatter content order is retained in bookmap. The allowed values are **retain** and **discard**; the default value is **discard**.

args.fo.userconfig

Specifies the user configuration file for FOP.

args.gen.task.lbl

Specifies whether to generate headings for sections within task topics. The allowed values are **YES** and **NO**; the default value is **NO**. Corresponds to the XSLT parameter **GENERATE-TASK-LABELS**.

args.rellinks

Specifies which links to include in the output. The following values are supported:

- **none** No links are included.
- **all** All links are included.
- **nofamily** Parent, child, next, and previous links are not included.

Default value depends on the transformation type.

args.xsl.pdf

Specifies an XSL file that is used to override the default XSL transformation

(plugins\org.dita.pdf2\xsl\fo\topic2fo_shell.xsl). You must specify the fully qualified file name.

custom.xep.config

Specifies the user configuration file for RenderX.

customization.dir

Specifies the customization directory.

pdf.formatter

Specifies the XSL processor. The following values are supported:

- **ah** Antenna House Formatter
- fop (default) Apache FOP
- xep RenderX XEP Engine

The full-easy-install package comes with Apache FOP; other XSL processors must be separately installed.

publish.required.cleanup

Specifies whether draft-comment and required-cleanup elements are included in the output. The allowed values are **yes** and **no**; the default value is the value of the **args.draft** parameter. Corresponds to XSLT parameter **publishRequiredCleanup**.

Note: This parameter is deprecated in favor of the **args.draft** parameter.

retain.topic.fo

Specifies whether to retain the generated FO file. The allowed values are **yes** and **no**; the default value is **no**. If the configuration property **org.dita.pdf2.use-out-temp** is set to

false, this parameter is ignored.

Parent topic: Ant parameters

Related concepts

PDF transformation

Related reference

Ant parameters: All transformations

Ant parameters: HTML5 and XHTML transformation

Certain parameters are specific to the HTML5 and XHTML transformation.

Figure 1. Ant parameters: HTML5 and XHTML transformation

args.xhtml.contenttarget

Specifies the value of the @target attribute on the <base> element in the TOC file. The default value is **contentwin**.

args.xhtml.toc

Specifies the base name of the TOC file. The default value is **index**.

args.xhtml.toc.class

Specifies the value of the @class attribute on the <body> element in the TOC file. Found in map2htmltoc.xsl.

Parent topic: Ant parameters

Related concepts

XHTML transformation HTML5 transformation

Related reference

Ant parameters: All transformations Ant parameters: Common HTML-based transformations

dita command arguments and options

The dita command takes mandatory arguments to process DITA, manage plug-in, or print information about the command. Options can be used modify the command behaviour or provide additional configuration.

Usage

dita -f name -i file [options] dita -install { file | url }

dita -uninstall *id*

dita -help

dita -version

Arguments

-f, -format name

Specifies the output format. You can create plug-ins to add new values for this parameter; by default, the following values are available:

- docbook
- eclipsehelp
- eclipsecontent
- html5
- htmlhelp
- javahelp
- odt
- pdf
- wordrtf
- troff
- xhtml

-i, -input file

Specifies the master file for your documentation project. Typically this is a DITA map, however it also can be a DITA topic if you want to transform a single DITA file. The path can be absolute, relative to **args.input.dir**, or relative to the directory where your project's ant build script resides if **args.input.dir** is not defined.

-install file -install url Install plug-in from a local ZIP file or from a URL.

-uninstall *id*

Uninstall plug-in with the ID.

-h, -help

Print command usage help.

-version

Print version information and exit.

Options

-o, -output dir

Specifies the name and location of the output directory. By default, the output is written to out subdirectory of the current directory.

-filter file

Specifies a filter file to be used to include, exclude, or flag content.

-temp dir

Specifies the location of the temporary directory. The temporary directory is where the DITA-OT writes temporary files that are generated during the transformation process.

-v, -verbose

Verbose logging.

-d, -debug

Debug logging.

-I, -logfile file

Output logging messages into a file.

-Dproperty=value

Specify a value for a property. Supported properties are the same as Ant parameters and are transformation type specific.

-propertyfile file

Load all properties from a file. Properties specified with **-D** option take precedence.

Parent topic: Reference

Command-line tool parameters

Certain parameters apply to all DITA-OT transformations. Other parameters are common to the HTML-based transformations. Finally, some parameters apply only to the specific transformation types.

You must supply the parameters to the command-line tool using the following syntax:

/parameter:value

Command-line tool parameters: All transformations Certain parameters apply to all transformations that are supported by the DITA Open Toolkit.

Command-line tool parameters: All HTML-based transformations

Certain parameters apply to all the HTML-based transformation types: Eclipse content, Eclipse help, HTML5, HTML Help, JavaHelp, TocJS, and XHTML.

<u>Command-line tool parameters: Eclipse content transformation</u> Certain parameters are specific to the Eclipse content transformation.

Command-line tool parameters: Eclipse help transformation Certain parameters are specific to the Eclipse help transformation.

<u>Command-line tool parameters: HTML help transformation</u> Certain parameters are specific to the HTML help transformation.

<u>Command-line tool parameters: JavaHelp transformation</u> Certain parameters are specific to the JavaHelp transformation.

<u>Command-line tool parameters: ODT transformation</u> Certain parameters are specific to the ODT transformation.

Command-line tool parameters: PDF transformation

Certain parameters are specific to the PDF2 transformation.

Command-line tool parameters: HTML5 and XHTML transformation

Certain parameters are specific to the HTML5 and XHTML transformation.

Parent topic: Reference

Related concepts Command-line tool

Related tasks Publishing DITA content from the command-line tool

Command-line tool parameters: All transformations

Certain parameters apply to all transformations that are supported by the DITA Open Toolkit.

Figure 1. Command-line tool parameters: All transformations

basedir

The directory where your project's ant build script resides. The DITA-OT will look for your .dita files relative to this directory. DITA-OT's default build script sets this as an attribute of the project, but you can also define it as a project property.

cleantemp

Specifies whether the DITA-OT deletes the files in the temporary directory after it finishes a build. The allowed values are **yes** and **no**; the default value is **yes**.

debug

Specifies whether debugging information is included in the log. The allowed values are **yes** and **no**; the default value is **no**.

ditadir

Specifies where the DITA-OT is installed.

ditalocale

Specifies the language locale file to use for sorting index entries.

Note: This parameter is not available for the XHTML transformation.

draft

Specifies whether the content of <draft-comment> and <required-cleanup> elements is included in the output. The

allowed values are **yes** and **no**; the default value is **no**. Corresponds to XSLT parameter **DRAFT** in most XSLT modules.

Tip: For PDF output, setting the **args.draft** parameter to **yes** causes the contents of the <titlealts> element to be rendered below the title.

filter

Specifies a filter file to be used to include, exclude, or flag content. Filter files must have a .ditaval or .DITAVAL extension.

Note: Deprecated in favor of the args.filter parameter.

grammarcache

Specifies whether the grammar-caching feature of the XML parser is used. The allowed values are **yes** and **no**; the default value is **no**.

Note: This option dramatically speeds up processing time. However, there is a known problem with using this feature for documents that use XML entities. If your build fails with parser errors about entity resolution, set this parameter to **no**.

i.

Specifies the master file for your documentation project. Typically this is a DITA map, however it also can be a DITA topic if you want to transform a single DITA file. The path can be absolute, relative to **args.input.dir**, or relative to the directory where your project's ant build script resides if **args.input.dir** is not defined.

logdir

Specifies the location where the DITA-OT places log files for your project.

outdir

Specifies the name and location of the output directory. By default, the output is written to *DITA-dir*\out.

tempdir

Specifies the location of the temporary directory. The temporary directory is where the DITA-OT writes temporary files that are generated during the transformation process.

transtype

Specifies the output format. You can create plug-ins to add new values for this parameter; by default, the following values are available:

- docbook
- eclipsehelp
- eclipsecontent
- html5
- htmlhelp
- javahelp
- odt
- pdf
- wordrtf
- troff
- xhtml

validate

Specifies whether the DITA-OT validates the content. The allowed values are **true** and **false**; the default value is **true**.

Parent topic: Command-line tool parameters

Command-line tool parameters: All HTML-based transformations

Certain parameters apply to all the HTML-based transformation types: Eclipse content, Eclipse help, HTML5, HTML Help, JavaHelp, TocJS, and XHTML.

Note: You must specify an absolute path as the value for the following parameters:

- ftr
- hdr
- hdf

Figure 1. Command-line tool parameters: All HTML-based transformations

args.css

Specifies the name of a custom .css file.

artlbl

Specifies whether to generate a label for each image; the label will contain the image file name. The allowed values are **yes** and **no**; the default value is **no**.

copycss

Specifies whether to copy the custom .css file to the output directory.

csspath

Specifies the location of a copied .css file relative to the output directory. Corresponds to XSLT parameter **CSSPATH**.

cssroot

Specifies the directory that contains the custom .css file. DITA-OT will copy the file from this location.

ftr

Specifies an XML file that contains content for a running footer. Corresponds to XSLT parameter **FTR**.

Note: The XML file must contain valid XML. A common practice is to place all content into a <div> element.

generateouter

Specifies whether to generate output files for content that is not located in or beneath the directory containing the DITA map file. The following values are supported:

- 1 (default) Do not generate output for content that is located outside the DITA map directory.
- **3** Shift the output directory so that it contains all output for the publication.

See <u>generate.outer.copy parameter</u> for more information.

hdf

Specifies an XML file that contains content to be placed in the document head.

hdr

Specifies an XML file that contains content for a running header. Corresponds to the XSLT parameter **HDR**.

Note: The XML file must contain valid XML. A common practice is to place all content into a <div> element.

indexshow

Specifies whether the content of <indexterm> elements are

rendered in the output. The allowed values are **yes** and **no**; the default value is **no**.

onlytopicinmap

Specifies whether files that are linked to, or referenced with a @conref attribute, generate output. If set to **yes**, only files that are referenced directly from the map will generate output.

outercontrol

Specifies how the DITA OT handles content files that are located in or below the directory containing the master DITA map. The following values are supported:

- **fail** Fail quickly if files are going to be generated or copied outside of the directory
- warn (default) Complete the operation if files will be generated or copied outside of the directory, but log a warning
- **quiet** Quietly finish with only those files; do not generate warnings or errors.

The **gen-list-without-flagging** Ant task generates a harmless warning for content that is located outside the map directory; you can suppress these warnings by setting the property to **quiet**.

Warning: Microsoft HTML Help Compiler cannot produce HTML Help for documentation projects that use outer content. The content files must reside in or below the directory containing the master DITA map file, and the map file cannot specify ".." at the start of the @href attributes for <topicref> elements.

usetasklabels

Specifies whether to generate headings for sections within task

topics. The allowed values are **YES** and **NO**; the default value is **NO**. Corresponds to the XSLT parameter **GENERATE-TASK-LABELS**.

xhtmlclass

Specifies whether to include the DITA class ancestry inside the XHTML elements .For example, the <prereq> element (which is specialized from section) would generate class="section prereq. The allowed values are **yes** and **no**; the default value is **yes**. Corresponds to the XSLT parameter **PRESERVE-DITA-CLASS**.

Note: Beginning with DITA OT release 1.5.2, the default value is **yes**. For release 1.5 and 1.5.1, the default value was **no**.

xsl

Specifies a custom XSL file to be used instead of the default XSL transformation (xsl\dita2xhtml.xsl). The parameter must specify a fully qualified file name.

Parent topic: Command-line tool parameters

Related reference

Command-line tool parameters: Eclipse content transformation Command-line tool parameters: Eclipse help transformation Command-line tool parameters: HTML help transformation Command-line tool parameters: JavaHelp transformation Command-line tool parameters: HTML5 and XHTML transformation

Command-line tool parameters: Eclipse content transformation

Certain parameters are specific to the Eclipse content transformation.

Figure 1. Command-line tool parameters: Eclipse content transformation

eclipsecontenttoc

Specifies the name of the TOC file

Parent topic: Command-line tool parameters

Related concepts Eclipse content transformation

Command-line tool parameters: Eclipse help transformation

Certain parameters are specific to the Eclipse help transformation.

Figure 1. Command-line tool parameters: Eclipse help transformation

eclipsehelptoc

Specifies the name of the TOC file.

provider

Specifies the name of the person or organization that provides the Eclipse help. The default value is **DITA**.

Tip: The toolkit ignores the value of this parameter when it processes an Eclipse map.

version

Specifies the version number to include in the output. The default value is **0.0.0**.

Tip: The toolkit ignores the value of this parameter when it processes an Eclipse map.

Parent topic: Command-line tool parameters

Related concepts

Eclipse help transformation

Related reference

Command-line tool parameters: HTML help transformation

Certain parameters are specific to the HTML help transformation.

Figure 1. Command-line tool parameters: HTML help transformation

htmlhelpincludefile

Specifies the name of a file that you want included in the HTML Help.

