DITA Open Toolkit 2.2

Table of Contents

DITA Open Toolkit 2.2 6

DITA Open Toolkit 2.2.3 Release Notes 7

Getting Started with the DITA Open Toolkit 13
- Installing the distribution package 13
- Building output using the dita command 13

DITA Open Toolkit User Guide 14
- DITA Open Toolkit Overview 14
  - DITA specification support 14
    - DITA 1.2 support 14
    - DITA 1.3 support 15
  - Tested platforms and tools 16
- Installing the DITA Open Toolkit 17
  - Prerequisite software 17
  - Installing the distribution package 18
- Publishing DITA content 18
  - Building output using the dita command 18
    - Setting build parameters with .properties files 19
    - Migrating Ant builds to use the dita command 20
  - Publishing DITA content from Ant 22
    - Ant 22
    - Building output using Ant 23
    - Creating an Ant build script 23
- DITA-OT transformations (output formats) 25
  - DITA to DocBook 25
  - DITA to Eclipse Content 25
  - DITA to Eclipse help 26
  - DITA to HTML5 26
  - DITA to HTML Help (CHM) 27
  - DITA to Java Help 27
  - DITA to Open Document Type 27
  - DITA to PDF (PDF2) 28
    - Generating revision bars 28
  - DITA to Rich Text Format 29
  - DITA to TocJS 29
  - DITA to troff 29
  - DITA to XHTML 29
- Globalizing DITA content 30
  - Globalization support offered by the DITA-OT 30
  - Supported languages: HTML-based transformations 30
  - Supported languages: PDF transformations 33

Error messages and troubleshooting 34
[/map/opentopic:map/topicref/topicref/topicref {"- map/topicref "}) (topicref]
Other error messages 34
Log files 35
Accessing help for the dita command 35
Checking the DITA-OT version 36
Enabling debug mode 36
Increasing Java memory allocation 36
Reducing processing time 37

DITA Open Toolkit Parameter Reference 38
DITA-OT parameters 38
Common 1
HTML-based output formats 1
generate.copy.outer parameter 0
HTML5 and XHTML 1
[/map/opentopic:map/topicref/topicref/topicref {"- map/topicref "}) (topicref]
Eclipse content 1
Eclipse Help 1
HTMLHelp 1
JavaHelp 1
Open Document Format 1
Other parameters 38
PDF 1
Arguments and options for the dita command 38
Configuration properties 40
plugin.properties file 40
configuration.properties file 40
Internal Ant properties 41

DITA Open Toolkit Developer Reference 42
Architecture of the DITA Open Toolkit 42
Processing structure 42
Processing modules 43
Processing order 44
Pre-processing modules 45
Generate lists (gen-list) 45
Debug and filter (debug-filter) 47
Resolve map references (mapref) 47
Copy related files (copy-files) 47
Resolve keyref (keyref) 47
Conref push (conrefpush) 48
Conref (conref) 48
Resolve code references (coderef) 48
Chunk topics (chunk) 49
Move metadata (move-meta-entries) and pull content into maps (mappull) 49
Map based linking (maplink) 50

Pull content into topics (topicpull) 50
Flagging in the toolkit 50
HTML-based processing modules 54
Common HTML-based processing 54
XHTML processing 54
HTML5 processing 54
Eclipse help processing 54
TocJS processing 55
HTML Help processing 56
JavaHelp processing 56
PDF processing modules 56
Open Document Format processing modules 57
Extending the DITA Open Toolkit 58
Installing plug-ins 58
Removing plug-ins 58
Rebuilding the DITA-OT documentation 59
Creating plug-ins 59
Overview of plug-ins 59
Benefits of plug-ins 59
Plug-in descriptor file 60
Plug-in dependencies 63
Extending an XML catalog file 63
Adding a new target to the Ant build process 64
Adding an Ant target to the pre-processing pipeline 65
Adding a new transformation type 66
Overriding an XSLT-processing step 67
Modifying or adding generated text 68
Adding parameters to existing XSLT steps 70
Adding a Java library to the DITA-OT classpath parameter 71
Adding new diagnostic messages 72
Creating a new plug-in extension point 73
Example plugin.xml file 75
Extension points 76
General extension points 76
Pre-processing extension points 76
XSLT-import extension points 78
XSLT-parameter extension points 79
Version and support information 81
[/map/opentopic:map/topicref/topicref/topicref {"- map/topicref "}) (topicref]
Customizing PDF output 81
Migrating customizations 84
Migrating to release 2.2 84
Migrating to release 2.1 84
Migrating to release 2.0 85
Migrating to release 1.8 85
Migrating to release 1.7 87
XHTML migration for flagging updates in DITA-OT 1.7 89
Migrating to release 1.6 92
Migrating to release 1.5.4 95

Implementation dependent features 97
Extended functionality 99

DITA and DITA-OT resources 101
DITA-OT project website at dita-ot.org 1
DITA-OT project archive at dita.xml.org 1
Yahoo! dita-users group 1
DITA-OT Users Google Group 1
DITA-OT Development Google Group 1
Home page for the OASIS DITA Technical Committee 1

Index 102

DITA Open Toolkit 2.2

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.

DITA-OT documentation

The DITA Open Toolkit documentation provides information about installing, running, configuring and extending the toolkit. Each part is available in the navigation panel on the left.

- See the DITA Open Toolkit 2.2.3 Release Notes for information on the changes in the current release.
- *Getting Started* provides 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 toolkit and building output.
- The *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 information on the supported DITA versions and the versions of components that have been tested for use with the toolkit.
- The *Parameter Reference* is designed to help users to locate information easily and quickly. It includes documentation for the DITA-OT parameters and configuration properties.
- The *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, extending the DITA-OT, and creating DITA-OT plug-ins.

Shortcuts to important information
Latest DITA-OT release
DITA-OT source code and development builds (GitHub)

DITA Open Toolkit 2.2.3 Release Notes

DITA Open Toolkit 2.2.3 is a maintenance release that includes fixes for issues reported in DITA-OT 2.2, which includes new features and enhancements and provides additional support for the OASIS DITA 1.3 specification.

Issue numbers correspond to the tracking number in the GitHub issues tracker.

Maintenance Release 2.2.3

DITA Open Toolkit Release 2.2.3 includes the following bug fixes.

- Specifying the source of a long quote with a `@keyref` attribute on a `<longquoteref>` element caused a Null Pointer Exception. The keyref parser has been modified to ensure that key references on long quote references are correctly resolved. #2246
- The keyscope cascading mechanisms for branch filtering have been modified to comply with the DITA 1.3 specification. In cases where no `@keyscope` attribute is specified on a filtered branch, the prefix and/or suffix values specified in `<dvrKeyscopePrefix>` / `<dvrKeyscopeSuffix>` are now used as the effective key scope names. Key scope prefixes and suffixes are no longer propagated to descendant key scopes, but only applied to the branch parent. #2240
- PDF: In DITA-OT 2.2.2, if a `<link>` element contained both `<linktext>` and a description in the `<desc>` element, the hyperlink was rendered with the contents of the `<desc>` element appended to the contents of the `<linktext>` element. Link processing has been modified to ensure that `<link>` elements with descriptions are rendered correctly. #2236
- PDF: Index term processing has been revised to ensure that indexes in PDF output generated with Apache FOP correctly include page ranges when `@start` and `@end` attributes are defined on `<indexterm>` elements and the terms of the index range appear on different pages. A single page reference is generated in cases where both occurrences of the index term appear on the same page. #2214
- ODT: The OpenDocument Text transformation has been revised to support SVG images for which image dimensions are not specified via `@height` and `@width` attributes. #2211

Maintenance Release 2.2.2

DITA Open Toolkit Release 2.2.2 included the following bug fixes.

- PDF: In earlier DITA-OT 2.x releases, short description content was not rendered when publishing from a bookmap with the args.chapter.layout parameter set to BASIC. Short descriptions are now rendered properly. #2023
- *Windows:* The name of the JAR file for Apache Ant has been corrected to fix the `ANT_HOME is set incorrectly or ant could not be located` error that appeared when running the bundled Ant version on Windows in DITA-OT 2.2. #2172
- The order in which plugins are integrated has been revised to guarantee that the base DITA-OT plugins always come first. This ensures that base target code cannot be overridden. #2180
- Content references with relative paths were not properly resolved to the target document in some cases. The conref base URI resolution mechanism was fixed to ensure that relative references are now resolved correctly. #2192

- The documentation has been updated with links to the final DITA 1.3 specification.

Maintenance Release 2.2.1

DITA Open Toolkit Release 2.2.1 included the following bug fixes.

- PDF: Several templates in the pr-domain.xsl file used `value-of` rather than `apply-templates` to output the element contents. This prevented flagging from working as intended in PDF output. This has been corrected to ensure that elements in the programming domain are properly flagged. #2170
- The name of the JAR file for the Apache Commons library that provides reusable Java components has been corrected in the integrator Ant script to fix the `NoClassDefFoundError` error that appeared when the integrator ran in DITA-OT 2.2. #2163
- PDF title metadata is now generated correctly for DITA maps that set the `@xml:lang` attribute. In certain cases, previous versions of DITA-OT displayed the `XMP structure: 1` label in the Document Properties dialog instead of the map title. #2154

Requirements

DITA Open Toolkit Release 2.2 requires the Java Runtime Environment (JRE) version 7 or later.

Release Highlights

DITA 1.3 support

DITA Open Toolkit 2.2 provides processing support for the OASIS DITA 1.3 specification. Initial preview support for this specification was added in version 2.0 of the toolkit; version 2.2 extends this foundation to support key scopes and branch filtering along with additional DITA 1.3 features.

Because DITA 1.3 is fully backwards compatible with previous DITA DTDs and schemas, DITA-OT 2.2 provides the 1.3 materials as the default DTDs for processing. The XML Catalog resolution maps any references for unversioned DITA doctypes to the 1.3 DTDs. All processing ordinarily dependent on the 1.0, 1.1, or 1.2 definitions continues to work as usual, and any documents that make use of the newer DITA 1.3 elements or attributes will be supported with specific new processing.

DITA Open Toolkit Release 2.2 extends DITA 1.3 support with the following enhancements:
Important: The DITA 1.3 grammars are now used as the default DTDs for processing #2094

- Initial implementation of DITA 1.3 branch filtering #1969, #1637
- Initial support for DITA 1.3 key scopes, including multiple scope names in a single `@keyscope` attribute #1979, #1648, #2004
- The `@keyref` attribute is now supported on `<object>` elements #1783
- Processing order has been revised to process any same topic fragments used in conrefs before the conref phase, to enable content references to elements in the same topic using a reference such as `<p conref="#./ID"/>` as reported in #1649. #1968

New HTML5 output

- The HTML5 transformation has been moved to a separate plugin to facilitate customization of modern HTML5-based output. The default CSS has been refactored in Sass as a foundation for further

extensions with CSS frameworks, custom plugins and future toolkit versions. (The original XHTML plugin remains available to support existing legacy HTML-based output formats such as TocJS, HTML Help and JavaHelp.) #2099
- The HTML5 transformation has been extended with a new nav-toc parameter that can be used to generate a table of contents in the HTML5 `<nav>` element of each page. The navigation can then be rendered in a sidebar or menu via CSS. This parameter is disabled by default (none), but can be enabled by setting the value to partial (which includes the current topic in the ToC along with its parents, siblings and children), or full (which generates a ToC for the entire map). #2103

PDF supports new languages

The PDF transformation has been extended to support additional languages with localized strings files and index collation. #2085, #2089

- Arabic
- Catalan
- Croatian
- Czech
- Danish
- Hungarian
- Icelandic
- Latvian
- Norwegian
- Polish
- Portuguese
- Portuguese (Brazil)
- Slovak
- Turkish

Related concepts
DITA 1.3 support
Related information
DITA 1.3 specification

Resolved issues

In addition to the highlights mentioned above, DITA Open Toolkit Release 2.2 includes the following changes.

