Authoring and publishing information for this document.

**Title**  
*DITA Open Toolkit User Guide*

**Edition, release, copyright**  
Third edition, April 18, 2007. Based on release 1.3.1 of DITA Open Toolkit. All files copyright 2006-2007 by VR Communications, Inc., unless otherwise indicated. Licensing and usage of this document and related materials is regulated by a Common Public License (CPL) granted by OASIS (Organization for the Advancement of Structured Information Standards),  

**Publishing information**  
DITA Open Toolkit is an open-source, reference implementation of the OASIS DITA standard (currently DITA 1.0).

**Authors**  
Anna van Raaphorst and Richard H. (Dick) Johnson, principals, VR Communications, Inc. ([http://www.vrcommunications.com](http://www.vrcommunications.com)). DITA logo artwork by Elizabeth Tichenor ([http://www.elizabethtichenor.com](http://www.elizabethtichenor.com)).

**Major changes in this edition**  
- The document title has been shortened  
- The document has been significantly restructured  
- New topics include key concepts, processing overview, localization, content reuse, building from within Eclipse, configuring the Eclipse XMLBuddy editor, creating Eclipse Infocenters, and publishing to XHTML using a frameset

**Acknowledgements**  
The authors would like to thank those members of the DITA community who have authored or contributed content to this document: Kylene Bruski, Don Day, Anne Gentle, JoAnn Hackos, Jennifer Linton, Deborah Pickett, Michael Priestley, Nat Wyatt, Stephen Zhang, and various anonymous IBM writers. We are also grateful to others in the community who have offered valuable encouragement, feedback, suggestions, and sources of information: Robert Anderson, Bernard Aschwanden, Ricardo Mattiazzi Baumgartner, Bob Doyle, Andy Hall, Erik Hennum, Carolyn Henry, Ellen Livengood, Shawn McKenzie, Scott Prentice, Paul Prescod, Todd Rose, Kate Wilhelm, Stacey Winn, Chris Wong, and Bonnie Yelverton.

**Invitation to contribute to future editions**  
The authors invite others in the DITA community to contribute to future editions of this document. To contact us: [http://www.vrcommunications.com/aboutus.htm](http://www.vrcommunications.com/aboutus.htm).
Installing the DITA Tookit full package distribution .......................................................... 52
Installing the DITA Tookit small package distribution ......................................................... 53
  Installing the small package on Windows ................................................................. 54
  Installing the small package on Linux .................................................................. 56
Installing on Mac OS ........................................................................................................... 58
Directories and files in the ditaot directory ................................................................. 58
Installing the optional tools .............................................................................................. 59
  Installing the HTML Help compiler on Windows .................................................. 60
  Installing FOP on Windows .................................................................................. 60
  Installing the JavaHelp processor on Windows ...................................................... 60
  Installing FOP on Linux .......................................................................................... 61
  Installing the JavaHelp processor on Linux ........................................................... 61
DITAblogs (installing and upgrading) ............................................................................. 61

DITA Open Toolkit plug-ins ............................................................................................... 63
  About DITA Open Toolkit plug-ins ........................................................................... 63
  Installing the plug-ins .............................................................................................. 64
    Installing the Idiom FO plug-in ............................................................................. 64
DITAblogs (plug-ins) .......................................................................................................... 64
For more information (plug-ins) ..................................................................................... 64

Setting up your working environment ........................................................................... 65
  Configuring your authoring tool to use the Toolkit catalog ........................................ 65
  Configuring your authoring tool to use DTD or schema URLs .................................... 69
  Configuring the Eclipse XMLBuddy editor for DITA ................................................ 70
  Setting up your source and output file directories .................................................... 71
DITAblogs (setting up) ....................................................................................................... 73
For more information (setting up) ................................................................................... 74

Processing (building) and publishing DITA documents ............................................... 75
  About Ant ................................................................................................................... 75
  About Ant scripts ........................................................................................................ 75
  Processing overview .................................................................................................. 80
  Ant processing parameters ....................................................................................... 82
  About the garage sample ............................................................................................ 86
  Processing to the key Toolkit output targets .............................................................. 87
    Processing to XHTML targets ............................................................................... 87
    Processing to HTML Help targets ....................................................................... 89
    Processing to PDF2 targets .................................................................................. 90
    Processing to Eclipse targets ............................................................................... 92
  Processing to other Toolkit output targets ................................................................ 98
    Processing to DocBook targets ............................................................................ 98
    Processing to JavaHelp targets ............................................................................ 100
    Processing from the Java command line .............................................................. 102
    Processing to PDF targets ................................................................................... 104
    Processing to troff targets ................................................................................... 104
    Processing to Word RTF targets .......................................................................... 105
Publishing content on a web server..........................................................188
For more information (distributing)..........................................................188

Migrating legacy content to DITA..............................................................189
About content migration...........................................................................189
DITAblogs (migrating content).................................................................189
For more information (migrating content)...............................................189

Sample files.............................................................................................191

Frequently asked questions (FAQs).........................................................193
Do I need to know XML to use DITA?......................................................193
Why is "Darwin" in the name of the DITA architecture and DITA Open Toolkit?..193
How does DITA differ from DocBook?....................................................194
Should I set my root element to <dita> or <concept> (or <task> or <reference>)?..194
How can I use conditional processing effectively?.................................195
What steps should I follow to architect and write a large DITA document?..196
How can I add a special character (for example, trademark) to a DITA file?..197

Release 1.3.1 and 1.3 information..........................................................199

Release history.......................................................................................203

DITA core vocabulary.............................................................................207
About the DITA core vocabulary.............................................................207
Administrator or manager audience category.........................................207
Apache Ant...............................................................................................207
Ant processing parameter........................................................................208
Ant script....................................................................................................208
Audience....................................................................................................208
Authoring tool (DITA)...............................................................................209
Best practice.............................................................................................209
Block element..........................................................................................209
Body element (<body>)............................................................................209
Build file...................................................................................................210
CMS (Content Management System).......................................................210
CSS (Cascading StyleSheet).................................................................210
Choice table.............................................................................................211
Collection-type attribute..........................................................................211
Command element (<cmd>).....................................................................211
Concept.....................................................................................................212
Concept analysis.....................................................................................212
Concept information type.........................................................................212
Conditional processing (filtering and flagging content)..........................212
Content.....................................................................................................213
Content inventory..................................................................................213
Content reference attribute.....................................................................213
Content reuse..........................................................................................213
Content specialist audience category....................................................213
Context element (<context>)...................................................................213
Introduction

This section contains introductory information about DITA, DITA Open Tooklit (OT), and this document.

Sections in this topic:

About DITA (Darwin Information Typing Architecture) on page 11
About DITA Open Toolkit (DITA OT) on page 11
About this document on page 12

About DITA (Darwin Information Typing Architecture)

DITA (Darwin Information Typing Architecture) is an XML-based, end-to-end architecture for authoring, producing, and delivering information (often called content) as discrete, typed topics. Typical information delivered using the DITA architecture is technical or scientific in nature and published as online help, through product support portals, or as print-ready PDF files.

DITA originated and is extensively used in the IBM Corporation; in 2005 it was adopted as an Organization for the Advancement of Structured Information Standards (OASIS) standard. DITA is currently used in many organizations world-wide, and is supported by an ever-growing list of commercial and open-source tools. DITA is actively being extended and enhanced under the direction of the OASIS DITA Technical Committee.

The DITA architecture, along with appropriate tools, is used to:

• Create, manage, and publish XML-based, structured information in a wide variety of environments and platforms
• Facilitate information sharing and reuse, and collaborative writing projects
• Reduce writing, publishing, and translation costs
• Integrate more tightly with related technologies

About DITA Open Toolkit (DITA OT)

DITA Open Toolkit is a Java-based implementation of the OASIS DITA Technical Committee’s specification for DITA DTDs and schemas. The Toolkit, which can be used in the Windows, Linux/UNIX, and Mac OS operating environments, transforms DITA content (maps and topics) into deliverable formats. The Toolkit uses Apache Ant for processing.

DITA Open Toolkit publishes to the following environments:

• Key targets
  • XHTML
  • HTML Help
  • PDF2 (processing using the Idiom plug-in and RenderX XEP)
  • Eclipse help
• Other targets
  • DocBook
  • Eclipse content
  • JavaHelp
• PDF (processing using standard Toolkit functionality and Apache FOP)

Note: The DITA Open Toolkit PDF transform, which was part of the 1.2.2 release, has been deprecated in favor of the PDF2 transform, which makes use of a plug-in from Idiom Technologies, Inc.

• troff
• Word RTF

About this document

DITA Open Toolkit User Guide (this document) is the definitive source of information about DITA Open Toolkit. It is also a product of the architecture and the recommended best practices, having been written entirely in DITA XML and produced using the principles and procedures described in the document.

Sections in this topic:

DITA Open Toolkit User Guide package on page 12
Target audience on page 12
Prerequisites on page 13
How and why this document was produced on page 13

DITA Open Toolkit User Guide package

This document is part of a package that includes the following components:

• XHTML, HTML Help, Eclipse help, and PDF versions of the published document
• The DITA source files for the document
• Two sets of sample files: garage and grocery shopping
• Seven PHP-based debugging, editing, and reporting tools


Target audience

An audience is a target group of users.

This document was written for both beginning and advanced users currently implementing or planning to implement DITA and DITA Open Toolkit to produce structured XML documents to be published through any of the supported channels.

Target audience categories and types for this document are:

Content specialists (for example, information architect; content creator and editor; and graphic, interface, print-document, and website designer)
Technical specialists (for example, application designer and developer, content manager, and database and system administrator)
Administrators and managers (for example, application designer and developer, content manager, and database and system administrator)
Vendors (for example, companies that want to embed the Toolkit in a software product)
Prerequisites

Before you use this document you should be familiar with your operating system environment (Windows, Linux/UNIX, or Mac OS) and the authoring tool you will use to create DITA files. We recommend using a DITA-aware editor. You may need to consult the documentation that came with your operating system and authoring tool as you use DITA and DITA Open Toolkit.