Parent topic: Command-line tool parameters

Related concepts

HTML help transformation

Related reference

Command-line tool parameters: JavaHelp transformation

Certain parameters are specific to the JavaHelp transformation.

Figure 1. Command-line tool parameters: JavaHelp transformation

javahelpmap

Specifies the name of the ditamap file for a JavaHelp project.

javahelptoc

Specifies the name of the file containing the TOC in your JavaHelp output. Default value is the name of the ditamap file for your project.

Parent topic: Command-line tool parameters

Related concepts

JavaHelp transformation

Related reference

Command-line tool parameters: ODT transformation

Certain parameters are specific to the ODT transformation.

Figure 1. Command-line tool parameters: ODT transformation

odtimgembed

Determines whether images are embedded as binary objects within the ODT file.

Parent topic: Command-line tool parameters

Related concepts ODT transformation

Related reference Command-line tool parameters: All transformations

Command-line tool parameters: PDF transformation

Certain parameters are specific to the PDF2 transformation.

Figure 1. Command-line tool parameters: PDF transformation

fooutputrellinks

Specifies whether to render related links in the output. The allowed values are **yes** and **no**; the default value is **no**. If the **args.fo.include.rellinks** parameter is specified, this parameter is ignored.

fouserconfig

Specifies the user configuration file for FOP.

retaintopicfo

Specifies whether to retain the generated FO file. The allowed values are **yes** and **no**; the default value is **no**. If the configuration property **org.dita.pdf2.use-out-temp** is set to **false**, this parameter is ignored.

xslpdf

Specifies an XSL file that is used to override the default XSL transformation (plugins\org.dita.pdf2\xsl\fo\topic2fo_shell.xsl). You must specify the fully qualified file name.

Parent topic: Command-line tool parameters

Related concepts

PDF transformation

Related reference

Command-line tool parameters: All transformations

Command-line tool parameters: HTML5 and XHTML transformation

Certain parameters are specific to the HTML5 and XHTML transformation.

Figure 1. Command-line tool parameters: HTML5 and XHTML transformation

xhtmltoc

Specifies the base name of the TOC file. The default value is **index**.

Parent topic: Command-line tool parameters

Related concepts

XHTML transformation HTML5 transformation

Related reference

lib/configuration.properties file

The lib/configuration.properties file controls certain common properties, as well as some properties that control PDF processing.

Table 1. Properties set in the lib/configuration.properties file

Property	Description
default.language	Specifies the language that is used if the input file does not have the @xml:lang attribute set on the root element. By default, this is set to en . The allowed values are those that are defined in IETF BCP 47, <u>Tags</u> for the Identification of Languages.
generate-debug-attributes	Specifies whether the @xtrf and @xtrc debugging attributes are generated in the temporary files. The following values are allowed:
	 true (default) — Enables generation of debugging attributes
	 false — Disables generation of debugging attributes
	Note: Disabling debugging attributes reduces the size of temporary files and thus reduces memory consumption. However, the log messages no longer have the source information available and thus the ability to debug problems might deteriorate.
	Specifies how the DITA-OT handles errors and error recovery. The following values are allowed:
	 strict — When an error is encountered, the DITA-OT stops processing.

processing-mode	• lax (default) — When an error is encountered, the DITA-OT attempts to recover from it.
	• skip — When an error is encountered, the DITA continues processing but does not attempt error recovery.
	(PDF transformation only) Enables I18N font processing. The following values are allowed:
org.dita.pdf2.i18n.enabled	 true (default) — Enables I18N processing
	• false — Disables I18N processing
org.dita.pdf2.use-out- temp	 (PDF transformation only) Specifies whether the XSL-FO processing writes the intermediate files (for example, the topic.fo file) to the output directory. The following values are allowed: true — Write intermediate files to the output directory false (default) — Write intermediate files to the temporary directory
plugindirs	A semicolon-separated list of directory paths that the DITA-OT searches for plug-ins to integrate; any relative paths are resolved against the DITA-OT base directory. Any immediate subdirectory that contains a plugin.xml file is integrated
plugin.ignores	A semicolon-separated list of directory names to be ignored during plug-in integration; any relative paths are resolved against the DITA-OT base directory.

Parent topic: Reference

DITA Open Toolkit Developer Reference

The *DITA Open Toolkit Developer Reference* is designed to provide more advanced information about the DITA-OT. It is geared to an audience that needs information about the DITA-OT architecture, configuring and extending the DITA-OT, and creating DITA-OT plug-ins.

Architecture of the DITA Open Toolkit The DITA Open Toolkit (DITA-OT) is an open-source implementation of the OASIS specification for the Darwin Information Typing Architecture (DITA). The toolkit uses Ant, XSLT, and Java to transform DITA content (maps and topics) into different deliverable formats.

Extending the DITA Open Toolkit

There are several methods that can be used to extend the toolkit; not all of them are recommended or supported. The best way to create most extensions is with a plug-in; extended documentation for creating plug-ins is provided in the next section.

Configuring the DITA Open Toolkit

The DITA OT uses .properties files that store configuration settings for the DITA OT and its plug-ins. The configuration properties are available to both Ant and Java processes, but unlike argument properties, they cannot be set at run time.

Creating DITA-OT plug-ins

The DITA Open Toolkit comes with a built in mechanism for adding in extensions through plug-ins. These plug-ins may do a wide variety of things, such as adding support for specialized DITA DTDs or Schemas, integrating processing overrides, or even providing entirely new output transforms. Plug-ins are the best way to extend the toolkit in a way that is consistent, easily sharable, and easy to preserve through toolkit upgrades.

Customizing PDF output

You can build a DITA-OT plug-in that contains a customized PDF

transformation.

Internal Ant properties

Reference list of Ant properties used by DITA-OT internally.

Implementation dependent features

Extended functionality

Architecture of the DITA Open Toolkit

The DITA Open Toolkit (DITA-OT) is an open-source implementation of the OASIS specification for the Darwin Information Typing Architecture (DITA). The toolkit uses Ant, XSLT, and Java to transform DITA content (maps and topics) into different deliverable formats.

DITA-OT processing structure The DITA-OT implements a multistage, map-driven architecture to process DITA content. Each stage in the process examines some or all of the content; some stages result in temporary files that are used by later steps, while others stages result in updated copies of the DITA content. Most of the processing takes place in a temporary working directory; the source files themselves are never modified.

DITA-OT processing modules

The DITA-OT processing pipeline is implemented using Ant. Individual modules within the Ant script are implemented in either Java or XSLT, depending on such factors as performance or requirements for customization. Virtually all Ant and XSLT modules can be extended by adding a plug-in to the toolkit; new Ant targets may be inserted before or after common processing, and new rules may be imported into common XSLT modules to override default processing.

DITA-OT processing order

The order of processing is often significant when evaluating DITA content. Although the DITA specification does not mandate a specific order for processing, the DITA-OT has determined that performing filtering before conref resolution best meets user expectations. Switching the order of processing, while legal, may give different results.

Pre-processing modules

The pre-processing operation is a set of steps that typically runs at the beginning of every DITA-OT transformation. Each step or stage

corresponds to an Ant target in the build pipeline; the preprocess target calls the entire set of steps.

HTML-based processing modules

The DITA-OT ships with several varieties of HTML output, each of which follows roughly the same path through the processing pipeline. All HTML-based transformation begin with the same call to the preprocessing module, after which they generate HTML files and then branch to create the transformation-specific navigation files.

PDF processing modules

The **PDF** (formerly known as **PDF2**) transformation process runs the pre-processing routine and follows it by a series of additional targets. These steps work together to create a merged set of content, convert the merged content to XSL-FO, and then format the XSL-FO file to PDF.

Open Document Format processing modules

The **odt** transformation creates a binary file using the OASIS Open Document Format.

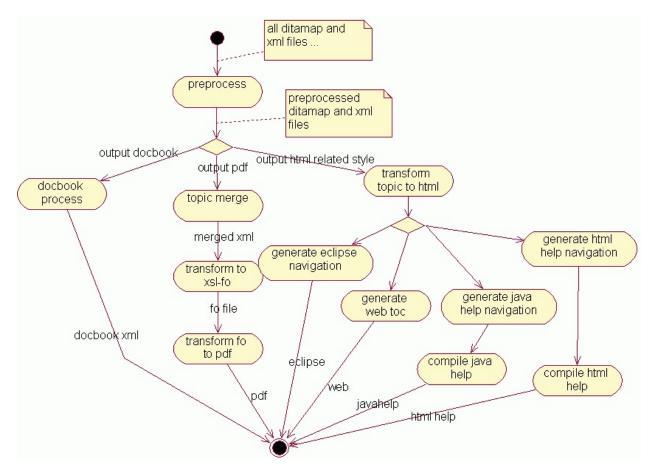
Parent topic: DITA Open Toolkit Developer Reference

DITA-OT processing structure

The DITA-OT implements a multi-stage, map-driven architecture to process DITA content. Each stage in the process examines some or all of the content; some stages result in temporary files that are used by later steps, while others stages result in updated copies of the DITA content. Most of the processing takes place in a temporary working directory; the source files themselves are never modified.

The DITA-OT is designed as a pipeline. Most of the pipeline is common to all output formats; it is known as the *pre-processing stage*. In general, any DITA process begins with this common set of pre-processing routines. Once the pre-processing is completed, the pipeline diverges based on the requested output format. Some processing is still common to multiple output formats; for example, Eclipse Help and HTML Help both use the same routines to generate XHTML topics, after which the two pipelines branch to create different sets of navigation files.

The following image illustrates how the pipeline works for some common output types: Docbook, PDF, Eclipse Help, XHTML, JavaHelp, and HTML Help.



Parent topic: Architecture of the DITA Open Toolkit

DITA-OT processing modules

The DITA-OT processing pipeline is implemented using Ant. Individual modules within the Ant script are implemented in either Java or XSLT, depending on such factors as performance or requirements for customization. Virtually all Ant and XSLT modules can be extended by adding a plug-in to the toolkit; new Ant targets may be inserted before or after common processing, and new rules may be imported into common XSLT modules to override default processing.

XSLT modules

The XSLT modules use shell files. Typically, each shell file begins by importing common rules that apply to all topics. This set of common processing rules may in turn import additional common modules, such as those used for reporting errors or determining the document locale. After the common rules are imported, additional imports can be included in order to support processing for DITA specializations.

For example, XHTML processing is controlled by the xsl\dita2xhtml.xsl file. The shell begins by importing common rules that are applicable to all general topics: xslhtml\dita2htmlImpl.xsl. After that, additional XSLT overrides are imported for specializations that require modified processing. For example, an override for reference topics is imported in order to add default headers to property tables. Additional modules are imported for tasks, for the highlighting domain, and for several other standard specializations. After the standard XSLT overrides occur, plugins may add in additional processing rules for local styles or for additional specializations.

Java modules

Java modules are typically used when XSLT is a poor fit, such as for processes that make use of standard Java libraries (like those used for index sorting). Java modules are also used in many cases where a step involves copying files, such as the initial process where source files are parsed and copied to a temporary processing directory.

Parent topic: Architecture of the DITA Open Toolkit

DITA-OT processing order

The order of processing is often significant when evaluating DITA content. Although the DITA specification does not mandate a specific order for processing, the DITA-OT has determined that performing filtering before conref resolution best meets user expectations. Switching the order of processing, while legal, may give different results.

The DITA-OT project has found that filtering first provides several benefits. Consider the following sample that contains a <note> element that both uses conref and contains a @product attribute:

```
<note conref="documentA.dita#doc/note" product="MyProd"/>
```

If the @conref attribute is evaluated first, then documentA must be parsed in order to retrieve the note content. That content is then stored in the current document (or in a representation of that document in memory). However, if all content with product="MyProd" is filtered out, then that work is all discarded later in the build.

If the filtering is done first (as in the DITA-OT), this element is discarded immediately, and documentA is never examined. This provides several important benefits:

- Time is saved by discarding unused content as early as possible; all future steps can load the document without this extra content.
- Additional time is saved case by not evaluating the @conref attribute; in fact, documentA does not even need to be parsed.
- Any user reproducing this build does not need documentA. If the content is sent to a translation team, that team can reproduce an error-free build without documentA; this means documentA can be kept back from translation, preventing accidental translation and increased costs.

If the order of these two steps is reversed, so that conref is evaluated first, it is possible that results will differ. For example, in the code sample above, the @product attribute will override the product setting on the

referencing note. Assume that the <note> elements in documentA is defined as follows:

```
<note id="note" product="SomeOtherProduct">This is an important r
```

A process that filters out product="SomeOtherProduct" will remove the target of the original conref before that conref is ever evaluated, which will result in a broken reference. Evaluating conref first would resolve the reference, and only later filter out the target of the conref. While some use cases can be found where this is the desired behavior, benefits such as those described above resulted in the current processing order used by the DITA-OT..