Features

DITA Open Toolkit Release 2.2 includes the following new features:

- The args.artlbl parameter previously provided in HTML-based transformations to show the source file name along with referenced images has been added to the PDF transformation. For inline images, the label appears immediately after the image; for standalone images, it appears on a line following the image. #2072, #2062
- In PDF output, any index entries that begin with special characters are now moved to a dedicated section of the index. (Previous toolkit versions issued a warning and dropped any unexpected index terms.) #2073, #2071

- The plugin configuration file resources/plugins.xml generated by the integration process now adds the `@xml:base` attribute to each plugin configuration element so that file paths in the plugin configuration can be resolved relative to the plugin folder. This makes it easier to share a DITA-OT distribution with others via a version control system or bundle it with an application without re-running the integration process in each installation location. #2018
- The `move-meta-entries` and `mappull` steps have been merged. The `mappull` step has been moved into `move-meta-entries` and an empty target for mappull is retained for backwards compatibility. #1995
- The generation of page masters and page sequence masters in the PDF plugin has been modularized to make it easier to extend or override the default set of masters. #1984
- The PDF plugin has been extended with two new extension points for `flagging-preprocess` and `i18n-postprocess` to alllow plugins to customize these steps of the flagging process. #1976
- The key definition reading process has been moved from the `gen-list` preprocessing module to the `keyref` module to further modularize the code and support additional DITA 1.3 use cases for keys, such as a map that includes a map that then includes another map with a key definition. #1962
- The mapref processing stage has been moved to the beginning of the preprocessing pipeline to simplify keyref processing and support DITA 1.3 branch filtering by allowing all processing to be performed on a single map. #1961
- The dita -install option can now be used to integrate multiple plugins at once. If no *file* or *url* argument is provided, the integration process reloads plugins from the plugins directory, so you can unzip multiple plugins to the plugins directory and run dita -install to integrate them all at once. #1957
- The PDF transformation has been extended with XML catalog support for RenderX XEP. The main DITA-OT catalog file is now used to ensure that doctypes such as SVG graphics pass validation when generating PDF output using XEP. #1955
- The Java code has been refactored to use URI-based processing (rather than File objects or Strings) wherever possible to permit automatic validation of values and support the use of external URIs. This allows non-file resources to be processed with commands such as dita -i http://example.com/test.ditamap -f html5. #1544, #1953
- Plugin folders are now sorted before the integration process runs to ensure predictable results and consistent order each time plugins are reloaded. #1905
- The metadata format in the plugin.xml file for the org.dita.xhtml plugin has been refactored with abstract transtypes that group common parameters used in multiple transformation types. This approach allows common parameters to be defined in one place and re-used for multiple output formats as necessary. #1866
- The dita command options have been extended to add -t as a synonym for the -temp option used to specify the location of the temporary directory. #1836, #2039
- The validation of the table group `@cols` attribute has been relaxed to support use cases in which tables containing auto-generated `@cols` values are reused via content references. #1835
- The format of the plugin.xml file has been extended to allow plugins to specify the list of public parameters added for each transformation type and announce extensions to a list of arguments defined in another transtype. #1757

Enhancements

DITA Open Toolkit Release 2.2 includes the following enhancements and changes to existing features:

- PDF: Table body rows now use the keep-together.within-page attribute to prevent page breaks within rows. #2118
- PDF: List item numbers are now aligned with the baseline to prevent issues when list items include icons or other inline elements that affect line spacing. #2117

- PDF: Step section bodies within task topics now honor the `$side-col-width` value from basic-settings.xsl, which defines a uniform indent relative to the page margin and aligns with other body text. (Earlier toolkit versions used a hard-coded 9-mm start-indent setting.) #2116
- PDF: The index generation process has been rewritten using XSL keys for better performance. The optimizations yield significantly faster PDF builds when using Apache FOP. #2098, #2086
- HTML5 and XHTML table border processing has been optimized to match the expected output based on the table width, column separation, row separation and frame settings in the source files, permit easier integration of CSS frameworks, and output valid documents. #2097
- The task headings (About this task, Procedure, etc.) and flags for Optional and Required steps in the PDF transformation have been synchronized with those available in the common string files, XHTML and ODT transformations. Source files that make use of these options should now yield more more consistent results when generating output in multiple formats. #2088
- PDF: The index groups for Numerics and Special Characters have been aligned for greater consistency across languages. #2080, #2074
- PDF: The “pointing finger” image hand.gif is no longer used to highlight `<note>` elements, as it may be considered offensive in some cultures. The image file is still available for backwards compatibility with any customization that references it, and the "Note Image Path" variables are still present to permit the use of custom image files, but they are now empty by default. Text-only note labels appear instead, and the default indentation is reduced by the width of the empty note image column. #2076, #1577
  Note: The warning.gif file is still used for Attention, Caution, Danger and Warning admonitions.
- The outer.control parameter description was corrected to clarify how the DITA-OT handles content files that are not located in or below the directory containing the master DITA map. #1707 #2066
- Formatter-specific code for XSL-FO rendering engines has been removed from the PDF plugin and split into separate plugins for Apache FOP, Antenna House Formatter and RenderX XEP. #2058
- The classpath order is now retained when generating the env.sh and env.bat environment files to ensure predictable results when a plugin that uses Java libraries presupposes a certain classpath order. #2053
- The PDF2 flagging step that converted stage1.xml to stage1a.xml in the PDF process has been refactored to take advantage of the flagging information added during the common preprocessing stage. #2049, #2047
- The dita.bat Windows batch file for the dita command now sets the *DITA_HOME* variable to point to the correct location of the DITA-OT. #2046
- PDF: A new axf.opt parameter has been added to specify the user configuration file for Antenna House Formatter. #2041
- Processing mode coverage has been improved to treat error messages as fatal errors, so the DITA-OT will now stop processing if any source files are missing when the processing-mode parameter is set to strict. #1986
- Table columns for which no width is defined in a `<colspec>` element are no longer set to `1*` per the CALS Table Model. Instead, empty `@colwidth` attributes are generated to allow formatter-specific auto-layout. The FO processor can then set the width of the columns based on the column content. #1970

Bugs

DITA Open Toolkit Release 2.2 provides fixes for the following bugs:

- Inconsistency in naming of flow name, region definition #2128
- Third relcolspec title on a reltable no longer taken into account for publishing #2119
- Topic in temp folder is not wellformed #2109

- gradle build fails with ‘Could not load FFI Provider ..’ on Windows #2108
- Behavior of the force-unique flag #2105
- Site builds fail after `html5` changes #2101
- Generated HTML table is invalid according to HTML5 specs #2095
- Fix table and figure list to include number, title #2093
- Remove obsolete info from codepage list #2091
- Add axf.jar into log-processor taskdef classpath #2090
- Add PFD2 index groups for a-breve, a-circ in Romanian #2081
- Ambiguous message for example with two titles #2078
- Table not localized in French translation org.dita.pdf2 - fr.xml #2061
- French translation of Table of contents is incorrect #2060
- Fix ODT title generation #2059, #2034
- Catch null FileInfo object being referenced in move-meta. #2051
- Flagging preprocess grabs too much with check for defaults #2050, #2048
- Error message not properly formatted #2027
- Can no longer publish to XHTML image with data protocol #2012
- Ensuring @chunk inside topicgroups functions as expected. #2009, #1991
- Copy-to usage with URI support does not properly work #2006
- Cannot publish remote HTTP DITA Map to XHTML #2003
- Branch filtering does not seem to work with entire DITA Maps #1992
- Add proper mappings for topicrefs with copy-to attributes in JavaHelp #1989
- Use the fragment part of KeyDef @href attribute when building @conref #1974
- Remove unwanted $PATH2PROJ remnant from $entry-file definition (glossary entry file resolution fails from term and abbreviated-form DOTX058W) #1967, #1966
- DITA-OT 2.0 - Build Error (Windows) - Illegal character - keyref target #1823
- abbreviated-form and term keyref links are not resolved when chunk="to-content" #1816
- Two levels of map ref causes good key ref to fail #1605

Contributors

DITA Open Toolkit Release 2.2 includes contributions by the following people:

1. Jarno Elovirta
2. Robert D. Anderson
3. Roger Sheen
4. Eero Helenius
5. Radu Coravu
6. Tom Glastonbury
7. Kendall Shaw
8. Eliot Kimber
9. Chris Nitchie
10. Stefan Eike

For the complete list of changes since the previous release, see the changelog on GitHub.

Getting Started with the DITA Open Toolkit

Getting Started provides 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 toolkit and building output.

Installing the distribution package

The DITA-OT distribution package can be installed on Linux, Mac OS X, and Windows. It contains everything that you need to run the toolkit except for Java.

- 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.

1. Download the dita-ot-2.2.3.zip package from the project website at www.dita-ot.org.
2. Extract the contents of the package to the directory where you want to install the DITA-OT.
3. **Optional:** Add the absolute path for the bin directory to the *PATH* system variable.

   This defines the necessary environment variable to run the dita command from the command line.

   Tip: This step is recommended, as it allows the the dita command to be run from any location on the file system and makes it easier to transform DITA content from any folder.

Building output using the dita command

You can generate output using the DITA Open Toolkit dita command-line tool. Build parameters can be specified on the command line or with .properties files.

The DITA-OT client is a command-line tool with no graphical user interface.

1. Open a terminal window by typing the following in the search bar:
   - On OS X and Linux, type `Terminal`.
   - On Windows, type `Command Prompt`.
2. At the command-line prompt, enter the following command:

   ```
    -input input-file -format format
   ```

   where:
   -

Run from , the following command generates HTML5 output for the sequence.ditamap file:

```
 -input sequence.ditamap -format html5
```

Most builds require you to specify more options than are described in this topic.

Related tasks
More information about building output with the dita command

DITA Open Toolkit User Guide

The *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 information on the supported DITA versions and the versions of components that have been tested for use with the toolkit.

DITA Open Toolkit Overview

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 that transform DITA content (maps and topics) into deliverable formats, including PDF, HTML, HTML Help, Eclipse Help, and JavaHelp.

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

Related information
Apache License, version 2.0

DITA specification support

DITA Open Toolkit 2.2 supports all standard versions of DITA, including 1.0, 1.1, and 1.2, with preview support for the DITA 1.3 specification.

DITA 1.2 support

DITA Open Toolkit 2.2 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.

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

DITA 1.3 support

DITA Open Toolkit 2.2 provides processing support for the OASIS DITA 1.3 specification. Initial preview support for this specification was added in version 2.0 of the toolkit; version 2.2 extends this foundation to support key scopes and branch filtering along with additional DITA 1.3 features.

Because DITA 1.3 is fully backwards compatible with previous DITA DTDs and schemas, DITA-OT 2.2 provides the 1.3 materials as the default DTDs for processing. The XML Catalog resolution maps any references for unversioned DITA doctypes to the 1.3 DTDs. All processing ordinarily dependent on the 1.0, 1.1, or 1.2 definitions continues to work as usual, and any documents that make use of the newer DITA 1.3 elements or attributes will be supported with specific new processing.

Initial Preview Support for DITA 1.3 in DITA-OT 2.0

The following DITA 1.3 features were implemented in version 2.0 of the toolkit. Issue numbers correspond to the tracking number in the GitHub issues tracker.

- Support DITA 1.3 link syntax (milestone 2) #1649
- Support DITA 1.3 cascade attribute (milestone 2) #1636
- Implement DITA 1.3 profiling (milestone 2) #1635
- Add new DITA 1.3 highlighting elements (milestone 4) #1651
- Add DITA 1.3 markup and xml domain support (milestone 4) #1652

- Add DITA 1.3 `<div>` element (milestone 4) #1654

Additional DITA 1.3 support in DITA-OT 2.2

The following DITA 1.3 features were implemented in version 2.2 of the toolkit.

Important: The DITA 1.3 grammars are now used as the default DTDs for processing #2094

- Initial implementation of DITA 1.3 branch filtering #1969, #1637
  The implementation is a separate module that is run before keyref processing. The process
  - Splits branches so that each branch contains a single ditavalref
  - Generates `@copy-to` attributes for each branch-generated `<topicref>`
  - Filters the map based on branch filters
  - Rewrites duplicate generated copy-to targets with a numbered `-#` suffix
  - Copies and filters generated copy-to targets
  - Filters topics that were not branch-generated
- Initial support for DITA 1.3 key scopes, including multiple scope names in a single `@keyscope` attribute #1979, #1648, #2004
- The `@keyref` attribute is now supported on `<object>` elements #1783
- Processing order has been revised to process any same topic fragments used in conrefs before the conref phase, to enable content references to elements in the same topic using a reference such as `<p conref="#./ID"/>` as reported in #1649. #1968

Note: For the latest status information on DITA 1.3-related features, see the DITA 1.3 label in the GitHub issues tracker.
Related information
DITA Version 1.3 Part 3: All-Inclusive Edition (HTML)
DITA Version 1.3 Part 3: All-Inclusive Edition (PDF)
DITA Version 1.3 (Distribution ZIP of the DITA source)
DITA Adoption Technical Committee

Tested platforms and tools

The DITA Open Toolkit has been tested against certain versions of Ant, ICU4J, JDK, operating systems, XML parsers, and XSLT processors.

Application	**Tested version**
Ant	• Ant 1.7.1 • Ant 1.8.2—1.8.4, 1.9.2-1.9.4
ICU for Java	• ICU4J 3.4.4 • ICU4J 49.1 • ICU4J 54.1

Application	Tested version
JDK	• IBM 1.7 • Oracle 1.7
Operating system	• Mac OS X 10.6—10.9 • SLES 10 • Windows XP • Windows 7
XML parser	• Xerces 2.9.0 • Xerces 2.11.0
XSLT processor	• Saxon 9 • Saxon-B 9.1 • Saxon-HE/PE/EE 9.5-9.6

Installing the DITA Open Toolkit

You can install the DITA Open Toolkit on Linux, Mac OS X, and Windows.

Prerequisite software

The prerequisite software that the DITA-OT requires depends on 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 http://www.oracle.com/technetwork/java/javase/downloads/.
Note: This is the *only* prerequisite software that you need to install. The remaining required software is included in the distribution packages.

Ant, version 1.7.1 or later
: Provides the standard setup and sequencing of processing steps. You can download Ant from http://ant.apache.org/.

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 http://www.icu-project.org/download/.

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

XSL-FO processor
: Required for generating PDF output. Apache™ FOP (Formatting Objects Processor) is included in the distribution packages. You can download FOP from http://xmlgraphics.apache.org/fop/download.html. You also can use commercial FO processors such as Antenna House Formatter or RenderX XEP.

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

Installing the distribution package

The DITA-OT distribution package can be installed on Linux, Mac OS X, and Windows. It contains everything that you need to run the toolkit except for Java.

- 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.