Before you use DITA Open Toolkit, be sure your operating environment meets the system requirements described in System requirements and supported applications on page 47.

How and why this document was produced

This document was produced as a collaborative effort by the two principals of VR Communications, Inc. (www.vrcommunications.com), Anna van Raaphorst and Richard H. (Dick) Johnson.

Anna van Raaphorst has worked for over 20 years helping organizations of all sizes with their information architecture, technical and marketing communications, and training needs. She is a frequent speaker on structured authoring and DITA tools and techniques. Anna is principal and content specialist at VR Communications, Inc.

Dick Johnson, with over 20 years of experience doing software design and development, is principal and software technology specialist for VR Communications. He has served as principal software engineer and team leader, user consultant, user group speaker, and technical writer in a number of software and scientific organizations.

We did the project for the following reasons:

• We have significant interest and prior involvement with structured writing, content management, scripting, and programming.
• It gave us a way to gain knowledge and experience in DITA and DITA Open Toolkit quickly.
• We wanted to help the DITA community by documenting and promoting DITA and the Toolkit.
• It was an opportunity for us to use our individual skill sets in a collaborative effort.

For more information about how we wrote the document, suggestions, guidelines, tips, and techniques, see the individual DITAblogs (notes on the production of this document) that are part of each chapter. For example, for our suggestions about troubleshooting the build process, see DITAblogs (troubleshooting) on page 126.
Key concepts (DITAdemystified)

This section contains information about some of the key concepts required to understand and use DITA and DITA Open Toolkit.

Sections in this topic:

Documentation lifecycle on page 15
Processing and publishing on page 18
Project components on page 23

Documentation lifecycle

Sections in this topic:

Traditional lifecycle on page 15
Typical DITA lifecycle on page 15
Technology trends in the documentation lifecycle on page 16
Creating and managing content on page 17

Traditional lifecycle

The key characteristics of the traditional (legacy) documentation lifecycle are listed and illustrated in the following figure.

<table>
<thead>
<tr>
<th>Planning</th>
<th>Writing</th>
<th>Editing</th>
<th>Production</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Writer</td>
<td>Editor</td>
<td>Production Staff</td>
<td>Print</td>
</tr>
<tr>
<td>Writer</td>
<td></td>
<td>Writer</td>
<td>Writer</td>
<td>Download</td>
</tr>
</tbody>
</table>

Characteristics
- Straight-line process, relatively long cycles
- Book paradigm
- Planning focuses on the product, not the needs of the user
- Docs owned by individual writers, little or no collaboration
- Few organizations concern themselves with information architecture
- "Developmental edits" are rare; copyedits done late in the cycle
- Production often done as a black box exercise
- Translation done after product general availability (in English)

Typical DITA lifecycle

The key characteristics of the DITA documentation lifecycle are listed and illustrated in the following figure.
Technology trends in the documentation lifecycle

Integration of documentation editing, processing, managing, and publishing technologies with other technologies in the product lifecycle drives many of the current trends in the documentation lifecycle. Customizing documentation to meet customer requirements (for example, translation) is also extremely important. "Keeping writers happy" with tools they have grown comfortable using over time is relatively unimportant in the larger scheme of things.

Some of the key trends of the documentation lifecycle are listed in the following figure.
Creating and managing content

Creating and managing DITA content involves writing and editing with a "DITA-aware" authoring tool, and (in most organizations) storing and managing content using a content management system. Some authoring tools and CMSs contain the embedded DITA Open Toolkit (or equivalent functionality) to allow processing within the tool.

A "DITA-aware" authoring tool is the software used to create DITA source content.

To be DITA-aware, an authoring tool must:

- Check for "well-formedness," that is, that the opening and closing tags match
  An example would be `<note>Text</note>.

- Check for validity, that is, that all DTD rules are obeyed
  An example would be a check to see whether the tags appearing in the document are nested according to rules in the document's DTDs.

DITA-aware authoring tools should insist on well-formedness and validity. That is, the tool should prevent you from saving or warn you before saving a file with errors.

Popular DITA-aware authoring tools include:

- Altova XMLSpy
- Arbortext Editor
- JustSystems XMetaL
- Adobe FrameMaker 7.2 with the Application Pack for DITA
- SyncRO Soft <oXygen/>
- Pixware XMLmind
- Stylus Studio
- Syntext Serna
- XMLBuddy (built into Eclipse)
- Xopus (front end for DocZone content management system)
If your DITA project involves large numbers of topics, many authors, or geographically distributed authoring and production teams, you may benefit from the features provided by a CMS, which can include:

- Workflow support
- Validation of topic links
- Support for the Semantic Web
- Localization (translation) support

To be truly effective, a CMS being used to store files for a DITA project must be aware of the tree-structured ("web") nature of the project, the content contained in the source files, and the relationships among the files and their content. The CMS must also be able to report on this kind of information: both meta information about the files, and syntactic and semantic information about the content. Ask your CMS vendor about the product's content analytical capabilities before deciding to buy.

It can also be useful for the CMS to include DITA Open Toolkit processing, and debugging and reporting aids that operate along with file processing. CMSs should also provide basic library (source control) functionality.

Popular CMSs include:

- Hosted (web-based)
  - Astoria
  - DocZone
  - XDocs
- Open-source
  - Alfresco
  - NetBeans
- Proprietary, server-based
  - Documentum
  - Ixiasoft
  - Vasont
- Hosted DITA training site: ditausers.org (includes the DITA Storm editor)

---

### Processing and publishing

Sections in this topic:

Processing and publishing overview on page 18
About XML on page 19
About DTDs on page 21
About schemas on page 21
About XSLT on page 22
About XPath on page 22
About Java on page 23

### Processing and publishing overview

Processing is producing output files from a set of DITA source files.

DITA Open Toolkit, working with other tools like the Java Development Kit (JDK) and Ant, provides a method to process DITA documents.
The following figure provides an overview of the processing and publishing of DITA documents using DITA Open Toolkit:

- In step 1, the Ant build tool is initialized (either through an optional batch script or directly from the command line), and arguments that will be passed from the Ant script to Ant are validated.
- In step 2, the Ant build tool calls the Toolkit, which produces the rendered temporary files. Input to this step is the .dita and .ditamap source files, and the DITA project DTDs or schemas.
- In step 3, the XSLT processor (SAXON or Xalan) produces the published output files. Input to this step are the XSLT stylesheets for the project, and the temporary files produced in step 2.

About XML

XML is a W3C-recommended general-purpose markup language that supports a wide variety of applications, and is extensible. XML is a simplified subset of Standard Generalized Markup Language (SGML). DITA is a "dialect" of XML.

The following figure shows a simple, annotated XML file.
The following figure shows the XML object model for the simple XML file.

The following figure shows a simplified tree structure model for a typical DITA project.
About DTDs

A DTD is the definition of the allowable elements, attributes, and other document pieces of an XML document. The DITA DTDs are (base) topic, concept, task, reference, map, and bookmap. Every DITA file must reference its DTD in the DOCTYPE declaration.

DTDs are not written in XML syntax.

The following figure shows a snippet of the DITA Concept DTD.

```xml
<!DOCTYPE concept[
<!ELEMENT concept (%title;, (%titlealt;)?,
(,%shortdesc; | %abstract;)?,
(,%prolog;)?, (%conbody;)?, (%related-links;)?,
(,%concept-info-types;)* )>
<!ATTLIST concept
id ID #REQUIRED
conref CDATA #IMPLIED
%select-atts;
%localization-atts;
%arch-atts;
outputclass CDATA #IMPLIED
domains CDATA "&included-domains;"
]>
```

About schemas

A schema defines the structure of an XML document. Schemas define elements and attributes valid for a particular namespace.

Schemas are W3C standards, and are written in XML syntax.

DITA schemas exist, but currently are seldom used. (DTDs are more common.)

The following figure shows the schema for DITA Concept.
About XSLT

XSLT, a W3C standard, is a language for transforming XML into other formats.

XSLT stylesheets are written in XML syntax.

In the final stage of processing, the DITA Open Toolkit runs XSLT stylesheet transforms (using either SAXON or Xalan) to produce output files. In certain cases, it is possible to override stylesheet processing to customize the output.

The following figure shows the XSLT stylesheet for the note element.

```
<xs:complexType name="concept.class">
  <xs:sequence>
    <xs:group ref="title"/>
    <xs:group ref="titlealtss" minOccurs="0"/>
    <xs:choice minOccurs="0">
      <xs:group ref="shortdesc"/>
      <xs:group ref="abstract"/>
    </xs:choice>
    <xs:group ref="prolog" minOccurs="0"/>
    <xs:group ref="combody" minOccurs="0"/>
    <xs:group ref="related-links" minOccurs="0"/>
    <xs:group ref="concept-info-types" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="conref" type="xs:string"/>
  <xs:attributeGroup ref="select-atts"/>
  <xs:attribute ref="ditaarch:DITAArchVersion"/>
  <xs:attribute name="outputclass" type="xs:string"/>
  <xs:attributeGroup ref="localization-atts"/>
  <xs:attributeGroup ref="global-atts"/>
  <xs:attribute ref="class" default=""/>
</xs:complexType>
```

About XPath

XPath traverses an XML document's tree structure to describe node sets in an XML document. XPath uses pattern-matching to find sets of elements (for example, `<note>Text</note>`).

XPath is a W3C standard and is used in XSLT.
The following figure shows XPath expressions in an XSLT document.

```xml
<xsl:template match="/*[contains(@class, ' map/map ')]">
  <xsl:param name="pathFromMaplist"/>
  <xsl:if test="./*[contains(@class, ' map/topicref ')][not(@toc='no')]">
    <UL><xsl:value-of select="$newline"/>
      <xsl:apply-templates select="*[contains(@class, ' map/topicref ')]">
        <xsl:with-param name="pathFromMaplist" select="$pathFromMaplist"/>
      </xsl:apply-templates>
    </UL><xsl:value-of select="$newline"/>
  </xsl:if>
</xsl:template>
```

About Java

Java is a general-purpose, object-oriented programming language.