Parent topic: Architecture of the DITA Open Toolkit

Pre-processing modules

The pre-processing operation is a set of steps that typically runs at the beginning of every DITA-OT transformation. Each step or stage corresponds to an Ant target in the build pipeline; the preprocess target calls the entire set of steps.

1. <u>Generate lists (gen-list)</u> The gen-list step examines the input files and creates lists of topics, images, document properties, or other content. These lists are used by later steps in the pipeline. For example, one list includes all topics that make use of the conref attribute; only those files are processed during the conref stage of the build. This step is implemented in Java.

2. Debug and filter (debug-filter)

The debug-filter step processes all referenced DITA content and creates copies in a temporary directory. As the DITA content is copied, filtering is performed, debugging information is inserted, and table column names are adjusted. This step is implemented in Java.

3. Copy related files (copy-files)

The copy-files step copies non-DITA resources to the output directory, such as HTML files that are referenced in a map or images that are referenced by a DITAVAL file.

4. Resolve keyref (keyref)

The keyref step examines all the keys that are defined in the DITA source and resolved the key references. Links that make use of keys are updated so that any @href value is replaced by the appropriate target; key-based text replacement is also performed. This step is implemented in Java.

5. Conref push (conrefpush)

The conrefpush step resolves "conref push" references. This step only processes documents that use conref push or that are updated due to the push action. This step is implemented in Java.

6. Conref (conref)

The conref step resolves conref attributes, processing only the DITA maps or topics that use the @conref attribute. This step is implemented in XSLT.

7. Move metadata (move-meta-entries)

The move-meta-entries step pushes metadata back and forth between maps and topics. For example, index entries and copyrights in the map are pushed into affected topics, so that the topics can be processed later in isolation while retaining all relevant metadata. This step is implemented in Java.

8. Resolve code references (codref)

The coderef step resolves references made with the <coderef> element. This step is implemented in Java.

9. Resolve map references (mapref)

The mapref step resolves references from one DITA map to another. This step is implemented in XSLT.

10. Pull content into maps (mappull)

The mappull step pulls content from referenced topics into maps, and then cascades data within maps. This step is implemented in XSLT.

11. Chunk topics (chunk)

The chunk step breaks apart and assembles referenced DITA content based on the @chunk attribute in maps. This step is implemented in Java.

12. Map based linking (maplink and move-links)

These two steps work together to create links based on a map and move those links into the referenced topics. The links are created based on hierarchy in the DITA map, the @collection-type attribute, and relationship tables. This step is implemented in XSLT and Java.

13. Pull content into topics (topicpull)

The topicpull step pulls content into <xref> and <link> elements. This step is implemented in XSLT.

14. Flagging in the toolkit

Beginning with DITA-OT 1.7, flagging support is implemented as a common preprocess module. The module evaluates the DITAVAL against all flagging attributes, and adds DITA-OT specific hints in to the topic when flags are active. Any extended transform type may use these hints to support flagging without adding logic to interpret the DITAVAL.

Parent topic: Architecture of the DITA Open Toolkit

Generate lists (gen-list)

The gen-list step examines the input files and creates lists of topics, images, document properties, or other content. These lists are used by later steps in the pipeline. For example, one list includes all topics that make use of the conref attribute; only those files are processed during the conref stage of the build. This step is implemented in Java.

The result of this list is a set of several list files in the temporary directory, including dita.list and dita.xml.properties.

List file property	List file	List pro
canditopicsfile	canditopics.list	candito
codereffile	coderef.list	coderef
conreffile	conref.list	conrefli
conrefpushfile	conrefpush.list	conrefp
conreftargetsfile	conreftargets.list	conrefta
copytosourcefile	copytosource.list	copytos
copytotarget2sourcemapfile	copytotarget2sourcemap.lis	t copytoti
flagimagefile	flagimage.list	flagima
fullditamapandtopicfile	fullditamapandtopic.list	fullditan

fullditamapfile	fullditamap.list	fullditan
fullditatopicfile	fullditatopic.list	fullditate
hrefditatopicfile	hrefditatopic.list	hrefdita
hreftargetsfile	hreftargets.list	hreftarg
htmlfile	html.list	htmllist
imagefile	image.list	imagelis
keyreffile	keyref.list	keyrefli
outditafilesfile	outditafiles.list	outditaf
relflagimagefile	relflagimage.list	relflagin
resourceonlyfile	resourceonly.list	resourc
skipchunkfile	skipchunk.list	skipchu
subjectschemefile	subjectscheme.list	subjects
subtargetsfile	subtargets.list	subtarg
tempdirToinputmapdir.relative.value		
uplevels		
user.input.dir		
user.input.file.listfile		
user.input.file		

Parent topic: Pre-processing modules

Next topic: Debug and filter (debug-filter)

Debug and filter (debug-filter)

The debug-filter step processes all referenced DITA content and creates copies in a temporary directory. As the DITA content is copied, filtering is performed, debugging information is inserted, and table column names are adjusted. This step is implemented in Java.

The following modifications are made to the DITA source:

- If a DITAVAL file is specified, the DITA source is filtered according to the entries in the DITAVAL file.
- Debug information is inserted into each element using the @xtrf and @xtrc attributes. The values of these attributes enable messages later in the build to reliably indicate the original source of the error.
 For example, a message might trace back to the fifth <ph> element in a specific DITA topic. Without these attributes, that count might no longer be available due to filtering and other processing.
- The table column names are adjusted to use a common naming scheme. This is done only to simplify later conref processing. For example, if a table row is pulled into another table, this ensures that a reference to "column 5 properties" will continue to work in the fifth column of the new table.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Generate lists (gen-list)</u> Next topic: <u>Copy related files (copy-files)</u>

Copy related files (copy-files)

The copy-files step copies non-DITA resources to the output directory, such as HTML files that are referenced in a map or images that are referenced by a DITAVAL file.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Debug and filter (debug-filter)</u> Next topic: <u>Resolve keyref (keyref)</u>

Resolve keyref (keyref)

The keyref step examines all the keys that are defined in the DITA source and resolved the key references. Links that make use of keys are updated so that any @href value is replaced by the appropriate target; key-based text replacement is also performed. This step is implemented in Java.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Copy related files (copy-files)</u> Next topic: <u>Conref push (conrefpush)</u>

Conref push (conrefpush)

The conrefpush step resolves "conref push" references. This step only processes documents that use conref push or that are updated due to the push action. This step is implemented in Java.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Resolve keyref (keyref)</u> Next topic: <u>Conref (conref)</u>

Conref (conref)

The conref step resolves conref attributes, processing only the DITA maps or topics that use the @conref attribute. This step is implemented in XSLT.

The values of the @id attribute on referenced content are changed as the elements are pulled into the new locations. This ensures that the values of the @id attribute within the referencing topic remain unique.

If an element is pulled into a new context along with a cross reference that references the target, both the values of the @id and @xref attributes are updated so that they remain valid in the new location. For example, a referenced topic might include a section as in the following example:

Figure 1. Referenced topic that contains a section and cross reference

```
<topic id="referenced_topic">
  <title>...</title>
  <body>
    <section id="sect"><title>Sample section</title>
    Figure <xref href="#referenced_topic/fig"/> contains an
    <fig id="fig"><title>Code sample</title>
        <codeblock>....</codeblock>
        </fig>
    </section>
  </body>
</topic>
```

When the section is referenced using a @conref attribute, the value of the @id attribute on the <fig> element is modified to ensure that it remains unique in the new context. At the same time, the <xref> element is also modified so that it remains valid as a local reference. For example, if the referencing topic has an @id set to "new_topic", then the conrefed <section> element may look like this in the intermediate document. *Figure 2. Resolved conrefed <section> element after the conref step*

```
<section id="sect"><title>Sample section</title>
    Figure <xref href="#new_topic/d1e25"/> contains an code sample</fig id="d1e25"><title>Code sample</title>
        <codeblock>....</codeblock>
        </fig>
</section>
```

In this case, the value of the @id attribute on the <fig> element has been changed to a generated value of "d1e25". At the same time, the <xref> element has been updated to use that new generated ID, so that the cross reference remains valid.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Conref push (conrefpush)</u> Next topic: <u>Move metadata (move-meta-entries)</u>

Move metadata (move-meta-entries)

The move-meta-entries step pushes metadata back and forth between maps and topics. For example, index entries and copyrights in the map are pushed into affected topics, so that the topics can be processed later in isolation while retaining all relevant metadata. This step is implemented in Java.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Conref (conref)</u> Next topic: <u>Resolve code references (codref)</u>

Resolve code references (codref)

The coderef step resolves references made with the <coderef> element. This step is implemented in Java.

The <coderef> is used to reference code stored externally in non-XML documents. During the pre-processing step, the referenced content is pulled into the containing <codeblock> element.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Move metadata (move-meta-entries</u>) Next topic: <u>Resolve map references (mapref</u>)

Resolve map references (mapref)

The mapref step resolves references from one DITA map to another. This step is implemented in XSLT.

Maps reference other maps by using the following sorts of markup:

```
<topicref href="other.ditamap" format="ditamap"/>
...
<mapref href="other.ditamap"/>
```

As a result of the mapref step, the element that references another map is replaced by the topic references from the other map. Relationship tables are pulled into the referencing map as a child of the root element (<map> or a specialization of <map>).

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Resolve code references (codref)</u> Next topic: <u>Pull content into maps (mappull)</u>

Pull content into maps (mappull)

The mappull step pulls content from referenced topics into maps, and then cascades data within maps. This step is implemented in XSLT.

The mappull step makes the following changes to the DITA map:

- Titles are pulled from referenced DITA topics. Unless the @locktitle attribute is set to "yes", the pulled titles replace the navigation titles specified on the <topicref> elements.
- The Inktext> element is set based on the title of the referenced topic, unless it is already specified locally.
- The <shortdesc> element is set based on the short description of the referenced topic, unless it is already specified locally.
- The @type attribute is set on <topicref> elements that reference local DITA topics. The value of the @type attribute is set to value of the root element of the topic; for example, a <topicref> element that references a task topic is given a @type attribute set to "task"".
- Attributes that cascade, such as @toc and print, are made explicit on any child <topicref >elements. This allows future steps to work with the attributes directly, without reevaluating the cascading behavior.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Resolve map references (mapref)</u> Next topic: <u>Chunk topics (chunk)</u>

Chunk topics (chunk)

The chunk step breaks apart and assembles referenced DITA content based on the @chunk attribute in maps. This step is implemented in Java.

The DITA-OT has implemented processing for the following values of the @chunk attribute:

- select-topic
- select-document
- select-branch
- by-topic
- by-document
- to-content
- to-navigation

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Pull content into maps (mappull)</u> Next topic: <u>Map based linking (maplink and move-links)</u>

Related information Chunking definition in the DITA 1.2 specification

Map based linking (maplink and move-links)

These two steps work together to create links based on a map and move those links into the referenced topics. The links are created based on hierarchy in the DITA map, the @collection-type attribute, and relationship tables. This step is implemented in XSLT and Java.

The maplink module runs an XSLT program that evaluates the map; it places all the generated links into a single file in the temporary directory. The move-links module then runs a Java program that pushes the generated links into the applicable topics.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Chunk topics (chunk)</u> Next topic:

Pull content into topics (topicpull)

The topicpull step pulls content into <xref> and <link> elements. This step is implemented in XSLT.

If an <xref> element does not contain link text, the target is examined and the link text is pulled. For example, a reference to a topic pulls the title of the topic; a reference to a list item pulls the number of the item. If the <xref> element references a topic that has a short description, and the <xref> element does not already contain a child <desc> element, a <desc> element is created that contains the text from the topic short description.

The process is similar for <link> elements. If the <link> element does not have a child <linktext> element, one is created with the appropriate link text. Similarly, if the <link> element does not have a child <desc> element, and the short description of the target can be determined, a <desc> element is created that contains the text from the topic short description.

Parent topic: <u>Pre-processing modules</u> Previous topic: <u>Map based linking (maplink and move-links)</u> Next topic: <u>Flagging in the toolkit</u>

Flagging in the toolkit

Beginning with DITA-OT 1.7, flagging support is implemented as a common preprocess module. The module evaluates the DITAVAL against all flagging attributes, and adds DITA-OT specific hints in to the topic when flags are active. Any extended transform type may use these hints to support flagging without adding logic to interpret the DITAVAL.

Evaluating the DITAVAL flags

Flagging is implemented as a reusable module during the preprocess stage. If a DITAVAL file is not used with a build, this step is skipped with no change to the file.

When a flag is active, relevant sections of the DITAVAL itself are copied into the topic as a sub-element of the current topic. The active flags are enclosed in a pseudo-specialization of the <foreign> element (referred to as a pseudo-specialization because it is used only under the covers, with all topic types; it is not integrated into any shipped document types).