1. Download the dita-ot-2.2.3.zip package from the project website at www.dita-ot.org.
2. Extract the contents of the package to the directory where you want to install the DITA-OT.
3. **Optional:** Add the absolute path for the bin directory to the *PATH* system variable.

   This defines the necessary environment variable to run the dita command from the command line.

   Tip: This step is recommended, as it allows the the dita command to be run from any location on the file system and makes it easier to transform DITA content from any folder.

Publishing DITA content

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

Building output using the dita command

You can generate output using the DITA Open Toolkit dita command-line tool. Build parameters can be specified on the command line or with .properties files.

At the command-line prompt, enter the following command:

```
 -input input-file -format format options
```

where:

-
-

For example, from , run:

```
dita -input sequence.ditamap -format html5 \
     -output output/sequence \
     -Dargs.input.dir= \
     -propertyfile properties/sequence-html5.properties
```

This builds sequence.ditamap to HTML5 output in output/sequence using the additional parameters specified in the properties/sequence-html5.properties file:

```
args.gen.task.lbl = NO
args.cssroot = ${args.input.dir}/css/
args.css = style.css
args.copycss = yes
args.csspath = branding
nav-toc = full
args.xhtml.toc = toc
```

Usually, you will want to specify a set of reusable build parameters in a .properties file.

Related reference
Arguments and options for the dita command
DITA-OT parameters
Internal Ant properties

Setting build parameters with .properties files

Usually, DITA builds require setting a number of parameters that do not change frequently. You can reference a set of build parameters defined in a .properties file when building output with the dita command. If needed, you can override any parameter by specifying it explicitly as an argument to the dita command.

About .properties files

A .properties file is a text file that enumerates one or more name-value pairs, one per line, in the format `name = value`. The .properties filename extension is customarily used, but is not required.

- Lines beginning with the `#` character are comments.
- Properties specified as arguments of the dita command override those set in .properties files.
  Restriction: For this reason, args.input and transtype can't be set in the .properties file.
- If you specify the same property more than once, the last instance is used.
- Properties not used by the selected transformation type are ignored.
- Properties can reference other property values defined elsewhere in the .properties file or passed by the dita command. Use the Ant `${property.name}` syntax.
- You can set properties not only for the default DITA-OT transformation types, but also for custom plugins.

1. Create your .properties file.

Note:
For example:

```
args.gen.task.lbl = NO
args.cssroot = ${args.input.dir}/css/
args.css = style.css
args.copycss = yes
args.csspath = branding
nav-toc = full
args.xhtml.toc = toc
```

2. Reference your .properties file with the dita command when building your output.

   ```
   dita -i my.ditamap -f html5 -propertyfile my.properties
   ```

3. If needed, pass additional arguments to the dita command to override specific build parameters.

   For example, to build output once with `<draft>` and `<required-cleanup>` content:

   ```
   dita -i my.ditamap -f html5 -propertyfile my.properties -Dargs.draft=yes
   ```

   Note:

Migrating Ant builds to use the dita command

Although Ant builds are still supported in the DITA Open Toolkit, you might want to switch to use the new dita command.

Building output with the dita command is often easier than using Ant. In particular, you can use .properties files to specify sets of DITA-OT parameters for each build.

You can include the dita command in shell scripts to perform multiple builds.

Note:

1. In your Ant build file, identify the properties set in each build target.
   Note: Some build parameters might be specified as properties of the project as a whole. You can refer to a build log to see a list of all properties that were set for the build.
2. Create a .properties file for each build and specify the needed build parameters, one per line, in the format `name = value`.
3. Use the dita command to perform each build, referencing your .properties file with the -propertyfile option.

Example: Ant build

Sample build file: *dita-ot-dir*/docsrc/samples/ant_sample/build-chm-pdf.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<project name="build-chm-pdf" default="all" basedir="">
```

```
  <property name="dita.dir" location="${basedir}/../../.."/>

  <target name="all" description="build CHM and PDF" depends="chm,pdf"/>

  <target name="chm" description="build CHM">
    <ant antfile="${dita.dir}/build.xml">
      <property name="args.input" location="../sequence.ditamap"/>
      <property name="transtype" value="htmlhelp"/>
      <property name="output.dir" location="../out/chm"/>
      <property name="args.gen.task.lbl" value="YES"/>
      <property name="args.breadcrumbs" value="yes"/>
    </ant>
  </target>

  <target name="pdf" description="build PDF">
    <ant antfile="${dita.dir}/build.xml">
      <property name="args.input" location="../taskbook.ditamap"/>
      <property name="transtype" value="pdf"/>
      <property name="output.dir" location="../out/pdf"/>
      <property name="args.gen.task.lbl" value="YES"/>
      <property name="args.rellinks" value="nofamily"/>
    </ant>
  </target>

</project>
```

Example: .properties files with dita command

The following .properties files and dita commands are equivalent to the example Ant build.

Sample .properties file: /properties/chm.properties

```
output.dir = out/chm
args.gen.task.lbl = YES
args.breadcrumbs = yes
```

Sample .properties file: /properties/pdf.properties

```
output.dir = out/pdf
args.gen.task.lbl = YES
args.rellinks = nofamily
```

Run from :

```
 -input sequence.ditamap -format htmlhelp -propertyfile properties/chm.properties
 -input taskbook.ditamap -format pdf -propertyfile properties/pdf.properties
```

Example: Call the dita command from an Ant build

In some cases, you might still want to use an Ant build to implement some pre- or post-processing steps, but also want the convenience of using the dita command with .properties files to define the parameters for each build. This can be accomplished with Ant's `<exec>` task.

This example uses a `<dita-cmd>` Ant macro defined in the /ant_sample/dita-cmd.xml file.

Sample build file: /ant_sample/build-chm-pdf-hybrid.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<project name="build-chm-pdf-hybrid" default="all" basedir="">

  <description>An Ant build that calls the dita command</description>

  <include file="dita-cmd.xml"/><!-- defines the <dita-cmd> macro -->

  <target name="all" depends="pre,main,post"/>

  <target name="pre">
    <description>Preprocessing steps</description>
  </target>

  <target name="main">
    <description>Build the CHM and PDF with the dita command</description>
    <dita-cmd input="../sequence.ditamap" format="htmlhelp" propertyfile="../properties/chm.properties"/>
    <dita-cmd input="../taskbook.ditamap" format="pdf" propertyfile="../properties/pdf.properties"/>
  </target>

  <target name="post">
    <description>Postprocessing steps</description>
  </target>

</project>
```

Publishing DITA content from Ant

You can use Ant to invoke the DITA Open Toolkit and generate output. You can use the complete set of parameters that are supported by the toolkit.

Related tasks
Migrating Ant builds to use the dita command

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 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.

Related tasks
Migrating Ant builds to use the dita command
Building output using Ant
Creating an Ant build script
Related reference
DITA-OT parameters
Apache Ant documentation

Building output using Ant

You can build output by using an Ant build script to provide the DITA-OT parameters.

1. Open a command prompt or terminal session, and then change to the directory where the DITA Open Toolkit is installed.
2. Issue the following command:

Option	Description
Linux or Mac OS X	bin/ant -f *build-script* *target*
Windows	bin\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.

Related concepts
Ant
Related tasks
Creating an Ant build script
Migrating Ant builds to use the dita command
Related reference
DITA-OT 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.

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="%project-name%" default="%default-target%" basedir=".">

  <property name="dita.dir" location="%path-to-DITA-OT%"/>

  <target name="%target-name%">
    <ant antfile="${dita.dir}/build.xml">
      <property name="args.input" value="%DITA-input%"/>
      <property name="transtype" value="html5"/>
    </ant>
  </target>

</project>
```

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

2. Specify project information:
   a. **Optional:** Set the value of the `@name` attribute to the name of your project.
   b. 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:
   a. Set the value of the `@name` attribute.
   b. Specify the value for the args.input property.
   c. Specify the value of the transtype property.
5. Save the build script.

The following Ant build script generates CHM and PDF output for the sample DITA maps.

```
<?xml version="1.0" encoding="UTF-8" ?>

<project name="build-chm-pdf" default="all" basedir="">

  <property name="dita.dir" location="${basedir}/../../.."/>

  <target name="all" description="build CHM and PDF" depends="chm,pdf"/>

  <target name="chm" description="build CHM">
    <ant antfile="${dita.dir}/build.xml">
      <property name="args.input" location="../sequence.ditamap"/>
      <property name="transtype" value="htmlhelp"/>
      <property name="output.dir" location="../out/chm"/>
      <property name="args.gen.task.lbl" value="YES"/>
      <property name="args.breadcrumbs" value="yes"/>
    </ant>
  </target>

  <target name="pdf" description="build PDF">
    <ant antfile="${dita.dir}/build.xml">
```

```
      <property name="args.input" location="../taskbook.ditamap"/>
      <property name="transtype" value="pdf"/>
      <property name="output.dir" location="../out/pdf"/>
      <property name="args.gen.task.lbl" value="YES"/>
      <property name="args.rellinks" value="nofamily"/>
    </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.

Another resource for learning about Ant scripts are the files in the *dita-ot-dir*/docsrc/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.
Related concepts
Ant
Related tasks
Building output using Ant
Migrating Ant builds to use the dita command
Related reference
DITA-OT parameters
Apache Ant documentation

DITA-OT transformations (output formats)

The DITA Open Toolkit ships with several core transformations that generate different output formats from DITA content. Each transformation represents an implementation of the 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.

Normalized DITA files have been through the DITA Open Toolkit pre-processing operation. In comparison to the source DITA files, the normalized DITA files 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.
Related reference
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
mapname.xml	Table of contents
index.xml	Index file
plugin.properties	
META-INF/MANIFEST.MF	

Related reference
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 sheet that overrides the default styles, or generate a `<nav>` element with a navigation TOC in topic pages.

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

Related reference
HTML transformation
HTML5 transformation

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

Related reference
HTML Help transformation

DITA to Java Help

The javahelp transformation will generate Java Help output, along with the control files needed to compile the project.

The Java Help output produces HTML files rather than XHTML files. In addition to the HTML output and CSS files, this transformation type will return the following files:

- Table of Contents (toc.xml)
- Sorted index (mapname_index.xml)
- Other Java Help project files (map.jhm and mapname_helpset.hs)
- If the Java Help compiler is located on the system, a compiled Java Help project will be returned.

Related reference
JavaHelp transformation

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).

Related reference
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.

Related reference
PDF transformation

Generating revision bars

If you use Antenna House Formatter or RenderX XEP, you can generate revision bars in your PDF output by using the `@changebar` attribute of the DITAVAL `<revprop>` element.

Note: FOP 1.1 does not support the XSL `fo:change-bar` formatting object.
The DITA specification for `@changebar` simply says:

@changebar
When flag has been set, specify a changebar color, style, or character, according to the changebar support of the target output format. If flag has not been set, this attribute is ignored.

The syntax for `@changebar` is a sequence of name and value pairs that are delimited by semicolons, for example:

```
<revprop action="flag" val="rev01"
 changebar="color:black;style:solid;width:0.5pt"
/>
```

To produce a revision bar, you must specify a value for style and should specify a value for width so you get a visible rule.

Each name and value pair corresponds to an attribute of the XSL-FO fo:change-bar-begin element. The following attributes and values are available:

style
The style to use for the line, as for other XSL-FO rules (@change-bar-style). The value solid produces a solid rule; the default value is none.

color
Any color value recognized by XSL-FO, including the usual color names or a hex color value. The default value is black.

offset
The space to offset the revision bar from the edge of the text column. You can use points (pt) or millimeters (mm).

placement
The side of the text column on which to place the revision bar. The allowed values are start (left side for left-to-right languages) and end (right side for left-to-right languages). The default value is start.

width

The width of the rule as a measurement value. Typical values are 1pt and 0.5pt, which renders a hairline rule.

XSL-FO 1.1 does not provide for revision bars that are not rules, so there is no way to get text revision indicators instead of rules, for example, using a number in place of a rule. Antenna House Formatter provides a proprietary extension to enable this, but the DITA-OT PDF transformation does not take advantage of it.

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.
- Certain output styles supported by other DITA-OT transformations are not supported.

DITA to TocJS

The tocjs transformation generates XHTML 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 XHTML output and uses a default frameset.

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 `<table>` 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 sheet 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.

Related reference
HTML transformation

Globalizing DITA content

The DITA standard supports content that is written in or translated to any language. In general, the DITA Open Toolkit 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 provide globalization support.

Related reference
Localization overview in the OASIS DITA standard
Modifying or adding generated text

Globalization support offered by the DITA-OT

The DITA Open Toolkit 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 HTML 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.

Supported languages: HTML-based transformations

The DITA Open Toolkit supports over 50 languages and language variants for the HTML-based transformations such as HTML5, XHTML, Eclipse Help, HTML Help, and TocJS.

Table 1. Supported languages: HTML-based transformations

Language	Language code
Arabic	ar or ar-EG
Belarusian	be or be-BY
Brazilian Portuguese	pt-BR

Language	Language code
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 or 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

Language	Language code
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

Language	Language code
Turkish	tr or tr-TR
Ukrainian	uk or uk-UA
Urdu	ur or ur-PK

Related reference
How to add support for new languages in HTML

Supported languages: PDF transformations

The DITA Open Toolkit 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.