The DITA Open Toolkit and other tools associated with it (for example, Apache Ant and Saxon) are written in Java. The fact that Java is portable allows the Toolkit to run in virtually any operating environment (for example, Windows, Linux/UNIX, and Mac OS).

The following figure shows a Java code example (Toolkit code).

```java
/*
 * This file is part of the DITA Open Toolkit project hosted on
 * SourceForge.net. See the accompanying license.txt file for
 * applicable licenses.
 */

/*
 * (c) Copyright IBM Corp. 2004, 2005 All Rights Reserved.
 */
package org.dita.dox.pipeline;

import org.dita.dox.exception.DITAOTException;
import org.dita.dox.module.AbstractPipelineModule;
import org.dita.dox.module.ModuleFactory;

/**
 * PipelineFacade implement AbstractFacade and control the constructing and executing
 * of a module.
 * 
 * @author Lian, Li
 *
 */
public class PipelineFacade implements AbstractFacade {

  /**
   * Automatically generated constructor: PipelineFacade
   */
  public PipelineFacade() {
  }
```

Project components

Key DITA project components include source files (for example, DITA source files, image files, ditaval files, and ditamaps), processing files (for example, batch scripts and Ant scripts), and output files (for example, XHTML topic files, XHTML TOC, HTML help TOC, HTML, index file, and PDF file). Most of the examples in
the following sections are from the garage sample, which is available as part of the DITA Open Toolkit User Guide package.

Sections in this topic:

Source files on page 24  
Processing files on page 26  
Output files on page 28

Source files

The following figure shows a complete concept file in the garage sample project.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" "./dtd/concept.dtd">
<concept id="garageTaskOverview" xml:lang="en-us">
  <title>Garage Tasks</title>
  <prolog>
    <author type="creator">IBM</author>
    <author type="contributor">Anna van Raaphorst</author>
    <publisher>OASIS (Organization for the Advancement of Structured Information Standards)</publisher>
    <copyright>
      <copyryear year="2001, 2005">
        <copyrholder>IBM Corporation</copyrholder>
      </copyryear>
      <created date="2001"/>
      <revised modified="2005"/>
      <revised modified="2008-June-29"/>
    </copyright>
    <crtdates>
      <crtdate>
        <metadata>
          <keywords>
            <indexterm>garage</indexterm> tasks related to <indexterm>indexterm</indexterm>
          </keywords>
          <proinfo>
            <prodname>DITA Open Toolkit</prodname>
          </proinfo>
        </metadata>
      </crtdate>
    </crtdates>
    <body>
      <p>When you go into the garage, be prepared to get your hands dirty!</p>
    </body>
  </prolog>
</concept>
```

The following figure shows a portion of a task file in the garage sample project.
The following figure shows the image that appears in the "washing the car" task. This is a .png file. The Toolkit can process .png, .jpg, and .gif.

Ditaval files are used in conditional processing. The following figure shows a .ditaval file that filters out topics in the garage sample that have to do with oil or snow.

Multiple .ditamap files can be used in a single project to process the same set of source files in different ways. The garage sample has a .ditamap file that processes the files in a hierarchy, with the tasks nested under an overview file about tasks, and the concepts nested under an overview file about concepts.

The following figure shows the hierarchy ditamap.
The following figure shows the ditamap that processes the topics as a sequence.

```
<?xml version="1.0" encoding="utf-8"?>
<DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "..//dtd//map.dtd">
<!-- (C) Copyright IBM Corporation 2001, 2006. All rights reserved. -->
<map title="Garage (sequence)" id="garage_top">
  <topicref href="concepts/lawnmower.dita" type="concept"/>
  <topicref href="concepts/oil.dita" type="concept"/>
  <topicref href="concepts/shelving.dita" type="concept"/>
  <topicref href="concepts/snowshovel.dita" type="concept"/>
  <topicref href="concepts/toolbox.dita" type="concept"/>
  <topicref href="concepts/tools.dita" type="concept"/>
  <topicref href="concepts/whiphose.dita" type="concept"/>
  <topicref href="concepts/wheelbarrow.dita" type="concept"/>
  <topicref href="concepts/workbench.dita" type="concept"/>
  <topicref href="tasks/changingtheoil.dita" type="task"/>
  <topicref href="tasks/organizing.dita" type="task"/>
  <topicref href="tasks/shovelingsnow.dita" type="task"/>
  <topicref href="tasks/sprayingpainting.dita" type="task"/>
  <topicref href="tasks/washingthecear.dita" type="task"/>
</map>
```

### Processing files

The following figure shows the (optional) batch script used to initiate Ant processing of the garage sample.
The following figure shows part of the "hierarchy" Ant script for the garage sample.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<target name="build" description="Building garage sample as a hierarchy of topics.">
  <echo>Building garage sample as a hierarchy of topics.</echo>
  <property name="toolkit_dir" value="${env.DITA_DIR}"/>
  <property name="baseldir" value="${env.DITA_DIR}"/>
  <property name="basedir" value="${env.DITA_DIR}"/>
  <property name="projdir" value="${basedir}/..DITASAMPLE_GARAGE_OUTPUT"/>
  <property name="outdir" value="${projdir}/..DITASAMPLE_GARAGE_OUTPUT"/>
  <property name="args.logdir" value="${outdir}"/>
</target>
```

The following figure shows an additional portion of the hierarchy Ant script for the garage sample.
Output files

The following figure shows the XHTML table of contents for the garage sample processed using the hierarchy Ant script.

- **Garage Tasks**
  - Changing the oil in your car
  - Organizing the workbench and tools
  - Shoveling snow
  - Taking out the garbage
  - Spray painting
  - Washing the car

- **Garage Concepts**
  - Lawnmower
  - Oil
  - Paint
  - Shelving
  - Snow shovel
  - Tool box
  - Tools
  - Water hose
  - Wheelbarrow
  - Workbench
  - Windshield washer fluid

The following figure shows the XHTML output for one of garage task files.
Taking out the garbage

Your municipality collects garbage from homes once a week, usually early in the morning.

1. Find out from the town what day of the week garbage is collected in your neighborhood.
2. The night before collection, place the garbage cans on the curb.
3. After the garbage has been collected, move the cans back into your garage.

Parent topic: Garage Tasks

The following figure shows the HTML Help table of contents for the garage sample processed using the hierarchy Ant script.

The following figure shows the HTML Help index for the garage sample.
The following figure shows PDF2 output (PDF output processed using the Idiom plug-in and XEP from RenderX).

**Organizing the workbench and tools**

Keep your garage neat by organizing the things you keep in the garage.

**Taking out the garbage**

Your municipality collects garbage from homes once a week, usually early in the morning.

1. Find out from the town what day of the week garbage is collected in your neighborhood.
2. The night before collection, place the garbage cans on the curb.
3. After the garbage has been collected, move the cans back into your garage.

The following figure shows a topic from this document (*DITA Open Toolkit User Guide*) displayed in an Eclipse Infocenter.
Processing to Eclipse help targets

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT directories in the root directory of your C: drive (or your home directory on Linux).

2. If you have not already done so, copy the garage sample files into the DITASAMPLE_GARAGE_SOURCE directory.

You should have a directory structure that looks like this:

```
DITASAMPLE_GARAGE_OUTPUT
DITASAMPLE_GARAGE_SOURCE
  web_sample
    concepts
    shared_files
    images
    styles
```

Getting started

If you are new to DITA and DITA Open Toolkit, we recommend that you follow these steps to get started.

1. Read the topics in *Evaluating DITA and DITA Open Toolkit* on page 41 for suggestions on how to evaluate for use in your environment, and how to choose your initial pilot project.

2. Be sure your system environment meets the requirements in *System requirements and supported applications* on page 47.

3. Install and become familiar with the authoring tool you plan to use to create DITA content.
   
   For information about choosing and installing an authoring tool, see topics in *Installation overview* on page 49.

4. Install DITA Open Toolkit and its prerequisite tools.
   
   For an overview of the installation process, see *Installation overview* on page 49. Step-by-step installation instructions are in the same section.

5. Set up your DITA source file and processing (build) environments.
   
   Information and instructions are in *Setting up your working environment* on page 65.

6. Process (build) the garage sample, provided for your use and reference.
   
   For information about processing, see *Processing (building) and publishing DITA documents* on page 75.

7. Create a few demo-level DITA topics and a map to aggregate them, and process them in the same way you did the garage sample files.
   
   For instructions, see *Creating DITA topics* on page 131 and *Creating DITA maps* on page 141.
   
   These two chapters make up a short tutorial that uses the grocery shopping sample. The sample contains template files and a complete Ant script.

8. When you feel comfortable with the basics, expand your DITA skill by reading introductory DITA texts, working through DITA tutorials, and taking courses or workshops on using DITA.
   
   For additional sources of information about DITA and DITA Open Toolkit, see *Getting information (general)* on page 35.

9. Before you jump into more ambitious DITA projects, do a thorough information analysis of your existing content and future requirements, and map out an expansion plan.

10. As you gain skill and tackle more complex DITA projects, use this document and other information you collect along the way for additional guidance and reference.
Getting information (general)

Sources of information about DITA and DITA Open Toolkit.