<ditaval-startprop>

When any flag is active on an element, a <ditaval-startprop> element will be created as the first child of the flagged element:

```
<ditaval-startprop class="+ topic/foreign ditaot-d/ditav</pre>
```

The <ditaval-startprop> element will contain the following:

- If the active flags should create a new style, that style is included using standard CSS markup on the @outputclass attribute. Output types that make use of CSS, such as XHTML, can use this value as-is.
- If styles conflict, and a <style-conflict> element exists in the DITAVAL, it will be copied as a child of <ditavalstartprop>.
- Any <prop> or <revprop> elements that define active flags will be copied in as children of the <ditaval-startprop> element. Any <startflag> children of the properties will be included, but <endflag> children will not.

<ditaval-endprop>

When any flag is active on an element, a <ditaval-endprop> element will be created as the last child of the flagged element:

```
<ditaval-endprop class="+ topic/foreign ditaot-d/ditaval
```

CSS values and <styleconflict> elements are not included on this element.

Any <prop> or <revprop> elements that define active flags will be copied in as children of <ditaval-prop>. Any <endflag> children of the properties will be included, but <startflag> children will not.

Supporting flags in overrides or custom transform types

For most transform types, the <foreign> element should be ignored by default, because arbitrary non-DITA content may not mix well unless coded for ahead of time. If the <foreign> element is ignored by default, or if a rule is added to specifically ignore <ditaval-startprop> and <ditaval-endprop>, then the added elements will have no impact on a transform. If desired, flagging support may be integrated at any time in the future.

The processing described above runs as part of the common preprocess, so any transform that uses the default preprocess will get the topic updates. To support generating flags as images, XSLT based transforms can use default fallthrough processing in most cases. For example, if a paragraph is flagged, the first child of will contain the start flag information; adding a rule to handle images in <ditaval-startprop> will cause the image to appear at the start of the paragraph content.

In some cases fallthrough processing will not result in valid output; for those cases, the flags must be explicitly processed. This is done in the XHTML transform for elements like , because fallthrough processing would place images in between and . To handle this, the code processes <ditaval-startprop> before starting the element, and <ditaval-endprop> at the end. Fallthrough processing is then disabled for those elements as children of .

Example DITAVAL

Assume the following DITAVAL file is in use during a build. This DITAVAL will be used for each of the following content examples.

```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <!-- Define what happens in the case of conflicting styles -->
  <style-conflict background-conflict-color="red"/>
  <!-- Define two flagging properties that give styles (no image)
 <prop action="flag" att="audience" style="underline" val="user"
  <prop action="flag" att="platform" style="overline" val="win" k
  <!-- Define a property that includes start and end image flags
  <prop action="flag" att="platform" val="linux" style="overline'</pre>
    <startflag imageref="startlin.png"><alt-text>Start linux</alt
    <endflag imageref="endlin.png"><alt-text>End linux</alt-text>
  </prop>
  <!-- Define a revision that includes start and end image flags
  <revprop action="flag" style="double-underline" val="rev2">
    <startflag imageref="start_rev.gif"><alt-text>sssssssssstart
    <endflag imageref="end rev.gif"><alt-text>eeeeeeeeeeeeeeeed</a</pre>
  </revprop>
</val>
```

Content example 1: adding style

Now assume the following paragraph exists in a topic. Class attributes are included, as they would normally be in the middle of the preprocess routine; @xtrf and @xtrc are left off for clarity.

```
Simple user; includes style but no images
```

Based on the DITAVAL above, audience="user" results in a style with underlining and with a green background. The interpreted CSS value is added to @outputclass on <ditaval-startprop>, and the actual property definition is included at the start and end of the element. The output from the flagging step looks like this (with newlines added for clarity, and class attributes added as they would appear in the temporary file):

The resulting file after the flagging step looks like this; for clarity, newlines are added, while @xtrf and @xtrc are removed:

Content example 2: conflicting styles

This example includes a paragraph with conflicting styles. When the audience and platform attributes are both evaluated, the DITAVAL indicates that the background color is both green and blue. In this situation, the <style-conflict> element is evaluated to determine how to style the content.

```
Conflicting styles (still no ir
```

The <style-conflict> element results in a background color of red, so this value is added to @outputclass on <ditaval-startprop>. As above, active properties are copied into the generated elements; the <styleconflict> element itself is also copied into the generated <ditavalstartprop> element.

The resulting file after the flagging step looks like this; for clarity, newlines are added, while @xtrf and @xtrc are removed:

```
   <ditaval-startprop class="+ topic/foreign ditaot-d/ditaval-staw
        outputclass="background-color:red;">
        <style-conflict background-color:red;">
        <style="color:red">
        <style-conflict background-color:red;">
        <style="color:red">
        <style-conflict background-color:red;">
        </style="color:red">
        <style-conflict background-color:red;">
        </style="color:red">
        <style="color:red"</style="color:red"</style="color:red">
        </style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="color:red"</style="colo:red"</style="colo:re
```

Content example 3: adding image flags

This example includes image flags for both @platform and @rev, which are defined in DITAVAL <prop> and <revprop> elements.

```
Generate images for platform="linux" and rev="2"
```

As above, the <ditaval-startprop> and <ditaval-endprop> nest the active property definitions, with the calculated CSS value on @outputclass. The <ditaval-startprop> drops the ending image, and <ditaval-endprop> drops the starting image. To make document-order processing more consistent, property flags are always included before revisions in <ditaval-startprop>, and the order is reversed for <ditaval-endprop>.

The resulting file after the flagging step looks like this; for clarity, newlines are added, while @xtrf and @xtrc are removed:

```
<ditaval-startprop class="+ topic/foreign ditaot-d/ditaval-star</pre>
         outputclass="background-color:blue;text-decoration:und
   <prop action="flag" att="platform" val="linux" style="overlin"
     <startflag imageref="startlin.png"><alt-text>Start linux</a
   </prop>
   <revprop action="flag" style="double-underline" val="rev2">
    </revprop>
 </ditaval-startprop>
 class="- topic/li ">Generate images for platform="linux" ar
 <ditaval-endprop class="+ topic/foreign ditaot-d/ditaval-endprop)</pre>
   <revprop action="flag" style="double-underline" val="rev2">
    </revprop>
   <prop action="flag" att="platform" val="linux" style="overlin"
     <endflag imageref="endlin.png"><alt-text>End linux</alt-te>
   </prop>
 </ditaval-endprop>
```

Parent topic: Pre-processing modules

Previous topic:

HTML-based processing modules

The DITA-OT ships with several varieties of HTML output, each of which follows roughly the same path through the processing pipeline. All HTML-based transformation begin with the same call to the pre-processing module, after which they generate HTML files and then branch to create the transformation-specific navigation files.

Common HTML-based processing After the pre-processing operation runs, HTML-based builds each run a common series of Ant targets to generate HTML file. Navigation may be created before or after this set of common routines.

XHTML processing

After the XHTML files are generated by the common routine, the dita.map.xhtml target is called by the **xhtml** transformation. This target generates a TOC file called index.html, which can be loaded into a frameset.

HTML5 processing

After the HTML5 files are generated by the common routine, the dita.map.xhtml target is called by the **html5** transformation. This target generates a TOC file called index.html, which can be loaded into a frameset.

Eclipse help processing

The **eclipsehelp** transformation generates XHTML-based output and files that are needing to create an Eclipse Help system plug-in. Once the normal XHTML process has run, the dita.map.eclipse target is used to create a set of control files and navigation files.

TocJS processing

The **tocjs** transformation was originally created as a plug-in that distributed outside of the toolkit, but it now ships bundled in the default packages. This HTML5-based output type creates a JavaScript based frameset with TOC entries that expand and

collapse.

HTML Help processing

The **htmlhelp** transformation created HTML Help control files. If the build runs on a system that has the HTML Help compiler installed, the control files are compiled into a CHM file.

JavaHelp processing

The **javahelp** transformation runs several additional Ant targets after the XHTML processing is completed in order to create control files for the JavaHelp output.

Parent topic: Architecture of the DITA Open Toolkit

Common HTML-based processing

After the pre-processing operation runs, HTML-based builds each run a common series of Ant targets to generate HTML file. Navigation may be created before or after this set of common routines.

After the pre-processing is completed, the following targets are run for all of the HTML-based builds:

- If the **arg.css** parameter is passed to the build to add a CSS file, the copy-css target copies the CSS file from its source location to the relative location in the output directory.
- If a DITAVAL file is used, the copy-revflag target copies the default start- and end-revision flags into the output directory.
- The DITA topics are converted to HTML files. Unless the @chunk attribute was specified, each DITA topic in the temporary directory now corresponds to one HTML file. Thedita.inner.topics.xhtml target is used to process documents that are in the map directory (or subdirectories of the map directory). The dita.outer.topics.xhtml target is used to process documents that are outside of the scope of the map, and thus might end up outside of the designated output directory. Various DITA-OT parameters control how documents processed by the dita.outer.topics.xhtml target are handled.

XHTML processing

After the XHTML files are generated by the common routine, the dita.map.xhtml target is called by the **xhtml** transformation. This target generates a TOC file called index.html, which can be loaded into a frameset.

HTML5 processing

After the HTML5 files are generated by the common routine, the dita.map.xhtml target is called by the **html5** transformation. This target generates a TOC file called index.html, which can be loaded into a frameset.

Eclipse help processing

The **eclipsehelp** transformation generates XHTML-based output and files that are needing to create an Eclipse Help system plug-in. Once the normal XHTML process has run, the dita.map.eclipse target is used to create a set of control files and navigation files.

Eclipse use multiple files to control the plug-in behavior. Some of these control files are generated by the build, while others might be created manually. The following Ant targets control the Eclipse help processing:

dita.map.eclipse.init

Sets up various default properties

dita.map.eclipse.toc

Creates the XML file that defines an Eclipse table of contents

dita.map.eclipse.index

Creates the sorted XML file that defines an Eclipse index

dita.map.eclipse.plugin

Creates the plugin.xml file that controls the behavior of an Eclipse plug-in

dita.map.eclipse.plugin.properties

Creates a Java properties file that sets properties for the plug-in, such as name and version information

dita.map.eclipse.manifest.file

Creates a MANIFEST.MF file that contains additional information used by Eclipse

copy-plugin-files

Checks for the presence of certain control files in the source

directory, and copies those found to the output directory

dita.map.eclipse.fragment.language.init

Works in conjunction with the dita.map.eclipse.fragment.language.country.init and dita.map.eclipse.fragment.error targets to control Eclipse fragment files, which are used for versions of a plug-in created for a new language or locale

Several of the targets listed above have matching templates for processing content that is located outside of the scope of the map directory, such as dita.out.map.eclipse.toc.

TocJS processing

The **tocjs** transformation was originally created as a plug-in that distributed outside of the toolkit, but it now ships bundled in the default packages. This HTML5-based output type creates a JavaScript based frameset with TOC entries that expand and collapse.

The following Ant targets control most of the TocJS processing:

tocjsInit

Sets up default properties. This target detects whether builds have already specified a name for JavaScript control file; if not, the default name toctree.js is used.

map2tocjs

Calls the dita.map.tocjs target, which generates the contents frame for TocJS output.

tocjsDefaultOutput

Ensures that the HTML5 processing module is run. If scripts are missing required information, such as a name for the default frameset, this target copies default style and control files. This target was add to the DITA-OT in version 1.5.4; earlier versions of the TocJS transformation created only the JavaScript control file by default.

HTML Help processing

The **htmlhelp** transformation created HTML Help control files. If the build runs on a system that has the HTML Help compiler installed, the control files are compiled into a CHM file.

Once the pre-processing and XHTML processes are completed, most of the HTML Help processing is handled by the following targets:

dita.map.htmlhelp

Create the HHP, HHC, and HHK files. The HHK file is sorted based on the language of the map.

dita.htmlhelp.convertlang

Ensures that the content can be processed correctly by the compiler, and that the appropriate code pages and languages are used.

compile.HTML.Help

Attempts to detect the HTML Help compiler. If the compiler is found, the full project is compiled into a single CHM file.

JavaHelp processing

The **javahelp** transformation runs several additional Ant targets after the XHTML processing is completed in order to create control files for the JavaHelp output.

There are two primary Ant targets:

dita.map.javahelp

Creates all of the files that are needed to compile JavaHelp, including a table of contents, sorted index, and help map file.

compile.Java.Help

Searches for a JavaHelp compiler on the system. If a compiler is found, the help project is compiled.

PDF processing modules

The **PDF** (formerly known as **PDF2**) transformation process runs the preprocessing routine and follows it by a series of additional targets. These steps work together to create a merged set of content, convert the merged content to XSL-FO, and then format the XSL-FO file to PDF.

The PDF process includes many Ant targets. During a typical conversion from map to PDF, the following targets are most significant.

map2pdf2

Creates a merged file by calling a common Java merge module. It then calls the publish.map.pdf target to do the remainder of the work.

publish.map.pdf

Performs some initialization and then calls the transform.topic2pdf target to do the remainder of processing.

transform.topic2pdf

Converts the merged file to XSL-FO, generates the PDF, and deletes the topic.fo file, unless instructed to keep it. Uses the following targets to perform those tasks:

transform.topic2fo

Convert the merged file to an XSL-FO file. This process is composed of several Ant targets.