Table 2. Supported languages: PDF transformation

Language	Language code
Arabic	ar or ar-EG
Catalan	ca-ES
Chinese (simplified)	zh-CN or zh-Hans
Croatian	hr or hr-HR
Czech	cs or cs-CZ
Danish	da or da-DK
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
Hungarian	hu or hu-HU
Icelandic	is or is-IS
Italian	it or it-IT

Language	Language code
Japanese	ja or ja-JP
Latvian	lv or lv-LV
Norwegian	no or no-NO
Polish	pl or pl-PL
Portuguese	pt or pt-PT
Portuguese (Brazil)	pt-PR
Romanian	ro or ro-RO
Russian	ru or ru-RU
Slovak	sk or sk-SK
Slovenian	sl or sl-SI
Spanish	es or es-ES
Swedish	sv or sv-SE
Turkish	tr or tr-TR

Error messages and troubleshooting

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

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 Increasing Java memory allocation.
2. Reduce memory consumption by setting the generate-debug-attributes 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 Increasing Java memory allocation.

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 -l 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 -l 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.)

Accessing help for the dita command

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

1.
2. Issue the following command:

Option	Description
Linux or Mac OS X	bin/dita -help
Windows	bin\dita -help

Note:

A brief description of the supported parameters appears in the command-line window.

Checking the DITA-OT version

You can determine the version of the DITA Open Toolkit from a command prompt.

1.
2. Issue the following command:

Option	Description
Linux or Mac OS X	bin/dita -version
Windows	bin\dita -version

Note:

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.

From the command prompt, add the following parameters:

Application	Parameters
dita command	-d or -debug
Ant	`-v -Dargs.debug=yes`

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 environment variable.

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

Platform	Command
Linux or Mac OS X	`export ANT_OPTS=$ANT_OPTS -Xmx1024M`
Windows	`set 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.

Reducing processing time

Several configuration changes can significantly reduce DITA-OT processing time.

Disable debug attribute generation

The generate-debug-attributes parameter determines whether debugging attributes are generated in the temporary files. By changing the value to `false`, DITA-OT will no longer generate the `@xtrf` and `@xtrc` debug attributes. This will make it more difficult to track down the source file location from which a given issue may have originated, but it will reduce the size of the temporary files. As a result, XML parsing will take less time and overall processing time will be reduced.

Use a fast disk for the temporary directory

DITA-OT keeps topic and map files as separate files and processes each file multiple times during preprocessing. Thus reading from disk, parsing XML, serializing XML, and writing to disk makes processing quite IO intensive. Use either an SSD or a RAM disk for temporary files, and never use a temporary directory that is not located on the same machine as where the processing takes place.

Reuse the JVM instance

For all but extremely large source sets, the JVM will not have enough time to warm-up. By reusing the same JVM instance, the first few DITA-OT conversions will be “normal”, but when the JIT starts to kick in, the performance increase may be 2-10 fold. This is especially noticeable with smaller source sets, as much of the DITA-OT processing is I/O intensive.

Use the latest Java version

DITA-OT 2.0 requires Java 7, but using the latest version Java 8 will further reduce processing time.

Collected links
SSD
RAM disk

DITA Open Toolkit Parameter Reference

The *Parameter Reference* is designed to help users to locate information easily and quickly. It includes documentation for the DITA-OT parameters and configuration properties.

DITA-OT parameters

Certain parameters apply to all DITA-OT transformations. Other parameters are common to the HTML-based transformations. Finally, some parameters apply only to specific transformation types. These parameters can be passed as options to the dita command using the -Dparameter-name=value syntax or included in build scripts as Ant properties.

Related tasks
Setting build parameters with .properties files

Other parameters

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

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.

Arguments and options for the dita command

The dita command takes mandatory arguments to process DITA content, manage plug-ins, or print information about the command. Options can be used to modify the command behavior or specify additional configuration parameters.

Usage

dita -i *file* ***-f*** *name [options]*

dita -install *[{ filename | URL }]*

dita -uninstall *id*

dita -help

```
dita -version
```

Arguments

-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 current directory if args.input.dir is not defined.

-f, -format *name*

-install *filename*
-install *URL*
: Install a single plug-in from a local ZIP file or from a URL.

-install

-uninstall *id*
: Uninstall a plug-in with the specified ID.

-h, -help
: Print command usage help.

-version
: Print version information and exit.

Options

-o, -output *dir*
: Specifies the path of the output directory; the path can be absolute or relative to the current directory. By default, the output is written to the out subdirectory of the current directory.

-filter *file*

-t, -temp *dir*

-v, -verbose
: Verbose logging.

-d, -debug
: Debug logging.

-l, -logfile *file*
: Write logging messages to a file.

-D*parameter*=*value*
: Specify a value for a DITA-OT or Ant build parameter.

Parameters not implemented by the specified transformation type or referenced in a .properties file are ignored.

Note:

-propertyfile *file*

Use build parameters defined in the referenced .properties file.

Build parameters specified on the command line override those set in the .properties file.

Related tasks
Building output using the dita command
Setting build parameters with .properties files
Related reference
DITA-OT parameters
Internal Ant properties

Configuration properties

The DITA-OT uses .properties files that store configuration settings for the toolkit 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.

configuration.properties file

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

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

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, Tags for Identifying Languages.

org.dita.pdf2.i18n.enabled

(PDF transformation only) Enables I18N font processing. The following values are allowed:
- true (default) — Enables I18N processing
- false — Disables I18N processing

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.

Internal Ant properties

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

`include.rellinks`

A space-separated list of link roles to be output; the `#default` value token represents links without an explicit role (those for which no `@role` attribute is defined). Defined by `args.rellinks`, but may be overridden directly. Valid roles include:

- parent
- child
- sibling
- friend
- next
- previous
- cousin
- ancestor
- descendant
- sample
- external
- other

DITA Open Toolkit Developer Reference

The *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, extending the DITA-OT, and creating DITA-OT plug-ins.

Architecture of the DITA Open Toolkit

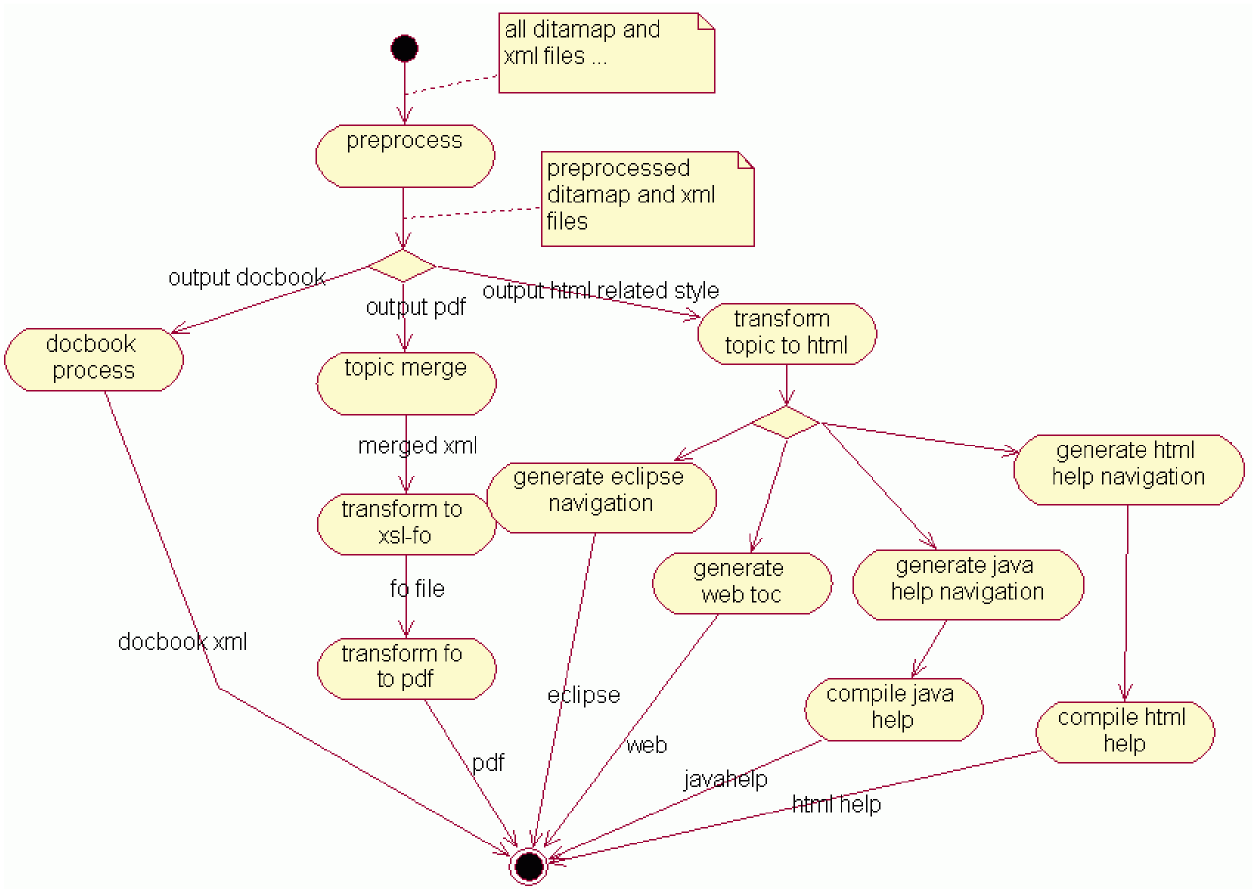
The DITA Open Toolkit is an open-source implementation of the OASIS specification for the Darwin Information Typing Architecture. The toolkit uses Ant, XSLT, and Java to transform DITA content (maps and topics) into different deliverable formats.

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.



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, plug-ins 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.

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 on the reference target will override the product setting on the referencing note. Assume that the referenced `<note>` element in documentA is defined as follows:

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

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.

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.

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 property	Usage
canditopicsfile	`canditopics.list`	canditopicslist	
codereffile	`coderef.list`	codereflist	topics with coderef
conreffile	`conref.list`	conreflist	Documents that contains conref attribute that need to be resolved in preprocess.
conrefpushfile	`conrefpush.list`	conrefpushlist	
conreftargetsfile	`conreftargets.list`	conreftargetslist	
copytosourcefile	`copytosource.list`	copytosourcelist	
copytotarget2sourcemapfile	`copytotarget2sourcemap.list`	copytotarget2sourcemaplist	
flagimagefile	`flagimage.list`	flagimagelist	
fullditamapandtopicfile	`fullditamapandtopic.list`	fullditamapandtopiclist	All of the ditamap and topic files that are referenced during the transformation. These may be referenced by

List file property	List file	List property	Usage
			href or conref attributes.
fullditamapfile	`fullditamap.list`	fullditamaplist	All of the ditamap files in dita.list
fullditatopicfile	`fullditatopic.list`	fullditatopiclist	All of the topic files in dita.list
hrefditatopicfile	`hrefditatopic.list`	hrefditatopiclist	All of the topic files that are referenced with an href attribute
hreftargetsfile	`hreftargets.list`	hreftargetslist	link targets
htmlfile	`html.list`	htmllist	resource files
imagefile	`image.list`	imagelist	Images files that are referenced in the content
outditafilesfile	`outditafiles.list`	outditafileslist	
relflagimagefile	`relflagimage.list`	relflagimagelist	
resourceonlyfile	`resourceonly.list`	resourceonlylist	
skipchunkfile	`skipchunk.list`	skipchunklist	
subjectschemefile	`subjectscheme.list`	subjectschemelist	
subtargetsfile	`subtargets.list`	subtargetslist	
tempdirToinputmapdir.relative.value			
uplevels			
user.input.dir			Absolute input directory path
user.input.file.listfile			Input file list file

List file property	List file	List property	Usage
user.input.file			Input file path, relative to input directory

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.

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>`).

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.

Resolve keyref (keyref)

The `keyref` step examines all the keys that are defined in the DITA source and resolves 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, and the key definition list file is written to the temporary directory. This step is implemented in Java.

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.

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 2. Referenced topic that contains a section and cross reference

```
<topic id="referenced_topic">
  <title>...</title>
  <body>
    <section id="sect"><title>Sample section</title>
      <p>Figure <xref href="#referenced_topic/fig"/> contains an code sample that demonstrates ... .</p>
      <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 element may look like this in the intermediate document.`<section>`
Figure 3. Resolved conrefed <section> element after the conref step

```
<section id="sect"><title>Sample section</title>
    <p>Figure <xref href="#new_topic/d1e25"/> contains an code sample that demonstrates ... .</p>
    <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.

Resolve code references (coderef)

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

The `<coderef>` element 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.

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

Related information
Chunking definition in the DITA 1.2 specification

Move metadata (move-meta-entries) and pull content into maps (mappull)

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.

Note: As of DITA-OT 2.2, the `move-meta-entries` and `mappull` steps have been merged. The `mappull` step has been moved into `move-meta-entries`.

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 `<linktext>` 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.

Map based linking (maplink)

This step collects links based on a map and moves 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 stylesheet that evaluates the map; it places all the generated links into a single file in memory. The module then runs a Java program that pushes the generated links into the applicable topics.

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.

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 to the topic when flags are active. Any extended transformation 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/ditaval-startprop ">
```

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 `<ditaval-startprop>`.
- 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-endprop ">
```

CSS values and `<style-conflict>` 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 `<startflag>` children of the properties will be included, but `<endflag>` children will not.

Supporting flags in overrides or custom transformation types

For most transformation 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 `<p>` 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 `<ol>`, because fallthrough processing would place images in between `<ol>` and `<li>`. 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 `<ol>`.