Sections in this topic:

- General Information on page 35
- Tools, downloads on page 35
- Training, education on page 36
- Consulting on page 36
- DITA and DITA Open Toolkit standards and product releases on page 36
- Usage Information on page 37
- User groups, forums on page 38
- Conferences on page 38

General Information

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>dita.xml.org: Official DITA community gathering place and information resource for the DITA OASIS Standard</td>
<td><a href="http://dita.xml.org">http://dita.xml.org</a></td>
</tr>
<tr>
<td>Includes a knowledge base, a wiki, news items about DITA, events, products and services, case studies, user groups, forums, blogs, and a resource directory. Hosted by OASIS.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cover Pages: Articles, presentations, and facts about DITA</td>
<td><a href="http://xml.coverpages.org/dita.html">http://xml.coverpages.org/dita.html</a></td>
</tr>
<tr>
<td>Hosted by OASIS.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
</table>

Tools, downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download site for the Toolkit and related/prerequisite software</td>
<td><a href="https://sourceforge.net/projects/dita-ot">https://sourceforge.net/projects/dita-ot</a></td>
</tr>
<tr>
<td>alphaWorks IBM TaskModeler: An Eclipse-based tool for rapidly creating and analyzing models of human activity for DITA and user experience design</td>
<td><a href="http://www.alphaworks.ibm.com/tech/taskmodeler">http://www.alphaworks.ibm.com/tech/taskmodeler</a></td>
</tr>
</tbody>
</table>
# Training, education

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction to DITA:</strong> A User Guide to the Darwin Information Typing Architecture</td>
<td><a href="http://www.comtech-serv.com">http://www.comtech-serv.com</a></td>
</tr>
</tbody>
</table>

| Article (level: beginner). Authors: Eric Hennum, Don Day, John Hunt, and Dave Schell. Publisher: IBM, updated September 2005. Roadmap for the Darwin Information Typing Architecture: what it is and how it applies to technical documentation. Also a product of the architecture, having been written entirely in XML and produced using the principles described. |


# Consulting

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of DITA consultants</strong></td>
<td><a href="http://dita.xml.org/taxonomy/term/48">http://dita.xml.org/taxonomy/term/48</a></td>
</tr>
</tbody>
</table>

# DITA and DITA Open Toolkit standards and product releases

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DITA 1.1 standard</strong></td>
<td><a href="http://docs.oasis-open.org/dita/v1.1/CD01/overview/overview.html">http://docs.oasis-open.org/dita/v1.1/CD01/overview/overview.html</a></td>
</tr>
<tr>
<td>Specification overview.</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Information source</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Roadmap for DITA development</td>
<td><a href="http://wiki.oasis-open.org/dita/Roadmap_for_DITA_development">http://wiki.oasis-open.org/dita/Roadmap_for_DITA_development</a></td>
</tr>
<tr>
<td>Lists of proposed and committed features for future releases.</td>
<td></td>
</tr>
<tr>
<td>Current DITA Open Toolkit release</td>
<td><a href="http://dita-ot.sourceforge.net/">http://dita-ot.sourceforge.net/</a></td>
</tr>
<tr>
<td>Bugs, support requests, patches, feature requests.</td>
<td></td>
</tr>
<tr>
<td>OASIS DITA Open Toolkit project home</td>
<td><a href="http://sourceforge.net/projects/dita-ot/">http://sourceforge.net/projects/dita-ot/</a></td>
</tr>
<tr>
<td>OASIS DITA Open Toolkit 1.3.1 features and issues tracking</td>
<td><a href="http://dita.xml.org/node/1282">http://dita.xml.org/node/1282</a></td>
</tr>
<tr>
<td>DITA Open Toolkit developers mailing list</td>
<td><a href="https://lists.sourceforge.net/lists/listinfo/dita-ot-developer">https://lists.sourceforge.net/lists/listinfo/dita-ot-developer</a></td>
</tr>
<tr>
<td>DITA 1.1 proposed features</td>
<td><a href="http://wiki.oasis-open.org/dita/DITA_1.1_Features">http://wiki.oasis-open.org/dita/DITA_1.1_Features</a></td>
</tr>
<tr>
<td>DITA 1.1 work items</td>
<td><a href="http://wiki.oasis-open.org/dita/List_of_DITA_1.1_Work_Items">http://wiki.oasis-open.org/dita/List_of_DITA_1.1_Work_Items</a></td>
</tr>
<tr>
<td>OASIS DITA standards committee website</td>
<td><a href="http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita">http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita</a></td>
</tr>
<tr>
<td>OASIS Standard, 09 May 2005.</td>
<td></td>
</tr>
<tr>
<td>OASIS DITA Architecture Specification version 1.1</td>
<td><a href="http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.html#concept-1">http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.html#concept-1</a></td>
</tr>
</tbody>
</table>

**Usage Information**

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Derby documentation</td>
<td>For an overview of the documentation, see <a href="http://db.apache.org/derby/manuals/index.html">http://db.apache.org/derby/manuals/index.html</a>.</td>
</tr>
<tr>
<td>Apache Derby, an Apache database subproject, is a relational database implemented entirely in Java, and available under the Apache license, version 2.0. The Derby documentation manuals are sourced in DITA.</td>
<td>For the DITA source files, see <a href="http://db.apache.org/derby/manuals/dita.html">http://db.apache.org/derby/manuals/dita.html</a>.</td>
</tr>
</tbody>
</table>
### User groups, forums

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forum for DITA questions and discussion</td>
<td><a href="http://groups.yahoo.com/group/dita-users">http://groups.yahoo.com/group/dita-users</a></td>
</tr>
<tr>
<td>Forum for Adobe FrameMaker/DITA questions</td>
<td><a href="http://groups.yahoo.com/group/framemaker-dita">http://groups.yahoo.com/group/framemaker-dita</a></td>
</tr>
<tr>
<td>List of DITA user and interest groups (by location)</td>
<td><a href="http://dita.xml.org/user-groups">http://dita.xml.org/user-groups</a></td>
</tr>
<tr>
<td>DITA forums on sourceforge.net (there are three forums: developers, help, and open discussion)</td>
<td><a href="https://sourceforge.net/forum/?group_id=132728">https://sourceforge.net/forum/?group_id=132728</a></td>
</tr>
</tbody>
</table>

### Conferences

**Note:** The following DITA-related conferences were or will be held in 2007. For the latest information about these conferences, see the referenced websites.

<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: March 26-28. Boston, MA. Conference organizer: Center for Information Development Management (CIDM).</td>
<td></td>
</tr>
<tr>
<td>Date: May 15-18. Location: Paris. XTech 2006 is the premier European conference for developers, information designers and managers working with web and standards-based technologies. Focus: The theme is “The Ubiquitous Web.”</td>
<td></td>
</tr>
<tr>
<td>X-Pubs 2007</td>
<td><a href="http://www.x-pubs.com">http://www.x-pubs.com</a></td>
</tr>
<tr>
<td>Date: June 4-5. Location: Reading, UK. Focus: The theme is “How did they do it?”</td>
<td></td>
</tr>
<tr>
<td>Date: October 6. Location: Raleigh, NC. DITA 2007. (This is the DITA East conference. DITA West was held in February in San Jose, CA.) Conference organizer: Bright Path Solutions.</td>
<td></td>
</tr>
<tr>
<td>XML 2007</td>
<td><a href="http://2007.xmlconference.org/public/content/home">http://2007.xmlconference.org/public/content/home</a></td>
</tr>
</tbody>
</table>
Evaluating DITA and DITA Open Toolkit

This section contains information on how to determine what it would take to create your DITA and DITA Open Toolkit authoring and production system.

"The word is out" among professional communicators and information development and publishing organizations about the benefits of using DITA and DITA Open Toolkit to produce high-quality, reusable, structured communications. However, for many individuals and organizations simply "knowing about" DITA and the Toolkit are not enough. Not having answers to two key questions prevent them from moving from the "active interest" stage to the "assessment and adoption" stage. These key questions are:

• "What would it take to put together a DITA-based authoring and production system that would scale as my needs expand?"
• "How can I get started with DITA at a 'hands-on' level without first making an expensive, time-consuming, long-term commitment that I may come to regret?"

A key goal of this chapter is to help answer those questions, and to provide pointers to additional sources of information.

Sections in this topic:

The DITA authoring/production framework on page 41
Use cases on page 43
Use case template on page 44

The DITA authoring/production framework

Sections in this topic:

Creating an expansible DITA authoring and production system on page 41
Base components required on page 41
Base skills required on page 42
System maturity levels, and the requirements of each on page 42

Creating an expansible DITA authoring and production system

One of the key questions for assessing DITA and DITA Open Toolkit is "What would it take to put together a DITA-based authoring and production system that would scale as my needs expand?" The answer depends on both your near-term goals as well as what you expect your ultimate system to look like. The information below lists and defines four "DITA maturity" levels: demo, pilot project, basic end-to-end system, and enterprise-level system, and the likely requirements for each.

Base components required

The base components, which apply to all levels, include the following:

• One of the supported operating systems (Microsoft Windows, Linux/UNIX, or Mac OS)
• The prerequisite processing components: Java Development Kit (JDK), Ant build tool, and an XSLT processor appropriate for the operating system (for example, SAXON or Xalan)
• DITA Open Toolkit
• A simple text editor (for example, Notepad for Windows) or "DITA-aware" editor (for example, XMLSpy) to create DITA source files
• Basic delivery components for your planned publishing environments (for example, if you publish in XHTML, you need a browser that can display XHTML)

Base skills required
The base skills, which also apply to all levels, include the following:
• A basic understanding of the operating environment where you will install the processing components and DITA Open Toolkit
• The ability to follow the installation and setup instructions for the prerequisite processing components and Toolkit
• The ability to follow instructions to create simple DITA source files

System maturity levels, and the requirements of each

Demo level
Materials written and processed at a demo level simply provide a proof-of-concept, or a way to demonstrate the end-to-end authoring and production processes. Two sets of such files (the garage sample and the grocery shopping sample) are provided with the DITA Open Toolkit User Guide package.

Only the base components and skills are required, unless you want to demo in a publishing environment other than XHTML (which is the default and assumed publishing environment, unless otherwise indicated).

Pilot project (prototype) level
You can do a pilot project with base-level tools and skills, without a library or content management system (CMS), and without doing extensive system-level integration. However, you may want to add the following to the base-level components:
• Components to support additional publishing environments (for example, if you plan to publish HTML Help you will need the HTML Help compiler)
• An authoring environment that supports DITA and has other features you require (for example, spellcheck and CSS support)
• If you plan to include non-text content, a specialized editing component (for example, for image content, video, or Flash)
• Tools for compiling and previewing your output in whatever publishing environment you are using (for example, you may want Microsoft's HTML Help Workshop, Sun's JavaHelp tools, or the Eclipse 2 Standalone Help System)
• If you plan to produce printed deliverables, tools that provide FO processing and preview
• Debugging and reporting tools, which may be available free or for purchase (such a toolset was created by the authors of this book, and is available free)

The following additional skills may be required:
• A good understanding of your target publishing environment(s)
• Some level of expertise with the DITA-aware editor
• Skill with other tools you plan to use

Basic end-to-end system level
Individuals and organizations working at this level need to create a more formal process, and plan for the acquisition of tools that will serve at this level and also at the enterprise level, if project plans extend that far. This level may require processing scripts (perhaps even with user interfaces for setting parameters and saving profiles) that link the components together and provide consistency for repeatable processing. Branding is a probably a factor important for the DITA output.