Ant target	Description
transform.topic2fo.index	Runs a Java process to set up index processing, based on the document language. This step generates the file stage1.xml in the

	temporary processing directory.
transform.topic2fo.flagging	Sets up preprocessing for flagging based on a DITAVAL file. This step generates the file stage1a.xml in the temporary processing directory.
transform.topic2fo.main	Does the bulk of the conversion from DITA to XSL-FO. It runs the XSLT based process that creates stage2.fo in the temporary processing directory
transform.topic2fo.i18n	Does additional localization processing on the FO file; it runs a Java process that converts stage2.fo into stage3.fo, followed by an XSLT process that converts stage3.fo into topic.fo.

transform.fo2pdf

Converts the topic.fo file into PDF using the specified FO processor (Antenna House, XEP, or Apache FOP).

delete.fo2pdf.topic.fo

Deletes the topic.fo file, unless otherwise specified by setting an Ant property or command-line option.

Parent topic: Architecture of the DITA Open Toolkit

Open Document Format processing modules

The **odt** transformation creates a binary file using the OASIS Open Document Format.

The **odt** transformation begins with pre-processing. It then runs either the dita.odt.package.topic Or dita.odt.package.map target, depending on whether the input to the transformation is a DITA topic or a DITA map. The following description focuses on the map process, which is made up of the following targets:

dita.map.odt

Converts the map into a merged XML file using the Java-based topicmerge module. Then an XSLT process converts the merged file into the content.xml file.

dita.map.odt.stylesfile

Reads the input DITA map, and then uses XSLT to create a styles.xml file in the temporary directory.

dita.out.odt.manifest.file

Creates the manifest.xml file

Once these targets have run, the generated files are zipped up together with other required files to create the output ODT file.

Parent topic: Architecture of the DITA Open Toolkit

Extending the DITA Open Toolkit

There are several methods that can be used to extend the toolkit; not all of them are recommended or supported. The best way to create most extensions is with a plug-in; extended documentation for creating plug-ins is provided in the next section.

- Creating a plug-in can be very simple to very complex, and is generally the best method for changing or extending the toolkit. Plug-ins can be used to accomplish almost any modification that is needed for toolkit processing, from minor style tweaks to extensive, complicated new output formats.
- The PDF process was initially developed independently of the toolkit, and created its own extension mechanism using customization directories. Many (but not quite all) of the capabilities available through PDF customization directories are now available through plug-ins.
- Using a single XSL file as an override by passing it in as a parameter. For example, when building XHTML content, the XSL parameter allows users to specify a single local XSL file (inside or outside of the toolkit) that is called in place of the default XHTML code. Typically, this code imports the default processing code, and overrides a couple of processing routines. This approach is best when the override is very minimal, or when the style varies from build to build. However, any extension made with this sort of override is also possible with a plug-in.
- Editing DITA-OT code directly may work in some cases, but is not advised. Modifying the code directly significantly increases the work and risk involved with future upgrades. It is also likely that such modifications will break plug-ins provided by others, limiting the function available to the toolkit.

<u>Manually installing plug-ins</u> Plug-ins are generally distributed as zip files. There are two steps to installing a plug-in: unzipping and

integrating.

Manually removing plug-ins

Plug-ins can be installed by removing the plug-in and running integration process.

Rebuilding the DITA-OT documentation

The DITA-OT ships with Ant scripts that enable you to rebuild the toolkit documentation. This is especially helpful if your environment contains plug-ins that integrate additional messages into the toolkit.

Parent topic: DITA Open Toolkit Developer Reference

Related tasks Installing plug-ins Removing plug-ins

Manually installing plug-ins

Plug-ins are generally distributed as zip files. There are two steps to installing a plug-in: unzipping and integrating.

About this task

It is possible to define a plug-in so that it may be installed anywhere, although most expect to be placed in plugins/ directory inside of the DITA-OT. Most plug-ins do not require a specific install directory and can go in either of the default locations, but some may come with instructions for a particular install directory.

Procedure

- The unzip the plug-in file to plugins subdirectory. The plug-in directory should be named after plug-in ID and version, for example plugins/com.example.xhtml_1.0.0.
- 2. Run plug-in integration process.
 - From the toolkit directory, run the following command to integrate all installed plug-ins:

```
ant -f integrator.xml
```

- Any build that uses the Java command line interface automatically runs the integrator before processing begins.
- Ant based builds may import the integrator.xml file, and add integrate to the start of the dependency chain for the build.

Note: The integration process in considered part of the installation process and running it before each conversion will incur a performance penalty.

The integration process has two modes, lax and strict. In the strict mode the integration process will immediately fail if it encounters errors in plug-in configurations or installation process. In the lax mode, the integration process will continue to finish regardless of errors; the lax mode does not imply error recovery and may leave the DITA-OT installation into a broken state. The default mode is lax due to backwards compatibility, to run the integration in strict mode:

```
ant -f integrator.xml strict
```

To get more information about the integration process, run Ant in verbose mode:

```
ant -f integrator.xml -verbose strict
```

Parent topic: Extending the DITA Open Toolkit

Manually removing plug-ins

Plug-ins can be installed by removing the plug-in and running integration process.

Procedure

- 1. Remove plug-in installation directory.
- 2. Run integration process.

```
ant -f integrator.xml
```

Parent topic: Extending the DITA Open Toolkit

Rebuilding the DITA-OT documentation

The DITA-OT ships with Ant scripts that enable you to rebuild the toolkit documentation. This is especially helpful if your environment contains plug-ins that integrate additional messages into the toolkit.

Procedure

- 1. Change to the doc directory.
- 2. Run the following command:

```
ant -f build.xml target
```

The *target* parameter is optional and specifies a specific transformation type. It takes the following values:

- build-html
- build-htmlhelp
- build-pdf

If you do not specify an Ant target, all three output formats (HTML5, HTML help, and PDF) are generated.

Parent topic: Extending the DITA Open Toolkit

Configuring the DITA Open Toolkit

The DITA OT uses .properties files that store configuration settings for the DITA OT and its plug-ins. The configuration properties are available to both Ant and Java processes, but unlike argument properties, they cannot be set at run time.

plugin.properties file The plugin.properties file is used to store configuration properties that are set by the integration process. The file is located in the lib\org.dita.dost.platform directory; it is regenerated each time the integration process is run and so should not be edited manually.

Parent topic: DITA Open Toolkit Developer Reference

plugin.properties file

The plugin.properties file is used to store configuration properties that are set by the integration process. The file is located in the lib\org.dita.dost.platform directory; it is regenerated each time the integration process is run and so should not be edited manually.

Parent topic: Configuring the DITA Open Toolkit

Creating DITA-OT plug-ins

The DITA Open Toolkit comes with a built in mechanism for adding in extensions through plug-ins. These plug-ins may do a wide variety of things, such as adding support for specialized DITA DTDs or Schemas, integrating processing overrides, or even providing entirely new output transforms. Plug-ins are the best way to extend the toolkit in a way that is consistent, easily sharable, and easy to preserve through toolkit upgrades.

A plug-in consists of a directory, typically stored directly within the plugins/ directory inside of the DITA-OT. Every plug-in is controlled by a file named plugin.xml, located in the plug-in's root directory.

Benefits of extending the toolkit through plug-ins include:

- Plug-ins are easily sharable with other users, teams, or companies; typically, all that is needed is to unzip and run a single integration step. With many builds, even that integration step is automatic.
- Allows overrides or customizations to grow from simple to complex over time, with no increased complexity to the extension mechanism.
- Plug-ins can be moved from version to version with an upgraded toolkit simply by unzipping again, or by copying the directory from one install to another; there is no need to re-integrate code based on updates to the core processing.
- Plug-ins can build upon each other. If you like a plug-in provided by one user, simply install that plug-in, and then create your own that builds on that extension. The two plug-ins can then be distributed to your team as a unit, or you can even share your own extensions with the original provider.

Plug-in configuration file The plugin.xml controls all aspects of a plug-in, making each extension visible to the rest of the toolkit. The file uses pre-defined extension points to locate changes, and integrates those changes into the core code.

Extending the XML Catalog

The XML Catalogs extension point is used to update the XML Catalogs used to resolve DTD or Schema document types, or to add URI mappings. This is required in order to support DITA specializations or new DITA document type shells.

Adding new targets to the Ant build process

The Ant conductor extension point is used to make new targets available to the Ant processing pipeline. This may be done as part of creating a new transform, extending pre-processing, or simply to provide Ant targets for the use of other plug-ins.

Adding Ant targets to the pre-process pipeline

Every step in the pre-process pipeline defines an extension point before and after the step, to allow plug-ins to integrate additional processing. This allows a plug-in to insert a new step before any preprocessing step, as well as before or after the entire preprocess pipeline.

Integrating a new transform type

Plug-ins may integrate an entire new transform type. The new transform type can be very simple, such as an XHTML build that creates an additional control file; it can also be very complex, adding any number of new processing steps.

Override styles with XSLT

The XSLT import extension points are used to override various steps of XSLT processing. For this, the extension attribute indicates the step that the override applies to; the file attribute is a relative path to the override within the current plugin. The plugin installer will add an XSL import statement to the default code so that your override becomes a part of the normal build.

Modifying or adding generated text

Generated text is the term for strings that are automatically added by the build, such as "Note" before the contents of a <note> element.

Passing parameters to existing XSLT steps

Plug-ins can define new parameters to be passed from the Ant build into existing XSLT pipeline stages, usually to have those parameters available as global <xs1:param> values within XSLT overrides.

Adding Java libraries to the classpath

If your Ant or XSLT extensions require additional Java libraries in the classpath, you can add them to the global DITA-OT classpath with the following feature.

Adding diagnostic messages

Plug-in specific warning and error messages can be added to the set of messages supplied by the DITA-OT. These messages can then be used by any XSLT override.

Managing plug-in dependencies

The <require> element in a plugin.xml file is used to create a dependency on another plug-in. The <require> element requires the plugin attribute in order to reference the dependency.

Version and support information

The following extension points are used by convention to define version and support info within a plug-in.

Creating a new plug-in extension point

If your plug-in needs to define its own extension point in an XML file, add the string "_template" to the filename before the file suffix. During integration, this file will be processed like the built-in DITA-OT templates.

Example plugin.xml file

The following is a sample of a plugin.xml file. This file adds support for a new set of specialized DTDs, and includes an override for the XHTML output processor.

Parent topic: DITA Open Toolkit Developer Reference

Plug-in configuration file

The plugin.xml controls all aspects of a plug-in, making each extension visible to the rest of the toolkit. The file uses pre-defined extension points to locate changes, and integrates those changes into the core code.

The root element of the plugin.xml file is <plugin>, and must specify an id attribute. The id attribute is used to identify the plug-in, as well as to identify whether pre-requisite plug-ins are available. The id attribute should follow the syntax rules:

```
id ::= token('.'token)*
token ::= ( [0..9] | [a..zA..Z] | '_' | '-' )+
```

The <plugin> element supports the following child elements:

• <feature> defines an *extension* to contribute to a defined *extension point*. The following attributes are supported:

Attribute	Description	Required
extension	extension point identifier	yes
value	comma separated string value of the extension	either value Or file
file	file path value of the extension, relative to plugin.xml	either value Or file
type	type of the value attribute	no

• extension-point defines new a *extension point* that can be used by other plug-ins. The following attributes are supported:

Attribute	Description	Required
id	extension point identifier	yes
name	extension point name	no

 <require> defines plug-in dependencies. The following attributes are supported:

Attribute	Description	Required
plugin	vertical bar separated list of plug-ins that are required	yes
importance	flag whether plug-in is required or optional	no

• <template> defines files that should be treated as *templates*. The following attributes are supported:

Attribute	Description	Required
file	file path to the template, relative to plugin.xml	yes

• <meta> defines metadata. The following attributes are supported:

Attribute	Description	Required
type	metadata name	yes
value	metadata value	yes

Any extension that is not recognized by the DITA-OT is ignored; all elements other than <plugin> are optional. Since version 1.5.3 multiple extension definitions within a plug-in configuration file are combined; in older versions only the last extension definition is used.

Extending the XML Catalog

The XML Catalogs extension point is used to update the XML Catalogs used to resolve DTD or Schema document types, or to add URI mappings. This is required in order to support DITA specializations or new DITA document type shells.

To do this, first create a catalog with only your new values, using the OASIS Catalog format, and place that in your plug-in. Local file references in the catalog should be relative to the location of the catalog. The following extension points are available to work with catalogs.

dita.specialization.catalog.relative dita.specialization.catalog

Adds the content of the catalog file defined in file attribute to main DITA-OT catalog file.

Remember: The dita.specialization.catalog extension is deprecated. Use dita.specialization.catalog.relative instead.

org.dita.pdf2.catalog.relative

Adds the content of the catalog file defined in file attribute to main PDF plug-in catalog file.