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" backcolor="green"/>
  <prop action="flag" att="platform" style="overline" val="win" backcolor="blue"/>

  <!-- Define a property that includes start and end image flags -->
  <prop action="flag" att="platform" val="linux" style="overline" backcolor="blue">
    <startflag imageref="startlin.png"><alt-text>Start linux</alt-text></startflag>
    <endflag imageref="endlin.png"><alt-text>End linux</alt-text></endflag>
  </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</alt-text></startflag>
    <endflag imageref="end_rev.gif"><alt-text>eeeeeeeeeeeeeend</alt-text></endflag>
  </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.

```
<p audience="user">Simple user; includes style but no images</p>
```

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:

```
<p audience="user" class="- topic/p ">
  <ditaval-startprop class="+ topic/foreign ditaot-d/ditaval-startprop "
           outputclass="background-color:green;text-decoration:underline;">
    <prop action="flag" att="audience" style="underline" val="user" backcolor="green"/>
  </ditaval-startprop>
  Simple user; includes style but no images
  <ditaval-endprop class="+ topic/foreign ditaot-d/ditaval-endprop ">
    <prop action="flag" att="audience" style="underline" val="user" backcolor="green"/>
  </ditaval-endprop>
</p>
```

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.

```
<p audience="user" platform="win">Conflicting styles (still no images)</p>
```

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 `<style-conflict>` element itself is also copied into the generated `<ditaval-startprop>` element.

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

```
<p audience="user" platform="win" class="- topic/p ">
  <ditaval-startprop class="+ topic/foreign ditaot-d/ditaval-startprop "
           outputclass="background-color:red;">
    <style-conflict background-conflict-color="red"/>
    <prop action="flag" att="audience" style="underline" val="user" backcolor="green"/>
    <prop action="flag" att="platform" style="overline" val="win" backcolor="blue"/>
  </ditaval-startprop>
  Conflicting styles (still no images)
  <ditaval-endprop class="+ topic/foreign ditaot-d/ditaval-endprop ">
    <prop action="flag" att="platform" style="overline" val="win" backcolor="blue"/>
    <prop action="flag" att="audience" style="underline" val="user" backcolor="green"/>
  </ditaval-endprop>
</p>
```

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.

```
<ol platform="linux" rev="rev2">
  <li>Generate images for platform="linux" and rev="2"</li>
</ol>
```

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:

```
<ol platform="linux" rev="rev2" class="- topic/ol ">
  <ditaval-startprop class="+ topic/foreign ditaot-d/ditaval-startprop "
           outputclass="background-color:blue;text-decoration:underline;text-decoration:overline;">
    <prop action="flag" att="platform" val="linux" style="overline" backcolor="blue">
      <startflag imageref="startlin.png"><alt-text>Start linux</alt-text></startflag>
    </prop>
    <revprop action="flag" style="double-underline" val="rev2">
      <startflag imageref="start_rev.gif"><alt-text>ssssssssssstart</alt-text></startflag>
    </revprop>
  </ditaval-startprop>
  <li class="- topic/li ">Generate images for platform="linux" and rev="2"</li>
  <ditaval-endprop class="+ topic/foreign ditaot-d/ditaval-endprop ">
```

```
    <revprop action="flag" style="double-underline" val="rev2">
      <endflag imageref="end_rev.gif"><alt-text>eeeeeeeeeeeeeend</alt-text></endflag>
    </revprop>
    <prop action="flag" att="platform" val="linux" style="overline" backcolor="blue">
      <endflag imageref="endlin.png"><alt-text>End linux</alt-text></endflag>
    </prop>
  </ditaval-endprop>
</ol>
```

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.

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

- If the args.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. The `dita.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 needed 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 uses 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 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.

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.

The `transform.topic2pdf` target 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.

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.

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.

Installing plug-ins

Use the dita command to install a plug-in.

At the command-line prompt, enter the following command:

```
 -install plug-in-zip
```

where:
-
-

Note:
Note:
Related reference
Arguments and options for the dita command

Removing plug-ins

Use the dita command to uninstall a plug-in.

At the command-line prompt, enter the following command:

```
 -uninstall plug-in-id
```

where:

-
-

Note:
Related reference
Arguments and options for the dita command

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.

1. Change to the docsrc directory.
2. Run the following command:

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

   The *target* parameter is optional and specifies a transformation type. It takes the following values:
   - html
   - htmlhelp
   - pdf

   If you do not specify an Ant target, HTML5 and PDF output is generated.

Creating plug-ins

You can use DITA-OT plug-ins to extend the DITA Open Toolkit.

Overview of plug-ins

Plug-ins enable users to extend the functionality of the DITA-OT. This might entail adding support for specialized document types, integrating processing overrides, or defining new output transformations.

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

Benefits of plug-ins

Plug-ins permit users to extend the toolkit in a way that is consistent, easy-to-share, and possible to preserve through toolkit upgrades.

The DITA-OT plug-in mechanism provides the following benefits:

- Plug-ins can easily be shared with other users, teams, or companies. Typically, all users need to do is to unzip and run a single integration step. With many builds, even that integration step is automatic.
- Plug-ins permit 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 of the DITA-OT simply by reinstalling or copying the directory from one installation to another. There is no need to re-integrate code based on updates to the core processing of the DITA-OT.
- Plug-ins can build upon each other. If you like a plug-in, simply install that plug-in, and then create your own plug-in that builds on top of it. The two plug-ins can then be distributed to your team as a unit, or you can share your own extensions with the original provider.

Plug-in descriptor file

The plug-in descriptor file (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 then integrates those changes into the core DITA-OT code.

The root element of the plugin.xml file is `<plugin>`. The `<plugin>` element must specify an `@id` attribute. The `@id` attribute is used to identify the plug-in; it also is used to identify whether prerequisite plug-ins are available. The value of the `@id` attribute must take the following form:

- One or more tokens, separated by periods
- A token can be composed of the following characters:
  - Numerals (0-9)
  - Lower-case letters (a-z)
  - Upper-case letters (A-Z)
  - Underscores (_)
  - Hyphens (-)

The `<plugin>` element can contain the following child elements:

`<feature>`

An optional element that defines the extension to the DITA-OT.

The following attributes are supported:

Attribute	Description	Required?
extension	Name of the DITA-OT extension point	Yes
value	Comma separated string value of the extension	Either the `@value` or `@file` attribute must be specified
file	File name, relative to the plugin.xml file	Either the `@value` or `@file` attribute must be specified
type	Type of the `@value` attribute	No

`<extension-point>`

An optional element that defines a new extension point that can be used by other DITA-OT plug-ins.

The following attributes are supported:

Attribute	Description	Required?
id	Extension point identifier	Yes
name	Extension point description	No

`<require>`

An optional element that defines plug-in dependencies.

The following attributes are supported:

Attribute	Description	Required?
plugin	A vertical-bar separated list of plug-ins that are required	Yes
importance	Identifies whether plug-in is required or optional	No

`<template>`

An optional element that defines files that should be treated as templates.

The following attributes are supported:

Attribute	Description	Required?
file	Name and path to the template file, relative to the plugin.xml file	Yes

`<meta>`

An optional element that defines metadata.

The following attributes are supported:

Attribute	Description	Required?
type	Metadata name	Yes
value	Metadata value	Yes

`<transtype>`

An optional element that defines a new output format (transformation type).

The following attributes are supported:

Attribute	Description	Required?
name	Transformation name	Yes
desc	Transformation type description	No
abstract	Defines an abstract transformation type that cannot be used directly	No
extends	Specifies the name of the transformation type being extended	No

The `<transtype>` element may define additional parameters for the transformation type using `<param>` child elements.

The following parameter attributes are supported:

Attribute	Description	Required?
name	Parameter name	Yes
desc	Parameter description	No
type	Parameter type (enum, file, string)	Yes

Enumeration parameters (`<param>` elements of type enum) define enumeration values with `<val>` child elements.

The following attributes are supported:

Attribute	Description	Required?
default	Defines an enumeration value as the default value (default="true")	Yes

Any extension that is not recognized by the DITA-OT is ignored. Since DITA-OT version 1.5.3, you can combine multiple extension definitions within a single plug-in.xml file; in older versions, only the last extension definition was used.

Related tasks
Adding a new transformation type
Related reference
#unique_111

Example plugin.xml file

Plug-in dependencies

A DITA-OT plug-in can be dependent on other plug-ins. Prerequisite plug-ins are integrated first, which ensures that the DITA-OT handles XSLT overrides correctly.

The `<require>` element in the plugin.xml file specifies whether the plug-in has dependencies. Use `<require>` elements to specify prerequisite plug-ins, in order from most general to most specific.

If a prerequisite plug-in is missing, the DITA-OT prints a warning during integration. To suppress the warning but keep the integration order if both plug-ins are present, add `importance="optional"` to the `<require>` element.

If a 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: Plug-in with a prerequisite plug-in

The following plug-in will only be installed if the plug-in with the ID `com.example.primary` is available. If that plug-in is not available, a warning is generated and the integration operation fails.

```
<plugin id="com.example.builds-on-primary">
  <!-- ... Extensions here -->
  <require plugin="com.example.primary"/>
</plugin>
```

Example: Plug-in that has optional plug-ins

The following plug-in will only be installed if either the plug-in with the ID `pluginA` or the plug-in with the ID `pluginB` is available. If neither of those plug-ins are installed, a warning is generated but the integration operation is completed.

```
<plugin id="pluginC">
  <!-- ...extensions here -->
  <require plugin="pluginA|pluginB" importance="optional"/>
</plugin>
```

Extending an XML catalog file

You can update either the main DITA-OT XML catalog or the XML catalog that is used by the PDF plug-in. This enables the DITA-OT to support new specializations and document-type shells.

You can use the `dita.specialization.catalog.relative` and `org.dita.pdf2.catalog.relative` extension points to update the DITA-OT catalog files.

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

1. Using the OASIS catalog format, create an XML catalog file that contains only the new values that you want to add to a DITA-OT catalog file.

2. Create a plug-in.xml file that contains the following content:

   ```
   <plugin id="plugin-id">
     <feature extension="extension-point" file="file"/>
   </plugin>
   ```

   where:
   - *plugin-id* is the plug-in identifier, for example, `com.example.catalog`.
   - *extension-point* is either `dita.specialization.catalog.relative` or `org.dita.pdf2.catalog.relative`.
   - *file* is the name of the new catalog file, for example, catalog-dita.xml.
3. Save the new XML catalog file to your plug-in. Be sure that the local file references are relative to the location of the catalog and plug-in.
4. Integrate the plug-in.

The catalog entries inside of the new catalog file are added to the core DITA-OT catalog file.

Example

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

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

Related reference
General extension points

Adding a new target to the Ant build process

Use the Ant conductor extension point (`dita.conductor.target.relative`) to make new targets available to the Ant processing pipeline. This can be done as part of creating a new transformation, extending pre-processing, or simply to make new Ant targets available to other plug-ins.

1. Create an Ant project file that contains the new target.
2. In the same directory as the Ant project file, create a wrapper file that imports the Ant project file:

   ```
   <dummy>
     <import file="Ant-file"/>
   </dummy>
   ```

   where *Ant-file* is the project file that contains the new target.
3. Create the plugin.xml file:

   ```
   <plugin id="plugin-id">
     <feature extension="dita.conductor.target.relative" file="wrapper-file"/>
   </plugin>
   ```

where:
   - *plugin-id* is the plug-in identifier, for example, `com.example.ant`.
   - *wrapper-file* is the wrapper file that imports the Ant project file.
4. Integrate the plug-in.

The imports from wrapper file are copied into the build.xml file, using the correct path. This makes the new Ant targets available to other processes.

Related reference
General extension points

Adding an Ant target to the pre-processing pipeline

You can add an Ant target to the pre-processing pipeline. This enables you to insert additional processing before or after the pre-processing chain or a specific step in the pre-processing operation.

You can use the `depend.preprocess.pre` and `depend.preprocess.post` extension points to run a target before or after the entire pre-processing operation. In addition, there are extension points that enable you ro run an Ant target before specific pre-processing steps.

1. Define and integrate the new Ant target.
2. Create the following plugin.xml file:

   ```
   <plugin id="plugin-id">
     <feature extension="extension-point" value="Ant-target"/>
   </plugin>
   ```

   where
   - *plugin-id* is the plug-in identifier.
   - *extension-point* is a pre-processing extension point.
   - *Ant-target* is the name of the Ant target.
3. Integrate the plug-in.

The new target is added to the Ant dependency list. The new target now is always run in conjunction with the specified step in the pre-processing pipeline.

Example

The following plugin.xml file specifies that the myAntTargetBeforeChunk target is always run before the `chunk` step in the pre-processing stage.

```
<plugin id="com.example.extendchunk">
  <feature extension="depend.preprocess.chunk.pre" value="myAntTargetBeforeChunk"/>
</plugin>
```

It assumes that the myAntTargetBeforeChunk target has already been defined and integrated.

Related reference
Pre-processing extension points

Adding a new transformation type

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

You can use the `dita.conductor.transtype.check` and `dita.transtype.print` extension points to define new transformation types.

When a transformation type is defined, the build expects Ant code to be integrated to define the transformation process. The Ant code must define a target based on the name of the transformation type; if the transformation type is "new-transform", the Ant code must define a target named dita2new-transform.