At this level, documentation is part of a formal product cycle. The information development process probably involves multiple people. Producing output may be a collaborative process between the information development and product development organizations. Writers are probably required to follow departmental
style and production guidelines. Information being developed goes through at least an informal review process. Candidate output documents may need to be formally verified or tested before being released.

When you move to this kind of production system, you may also need:

- A make or macro component to program a sequence of processes
- A packaging tool for distribution

Projects at this stage may require a number of additional specialized skills, for example:

- Information architecture, planning, and workflow expertise
- The ability to install and set up additional tools of a more complex nature
- The ability to create and adapt XSLT
- Graphic design expertise
- User interface and user experience skills

**Enterprise system level**

This level has specific support for enterprise business rules, for editor wizards and for what the DITA architecture calls the “delivery context” layer (for example, true book-like output or specific mapping methodologies). The content may be translated into one or more languages. The distribution or fulfillment process is complex. A customer feedback mechanism needs to be present.

There is a need for extensive record-keeping, including the analysis of metadata. Content terminology and topics may be shared among a number of organizations (for example, marketing, technical writing, and training). There may be coordinated or shared content among other internal product or component development groups, or even external groups like business partners.

Writers may be required to adhere to an organization-wide set of style and terminology guidelines. Content may be formally reviewed by a group of editors, according to a complex schedule. Verification and testing is probably done by a separate test organization.

Tools at this level could include a library system, a content management system, business analysis software, and a customer feedback mechanism or system.

---

### Use cases

Sections in this topic (use case categories and example types):

- **Problem/solution scenarios** on page 43
- **Industry scenarios** on page 44
- **Publishing environment scenarios** on page 44
- **Access and distribution scenarios** on page 44
- **Authoring within an integrated development environment (IDE)** on page 44

### Problem/solution scenarios

Use cases in this category describe a particular problem and its solution using DITA and DITA Open Toolkit.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation/localization</td>
<td>Producing documentation that needs to be translated into other languages. Working with translation centers to produce high-quality documentation in all target languages on the same schedule as the English-language version of the documentation.</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Working with internal or business</td>
<td>Coordinating with internal (for example, training) or external (for example, business partner) organizations. Delivering simultaneously or on separate schedules. Strong need to coordinate terminology and coverage of key content.</td>
</tr>
<tr>
<td>partners</td>
<td></td>
</tr>
<tr>
<td>Using a library system or content</td>
<td>Tips on architecting, organizing, creating, and processing DITA information when using a library system or CMS.</td>
</tr>
<tr>
<td>management system</td>
<td></td>
</tr>
<tr>
<td>Using a controlled vocabulary,</td>
<td>How use of a controlled vocabulary (ontology, taxonomy) relates to architecting, organizing, creating, and processing DITA content.</td>
</tr>
<tr>
<td>taxonomy, or ontology</td>
<td></td>
</tr>
<tr>
<td>Modeling content</td>
<td>Using content modeling tools.</td>
</tr>
<tr>
<td>Migrating legacy content</td>
<td>Issues, tools, and techniques related to migration from books to topic-based information, or from unstructured to structured documentation. A related topic could be migrating to new tools (for example, from unstructured to structured FrameMaker, or from Microsoft Word to Arbortext Editor).</td>
</tr>
<tr>
<td>Migrating to a new tool</td>
<td>For example, from unstructured to structured FrameMaker, or from Microsoft Word to Arbortext Editor.</td>
</tr>
<tr>
<td>Prototyping</td>
<td>How and when to prototype. Use of tools to facilitate prototyping.</td>
</tr>
<tr>
<td>Lone writer scenario</td>
<td>Small organization where a single person needs to have all skills (architecture, communication, and technical) and wear all hats.</td>
</tr>
</tbody>
</table>

**Industry scenarios**

Examples might focus on the particular needs of the software or hardware industry, biotechnology, insurance, or finance.

**Publishing environment scenarios**

Use cases in this category describe scenarios particular to one or more publishing environments, for example Eclipse help, HTML Help, PDF, XHTML, or a combination of multiple publishing environments.

**Access and distribution scenarios**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSS</td>
<td>Using RSS to distribute your published output.</td>
</tr>
<tr>
<td>RDF/OWL</td>
<td>Using RDF/OWL to improve access to your published information. Using tools like SPARQL to query RDF information.</td>
</tr>
</tbody>
</table>

**Authoring within an integrated development environment (IDE)**

Examples might include popular software development environments like Eclipse and NetBeans.

**Use case template**

This sample template was included to promote consistency in writing up use cases for DITA or DITA Open Toolkit.

**Summary**

**Organization name**
Author name
Date
Industry, sector
Category
Examples: cost; translation, localization; working with a business or internal partner; using a library system or CMS; metadata (controlled vocabulary, ontology, taxonomy); content modeling; migrating legacy content; prototyping; lone writer scenario; output type (XHTML, HTML Help, PDF, etc); industry; distribution or access scenario.

Prime motivation
Examples: contain spiraling translation costs, reduce time to market, work more effectively with business partners

Problem
What specific problem were you trying to solve? (100-200 words)

Alternatives
What alternatives did you explore or try, and what were the pros and cons of each? (50-100 words)

Solution
What was the solution, and what DITA-related tools and techniques did you use? (200-400 words)

Result
Was the original problem completely solved? What was the user reaction? Include testimonials, if possible. (100-200 words)

Future plans
Is any follow-on work planned? If so, how did this project set the stage? (50-100 words)
Installing and upgrading DITA Open Toolkit

This section contains information on how to install and upgrade DITA Open Toolkit on Windows, Linux/UNIX, and Mac OS.

Sections in this topic:

- System requirements and supported applications on page 47
- Installation overview on page 49
- Installing your DITA authoring tool on page 51
- Installing the Java SDK (JDK) on page 51
- Installing the DITA Tookit full package distribution on page 52
- Installing the DITA Tookit small package distribution on page 53
- Installing on Mac OS on page 58
- Directories and files in the ditaot directory on page 58
- Installing the optional tools on page 59
- DITAblogs (installing and upgrading) on page 61

System requirements and supported applications

System requirements

DITA Open Toolkit is written in Java and requires at least a minimal set of Java applications be installed. Java SDK 1.4.2, 1.5, or 1.6 must be used to execute the applications and the Toolkit Java code.

It is highly likely that any operating system environment where the supported Java SDK can be installed will support basic Toolkit functionality. The Toolkit has been successfully installed and used on Windows XP, Mac OS X, various UNIX and Linux distributions including FreeBSD, Ubuntu Linux, NexentaGNU/OpenSolaris, Solaris, and other operating environments.

Some optional applications can be installed and run only on Windows, for example the HTML Help compiler.

Required tools

Note: All of the required tools except the JDK are bundled in the full package installation. For more information about the installation packages, see Installation considerations on page 50

The following tools are required to use DITA Open Toolkit at the 1.3.1 release level.

Java Development Kits (SDKs)

Sun. You can download the Sun JDK from http://java.sun.com/j2se/1.4.2/download.html.

IBM. You can download the IBM JDK from http://www.ibm.com/developerworks/java/jdk.

Ant

Ant 1.6.5. You can download Ant from http://ant.apache.org/bindownload.cgi.

Either the SAXON or Xalan XSLT processor

SAXON 6.5. You can download SAXON from http://saxon.sourceforge.net/.


Supported languages
DITA and DITA Open Toolkit support the languages listed in the following table.

<table>
<thead>
<tr>
<th>Language</th>
<th>xml:lang value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>ar-eg</td>
</tr>
<tr>
<td>Belarusian</td>
<td>bg-bg</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>be-by</td>
</tr>
<tr>
<td>Catalan</td>
<td>ca-es</td>
</tr>
<tr>
<td>Chinese (Simplified)</td>
<td>zh-cn</td>
</tr>
<tr>
<td>Chinese (Traditional)</td>
<td>zh-tw</td>
</tr>
<tr>
<td>Croatian</td>
<td>hr-hr</td>
</tr>
<tr>
<td>Czech</td>
<td>cs-cz</td>
</tr>
<tr>
<td>Danish</td>
<td>da-dk</td>
</tr>
<tr>
<td>Dutch</td>
<td>nl-nl</td>
</tr>
<tr>
<td>Dutch (Belgian)</td>
<td>nl-be</td>
</tr>
<tr>
<td>English (Canadian)</td>
<td>en-ca</td>
</tr>
<tr>
<td>English (UK)</td>
<td>en-gb</td>
</tr>
<tr>
<td>English (US)</td>
<td>en-us</td>
</tr>
<tr>
<td>Estonian</td>
<td>et-ee</td>
</tr>
<tr>
<td>Finnish</td>
<td>fi-fi</td>
</tr>
<tr>
<td>French</td>
<td>fr-fr</td>
</tr>
<tr>
<td>French (Belgian)</td>
<td>fr-be</td>
</tr>
<tr>
<td>French (Canadian)</td>
<td>fr-ca</td>
</tr>
<tr>
<td>French (Swiss)</td>
<td>fr-ch</td>
</tr>
<tr>
<td>German</td>
<td>de-de</td>
</tr>
<tr>
<td>German (Swiss)</td>
<td>de-ch</td>
</tr>
<tr>
<td>Greek</td>
<td>el-gr</td>
</tr>
<tr>
<td>Hebrew</td>
<td>he-il</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hu-hu</td>
</tr>
<tr>
<td>Icelandic</td>
<td>is-is</td>
</tr>
<tr>
<td>Italian</td>
<td>it-it</td>
</tr>
<tr>
<td>Italian (Swiss)</td>
<td>it-ch</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja-jp</td>
</tr>
<tr>
<td>Korean</td>
<td>ko-kr</td>
</tr>
<tr>
<td>Latvian</td>
<td>lv-lv</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>lt-lt</td>
</tr>
<tr>
<td>Macedonian</td>
<td>mk-mk</td>
</tr>
<tr>
<td>Norwegian</td>
<td>no-no</td>
</tr>
</tbody>
</table>
DITA Open Toolkit distributions

DITA Open Toolkit is available in the following distribution formats:

- Full package distribution
- Small package distribution
- Source distribution

These distributions are all available for download from [http://sourceforge.net/projects/dita-ot](http://sourceforge.net/projects/dita-ot).