Example

This example assumes that "catalog-dita.xml" contains an OASIS catalog for any DTDs or Schemas inside this plug-in. The catalog entries inside of catalog-dita.xml are relative to the catalog itself; when the plug-in is integrated, they will be added to the core DITA-OT catalog (with the correct path).

```
<plugin id="com.example.catalog">
<feature extension="dita.specialization.catalog.relative" file=
</plugin>
```

Adding new targets to the Ant build process

The Ant conductor extension point is used to make new targets available to the Ant processing pipeline. This may be done as part of creating a new transform, extending pre-processing, or simply to provide Ant targets for the use of other plug-ins.

```
dita.conductor.target.relative
dita.conductor.target
```

Add Ant import to main Ant build file.

Remember: The dita.conductor.target extension is deprecated. Use dita.conductor.target.relative instead.

Example

To extend And processing, first place your extensions in an Ant project file within your plug-in, such as myAntStuff.xml. Next, create a small wrapper file myAntStuffWrapper.xml in the same directory:

```
<dummy> <import file="myAntStuff.xml"/> </dummy>
```

Then create the following feature:

```
<plugin id="com.example.ant">
<feature extension="dita.conductor.target.relative" file="myAnt
</plugin>
```

When the plug-in is integrated, the imports from myAntStuffWrapper.xml will be copied into build.xml (using the correct path). This makes targets in myAntStuff.xml available to any other processing.

Adding Ant targets to the pre-process pipeline

Every step in the pre-process pipeline defines an extension point before and after the step, to allow plug-ins to integrate additional processing. This allows a plug-in to insert a new step before any pre-processing step, as well as before or after the entire preprocess pipeline.

The group of preprocessing steps defines extension points before and after the full preprocessing chain.

depend.preprocess.pre

Preprocessing pre-target; extending this target runs your Ant target before the full preprocess routine begins.

depend.preprocess.post

Preprocessing post-target; extending this target runs your Ant target after the full preprocess routine completes.

In addition, there are extension points to execute an Ant target before individual preprocessing steps.

depend.preprocess.clean-temp.pre

Clean temp pre-target

depend.preprocess.gen-list.pre

Generate list pre-target

depend.preprocess.debug-filter.pre

Debug and filter pre-target

depend.preprocess.conrefpush.pre

Content reference push pre-target

depend.preprocess.move-meta-entries.pre

Move meta entries pre-target

depend.preprocess.conref.pre

Content reference pre-target depend.preprocess.coderef.pre Code reference pre-target depend.preprocess.mapref.pre Map reference pre-target depend.preprocess.keyref.pre Resolve key reference pre-target depend.preprocess.mappull.pre Map pull pre-target depend.preprocess.chunk.pre Chunking pre-target depend.preprocess.maplink.pre Map link pre-target depend.preprocess.move-links.pre Move links pre-target depend.preprocess.topicpull.pre Topic pull pre-target depend.preprocess.copy-files.pre Copy files pre-target depend.preprocess.copy-image.pre Copy images pre-target

depend.preprocess.copy-html.pre

Copy HTML pre-target

depend.preprocess.copy-flag.pre

Copy flag pre-target

depend.preprocess.copy-subsidiary.pre

Copy subsidiary pre-target

depend.preprocess.copy-generated-files.pre

Copy generated files pre-target

Example

The following feature adds "myAntTargetBeforeChunk" Ant target to be executed before the chunk step in preprocessing. It assumes that an Ant file defining that target has already been integrated.

```
<plugin id="com.example.extendchunk">
<feature extension="depend.preprocess.chunk.pre" value="myAntTa
</plugin>
```

When integrated, the Ant target "myAntTargetBeforeChunk" will be added to the Ant dependency list so that it always runs immediately before the Chunk step.

Integrating a new transform type

Plug-ins may integrate an entire new transform type. The new transform type can be very simple, such as an XHTML build that creates an additional control file; it can also be very complex, adding any number of new processing steps.

The transtype extension point is used to define a new "transtype", or transform type, which makes use of targets in your Ant extensions. When a transform type is defined, the build expects Ant code to be integrated to define the transform process. The Ant code must define a target based on the name of the transform type; if the transform type is "mystuff", the Ant code must define a target named dita2mystuff.

dita.conductor.transtype.check

Add new value to list of valid transformation type names.

dita.transtype.print

Declare transtype as a print type.

Example

The following feature defines a transform type of "newtext" and declares it as a print type; using this transform type will cause the build to look for a target dita2newtext, defined in a related Ant extension from the third feature:

```
<plugin id="com.example.newtext">
    <feature extension="dita.conductor.transtype.check" value="new1
    <feature extension="dita.transtype.print" value="newtext"/>
    <feature extension="dita.conductor.target.relative" file="antW1
</plugin>
```

Override styles with XSLT

The XSLT import extension points are used to override various steps of XSLT processing. For this, the extension attribute indicates the step that the override applies to; the file attribute is a relative path to the override within the current plugin. The plugin installer will add an XSL import statement to the default code so that your override becomes a part of the normal build.

The following XSLT steps are available to override in the core toolkit:

dita.xsl.xhtml

Overrides default (X)HTML output (including HTML Help and Eclipse Help). The referenced file is integrated directly into the XSLT step that generates XHTML.

dita.xsl.xslfo

Overrides default PDF output (formerly known as PDF2). The referenced file is integrated directly into the XSLT step that generates XSL-FO for PDF.

dita.xsl.docbook

Overrides default DocBook output.

dita.xsl.rtf

Overrides default RTF output.

dita.xsl.eclipse.plugin

Overrides the step that generates plugin.xml for Eclipse.

dita.xsl.conref

Overrides the preprocess step that resolves conref.

dita.xsl.topicpull

Overrides the preprocess step "topicpull" (the step that pulls text into <xref> elements, among other things).

dita.xsl.mapref

Overrides the preprocess step "mapref" (the step that resolves references to other maps).

dita.xsl.mappull

Overrides the preprocess step "mappull" (the step that updates navtitles in maps and causes attributes to cascade).

dita.xsl.maplink

Overrides the preprocess step "maplink" (the step that generates map-based links).

Example

The following two files represent a complete, simple style plug-in. The plugin.xml file declares an XSLT file that extends XHTML processing; the XSLT file overrides default header processing to provide a (theoretical) banner.

Modifying or adding generated text

Generated text is the term for strings that are automatically added by the build, such as "Note" before the contents of a <note> element.

The generated text extension point is used to add new strings to the default set of generated text. There are several reasons you may want to use this:

- It can be used to add new text for your own processing extensions; for example, it could be used to add localized versions of the string "User response" to aid in rendering troubleshooting information.
- It can be used to override the default strings in the toolkit; for example, it could be used to reset the English string "Figure" to "Fig".
- It can be used to add support for new languages (for non-PDF transforms only; PDF requires more complicated localization support). For example, it could be used to add support for Vietnamese or Gaelic; it could also be used to support a new variant of a previously supported language, such as Australian English.

dita.xsl.strings

Add new strings to generated text file.

Example: adding new strings

First copy the file xsl/common/strings.xml to your plug-in, and edit it to contain the languages that you are providing translations for ("en-us" must be present). For this sample, copy the file into your plug-in as xsl/my-new-strings.xml. The new strings file will look something like this:

```
<!-- Provide strings for my plug-in; this plug-in supports
     English, Icelandic, and Russian. -->
<langlist>
  <lang xml:lang="en"
                          filename="mystring-en-us.xml"/>
  <lang xml:lang="en-us"
                          filename="mystring-en-us.xml"/>
 <lang xml:lang="is"
                          filename="mystring-is-is.xml"/>
                          filename="mystring-is-is.xml"/>
 <lang xml:lang="is-is"
                          filename="mystring-ru-ru.xml"/>
 <lang xml:lang="ru"
 <lang xml:lang="ru-ru"
                          filename="mystring-ru-ru.xml"/>
</langlist>
```

Next, copy the file xsl/common/strings-en-us.xml to your plug-in, and replace the content with your own strings (be sure to give them unique name attributes). Do the same for each language that you are providing a translation for. For example, the file mystring-en-us.xml might contain:

```
<strings xml:lang="en-us">
    <str name="String1">English generated text</str>
    <str name="Another String">Another String in English</str>
</strings>
```

Use the following extension code to include your strings in the set of generated text:

```
<plugin id="com.example.strings">
<feature extension="dita.xsl.strings" file="xsl/my-new-strings
</plugin>
```

The string is now available to the "getString" template used in many DITA-OT XSLT files. For example, if processing in a context where the xml:lang value is "en-us", the following call would return "Another String in English":

```
<xsl:call-template name="getString">
<xsl:with-param name="stringName" select="'Another String'"/>
```

</xsl:call-template>

Note: If two plug-ins define the same string, the results will be nondeterministic, so multiple plug-ins should not try to create the same generated text string. One common way to avoid this problem is to ensure the name attributes used to look up the string value are related to the ID or purpose of your plug-in.

Example: modifying existing strings

The process for modifying existing generated text is exactly the same as for adding new text, except that the strings you provide override values that already exist. To begin, set up the xsl/my-new-strings.xml file in your plug-in as in the previous example.

Next, copy the file xsl/common/strings-en-us.xml to your plug-in, and choose the strings you wish to change (be sure to leave the name attribute unchanged, because this is the key used to look up the string). Create a strings file for each language that needs to modify existing strings. For example, the new file mystring-en-us.xml might contain:

```
<strings xml:lang="en-us">
<str name="Figure">Fig</str>
<str name="Draft comment">ADDRESS THIS DRAFT COMMENT</str>
</strings>
```

To integrate the new strings, use the same method as above to add these strings to your plugin.xml file. Once this plug-in is integrated, where XHTML output previously generated the term "Figure", it will now generate "Fig"; where it previously generated "Draft comment", it will now generate "ADDRESS THIS DRAFT COMMENT". The same strings in other languages will not be modified unless you also provide new versions for those languages.

Note: If two plug-ins override the same string in the same language, the results will be non-deterministic (either string may be used under different conditions). Multiple plug-ins should not override the same generated text string for a single language.

Example: adding a new language

The process for adding a new language is exactly the same as for adding new text, except you are effectively just translating an existing strings file. To begin, set up the xsl/my-new-strings.xml file in your plug-in as in the previous examples. In this case, the only difference is that you are adding a mapping to new languages; for example, the following file would be used to set up support for Vietnamese:

Next, copy the file xsl/common/strings-en-us.xml to your plug-in, and rename it to match the language you wish to add. For example, to support Vietnamese strings you may want to pick a name like strings-vi.xml. In that file, change the xml:lang attribute on the root element to match your new language.

Once the file is ready, translate the contents of each <str> element (be sure to leave the name attribute unchanged). Repeat this process for each new language you wish to add.

To integrate the new languages, use the same method as above to add these strings to your plugin.xml file. Once this plug-in is integrated, non-PDF builds will include support for Vietnamese; instead of generating the English word "Caution", the element <note type="caution" xml:lang="vi"> may generate something like "chú ý".

Note: If two plug-ins add support for the same language using different values, the results will be non-deterministic (translations from either plug-in may be picked up under different conditions).

Related reference

Languages supported by the core toolkit

Passing parameters to existing XSLT steps

Plug-ins can define new parameters to be passed from the Ant build into existing XSLT pipeline stages, usually to have those parameters available as global <xsl:param> values within XSLT overrides.

To create new parameters, create a file insertParameters.xml which contains one or more Ant <param> elements. It also needs a <dummy> wrapper element around the parameters. For example, the following parameter will be passed in to the XSLT file with a value of \${antProperty}, but only if that parameter is defined:

```
<dummy>
  <!-- Any Ant code allowed in xslt task is possible. Common exar
  <param name="paramNameinXSLT" expression="${antProperty}" if="&
  </dummy>
```

Pass the value using the following extensions:

```
dita.conductor.html.param
```

Pass parameters to HTML and HTML Help XSLT

dita.conductor.xhtml.param

Pass parameters to XHTML and Eclipse Help XSLT

dita.conductor.xhtml.toc.param

Pass parameters to XHTML TOC XSLT

dita.conductor.eclipse.toc.param

Pass parameters to Eclipse Help TOC XSLT

dita.preprocess.conref.param

Pass parameters to conref XSLT

dita.preprocess.mapref.param

Pass parameters to mapref XSLT

dita.preprocess.mappull.param

Pass parameters to mappull XSLT

dita.preprocess.maplink.param

Pass parameters to maplink XSLT

dita.preprocess.topicpull.param

Pass parameters to topicpull XSLT

dita.conductor.pdf2.param

Pass parameters to PDF2 XSLT

Example

The following plug-in will pass the parameters defined inside of insertParameter.xml as input to the XHTML process. Generally, an additional XSLT override will make use of the parameter to do something new with the generated content.

```
<plugin id="com.example.newparam">
<feature extension="dita.conductor.xhtml.param" file="insertPar
</plugin>
```

Adding Java libraries to the classpath

If your Ant or XSLT extensions require additional Java libraries in the classpath, you can add them to the global DITA-OT classpath with the following feature.

dita.conductor.lib.import

Add Java libraries to DITA-OT classpath.