1. Create an Ant project file for the new transformation. This project file must define a target named "dita2*new-transtype*," where *new-transtype* is the name of the new transformation type.
2. Create the following feature:

   ```
   <plugin id="plugin-id ">
     <feature extension="dita.conductor.transtype.check" value="new-transtype"/>
     <feature extension="dita.transtype.print" value="new-transtype"/>
     <feature extension="dita.conductor.target.relative" file="ant-file"/>
   </plugin>
   ```

   where:

   - *plugin-id* is the plug-in identifier, for example, com.dita-ot.pdf.
   - *new-transtype* is the name of the new transformation, for example, dita-ot-pdf.
   - *ant-file* is the name of the Ant file, for example, build-dita-ot-pdf.xml.

   Exclude the content that is highlighted in bold if the transformation is not intended for print.

3. Integrate the plug-in.

You now can use the new transformation.

Examples

The following plugin.xml file defines a new transformation type named "newtext"; it also defines the transformation type to be a print type. The build will look for a dita2newtext target.

```
<plugin id="com.example.newtext">
  <feature extension="dita.conductor.transtype.check" value="newtext"/>
  <feature extension="dita.transtype.print" value="newtext"/>
  <feature extension="dita.conductor.target.relative" file="build-newtext.xml"/>
</plugin>
```

The following example shows how the org.dita.html5 plugin uses the `<transtype>` element to extend the common HTML transformation with a new html5 transformation type and define a new nav-toc parameter with three possible values:

```
<transtype name="html5" extends="common-html" desc="HTML5">
  <param name="nav-toc" type="enum"
        desc="Specifies whether to generate a navigation TOC in topic pages.">
    <val default="true" desc="No TOC">none</val>
    <val desc="Partial TOC that shows the current topic">partial</val>
    <val desc="Full TOC">full</val>
  </param>
</transtype>
```

Related reference
General extension points
Plug-in descriptor file
#unique_111

Overriding an XSLT-processing step

You can override specific XSLT-processing steps in both the pre-processing pipeline and certain DITA-OT transformations.

1. Develop an XSL file that contains the XSL override.
2. Construct a plugin.xml file that contains the following content:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <plugin id="plugin-id">
     <feature extension="extension-point" file="relative-path"/>
   </plugin>
   ```

   where:
   - *plugin-id* is the plug-in identifier, for example, com.example.brandheader.
   - *extension-point* is the DITA-OT extension point, for example, `dita.xsl.xhtml`. This indicates the DITA-OT processing step that the XSL override applies to.
   - *relative-path* is the relative path and name of the XSLT file, for example, xsl/header.xsl.
3. Integrate the plug-in.

The plug-in installer adds an XSL import statement to the default DITA-OT code, so that the XSL override becomes part of the normal build.

Example: Overriding XHTML header processing

The following two files represent a complete, simple style plug-in.

The plugin.xml file declares an XSLT file that extends XHTML processing:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="com.example.brandheader">
  <feature extension="dita.xsl.xhtml" file="xsl/header.xsl"/>
</plugin>
```

The xsl/header.xsl XSLT file referenced in plugin.xml overrides the default header processing to add a banner:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template name="gen-user-header">
    <div><img src="http://www.example.com/company_banner.jpg"
              alt="Example Company Banner"/></div>
  </xsl:template>
</xsl:stylesheet>
```

Related reference
XSLT-import extension points

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:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 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"/>
  <lang xml:lang="is-IS"  filename="mystring-is-is.xml"/>
  <lang xml:lang="ru"     filename="mystring-ru-ru.xml"/>
  <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:

```
<?xml version="1.0" encoding="utf-8"?>
<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.xml"/>
</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 non-deterministic, 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:

```
<?xml version="1.0" encoding="utf-8"?>
<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:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Map languages with xml:lang="vi" or xml:lang="vi-vn"
     to the translations in this plug-in. -->
<langlist>
  <lang xml:lang="vi"     filename="strings-vi.xml"/>
  <lang xml:lang="vi-VN"  filename="strings-vi.xml"/>
</langlist>
```

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 tasks
Globalizing DITA content
Related reference
Languages supported by the core toolkit

Adding parameters to existing XSLT steps

You can pass parameters from the Ant build to existing XSLT steps in both the pre-processing pipeline and certain DITA-OT transformations. This can be useful if you want to make the parameters available as global `<xsl:param>` values within XSLT overrides.

1. Create an XML file that contains one or more Ant `<param>` elements nested within a `<dummy>` wrapper element.

    ```
    <dummy>
      <!-- Any Ant code allowed in xslt task is possible. Common example: -->
      <param name="paramNameinXSLT" expression="${antProperty}" if="antProperty"/>
    </dummy>
    ```

2. Construct a plugin.xml file that contains the following content:

   ```
   <plugin id="plugin-id">
     <feature extension="extension-point" file="file"/>
   </plugin>
   ```

   where:
   - *plugin-id* is the plug-in identifier, for example, `com.example.newparam`.
   - *extension-point* is the DITA-OT extension point, for example, `dita.conductor.xhtml.param`. This indicates the DITA-OT processing step where the parameters will be available.
   - *file* is the name of the XML file that you created in step 1, for example, insertParameters.xml.
3. Integrate the plug-in.

The plugin.xml file passes the parameters to the specified transformation or pre-processing module.

Example

The following plug-in passes the parameters defined in the insertParameters.xml file as input to the XHTML process. Generally, an additional XSLT override will make use of the parameters to do something new with the generated content.

```
<plugin id="com.example.newparam">
  <feature extension="dita.conductor.xhtml.param" file="insertParameters.xml"/>
</plugin>
```

Related reference
XSLT-parameter extension points

Adding a Java library to the DITA-OT classpath parameter

You can use the `dita.conductor.lib.import` extension point to add an additional Java library to the DITA-OT classpath parameter.

1. If necessary, compile the Java code into a JAR file.
2. Create a plugin.xml file that contains the following code:

   ```
   <plugin id="plugin-id">
     <feature extension="dita.conductor.lib.import" file="file"/>
   </plugin>
   ```

   where:
   - *plugin-id* is the plug-in identifier, for example, com.example.addjar.
   - *file* is the name of the JAR file, for example, myJavaLibrary.jar.
3. Integrate the plug-in.

The Ant or XSLT code now can make use of the Java code.

In the following extended example, the myJavaLibrary.jar file performs a validation step during processing, and you want it to run immediately before the `conref` step. To accomplish this, you will need to use several features:

- The JAR file must be added to the classpath.
- The Ant target must be added to the dependency chain for conref.
- An Ant target must be created that uses this class, and the Ant wrapper integrated into the code.

The files might look like the following:

Figure 4. plugin.xml file

```
<?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.example.sampleValidation.jar"/>
  <!-- Integrate the Ant code -->
  <feature extension="dita.conductor.target.relative" file="antWrapper.xml"/>
  <!-- Define the Ant target that is called, and the location (before conref) -->
  <feature extension="depend.preprocess.conref.pre" value="validateWithJava"/>
</plugin>
```

Figure 5. antWrapper.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<dummy>
  <import file="calljava-antcode.xml"/>
</dummy>
```

Figure 6. calljava-antcode.xml file

```
<?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 classpath) -->
      <classpath refid="dost.class.path"/>
    </java>
  </target>
</project>
```

Related reference
General extension points

Adding new diagnostic messages

Use the `dita.xsl.messages` extension point to add plug-in-specific messages to the diagnostic messages that are generated by the DITA-OT. These messages then can be used by any XSLT override.

1. Create an XML file that contains the messages that you want to add. Be sure to use the following format for the XML file:

```
<dummy>
  <!-- See resources/messages.xml for the details. -->
  <message id="DOTXnumberletter" type="error-severity">
    <reason>Message text</reason>
    <response>How to resolve</response>
  </message>
</dummy>
```

where:
   - *number* is a three-digit integer.
   - *letter* is one of the following upper-case letters: I, W, E, F. It should match the value that is specified for the `@type` attribute.
   - *error-severity* specifies the severity of the error. It must be one of the following values:
     - INFO
     - WARN
     - ERROR
     - FATAL
2. Create a plug-in.xml file that contains the following content:

```
<plugin id="plugin-id">
  <feature extension="dita.xsl.messages" file="file"/>
</plugin>
```

where:

   - *plugin-id* is the plug-in identifier, for example, `com.example.newmsg`.
   - *file* is the name of the new messages file, for example, myMessages.xml.
3. Integrate the new plug-in.

You now can use the following call in XSLT modules to generate a message when a specific condition occurs:

```
<xsl:call-template name="output-message">
  <xsl:with-param name="msgnum">numberletter</xsl:with-param>
  <xsl:with-param name="msgsev">letter</xsl:with-param>
</xsl:call-template>
```

Related reference
General extension points

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://dita-ot.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 `behavior` attribute defines which processing action is used.

Supported values for `behavior` attribute:

`org.dita.dost.platform.CheckTranstypeAction`
Create Ant condition elements to check if the `${transtype}` property value equals a supported transformation type value.

`org.dita.dost.platform.ImportAntLibAction`
Create Ant `pathelement` elements for library imported extension 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 generated text extension point. The `id` attribute is used to define the extension point ID.

`org.dita.dost.platform.ImportXSLAction`
Create `xsl: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 Ant import extension point. The `id` attribute is used to define the extension point ID.

`org.dita.dost.platform.InsertAntActionRelative`
Include plug-in conductor content based on relative Ant import extension point. The `id` attribute is used to define the extension point ID.

`org.dita.dost.platform.InsertCatalogActionRelative`
Include plug-in catalog content based on catalog import extension point. The `id` attribute is used to define the extension point ID.

`org.dita.dost.platform.ListTranstypeAction`
Create a pipe delimited list of supported transformation types.

`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.

```
<plugin id="com.example.new-extensions">
  <extension-point id="com.example.new-extensions.pre"
                   name="Custom target preprocess"/>
  <extension-point id="com.example.new-extensions.content"
                   name="Custom target content"/>
  <template file="myBuildFile_template.xml"/>
</plugin>
```

When the integrator runs, this will be used to recreate myBuildFile.xml, replacing Ant file content based on extension point use.

```
<project xmlns:dita="http://dita-ot.sourceforge.net">
  <target name="dita2custom"
          depends="dita2custom.init,
                   {com.example.new-extensions.pre},
                   dita2xhtml"
          dita:extension="depends org.dita.dost.platform.InsertDependsAction">
    <dita:extension id="com.example.new-extensions.content"
                    behavior="org.dita.dost.platform.InsertAction"/>
  <target>
</project>
```

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 DITA-OT download page at SourceForge.

```
<plugin id="org.metadita.specialization.music">
  <feature extension="dita.specialization.catalog.relative" file="catalog-dita.xml">
  <feature extension="dita.xsl.xhtml" file="xsl/music2xhtml.xsl"/>
</plugin>
```

Related reference
Plug-in descriptor file
#unique_111

Extension points

The DITA Open Toolkit provides a series of extension points that can be used to integrate changes into the core code. Extension points are defined in the plugin.xml file for each plug-in. The DITA-OT integration process makes each extension visible to the rest of the toolkit.

General extension points

These extension points enable you to extend the DITA-OT. You can add Ant targets or imports; add a Java library to the classpath parameter; add a new transformation type; extend a catalog file; add new diagnostic messages, and more.

dita.conductor.lib.import
Adds a Java library to the DITA-OT classpath.

dita.conductor.target
Adds an Ant import to the main Ant build file. This extension point is deprecated; use `dita.conductor.target.relative` instead.

dita.conductor.target.relative
Adds an Ant import to the main Ant build file.

dita.conductor.transtype.check
Adds a new value to the list of valid transformation types.

dita.specialization.catalog
Adds the content of a catalog file to the main DITA-OT catalog file. This extension point is deprecated; use `dita.specialization.catalog.relative` instead.

dita.specialization.catalog.relative
Adds the content of a catalog file to the main DITA-OT catalog file.

dita.transtype.print
Defines a transformation as a print type.

dita.xsl.messages
Adds new diagnostic messages to the DITA-OT.

org.dita.pdf2.catalog.relative
Adds the content of a catalog file to the main catalog file for the PDF plug-in.

Pre-processing extension points

You can use these extension points to run an Ant target before or after the pre-processing operation; you also run an Ant target before or after a specific step in the pre-processing operation.

depend.preprocess.chunk.pre
Runs an Ant target before the `chunk` step in the pre-processing stage.

depend.preprocess.coderef.pre
Runs an Ant target before the `coderef` step in the pre-processing stage.

depend.preprocess.conref.pre
Runs an Ant target before the `conref` step in the pre-processing stage.

depend.preprocess.conrefpush.pre
Runs an Ant target before the `conrefpush` step in the pre-processing stage.

depend.preprocess.clean-temp.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.copy-files.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.copy-flag.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.copy-generated-files.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.copy-html.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.copy-image.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.copy-subsidiary.pre
Runs an Ant target before the `x` step in the pre-processing stage.

depend.preprocess.debug-filter.pre
Runs an Ant target before the `debug-filter` step in the pre-processing stage.

depend.preprocess.gen-list.pre
Runs an Ant target before the `gen-list` step in the pre-processing stage.

depend.preprocess.keyref.pre
Runs an Ant target before the `keyref` step in the pre-processing stage.

depend.preprocess.maplink.pre
Runs an Ant target before the `maplink` step in the pre-processing stage.

depend.preprocess.mappull.pre
Runs an Ant target before the `mappull` step in the pre-processing stage.

depend.preprocess.mapref.pre
Runs an Ant target before the `mapref` step in the pre-processing stage.

depend.preprocess.move-links.pre
Runs an Ant target before the `move-links` step in the pre-processing stage.

depend.preprocess.move-meta-entries.pre
Runs an Ant target before the `move-meta-entries` step in the pre-processing stage.

depend.preprocess.pre
Runs an Ant target before the pre-processing stage.

depend.preprocess.post
Runs an Ant target after the pre-processing stage.

depend.preprocess.topicpull.pre
Runs an Ant target before the `topicpull` step in the pre-processing stage.

XSLT-import extension points

You can use these extension points to override XSLT processing steps in pre-processing and certain transformation types. The value of the `<file>` attribute in the `<feature>` element specifies a relative path to an XSL file in the current plug-in. The plug-in installer adds a XSL import statement to the default DITA-OT code, so that the XSL override becomes part of the normal build.

Pre-processing

You can use the following extension points to add XSLT processing to modules in the pre-processing pipeline:

dita.xsl.conref
Overrides the pre-processing step that resolves conref.

dita.xsl.maplink
Overrides the `maplink` step in the pre-processing pipeline. This is the step that generates map-based links.

dita.xsl.mappull
Overrides the `mappull` step in the pre-processing pipeline. This is the step that updates navigation titles in maps and causes attributes to cascade.

dita.xsl.mapref
Overrides the `mapref` step in the pre-processing pipeline. This is the step that resolves references to other maps.

dita.xsl.topicpull
Overrides the `topicpull` step in the pre-processing pipeline. This is the step that pulls text into `<xref>` elements, as well as performing other tasks.

Transformations

You can use the following extension points to add XSLT processing to modules in DITA-OT transformations:

dita.xsl.docbook
Overrides the default DocBook transformation.

dita.xsl.eclipse.plugin
Overrides the step that generates the plugin.xml file for Eclipse Help.

dita.xsl.rtf
Overrides the default rich-text format (RTF) transformation.

dita.xsl.troff-ast
Overrides the intermediate block-and-phrase format that is generated as input to troff processing.

dita.xsl.troff
Overrides the XSL that converts block-and-phrase intermediate markup into troff.

dita.xsl.xhtml
Overrides the default HTML or XHTML transformation, including HTML Help and Eclipse Help. The referenced file is integrated directly into the XSLT step that generates XHTML.

dita.xsl.xslfo
Overrides the default PDF transformation (formerly known as PDF2). The referenced XSL file is integrated directly into the XSLT step that generates the XSL-FO.

Example

The following two files represent a complete (albeit simple) plug-in that adds a company banner to the XHTML output. The plugin.xml file declares an XSLT file that extends the XHTML processing; the xsl/header.xsl file overrides the default header processing to provide a company banner.

Figure 7. Contents of the plugin.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="com.example.brandheader">
  <feature extension="dita.xsl.xhtml" file="xsl/header.xsl"/>
</plugin>
```