**Full package distribution**

The full package distribution contains the Toolkit and most of the basic tools required for doing document builds. Included in the full package are:

- DITA Open Toolkit
- Ant build processor
- XML catalog resolver
- FOP processor for creating PDF outputs
- icu4j (ICU) globalization routines
- Xalan XSLT processor
- Shell scripts for setting the necessary runtime environment variables

To process DITA documents you must also download and install the Java J2SE SDK.
If one of your target output types is HTML Help, you will probably want to install the Microsoft HTML Help compiler. If you one of your target output types is JavaHelp, you will probably want to install the JavaHelp processor.

**Small package distribution**

The small package distribution contains only DITA Open Toolkit. You must separately install all the other required and optional processors to create a functioning build environment.

Release 1.2.2 and prior releases of the Toolkit were distributed only in this way.

You might want to download this distribution if you have a prior version of the Toolkit already installed, since the release 1.3 and 1.3.1 Toolkit prerequisites are still the same as those for release 1.2.2.

The small distribution is typically the one used to embed the Toolkit in other products.

**Source distribution**

The source distribution contains the source and executable code for the Toolkit (and it also contains the source code for this document). You might download this distribution if you need to modify Toolkit Java code or if you want a detailed look at how the Toolkit works.

**Upgrade considerations**

Before upgrading to a new version of DITA Open Toolkit, be sure to back up your current version so you can reapply modifications after your Toolkit upgrade. Such modifications might include:

- Specialization DTDs you added to the dtd directory and the corresponding updates you made to the catalog-dita-template.xml file
- XSLT stylesheets you have added to the xsl directory to override the standard stylesheets
- Plug-ins you have installed
- Catalogs for XML editors

**Installation considerations**

Consider the following important points before selecting an authoring tool, and before downloading and installing DITA Open Toolkit and its prerequisite tools.

**The tools you use need to work together as a set.**

This means, for example, that the DITA-aware authoring tool you are already using may not be "aware" of the version of the DTDs that come with the Toolkit. It could also mean that the version of SAXON you already have installed on your laptop doesn't work with the Toolkit. Read System requirements and supported applications on page 47 and check your current system environment before installing or upgrading.

**You may not need to install the prerequisite tools separately.**

DITA Open Toolkit 1.3.1 full package comes with all required tools except the Java JDK.

The Linux distribution Fedora Core 5 already comes with the JDK, Ant, and Xalan. The Mac OS X installation DVD also comes with some of the required components.

**You may need to move or uninstall one or more tools in your current environment before installing the Toolkit and its prerequisites.**

For example, if the version of one of the tools in your Fedora or Mac OS X package is incompatible with the version of the Toolkit you are installing, you may have to change your system environment.
Installing your DITA authoring tool

You can create DITA source files with a plain text editor, for example Microsoft Notepad for Windows. Editors at varying levels of "DITA-awareness" are also available, both free and for purchase. These editors help you create DITA documents that are well-formed and valid.

For a list of popular DITA-aware authoring tools, see Creating and managing content on page 17

Authoring tools are not part of DITA Open Toolkit or any of the prerequisite tools for the Toolkit.

Some of these DITA authoring products contain embedded copies of the DITA Open Toolkit and hence you will not need to install the Toolkit separately.

Installing the Java SDK (JDK)

Both the small and full package distributions require that the Java SDK be installed separately.

Installing the JDK on Windows

1. For the Sun version of the JDK, enter the URL http://java.sun.com/j2se/1.4.2/download.html.
2. From the Sun Developer Network page, scroll to find the heading J2SE v 1.4.2_13 SDK (that is, _13 or whatever the latest version is that works with the Toolkit).
3. Select Download J2SE SDK.
4. From the Sun Developer Network page, accept the license agreement and scroll to the heading "Windows Platform - Java(TM) 2 SDK, Standard Edition 1.4.2_13".
5. Select and download Windows installation, Multi-language.
6. Save and install the .exe file.
7. If prompted, install the JDK to C:\j2sdk1.4.2_13.
8. Set the JAVA_HOME environment variable to C:\j2sdk1.4.2_13.

Note: For the IBM version of the JDK, enter http://www.ibm.com/developerworks/java/jdk.

Installing the JDK on Linux

1. Enter the URL: http://java.sun.com/j2se/1.4.2/download.html.
2. From the Sun Developer Network page, scroll to find the heading J2SE v 1.4.2_13 SDK.
3. Select Download J2SE SDK.
4. From the Sun Developer Network page, accept the license agreement and scroll to the heading "Linux Platform - Java(TM) 2 SDK, Standard Edition 1.4.2_13".
5. Select and download self-extracting file.
6. Run and install into a Linux home directory.
7. Set the JAVA_HOME environment variable using export JAVA_HOME=${java_dir}. 
Installing the DITA Toolkit full package distribution

Installing the full package distribution

To install the full package:

- Unzip the package into the C:\ditaot directory on Windows, or into a home directory on Linux.
- On Windows, create a new shortcut for C:\ditaot\startcmd.bat to be used to run DITA builds. On Linux execute the shell script startcmd.sh before running a DITA build.

Note: You cannot run a DITA Toolkit build until you have installed the Java SDK.

Note: You may want to install other optional tools to complete your build environment.

Verifying the installation

Unzip or extract the "fullpackage" zip file to a convenient directory, such as your C: drive's root directory. The package will install a directory called dita-ot1.3.1 that contains not only the usual Toolkit materials but also all the run-time components needed to run the Toolkit in a basic evaluation mode.

Browse to this new directory and double-click on the startcmd.bat file in that directory. A new command shell window will open up, with the environment variables already set to enable the Toolkit to run within that shell.

At the command prompt (usually C:\dita-ot1.3.1 for this version), type ant samples.web -f build_demo.xml After a series of processing messages, there should be a new \out directory in the dita-ot1.3.1 directory that contains a folder with the resulting HTML output in it.

Now try the full set of transforms from a single command: ant all -f build_demo.xml This command will process every DITA example in the Toolkit into each of the supported output types. After a much longer flurry of messages stops, the \out directory should have a number of folders in it, each with several forms of deliverable produced by the Toolkit demos. If you have the Microsoft HTMLHelp Workshop or the JavaHelp toolset installed, you will even get ready-to use .chm and .javahelp output files. By comparing the outputs with the various source materials in the distribution, you can get an idea about how the processing works. See Processing (building) and publishing DITA documents on page 75 for more information on processing.

Demo targets

<table>
<thead>
<tr>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Build all output</td>
</tr>
<tr>
<td>clean</td>
<td>Delete all output</td>
</tr>
<tr>
<td>clean.demo</td>
<td>Remove the demo output</td>
</tr>
<tr>
<td>clean.demo.book</td>
<td>Remove the book demo output</td>
</tr>
<tr>
<td>clean.demo.elementref</td>
<td>Remove the Element Reference demo output</td>
</tr>
<tr>
<td>clean.demo.enote</td>
<td>Remove the eNote demo output</td>
</tr>
<tr>
<td>clean.demo.faq</td>
<td>Remove the FAQ demo output</td>
</tr>
<tr>
<td>clean.demo.langref</td>
<td>Remove the Language Reference demo output</td>
</tr>
<tr>
<td>clean.doc</td>
<td>Remove the documentation output</td>
</tr>
<tr>
<td>clean.doc.articles</td>
<td>Delete the articles directory in doc.</td>
</tr>
<tr>
<td>clean.doc.langref</td>
<td>Delete the langref directory in doc.</td>
</tr>
<tr>
<td>clean.docbook</td>
<td>Remove the docbook output</td>
</tr>
<tr>
<td>clean.samples</td>
<td>Remove the sample output</td>
</tr>
<tr>
<td>clean.samples.eclipse</td>
<td>Remove the sample Eclipse output</td>
</tr>
<tr>
<td>clean.samples.htmlhelp</td>
<td>Remove the sample HTMLHelp output</td>
</tr>
<tr>
<td>clean.samples.javahelp</td>
<td>Remove the sample JavaHelp output</td>
</tr>
<tr>
<td>clean.samples.pdf</td>
<td>Remove the sample PDF output</td>
</tr>
<tr>
<td>clean.samples.web</td>
<td>Remove the sample web output</td>
</tr>
<tr>
<td>demo</td>
<td>Build the demos</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>demo.book</td>
<td>Build the book demo</td>
</tr>
<tr>
<td>demo.elementref</td>
<td>Build the element reference demo</td>
</tr>
<tr>
<td>demo.enote</td>
<td>Build the eNote demo</td>
</tr>
<tr>
<td>demo.faq</td>
<td>Build the FAQ demo</td>
</tr>
<tr>
<td>demo.langref</td>
<td>Build the Language Reference book as a demo</td>
</tr>
<tr>
<td>demo.langref.compilehelp</td>
<td>Build the Language Reference as HTML Help (if the workshop is installed)</td>
</tr>
<tr>
<td>doc</td>
<td>Build the documentation</td>
</tr>
<tr>
<td>doc.articles.chm</td>
<td>Build the articles of dita as document.</td>
</tr>
<tr>
<td>doc.articles.pdf</td>
<td>Build the articles of dita as document.</td>
</tr>
<tr>
<td>doc.articles.web</td>
<td>Build the articles of dita as document.</td>
</tr>
<tr>
<td>doc.langref.chm</td>
<td>Build the langref document</td>
</tr>
<tr>
<td>doc.langref.pdf</td>
<td>Build the langref document</td>
</tr>
<tr>
<td>doc.langref.web</td>
<td>Build the langref document</td>
</tr>
<tr>
<td>docbook</td>
<td>Transform the samples to DocBook</td>
</tr>
<tr>
<td>prompt</td>
<td>Prompt to build anything</td>
</tr>
<tr>
<td>samples</td>
<td>Build the sample output</td>
</tr>
<tr>
<td>samples.eclipse</td>
<td>Build the samples for Eclipse</td>
</tr>
<tr>
<td>samples.htmlhelp</td>
<td>Build the samples for HTMLHelp</td>
</tr>
<tr>
<td>samples.javahelp</td>
<td>Build the samples for JavaHelp</td>
</tr>
<tr>
<td>samples.pdf</td>
<td>Build the samples as PDF</td>
</tr>
<tr>
<td>samples.troff</td>
<td>Build the samples as troff</td>
</tr>
<tr>
<td>samples.web</td>
<td>Build the samples for the web</td>
</tr>
</tbody>
</table>

If you do not specify a target for `build_demo.xml`, the default target is `prompt`.