Example

The following plug-in adds the compiled Java code from myJavaLibrary.jar into the global DITA-OT classpath. XSLT or Ant code can then make use of the added code.

```
<plugin id="com.example.addjar">
<feature extension="dita.conductor.lib.import" file="myJavaLibu
</plugin>
```

Now assume that in this case myJavaLibrary.jar performs some validation step in the middle of processing, and you always want it to run immediately before the conref step. In that case you need to make use of several features in this plug-in

- The JAR file must be added to the classpath.
- An Ant target must be created that uses this class, and the Ant wrapper integrated into the code.
- The Ant target must be added to the dependency chain for conref.

In this extended example, the files might look something like this.

```
plugin.xml:
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="com.example.samplejava">
  <!-- Add the JAR file to the DITA-OT CLASSPATH -->
  <feature extension="dita.conductor.lib.import" file="com.examp]
  <!-- Integrate the Ant code -->
  <feature extension="dita.conductor.target.relative" file="antWi
  <!-- Define the Ant target that is called, and the location (be
  <feature extension="depend.preprocess.conref.pre" value="validation"
</plugin>
antWrapper.xml imports the new Ant code:
<?xml version="1.0" encoding="UTF-8"?>
<dummy>
  <import file="calljava-antcode.xml"/>
</dummy>
calljava-antcode.xml:
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project default="validateWithJava">
  <target name="validateWithJava">
    <java classname="com.example.sampleValidation">
        <!-- The class was added to dost.class.path (the DITA-OT cl
        <classpath refid="dost.class.path"/>
        </java>
    </target>
</project>
```

Parent topic: Creating DITA-OT plug-ins

Adding diagnostic messages

Plug-in specific warning and error messages can be added to the set of messages supplied by the DITA-OT. These messages can then be used by any XSLT override.

dita.xsl.messages

Add new messages to diagnostic message file.

Example

To add your own messages, create the new messages in an XML file such as myMessages.xml:

```
<dummy>
  <!-- See resource/messages.xml for the details. -->
  <message id="DOTXmy-msg-numW" type="WARN">
        <reason>Message text</reason>
        <response>How to resolve</response>
        </message>
  </dummy>
```

There are three components to the message ID:

- 1. The prefix DOTX is used by all DITA-OT XSLT transforms, and must be part of the ID.
- 2. This is followed by the message number ("my-msg-num" in the sample above). By convention, this should be a three digit integer.
- 3. Finally, a letter corresponds to the severity. This should be one of:
 - I = Informational, used with type="INFO"
 - W = Warning, used with type="WARN"
 - E = Error, used with type="ERROR"
 - F = Fatal, used with type="FATAL"

Once the message file is defined, it is incorporated with this extension:

```
<plugin id="com.example.newmsg">
<feature extension="dita.xsl.messages" file="myMessages.xml"/>
</plugin>
```

XSLT modules can then generate the message using the following call:

```
<xsl:call-template name="output-message">
<xsl:with-param name="msgnum">my-msg-num</xsl:with-param>
<xsl:with-param name="msgsev">W</xsl:with-param>
</xsl:call-template>
```

Parent topic: Creating DITA-OT plug-ins

Managing plug-in dependencies

The <require> element in a plugin.xml file is used to create a dependency on another plug-in. The <require> element requires the plugin attribute in order to reference the dependency.

If the current plug-in requires a plug-in with id="plugin-id" before it can be installed, it would include the following:

```
<require plugin="plugin-id">
```

Prerequisite plug-ins are integrated before the current plug-in is integrated. This does the right thing with respect to XSLT overrides. If your plug-in is a specialization of a specialization, it should <require> its base plug-ins, in order from general to specific.

If a prerequisite plug-in is missing, a warning will be printed during integration. To suppress this, but keep the integration order if both plug-ins are present, add importance="optional" to the <require> element.

If your plug-in can depend on any one of several optional plug-ins, separate the plug-in ids with a vertical bar. This is most useful when combined with importance="optional":

Example

The following plug-in will only be installed if the plug-in with id="com.example.primary" is available. If that one is not available, a warning will be generated during the integration process.

```
<plugin id="com.example.builds-on-primary">
  <!-- ...extensions here -->
  <require plugin="com.example.primary"/>
  </plugin>
```

The following plug-in will only be installed if either the plug-in with id="pluginA" or the plug-in with id="pluginB" are available. If neither of those are installed, the current plug-in will be ignored.

```
<plugin id="pluginC">
    <!-- ...extensions here -->
    <require plugin="pluginA|pluginB" importance="optional"/>
</plugin>
```

Parent topic: Creating DITA-OT plug-ins

Version and support information

The following extension points are used by convention to define version and support info within a plug-in.

- package.support.name
- package.support.email
- package.version

Note:

The toolkit does not currently do anything with these values, but may do so in the future.

The package.version value should follow the syntax rules:

```
version ::= major ( '.' minor ( '.' micro ( '.' qualifier )? )'
major ::= number
minor ::= number
micro ::= number
qualifier ::= ( [0..9] | [a..zA..Z] | '_' | '-' )+
```

The default value is 0.0.0.

Example

```
<plugin id="com.example.WithSupportInfo">
    <feature extension="package.support.name" value="Joe the Authon
    <feature extension="package.support.email" value="joe@example.c
    <feature extension="package.version" value="1.2.3"/>
</plugin>
```

Parent topic: Creating DITA-OT plug-ins

Creating a new plug-in extension point

If your plug-in needs to define its own extension point in an XML file, add the string "_template" to the filename before the file suffix. During integration, this file will be processed like the built-in DITA-OT templates.

Template files are used to integrate most DITA-OT extensions. For example, the file dita2xhtml_template.xsl contains all of the default rules for converting DITA topics to XHTML, along with an integration point for plug-in extensions. When the integrator runs, the file dita2xhtml.xsl is recreated, and the integration point is replaced with references to all appropriate plug-ins.

To mark a new file as a template file, use the <template> element.

The template extension namespace has the URI http://ditaot.sourceforge.net. It is used to identify elements and attributes that have a special meaning in template processing. This documentation uses a prefix of dita: for referring to elements in the template extension namespace. However, template files are free to use any prefix, provided that there is a namespace declaration that binds the prefix to the URI of the template extension namespace.

dita:extension element

The dita:extension elements are used to insert generated content during integration process. There are two required attributes:

- The id attribute defines the extension point ID which provides the argument data.
- The behaviour attribute defines which processing action is used.

Supported values for behavior attribute:

org.dita.dost.platform.CheckTranstypeAction

Create Ant condition elements to check if \${transtype} property value equals a supported transtype value.

org.dita.dost.platform.ImportAntLibAction

Create Ant pathelement elements for <u>library imported extension</u> point. The id attribute is used to define the extension point ID.

org.dita.dost.platform.ImportPluginCatalogAction

Include plug-in metadata catalog content.

org.dita.dost.platform.ImportPluginInfoAction

Create plug-in metadata Ant properties.

org.dita.dost.platform.ImportStringsAction

Include plug-in string file content base on <u>generated text</u> <u>extension point</u>. The id attribute is used to define the extension point ID.

org.dita.dost.platform.ImportXSLAction

Create xs1:import elements based on XSLT import extension

point. The id attribute is used to define the extension point ID.

org.dita.dost.platform.InsertAction

Include plug-in conductor content based on <u>Ant import extension</u> point. The id attribute is used to define the extension point ID.

org.dita.dost.platform.InsertAntActionRelative

Include plug-in conductor content based on <u>relative Ant import</u> <u>extension point</u>. The id attribute is used to define the extension point ID.

org.dita.dost.platform.InsertCatalogActionRelative

Include plug-in catalog content based on <u>catalog import</u> <u>extension point</u>. The id attribute is used to define the extension point ID.

org.dita.dost.platform.ListTranstypeAction

Create a pipe delimited list of supported transtypes.

dita:extension attribute

The dita:extension attribute is used to process attributes in elements which are not in template extension namespace. The value of the attribute is a space delimited tuple, where the first item is the name of the attribute to process and the second item is the action ID.

Supported values:

depends org.dita.dost.platform.InsertDependsAction

Ant target dependency list is processed to replace all target names which start with an open curly bracket and end with a close curly bracket. The value of the extension point is the ID between the curly brackets.

Example

The following plug-in defines myBuildFile_template.xml as a new template for extensions, and two new extension points.

When the integrator runs, this will be used to recreate myBuildFile.xml, replacing Ant file content based on extension point use.

Parent topic: Creating DITA-OT plug-ins

Example plugin.xml file

The following is a sample of a plugin.xml file. This file adds support for a new set of specialized DTDs, and includes an override for the XHTML output processor.

This plugin.xml file would go into a directory such as DITA-OT\plugins\music\ and referenced supporting files would also exist in that directory. A more extensive sample using these values is available in the actual music plug-in, available at the <u>DITA-OT download page</u> at SourceForge

```
<plugin id="org.metadita.specialization.music">
<feature extension="dita.specialization.catalog.relative" file=
<feature extension="dita.xsl.xhtml" file="xsl/music2xhtml.xsl",
</plugin>
```

Parent topic: Creating DITA-OT plug-ins

XHTML migration for flagging updates in DITA-OT 1.7

This topic is primarily of interest to developers with XHTML transform overrides written prior to DITA-OT 1.7. Due to significant changes in the flagging process with the 1.7 release, some changes may be needed to make overrides work properly with DITAVAL based flagging. The new design is significantly simpler than the old design; in many cases, migration will consist of deleting old code that is no longer needed.

Which XHTML overrides need to migrate?

If your override does not contain any code related to DITAVAL flagging, then there is nothing to migrate.

If your builds do not make use of DITAVAL based flagging, but calls the deprecated flagging templates, then you should override but there is little urgency. You will not see any difference in the output, but those templates will be removed in a future release.

If you do make use of DITAVAL based flagging, try using your override with 1.7. Check the elements you override:

- 1. In some cases flags may be doubled. This will be the case if you call routines such as "start-flagit".
- 2. In some cases flags may be removed. This will be the case if you call shortcut routines such as "revtext" or "revblock".
- 3. In other cases, flags may still appear properly, in which case migration is less urgent

For any migration that needs migration, please see the instructions that follow.

Deprecated templates in DITA-OT 1.7

All of the old DITAVAL based templates are deprecated in DITA-OT 1.7. If your overrides include any of the following templates, they should be migrated for the new release; in many cases the templates below will not have any effect on your output, but all instances should be migrated.

- The "gen-style" template used to add CSS styling
- The "start-flagit" and "end-flagit" templates used to generate image flags based on property attributes like @audience
- The "start-revflag" and "end-revflag" templates, used to generate images for active revisions
- Shortcut templates that group these templates into a single call, such as:
 - "start-flags-and-rev" and "end-flags-and-rev", used to combine flags and revisions into one call
 - "revblock" and "revtext", both used to output start revisions, element content, and end revisions
 - The modes "outputContentsWithFlags" and "outputContentsWithFlagsAndStyle", both used to combine processing for property/revision flags with content processing
- All other templates that make use of the *flagrules* variable, which is no longer used in any of the DITA-OT 1.7 code
- All templates within flag.xsl that were called from the templates listed above
- Element processing handled with mode="elementname-fmt", such as mode="ul-fmt" for processing unordered lists and mode="section-fmt" for sections.

What replaces the templates?

The new flagging design described in the preprocess design section now adds literal copies of relevant DITAVAL elements, along with CSS based flagging information, into the relevant section of the topic. This allows most flags to be processed in document order; in addition, there is never a need to read the DITAVAL, interpret CSS, or evaluate flagging logic. The htmlflag.xsl file contains a few rules to match and process the start/end flags; in most cases, all code to explicitly process flags can be deleted.

For example, the common logic for most element rules before DITA-OT 1.7 could be boiled down to the following:

Match element

Create "flagrules" variable by reading DITAVAL for active flags Output start tag such as <div> Or

Call "commonattributes" and ID processing

Call "gen-style" with \$flagrules, to create DITAVAL based CSS

Call "start-flagit" with \$flagrules, to create start flag images

Call "start-revflag" with \$flagrules, to create start revision images Output contents

Call "end-revflag" with \$flagrules, to create end revision images

Call "end-flagit" with \$flagrules, to create end flag images

Output end tag such as </div> or

In DITA-OT 1.7, style and images are typically handled with XSLT fallthrough processing. This removes virtually all special flag coding from element rules, because flags are already part of the document and processed in document order. The sample above is reduced to:

Match element

Output start tag such as <div> Or

Call "commonattributes" and ID processing

Output contents

Output end tag such as </div> or

Migrating "gen-style" named template

Calls to the "gen-style" template should be deleted. There is no need to replace this call for most elements.