Figure 8. Contents of the xsl/header.xsl file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template name="gen-user-header">
    <div><img src="http://www.example.com/company_banner.jpg"
              alt="Example Company Banner"/></div>
  </xsl:template>
</xsl:stylesheet>
</xml>
```

XSLT-parameter extension points

You can use theses extension points to pass parameters into existing XSLT steps in both the pre-processing pipeline and DITA-OT transformation. The parameters generally will be available as global `<xsl:param>` values with XSLT overrides.

Pre-processing

You can use the following extension points to pass parameters to modules in the pre-processing pipeline:

dita.preprocess.conref.param
Pass parameters to the `conref` module in the pre-processing pipeline

dita.preprocess.maplink.param
Pass parameters to the `maplink` module in the pre-processing pipeline

dita.preprocess.mappull.param
Pass parameters to the `mappull` module in the pre-processing pipeline

dita.preprocess.mapref.param
Pass parameters to the `mapref` module in the pre-processing pipeline

dita.preprocess.topicpull.param
Pass parameters to the `topicpull` module in the pre-processing pipeline

Transformations

You can use the following extension points to pass parameters to modules in DITA-OT transformations:

dita.conductor.eclipse.toc.param
Pass parameters to the XSLT step that generates the Eclipse Help table of contents (TOC).

dita.conductor.html.param
Pass parameters to the HTML and HTML Help transformations.

dita.conductor.pdf2.param
Pass parameters to the PDF (formerly PDF2) transformation.

dita.conductor.xhtml.param
Pass parameters to the XHTML and Eclipse Help transformations.

dita.conductor.xhtml.toc.param
Pass parameters to to the XSLT step that generates the XHTML table of contents (TOC).

Example

The following two files represent a complete (albeit simple) plug-in that passes the parameters defined in the insertParameters.xml file to the XHTML transformation process.

Figure 9. Contents of the plugin.xml file

```
<plugin id="com.example.newparam">
  <feature extension="dita.conductor.xhtml.param" file="insertParameters.xml"/>
</plugin>
```

Figure 10. Contents of the insertParameters.xml

```
<dummy>
  <!-- Any Ant code allowed in xslt task is possible. Common example: -->
  <param name="paramNameinXSLT" expression="${antProperty}" if="antProperty"/>
</dummy>
```

Version and support information

You can use these extension points to define version and support information for a plug-in. Currently, the DITA-OT does not do anything with this information, but it might do so in the future.

package.support.name
Specifies the person who provides support for the DITA-OT plug-in

package.support.email
Specifies the e-mail address for the person who provides support for the DITA-OT plug-in

package.version

Specifies the version of the DITA-OT plug-in. The value uses the following syntax:

```
major.minor.micro.qualifier
```

where:

- *major* is a number and is required.
- *minor* is a number and is optional.
- *micro* is a number and is optional.
- *qualifier* is optional and can be composed of numerals, uppercase or lower case letters, underscores, and hyphens.

By default, the package.version value is set to `0.0.0`.

Example

```
<plugin id="com.example.WithSupportInfo">
  <feature extension="package.support.name" value="Joe the Author"/>
  <feature extension="package.support.email" value="joe@example.com"/>
  <feature extension="package.version" value="1.2.3"/>
</plugin>
```

Customizing PDF output

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

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

1. 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 11. 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="print-pdf"/>
  <feature extension="dita.transtype.print" value="print-pdf"/>
  <feature extension="dita.conductor.target.relative" file="integrator.xml"/>
</plugin>
```

3. Add an Ant script (integrator.xml) to define the transformation type.
   Figure 12. 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.plugin.com.example.print-pdf.dir}/cfg"/>
  </target>
  <target name="dita2print-pdf" depends="dita2print-pdf.init, dita2pdf2"/>
</project>
```

4. In the new plug-in directory, add a cfg/catalog.xml file that specifies the custom XSLT style sheets.
   Figure 13. cfg/catalog.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog prefer="system" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="cfg:fo/attrs/custom.xsl" uri="fo/attrs/custom.xsl"/>
  <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 14. cfg/fo/attrs/custom.xsl file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
  <!-- Change page size to A4 -->
  <xsl:variable name="page-width">210mm</xsl:variable>
  <xsl:variable name="page-height">297mm</xsl:variable>
</xsl:stylesheet>
```

6. 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 15. cfg/fo/xsl/custom.xsl file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version="2.0">
  <!-- Move figure title to top and description to bottom -->
  <xsl:template match="*[contains(@class,' topic/fig ')]">
```

```
    <fo:block xsl:use-attribute-sets="fig">
      <xsl:call-template name="commonattributes"/>
      <xsl:if test="not(@id)">
        <xsl:attribute name="id">
          <xsl:call-template name="get-id"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates select="*[contains(@class,' topic/title ')]"/>
      <xsl:apply-templates select="*[not(contains(@class,' topic/title ') or contains(@class,' topic/desc 
      <xsl:apply-templates select="*[contains(@class,' topic/desc ')]"/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

7. 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 16. 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"/></variable>
  <!-- Change unordered list bullet to an em dash -->
  <variable id="Unordered List bullet">&#x2014;</variable>
</vars>
```

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
```

Run the integration process to install the plug-in and make the print-pdf transformation available.

Migrating customizations

If you have XSL transformation overrides, plugins or other customizations written prior to DITA-OT 2.2, you may need to make changes to ensure your overrides work properly with the latest toolkit versions.

In some cases, you may be able to remove old code that is no longer needed. In other cases, you may need to refactor your code to point to the modified extension points, templates or modes in recent toolkit versions.

Migrating to release 2.2

In DITA-OT 2.2, the HTML5 transformation was refactored as its own plugin and separate plugins were created for each of the rendering engine-specific PDF transformations.

HTML5

The HTML5 transformation introduced in release 2.0 as part of the XHTML plugin has been moved to a separate HTML5 plugin. Customizations that extended the previous HTML5 output under the XHTML plugin will probably need to be refactored on the new HTML5 plugin.

PDF

Processing specific to Apache FOP, Antenna House Formatter, and RenderX XEP has been separated into separate plugins for each of those rendering engines. Customizations that extended this processing might need to extend the new org.dita.pdf2.fop, org.dita.pdf2.axf, or org.dita.pdf2.xep plugins.

PDF customizations that are not specific to a rendering engine can continue to extend the org.dita.pdf2 plugin as before.

Migrating to release 2.1

In DITA-OT 2.1, the `insertVariable` template was deprecated for PDF transformations.

PDF

The following template has been deprecated:

- `insertVariable`

Calls to that template will result in warnings in the build log. To update your plugin, make the following changes:

```
<xsl:call-template name="~~insertVariable~~getVariable">
  <xsl:with-param name="~~theVariableID~~id" select="var-id"/>
  <xsl:with-param name="~~theParameters~~params">
    params
  </xsl:with-param>
</xsl:call-template>
```

Migrating to release 2.0

In DITA-OT 2.0, XSLT templates were converted to XSLT 2.0, variable typing was implemented, and some older templates were refactored or removed. In addition, the dita command simplifies distribution of plugins by allowing installation from a URL.

All Transformations — Variable Typing

XSLT stylesheets were converted to XSLT 2.0. With that change, variable types were also implemented. Plugins that change template variable values will need to make the following changes:

- Declare the same types defined in the default templates with `@as`.
- Ensure that the generated values conform to the declared type.

For example:

```
<xsl:variable name="urltest">
<xsl:variable name="urltest" as="xs:boolean">
```

All Transformations — Refactoring

Much of the toolkit code was refactored for release 2.0. Customization changes that were based on a specific template in a previous version of the toolkit might not work because the modified template is no longer used. If this is the case, the changes will need to be reimplemented based on the new XSLT templates.

HTML5

A new HTML5 transformation type has been added. Customizations that previously modified the XHTML output to generate valid HTML5 should still work, but basing your customization on the new transformation type might simplify the customization and reduce the work required to maintain compatibility with future versions of the toolkit.

Note: The HTML5 transformation was refactored with release 2.2. Before basing your customization on the changes in release 2.0, consider whether you might want to move to release 2.2 instead. See Migrating to release 2.2.

Plugin Installation and Distribution

Plugins can now be installed or uninstalled from a ZIP archive using the new dita command. Plugins can also be installed from a referenced URL. See Arguments and options for the dita command.

Migrating to release 1.8

In DITA-OT 1.8, certain stylesheets were moved to plug-in specific folders and various deprecated Ant properties, XSLT stylesheets, parameters and modes were removed from the XHTML, PDF and ODT transformations.

Stylesheets for the following transformation types have moved to plug-in specific folders:

- eclipsehelp
- htmlhelp
- javahelp
- odt
- xhtml

Preprocessing

The following deprecated Ant properties have been removed:

- `dita.script.dir`, use `${dita.plugin.`*`id`*`.dir}` instead
- `dita.resource.dir`, use `${dita.plugin.org.dita.base.dir}/resource` instead
- `dita.empty`
- `args.message.file`

XHTML

XSLT Java extension `ImgUtils` has been removed from stylesheets and been replaced with preprocessing module `ImageMetadataModule`. The old `ImgUtils` Java classes are still included in the build.

PDF

The following deprecated XSLT stylesheets have been removed:

- artwork-preprocessor.xsl
- otdita2fo_frontend.xsl

The following deprecated XSLT templates have been removed:

- `insertVariable.old`

The following deprecated XSLT modes have been removed:

- `layout-masters-processing`
- `toc-prefix-text`, use `tocPrefix` mode instead
- `toc-topic-text`, use `tocText` mode instead

Link generation has been simplified by removing deprecated arguments in favor of `args.rellinks`. The following deprecated Ant properties have been removed:

- `args.fo.include.rellinks`

The following XSLT parameters have been removed:

- `antArgsIncludeRelatedLinks`
- `disableRelatedLinks`

A call to a named template `pullPrologIndexTerms.end-range` has been added to `processTopic*` templates to handle topic wide index ranges.

Legacy PDF

The following deprecated XSLT stylesheets have been removed:

- dita2fo-shell_template.xsl
- topic2fo-shell.xsl

ODT

Link generation has been simplified by removing deprecated arguments in favor of `args.rellinks`. The following deprecated Ant properties have been removed:

- `args.odt.include.rellinks`

The following XSLT parameters have been added:

- `include.rellinks`

The following XSLT parameters have been removed:

- `disableRelatedLinks`

Migrating to release 1.7

In DITA-OT 1.7, a new preprocessing step implements flagging for HTML-based output formats. PDF processing was corrected with regard to `shortdesc` handling, and a new XSLT template mode was introduced for HTML TOC processing. Several stylesheets were moved to plug-in specific folders and deprecated properties and XSLT variables were removed.