You can also try your hand at modifying some of the sample scripts in the `ant` directory. These represent the kind of driver files that you would create for your own projects. You can easily adapt these to process your own test DITA files. Run the other ant samples using this example:

```
C:\dita-ot1.3.1>ant -f ant/sample_xhtml.xml
```

This is basically the same as running `ant samples.web -f build_demo.xml`, but intended for you to modify.

You will find the output for this exercise in the `ant` directory itself. You can add parameters to the `sample_xhtml.xml` file to change where your outputs end up, and also to modify the build process in other ways. See *Ant processing parameters* on page 82 to learn more about processing options.

### Installing the DITA Toolkit small package distribution

#### Installing the small package distribution

To install the small package:

2. Unzip the package into the `C:\ditaot` directory on Windows, or into a home directory on Linux.
3. On Windows, add `C:\ditaot\lib;C:\ditaot\lib\dost.jar;C:\ditaot\lib\resolver.jar` to your `CLASSPATH` environment variable.
4. On Linux, set up your environment variable CLASSPATH. For example:

```
export CLASSPATH=$CLASSPATH:$ditaot_dir/lib:$ditaot_dir/lib/dost.jar:$ditaot_dir/lib/resolver.jar
```

#### Installation considerations

Before installing the DITA Open Toolkit small package and its prerequisite software on Windows, check to see if any of the required tools are already installed on your system and, if so, whether the version you have is supported (see *System requirements and supported applications* on page 47). For any tools you need to install, complete the tasks below in the order shown.
Installing the small package on Windows

Before installing the DITA Open Toolkit small package and its prerequisite software on Windows, check to see if any of the required tools are already installed on your system and, if so, whether the version you have is supported (see System requirements and supported applications on page 47). For any tools you need to install, complete the tasks below in the order shown.

Installing Ant on Windows

1. Enter the URL: http://ant.apache.org/bindownload.cgi.
2. On the Apache Ant Project page, find the heading Current Release of Ant.
3. Select apache-ant-1.6.5-bin.zip [PGP] [SHA1] [MD5]
4. Click Save to unzip the apache-ant-1.6.5-bin.zip [PGP] [SHA1] [MD5] file and save it to your C:\ directory as ant.
5. Add the bin directory to your PATH environment variable.
6. Add the ANT_HOME environment variable set to C:\ant.
7. Add the ANT_OPTS environment variable set to -Xmx256M.

Installing SAXON on Windows

1. Enter the URL: http://saxon.sourceforge.net/.
2. From SAXON: The XSLT and XQuery Processor page, scroll to find the heading Saxon 6.5.5.
3. Select Download.
   The SourceForge.net page opens with a list of download options.
4. Select any of the images to start the download.
5. Click Save to unzip the saxon6.5.5.zip file and save it to the C:\ directory as saxon.
6. Add C:saxon\saxon.jar to the CLASSPATH environment variable.
7. Set up ANT_OPTS. For example: set ANT_OPTS=%ANT_OPTS% -Djavax.xml.transform.TransformerFactory=com.icl.saxon.TransformerFactoryImpl

Installing Xalan on Windows

1. Enter the URL: http://archive.apache.org/dist/xml/xalan-j/.
2. From Xalan: The Xalan Processor page, scroll to find the heading xalan-j_2_7_0-bin.zip. Click to download.
3. Save and unzip the xalan-j_2_7_0-bin.zip file to C:\ directory as xalan.
4. Add C:xalan\bin to the CLASSPATH environment variable.

Setting environment variables on Windows

1. From the Start Menu, select Start > Settings > Control Panel.
2. Double-click System to open the System Properties window.
3. On the Advanced tab, select environmental variables .
4. Modify each environmental or system variable.

Set the PATH environment variable to include the directory where you installed the Ant bin directory:

1. Find the PATH environment variable in the list. If PATH is not listed, click on New under the System variables section.
2. Type %ANT_HOME%\bin;%JAVA_HOME%\bin;
Important: If there are other variables listed, create a new variable separated by a semicolon. Ensure there are no spaces before or after the semicolon.

Set the \texttt{ANT\_HOME} environment variable to the directory where you installed Ant:

1. Click on \texttt{New} under the System variables section.
2. Type \texttt{ANT\_HOME} in the variable name field.
3. Type \texttt{C:\ant} in the variable value field.

Set the \texttt{ANT\_OPTS} environment variable to the directory where you installed Ant:

1. Click \texttt{New} under the System variables section.
2. Type \texttt{ANT\_OPTS} in the variable name field.
3. Type \texttt{-Xmx256M} in the variable value field.

Set the \texttt{JAVA\_HOME} environment variable to the directory where you installed the J2SE SDK application:

1. Click on \texttt{New} under the System variables section.
2. Type \texttt{JAVA\_HOME} in the variable name field.
3. Type \texttt{C:\j2sdk1.4.2_13} in the variable value field.

Set the \texttt{JHHOME} environment variable to the directory where you installed the JavaHelp application:

1. Click on \texttt{New} under the System variables section.
2. Type \texttt{JHHOME} in the variable name field.
3. Type \texttt{C:\javahelp\jh2.0} in the variable value field.

Create or append to the \texttt{CLASSPATH} environment variable for DITA-OT:

1. Find the \texttt{CLASSPATH} environment variable in the list. If \texttt{CLASSPATH} is not listed, click \texttt{New} under the System variables section.
2. Type \texttt{C:\ditaot\lib;C:\ditaot\lib\dost.jar;C:\ditaot\lib\resolver.jar}

   Important: If there are other variables listed, create a new variable separated from the others by a semicolon. Ensure there are no spaces before or after the semicolon.

Create or append to the \texttt{CLASSPATH} environment variable for the Apache FOP application:

1. Find the \texttt{CLASSPATH} environment variable in the list. If \texttt{CLASSPATH} is not listed, click \texttt{New} under the System variables section.
2. Type \texttt{C:\fop-0.20.5\build\fop.jar;C:\fop-0.20.5\lib\batik.jar;C:\fop-0.20.5\lib\avalon-framework-cvs-20020806.jar}

   Important: If there are other variables listed, create a new variable separated from the others by a semicolon. Ensure there are no spaces before or after the semicolon.

(If you use SAXON) Create or append to environment variables for SAXON:

1. Find the \texttt{CLASSPATH} environment variable in the list. If \texttt{CLASSPATH} is not listed, click on \texttt{New} under the System variables section.
2. Type \texttt{C:\saxon\saxon.jar}

   Important: If there are other variables listed, create a new variable separated by a semicolon. Ensure there are no spaces before or after the semicolon.

3. Set up \texttt{ANT\_OPTS}. For example:
   
   set \texttt{ANT\_OPTS}=%\texttt{ANT\_OPTS}% -Djavax.xml.transform.TransformerFactory=com.icl.saxon.TransformerFactoryImpl

(If you use Xalan) Set the \texttt{CLASSPATH} environment variable for Xalan:
1. Find the CLASSPATH environment variable in the list. If CLASSPATH is not listed, click on New under the System variables section.

2. Type C:\xalan\bin

   Important: If there are other variables listed, create a new variable separated by a semicolon. Ensure there are no spaces before or after the semicolon.

---

**Verifying the installation on Windows**

1. From the toolbar, click Start > Run.
2. In the Open field, type cmd.
3. Change the command prompt according to the following table.

<table>
<thead>
<tr>
<th>If this prompt displays,</th>
<th>type the following command</th>
</tr>
</thead>
<tbody>
<tr>
<td>D:\</td>
<td>C:\</td>
</tr>
<tr>
<td>H:\</td>
<td>C:\</td>
</tr>
<tr>
<td>C:\My Documents...</td>
<td>cd \</td>
</tr>
</tbody>
</table>

4. At the prompt, type cd ditaot
   The command prompt changes to C:\ditaot
5. Type ant -f build_demo.xml all and press Enter to process the DITA files in the demo, doc, docbook, and samples directories. This procedure also verifies the Toolkit installation.
   The testing process completes in 3-10 minutes depending on the speed of your machine. When testing completes, the confirmation message BUILD SUCCESSFUL displays.

Be sure the directories and files in your ditaot are as described in *Directories and files in the ditaot directory* on page 58.

---

**Installing the small package on Linux**

Before installing DITA Open Toolkit small package on Linux, check to see if any of the required tools are already installed on your system and, if so, whether the version you have is supported (see *System requirements and supported applications* on page 47)

**Note:** As an example, if you are using Fedora Core 5 Linux, software installation is done using the Package Manager. From this application, if you install the Java Development package within the Development group, you will install:

- Ant 1.6.5
- Java SDK 1.4.2
- Xalan-J 2.6.0

For any tools you do need to install, complete the tasks below in the order shown.