The "gen-style" template was designed to read a DITAVAL file, find active style-based flagging (such as colored or bold text), and add it to the generated @style attribute in HTML.

With DITA-OT 1.7, the style is calculated in the pre-process flagging module. The result is created as @outputclass on a <ditaval-startprop> sub-element. The "commonattributes" template now includes a line to process that value; the result is that for every element that calls "commonattributes", DITAVAL style will be processed when needed. Because virtually every element includes a call to this common template, there is little chance that your override needs to explicitly process the style. The new line in "commonattributes" that handles the style is:

<xsl:apply-templates select="*[contains(@class,' ditaot-d/ditava]

Migrating "start-flagit", "start-revflag", "end-flagit", and "end-flagit" named templates

Calls to these templates fall into two general groups.

If the flow of your element rule is to create a start tag like <div>, "start-flagit"/"start-revflag", process contents, "end-revflag"/"end-flagit", end tag - you just need to delete the calls to these templates. Flags will be generated simply by processing the element contents in document order.

If the flow of your element rule processes flags outside of the normal document-order. There are generally two reasons this is done. The first case is for elements like , where flags must appear before the in order to create valid XHTML. The second is for elements like <section>, where start flags are created, followed by the title or some generated text, element contents, and finally end flags. In either of these cases, support for processing flags in document order is disabled, so they must be explicitly processed out-of-line. This is done with the following two lines (one for start flag/revision, one for end flag/revision):

```
Create starting flag and revision images:
<xsl:apply-templates select="*[contains(@class,' ditaot-d/ditava]
Create ending flag and revision images:
<xsl:apply-templates select="*[contains(@class,' ditaot-d/ditava]
```

For example, the following lines are used in DITA-OT 1.7 to process the element (replacing the 29 lines used in DITA-OT 1.6):

Migrating "start-flags-and-rev" and "end-flags-and-rev"

- "start-flags-and-rev" is equivalent to calling "start-flagit" followed by "start-revflag"; it should be migrated as in the previous section.
- "end-flags-and-rev" is equivalent to calling "end-revflag" followed by "end-flagit"; it should be migrated as in the previous section.

Migrating "revblock" and "revtext"

Calls to these two templates can be replaced with a simple call to <xsl:apply-templates/>.

Migrating modes "outputContentsWithFlags" **and** "outputContentsWithFlagsAndStyle"

Processing an element with either of these modes can be replaced with a simple call to <xsl:apply-templates/>.

Migrating mode="elementname-fmt"

Prior to DITA-OT 1.7, many elements were processed with the following logic:

```
Match element
Set variable to determine if revisions are acti
If active
    create division with rev style
    process element with mode="elementname-
    end division
Else
    process element with mode="elementname-fmt"
Match element with mode="elementname-fmt"
Process as needed
```

Beginning with DITA-OT 1.7, styling from revisions is handled automatically with the "commonattributes" template. This means there is no need for the extra testing, or the indirection to mode="elementnamefmt". These templates are deprecated, and element processing will move into the main element rule. Overrides that include this indirection may remove it; overrides should also be sure to match the default rule, rather than matching with mode="elementname-fmt".

Parent topic: DITA Open Toolkit Developer Reference

Customizing PDF output

You can build a DITA-OT plug-in that contains a customized PDF transformation.

About this task

This topic demonstrates the process of building a plug-in (com.example.print-pdf) that creates a new transformation type: **print-pdf**. The **print-pdf** transformation has the following characteristics:

- Uses A4 paper
- Renders figures with a title at the top and a description at the bottom
- Use em dashes as the symbols for unordered lists

Procedure

- In the plugins directory, create a directory named com.example.print-pdf.
- 2. In the new com.example.print-pdf directory, create a plug-in configuration file (plugin.xml) that declares the new **print-pdf** transformation and its dependencies.

Figure 1. plugin.xml file

```
<?xml version='1.0' encoding='UTF-8'?>
<plugin id="com.example.print-pdf">
    <require plugin="org.dita.pdf2"/>
    <feature extension="dita.conductor.transtype.check" value='
    <feature extension="dita.transtype.print" value="print-pdf"
    <feature extension="dita.conductor.target.relative" file=":
</plugin>
```

3. Add an Ant script (integrator.xml) to define the transformation type. *Figure 2. integrator.xml file*

```
<?xml version='1.0' encoding='UTF-8'?>
<project name="com.example.print-pdf">
        <target name="dita2print-pdf.init">
            <property name="customization.dir" location="${dita.plug:
            </target>
            <target name="dita2print-pdf" depends="dita2print-pdf.init"
</project>
```

4. In the new plug-in directory, add a cfg/catalog.xml file that specifies the custom XSLT style sheets.

Figure 3. cfg/catalog.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog prefer="system" xmlns="urn:oasis:names:tc:entity:xm.
<uri name="cfg:fo/attrs/custom.xsl" uri="fo/attrs/custom.x:
```

```
<uri name="cfg:fo/xsl/custom.xsl" uri="fo/xsl/custom.xsl"/:
</catalog>
```

5. Create the cfg/fo/attrs/custom.xsl file, and add attribute and variable overrides to it.

For example, add the content highlighted with bold to change the page size to A4.

Figure 4. cfg/fo/attrs/custom.xs1 file

 Create the cfg/fo/xsl/custom.xsl file, and add XSLT overrides to it. For example, the following code changes the rendering of <figure> elements.

Figure 5. cfg/fo/xs1/custom.xs1 file

 Create an English-language variable-definition file (cfg/common/vars/en.xml) and make any necessary modifications to it.

For example, the following code removes the period after the number for an ordered-list item; it also specifies that the bullet for an unordered list item should be an em dash.

Figure 6. cfg/common/vars/en.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<vars xmlns="http://www.idiominc.com/opentopic/vars">
    <!-- Remove dot from list number -->
        <variable id="Ordered List Number"><param ref-name="number"
        <!-- Change unordered list bullet to an em dash -->
        <variable id="Unordered List bullet">&#x2014;</variable>
</vars>
```

Results

The new plug-in directory has the following layout and files:

```
com.example.print-pdf/
cfg/
    common/
    vars/
    en.xml
    fo/
    attrs/
        custom.xsl
        xsl/
        custom.xsl
        catalog.xml
    integrator.xml
    plugin.xml
```

What to do next

Run the integration process to install the plug-in and make the **print-pdf** transformation available.

Parent topic: DITA Open Toolkit Developer Reference

Internal Ant properties

Reference list of Ant properties used by DITA-OT internally.

include.rellinks

Space separated list of link roles to be output; value token #default denotes default role value. Property default value depends on transformation type. Defined by args.rellinks, but may be overridden directly.

Parent topic: DITA Open Toolkit Developer Reference

Implementation dependent features

Chunking

Supported chunking methods:

- select-topic
- select-document
- select-branch
- by-topic
- by-document
- to-content
- to-navigation.

When no chunk attribute values are given, no chunking is performed.

Note: In effect, for HTML based transformation types this is equivalent to select-document and by-document defaults.

Error recovery:

- When two tokens from the same category are used, no error or warning is thrown.
- When an unrecognized chunking method is used, no error or warning is thrown.

Filtering

Error recovery:

- When there are multiple revprop elements with the same val attribute, no error or warning is thrown
- When multiple prop elements define a duplicate attribute and value combination, attribute default, or fall-back behaviour, DOTJ007E error is thrown.

Debug attributes

The debug attributes are populated as follows:

xtrf

absolute system path of the source document

xtrc

element counter that uses the format

```
element-name ":" integer-counter ";" line-number ":" col
```

Image scaling

If both height and width attributes are given, image is scaled nonuniformly.

If scale attribute is not an unsigned integer, no error or warning is thrown during preprocessing.

Map processing

When a topicref element that references a map contains child topicref elements, DOTX068W error is thrown and the child topicref elements are ignored.

Link processing

When the value of href attribute is not a valid URI reference, DOTJ054E error is thrown. Depending on <u>error recover mode</u>, error recover may be attempted.

Copy-to processing

When the copy-to attribute is specified on a topicref, the content of the shortdesc element is not used to override the short description of the topic.

Parent topic: DITA Open Toolkit Developer Reference

Extended functionality

Parent topic: DITA Open Toolkit Developer Reference

Code reference processing

Charset definition

DITA-OT supports defining the code reference target file encoding using the format attribute. The supported format is:

```
format (";" space* "charset=" charset)?
```

If charset is not defined system default charset will be used. If charset is not recognized or supported, DOTJ052E error is thrown and system default charset is used as a fall-back.

```
<coderef href="unicode.txt" format="txt; charset=UTF-8"/>
```

Line range extraction

Code reference can extract only a given line ranges with line-range pointer in the URI fragment. The format is:

uri ("#line-range(" start ("," end)? ")")?

Start and end line numbers start from 1 and are inclusive. If end range is omitted, range ends in last line of the file.

```
<coderef href="Parser.scala#line-range(5, 10)" format="scala"/>
```

Only lines from 5 to 10 will be included in the output.

RFC 5147

DITA-OT implements line position and range from <u>RFC 5147</u>. The format for line range is:

```
uri ("#line=" start? "," end? )?
```

Start and end line numbers start from 0 and are inclusive and exclusive, respectively. If the start range is omitted, range starts from the first line; if end range is omitted, range ends in last line of the file. The format for line

position is:

uri ("#line=" position)?

Position line number starts from 0.

<coderef href="Parser.scala#line=4,10" format="scala"/>

Only lines from 5 to 10 will be included in the output.

DITA and DITA-OT resources

In addition to the DITA-OT documentation, there are other resources about DITA and the DITA-OT that you might find helpful.

DITA-OT project page at dita.xml.org The DITA-OT project page at dita.xml.org provides news about the latest toolkit builds, plans for the next milestone release, and other rapidly-changing information. It also contains release notes for all past and upcoming releases.

Yahoo! dita-users group

This list-serv is a vital resource for the DITA community. People post regularly, both asking for and offering help. While the archived messages can be difficult to search, this is a treasure trove of information.

Home page for the OASIS DITA Technical Committee

The OASIS DITA Technical Committee develops the DITA standard.

Web-based resources

There are many vital DITA resources online, including the Yahoo! dita-users group and the DITA-OT project page at dita.xml.org.

developerWorks articles

Between 2001 and 2005, IBM DITA experts published an important collection of articles on the developerWorks Web site.

Web-based resources

There are many vital DITA resources online, including the Yahoo! ditausers group and the DITA-OT project page at dita.xml.org.

DITA-OT project page at dita.xml.org

The DITA-OT project page at dita.xml.org provides news about the latest toolkit builds, plans for the next milestone release, and other rapidly-changing information. It also contains release notes for all past and upcoming releases.

Yahoo! dita-users group

The DITA-OT project page at dita.xml.org provides news about the latest toolkit builds, plans for the next milestone release, and other rapidly-changing information. It also contains release notes for all past and upcoming releases.

Home page for the OASIS DITA Technical Committee

The OASIS DITA Technical Committee develops the DITA standard.

Parent topic: DITA and DITA-OT resources

developerWorks articles

Between 2001 and 2005, IBM DITA experts published an important collection of articles on the developerWorks Web site.

- Introduction to the Darwin Information Typing Architecture
- Specializing topic types in DITA
- Specializing domains in DITA
- Frequently Asked Questions about the Darwin Information Typing Architecture
- Why use DITA to produce HTML deliverables?
- Design patterns for information architecture with DITA map domains
- <u>Migrating HTML to DITA, Part 1: Simple steps to move from HTML to DITA</u>
- Migrating HTML to DITA, Part 2: Extend the migration for more robust results
- Transform Eclipse navigation files to DITA navigation files

Parent topic: DITA and DITA-OT resources

generate.outer.copy parameter

Elaboration on how the **generate.outer.copy** parameter functions.

Background

This is an issue in the following situations:

- The DITA map is in a directory that is a peer to directories that contain referenced objects.
- The DITA map is in a directory that is below the directories that contain the referenced objects.

Let's assume that the directory structure for the DITA content looks like the following:

maps topics images

The DITA map is in the maps directory, the topics are in the topics directory, and the images are in the images directory.

Setting the generate.outer.copy parameter to 1

Let's assume that you run the HTML5 transformation and specify an output directory of c:\A-test. By default, The DITA-OT uses the **generate.outer.copy** parameter with a value of **1**. Output is not built for the topics. You receive only the following output:

- C:\A-test
- --- index.html
- --- commonltr.css
- --- commonrtl.css

The index.html file contains the navigation structure, but all the links are broken, since no HTML5 files were built for the topics.

How do you fix this? By specifying a value of **3** for the **generate.outer.copy** parameter.

Setting the generate.outer.copy parameter to 3

Now your output directory structure looks like this:

C:\A-test --- images\ --- maps\ --- topics\

The index.html file is in the maps directory, and the CSS and other files are located in the output directory, c:\A-test. Copying the output directory is simplified.

Parent topic: Ant parameters: Common HTML-based transformations