A new job status file .job.xml has been introduced and replaces dita.list and dita.xml.properties as the normative source for job status. If you have custom processing which modifies the job properties, you should change your code to modify .job.xml instead.

Support for the following deprecated properties has been removed:

- `dita.input`
- `dita.input.dirname`
- `dita.extname`

Stylesheets for the following transformation types have moved to plug-in specific folders:

- docbook
- eclipsecontent
- troff
- wordrtf

If custom plug-ins have hard coded paths to these stylesheets, update references to use either `plugin` URIs in `xsl:import` instructions or use `dita.plugin.*` Ant properties.

The integration process has been changed to use strict mode by default. For old plug-ins which are not valid, lax processing mode can still be used.

Plug-ins that use the `MessageUtils` Java class must use `getInstance` method to access the `MessageUtils` instance, as `getMessage` methods have been changed to instance methods.

Preprocessing

The preprocessing Ant dependency chain has been cleaned up. Tasks no longer depend on the previous task in the default chain, but rather the whole preprocess dependency chain is defined by the `preprocess` task.

HTML

Core TOC generation has been moved to a separate XSLT stylesheet xsl/map2htmtoc/map2htmlImpl.xsl and the new templates use the mode `toc`. Plug-ins which override HTML TOC processing should change the map processing templates to `toc` mode.

HTML and extended transformation types

Flagging logic has been pulled out of the core X/HTML code and moved to a preprocess step. This significantly simplifies and optimizes the X/HTML code, while making flagging logic available to any other transformation type. The new preprocess step implements all flagging logic; for each active flag, it adds a DITA-OT specific hint into the intermediate topics (implemented as a specialization of the DITA <foreign> element). As part of this change, all flagging-related templates in the XHTML code (such as start-flagit and gen-style) are deprecated.

If you override the X/HTML transforms, you may need to update your overrides to use the new flagging logic. In most cases this just means deleting calls to the deprecated templates; in some cases, the calls can be replaced with 2 lines to process flags in new places. You should compare your override to the updated XHTML code and update as needed. See XHTML migration for flagging updates in DITA-OT 1.7 for details.

Plug-ins that provide support for new transforms need to ensure that they properly support the DITA <foreign> element, which should be ignored by default; if so, this change will have no immediate impact. Support for flagging new transformation types may be more easily added based on this update, because there is no need to re-implement flagging logic, but this is not required. See Flagging in the toolkit for details on how to add flagging support.

PDF

The following deprecated XSLT variables have been removed:

- `page-margin-left`
- `page-margin-right`

XSLT stylesheets have been split to separate specialization topic code and new `xsl:import` instructions have been added to topic2fo.xsl. Plug-ins which define their own shell stylesheet should be revised to import all the required stylesheet modules.

PDF processing used to replace topic `shortdesc` with map `shortdesc`, but this behavior was incorrect and was removed to comply with the DITA specification.

A new `#note-separator` variable string was added to facilitate customization.

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 <span>
    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 </span>
```

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 <span>
    Call "commonattributes" and ID processing
    Output contents
  Output end tag such as </div> or </span>
```

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/ditaval-startprop ')]/@outputclass" mode="add-ditaval-sty
```

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 `<ol>`, where flags must appear before the `<ol>` 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/ditaval-startprop ')]" mode="out-of-line"/>

Create ending flag and revision images:
<xsl:apply-templates select="*[contains(@class,' ditaot-d/ditaval-endprop ')]" mode="out-of-line"/>
```

For example, the following lines are used in DITA-OT 1.7 to process the `<ul>` element (replacing the 29 lines used in DITA-OT 1.6):

```
<xsl:template match="*[contains(@class,' topic/ul ')]">
  <xsl:apply-templates select="*[contains(@class,' ditaot-d/ditaval-startprop ')]" mode="out-of-line"/>
  <xsl:call-template name="setaname"/>
  <ul>
    <xsl:call-template name="commonattributes"/>
    <xsl:apply-templates select="@compact"/>
    <xsl:call-template name="setid"/>
    <xsl:apply-templates/>
  </ul>
  <xsl:apply-templates select="*[contains(@class,' ditaot-d/ditaval-endprop ')]" mode="out-of-line"/>
  <xsl:value-of select="$newline"/>
</xsl:template>
```

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 active and $DRAFT is on
    If active
        create division with rev style
            process element with mode="elementname-fmt"
        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="elementname-fmt"`. 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"`.

Migrating to release 1.6

In DITA-OT 1.6, various demo plugins were removed along with many deprecated properties, targets, templates and modes. The PDF2 transformation no longer supports the beta version of DITA from IBM, the "bkinfo" demo plug-in, or layout-masters.xml configuration.

Support for the old DITAVAL format (used before OASIS added DITAVAL to the standard in 2007) has been removed.

The demo folder has been deprecated and the following plug-ins have been moved to the plugins folder:

old path	**new path**
demo/dita11	plugins/org.dita.specialization.dita11
demo/dita132	plugins/org.dita.specialization.dita132
demo/eclipsemap	plugins/org.dita.specialization.eclipsemap
demo/fo	plugins/org.dita.pdf2
demo/tocjs	plugins/com.sophos.tocjs

old path	**new path**
demo/h2d	plugins/h2d
demo/legacypdf	plugins/legacypdf

The remaining plug-ins in the demo folder have been moved to a separate repository at github.com/dita-ot/ext-plugins.

The deprecated property `dita.input.valfile` should be replaced with the new argument property `args.filter`.

The `dita-preprocess` target has been removed and dependencies should be replaced with a target sequence `build-init, preprocess`.

Support for the `args.message.file` argument has been removed as message configuration has become static configuration.

The `workdir` processing instruction has been deprecated in favor of `workdir-uri`. The only difference between the two processing instructions is that `workdir-uri` contains a URI instead of a system path.

Preprocessing

The following deprecated templates and modes have been removed in topic pull stylesheets:

- inherit
- get-stuff
- verify-type-attribute
- classval
- getshortdesc
- getlinktext
- blocktext
- figtext
- tabletext
- litext
- fntext
- dlentrytext
- firstclass
- invalid-list-item
- xref

PDF2

The following deprecated items are no longer supported in the PDF transform:

- Support for the beta version of DITA, available from IBM before the OASIS standard was created in 2005.
- Support for the "bkinfo" demo plug-in, used to support book metadata before OASIS created the BookMap format in 2007.

- Support for layout-masters.xml configuration. Plug-ins should use the `createDefaultLayoutMasters` template instead.

The following extension-points have been added:

- `dita.conductor.pdf2.param` to add XSLT parameters to XSL FO transformation.

Custom PDF2 shell stylesheets need to be revised to not include separate IBM and OASIS DITA stylesheets. The *_1.0.xsl stylesheets have been removed and their imports must be removed from shell stylesheets.

The following template modes have been deprecated:

- toc-prefix-text
- toc-topic-text

The following named templates have been removed:

- processTopic
- createMiniToc
- processTopicTitle
- createTopicAttrsName
- processConcept
- processReference
- getTitle
- placeNoteContent
- placeImage
- processUnknowType
- insertReferenceTitle
- buildRelationships
- processTask

The main FO generation process now relies on the merging process to rewrite duplicate IDs. The default merging process did this already in previous releases, but now also custom merging processes must fulfill the duplicate ID rewrite requirement.

XHTML

The following named templates have been deprecated:

- make-index-ref

The following deprecated templates have been removed:

- revblock-deprecated
- revstyle-deprecated
- start-revision-flag-deprecated
- end-revision-flag-deprecated
- concept-links
- task-links
- reference-links
- relinfo-links

- sort-links-by-role
- create-links
- add-linking-attributes
- add-link-target-attribute
- add-user-link-attributes

The removed templates have been replaced by other templates in earlier releases and plug-ins should be changed to use the new templates.

ODT

The following deprecated templates have been removed:

- revblock-deprecated
- revstyle-deprecated
- start-revision-flag-deprecated
- end-revision-flag-deprecated

The removed templates have been replaced by other templates in earlier releases and plug-ins should be changed to use the new templates.

Migrating to release 1.5.4

DITA-OT 1.5.4 adds new extension points to configure behavior based on file extensions, declare print transformation types and add mappings to the PDF configuration catalog file. PDF output supports mirrored page layout and uses new font family definitions. Support for several new languages was added for PDF and XHTML output.

Configuration properties file changes

In previous versions, the lib/configuration.properties file was generated by the integration process. Integration has been changed to generate lib/org.dita.dost.platform/plugin.properties and the role of the old lib/configuration.properties has been changed to contain defaults and configuration options, such as default language.

The `dita.plugin.org.dita.*.dir` properties have been changed to point to the DITA-OT base directory.

To allow access to configuration files, the lib directory needs to be added to the Java classpath.

New plug-in extension points

New plug-in extension points have been added allow configuring DITA-OT behavior based on file extensions.

Extension point	Description	Default values
`dita.topic.extension`	DITA topic	.dita, .xml
`dita.map.extensions`	DITA map	.ditamap

Extension point	Description	Default values
`dita.html.extensions`	HTML file	.html, .htm
`dita.resource.extensions`	Resource file	.pdf, .swf

Both HTML and resource file extensions are used to determine if a file in source is copied to output.

A new plug-in extension point has been added to declare transformation types as print types.

Extension point	Description
`dita.transtype.print`	Declare transformation type as a print type.

The `print_transtypes` property in integrator.properties has been deprecated in favor of `dita.transtype.print`.

Plugin URI scheme

Support for the plugin URI scheme has been added to XSLT stylesheets. Plug-ins can refer to files in other plug-ins without hard-coding relative paths, for example:

```
<xsl:import href="plugin:org.dita.pdf2:xsl/fo/topic2fo_1.0.xsl"/>
```

XHTML

Support for the following languages has been added:

- Indonesian
- Kazakh
- Malay

PDF

Support for mirrored page layout was added. The default is the unmirrored layout. The following XSLT configuration variables have been deprecated:

- `page-margin-left`
- `page-margin-right`

The following variables should be used instead to control page margins:

- `page-margin-outside`
- `page-margin-inside`

The `args.bookmap-order` property has been added to control how front and back matter are processed in bookmaps. The default is to reorder the frontmatter content as in previous releases.

A new extension point has been added to add mappings to the PDF configuration catalog file.

Extension point	Description
`org.dita.pdf2.catalog.relative`	Configuration catalog includes.

Support for the following languages has been added:

- Finnish
- Hebrew
- Romanian
- Russian
- Swedish

PDF processing no longer copies images or generates XSL FO to output directory. Instead, the temporary directory is used for all temporary files and source images are read directly from source directory. The legacy processing model can be enabled by setting `org.dita.pdf2.use-out-temp` to `true` in configuration properties; support for the legacy processing model may be removed in future releases.

Support for FrameMaker index syntax has been disabled by default. To enable FrameMaker index syntax, set `org.dita.pdf2.index.frame-markup` to `true` in configuration properties.

A configuration option has been added to disable internationalization (I18N) font processing and use stylesheet-defined fonts. To disable I18N font processing, set `org.dita.pdf2.i18n.enabled` to `false` in configuration properties.

The XSLT parameters `customizationDir` and `fileProfilePrefix` have been removed in favor of the `customizationDir.url` parameter.

A new shell stylesheet has been added for FOP and other shell stylesheets have also been revised. Plug-ins which have their own shell stylesheets for PDF processing should make sure all required stylesheets are imported.

Font family definitions in stylesheets have been changed from Sans, Serif, and Monospaced to sans-serif, serif, and monospace, respectively. The I18N font processing still uses the old logical names and aliases are used to map the new names to old ones.

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: For HTML-based transformation types, this is effectively 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 behavior, the 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 ":" column-number
```

Image scaling

If both height and width attributes are given, the image is scaled non-uniformly.

If the 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, the 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, the DOTJ054E error is thrown. Depending on the processing-mode setting, error recovery 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.

Coderef processing

When `coderef` elements are used within codeblocks to reference external files with literal code samples, the system default character set is used as the target file encoding unless a different character set is explicitly defined via the mechanisms described under Character set definition.

Extended functionality

Code reference processing

Character set definition

DITA-OT supports defining the code reference target file encoding using the `format` attribute. The supported format is:

```
format (";" space* "charset=" charset)?
```

If a character set is not defined, the system default character set will be used. If the character set is not recognized or supported, the DOTJ052E error is thrown and the system default character set is used as a fall-back.

```
<coderef href="unicode.txt" format="txt; charset=UTF-8"/>
```

Line range extraction

Code references can be limited to extract only a specified line range by defining the `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 the end range is omitted, the range ends on the 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 RFC 5147. 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, the range starts from the first line; if the end range is omitted, the range ends on the 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 Open Toolkit documentation, there are other resources about DITA and the DITA-OT that you might find helpful.

Index

A

Ant: 22

    overview: 22

B

build.xml: 22

C

configuration properties : 40

    default.language: 40

    generate-debug-attributes: 40

    org.dita.pdf2.i18n.enabled: 40

    org.dita.pdf2.use-out-temp: 40

    processing-mode: 40

D

dita command: 38

    syntax: 38

F

files: 22, 40, 40

    build.xml: 22

    lib/configuration.properties file: 40

    plugin.properties: 40

    topic.fo file: 40

P

PDF processing : 40

configuration properties: 40

plugin.properties file: 40

T

topic.fo : 40

location of: 40