**Installing Ant on Linux**

1. Enter the URL: http://ant.apache.org/bindownload.cgi.
2. On the Apache Ant Project page, find the heading Current Release of Ant.
3. Select apache-ant-1.6.5-bin.tar.gz [PGP] [SHA1] [MD5].
4. Save and extract the package file into a Linux home directory.
5. Set the ANT_OPTS environment variable: export ANT_OPTS="-Xmx256M"
6. Set the `ANT_HOME` environment variable to the directory where you installed Ant: `export ANT_HOME=${ant_dir}`

7. Set the `PATH` environment variable to include the directory where you installed the Ant bin directory: `export PATH=${ANT_HOME}/bin:${JAVA_HOME}/bin:${PATH}`

---

**Installing Saxon on Linux**

1. Enter the URL: [http://saxon.sourceforge.net/](http://saxon.sourceforge.net/)
2. From Saxon: The XSLT and XQuery Processor page, scroll to find the heading **Saxon 6.5.5**
3. Select **Download (3265 Kbytes)**. The SourceForge.net page opens with a list of download options.
4. Select any of the images to start the download.

   If Saxon does not appear to be downloading, wait a few minutes before selecting another image.

   You may have to select more than one image until you find one that works.

5. Download and unzip the Saxon6.5.5.zip file and save it to a Linux home directory.
6. Add Saxon to your **CLASSPATH** environment variable: `export CLASSPATH=${CLASSPATH}:${saxon_dir}/saxon.jar`

---

**Installing Xalan on Linux**

2. From Saxon: The Xalan Processor page, scroll to find the heading **xalan-j_2_7_0-bin.tar.gz**. Click to download.
3. Save and unzip the xalan-j_2_7_0-bin.tar.gz file to a Linux home directory.
4. Add Xalan to the **CLASSPATH** environment variable: `export CLASSPATH=${CLASSPATH}:${xalan_dir}/bin`

---

**Setting environment variables on Linux**

1. Type in the Linux Console.
2. Modify each environmental or system variable.
   
   **Set the PATH environment variable** to include the directory where you installed the Ant bin directory: `export PATH=${ANT_HOME}/bin:${JAVA_HOME}/bin:${PATH}`
   
   **Set the ANT_HOME environment variable** to the directory where you installed Ant: `export ANT_HOME=${ant_dir}`
   
   **Set the ANT_OPTS environment variable** to the directory where you installed Ant: `export ANT_OPTS="-Xmx256M"`
   
   **Set the JAVA_HOME environment variable** to the directory where you installed the J2SE SDK application: `export JAVA_HOME=${java_dir}`
   
   **Set the JHHOME environment variable** to the directory where you installed the JavaHelp application: `export JHHOME=${javahelp_dir}`
   
   **Set the CLASSPATH environment variable** for DITA-OT: Set up your environment variable CLASSPATH to include the dost.jar. For example: `export CLASSPATH=${ditaot_dir}/lib:${ditaot_dir}/lib/dost.jar:${ditaot_dir}/lib/resolver.jar`
Set the CLASSPATH environment variable for the Apache FOP application: Set up your environment variable CLASSPATH to include the fop.jar, batik.jar and avalon.jar files in the FOP directory. For example:

```bash
```

(If you use SAXON) Set environment variables for SAXON:

1. Set up CLASSPATH to include the saxon.jar file. For example:
   ```bash
   export CLASSPATH=${CLASSPATH}:${saxon_dir}/saxon.jar
   ```

2. Set up ANT_OPTS. For example:
   ```bash
   export ANT_OPTS=${ANT_OPTS}
   -Djavax.xml.transform.TransformerFactory=com.icl.saxon.TransformerFactoryImpl
   ```

(If you use Xalan) Set environment variables for Xalan: Set up CLASSPATH to include the xalan.jar file and the xercesImpl.jar file. For example:

```bash
export CLASSPATH=${CLASSPATH}:${xalan_dir}/bin
```

**Verifying the installation on Linux**

1. In the console, type `cd {ditaot_dir}`
2. Type `ant -f build.demo.xml all` and press Enter to begin to process the DITA files in the demo, doc, docbook, and samples directories. This procedure also verifies the Toolkit installation. The testing process completes in 3-10 minutes depending on the speed of your machine. When testing completes, the confirmation message BUILD SUCCESSFUL displays.

Be sure the directories and files in your ditaot are as described in *Directories and files in the ditaot directory* on page 58.

**Installing on Mac OS**

**MacOS installation considerations**

Before installing DITA Open Toolkit and its prerequisite software on Mac OS X, check to see if any of the tools are already installed on your system and, if so, whether the version you have is supported (see *System requirements and supported applications* on page 47). Java is a core component of Mac OS X. Newer versions of Mac OS X include the full version of the Java JDK 1.4.2 by default. This version of the JDK includes Ant and the Xalan-J XSLT processor, as well. Other tools you want to install may be included on the Mac OS X Developer's Tools on the product DVD.

**Installing on MacOS**

To install the Toolkit, extract the zip file to your HOME directory, then edit your login rc file to include the Toolkit in your CLASSPATH.

**Directories and files in the ditaot directory**

When you have installed DITA Open Toolkit, the following directories and subdirectories should be in your root ditaot directory.
<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>root (ditaot)</td>
<td>System-level Ant scripts and other system files (for example, conductor.xml, build.xml, and integrator.xml). System-level scripts handle DITA source file processing and transformation into published output. They are an integral part of DITA Open Toolkit and should never be modified by users. For more information, see About Ant scripts on page 75.</td>
</tr>
<tr>
<td>ant</td>
<td>Ant scripts that can be used as-is to process sample files or modified for your use.</td>
</tr>
<tr>
<td>css</td>
<td>Sample CSS (cascading style sheet) files.</td>
</tr>
<tr>
<td>demo</td>
<td>Specializations, plug-ins, and validators that demonstrate extensions to the base DiTA language. Includes:</td>
</tr>
<tr>
<td></td>
<td>• book: bookmark specialization</td>
</tr>
<tr>
<td></td>
<td>• elementref: simple element reference description markup</td>
</tr>
<tr>
<td></td>
<td>• enote: data object specialization</td>
</tr>
<tr>
<td></td>
<td>• faq: faq (frequently asked questions) specialization</td>
</tr>
<tr>
<td></td>
<td>• fo: plug-in files to produce PDF output</td>
</tr>
<tr>
<td></td>
<td>• FrameMaker_adapter: plug-in to produce a structured FrameMaker (7.0+) input file</td>
</tr>
<tr>
<td></td>
<td>• h2d: plug-in to convert XHTML to DITA topics</td>
</tr>
<tr>
<td></td>
<td>• java: validators for DITA schemas</td>
</tr>
<tr>
<td></td>
<td>Many of these directories have README files that provide information about how to use the specializations.</td>
</tr>
<tr>
<td>doc</td>
<td>DITA documentation: language reference and application notes.</td>
</tr>
<tr>
<td>dtd</td>
<td>Core DiTA definitions in XML DTD format.</td>
</tr>
<tr>
<td>lib</td>
<td>Contains dost.jar, the executable .jar file and other .jar files it depends on.</td>
</tr>
<tr>
<td>plugins</td>
<td>DITA Open Toolkit plug-ins.</td>
</tr>
<tr>
<td>resource</td>
<td>Miscellaneous resource files, including the default (common) CSS files and error messages.</td>
</tr>
<tr>
<td>samples</td>
<td>Sample DITA source files and Ant scripts.</td>
</tr>
<tr>
<td>schema</td>
<td>Core DITA definitions in XML Schema format.</td>
</tr>
<tr>
<td>tools</td>
<td>Ant 1.6.5 processor.</td>
</tr>
<tr>
<td>xsl</td>
<td>Core and process-specific stylesheets. Includes:</td>
</tr>
<tr>
<td></td>
<td>• common: stylesheets that can be used by any process (for example, internationalization)</td>
</tr>
<tr>
<td></td>
<td>• docbook: stylesheets used in converting DITA source content into DocBook source</td>
</tr>
<tr>
<td></td>
<td>• preprocess: code for conditional, conref, and link resolution</td>
</tr>
<tr>
<td></td>
<td>• troff: stylesheets used in converting DITA source content into troff source</td>
</tr>
<tr>
<td></td>
<td>• xslfo: code to support the processing of Formatting Objects (FO) output</td>
</tr>
<tr>
<td></td>
<td>• xslhtml: code to support XHTML processing</td>
</tr>
<tr>
<td></td>
<td>• xsrltf: code to support RTF processing</td>
</tr>
</tbody>
</table>

**Installing the optional tools**

Sections in this topic:

- Installing the HTML Help compiler on Windows on page 60
- Installing FOP on Windows on page 60
Depending on the kind of output you expect to produce, you may want to install the following tools.

(If you plan to publish HTML Help) The Microsoft HTML Help processor
(If you plan to publish JavaHelp files) The Sun JavaHelp processor
(If you plan to publish PDF files) The Apache FOP processor or the RenderX XEP processor

The default processing script uses Apache FOP for converting FO files into PDF. With some modification of the build scripts, you can use the RenderX XEP processor, instead. FOP is free. XEP is free for personal use.

(If you plan to publish Eclipse content) The IBM Eclipse content processor
(If you plan to publish Eclipse help) The IBM Eclipse help processor

Installing the HTML Help compiler on Windows

2. From the MSDN page, scroll to find the heading HTML Help Workshop.
3. Select Download Htmlhelp.exe.
4. Click Run and navigate to a C:\ directory as C:\Program Files\HTML Help Workshop.
5. Follow the steps in the HTML Help install guide wizard to complete the installation.

If you install the Help compiler to a drive other than the C: drive, you may need to customize the <property> value for hhc.dir in some of the build.xml scripts in the Toolkit root directory. The Toolkit assumes the compiler is installed on your C: drive.

Installing FOP on Windows

1. Enter the URL: http://apache.tradebit.com/pub/xml/fop/
2. From the FOP page, in the Name column, select "fop-0.20.5-bin.zip".
3. Click Save to unzip the fop-0.20.5-bin.zip file and save it to the C: \ directory as fop-0.20.5.
4. Add to the CLASSPATH for the following jar files:
   C:\fop-0.20.5\build\fop.jar;C:\fop-0.20.5\lib\batik.jar;C:\fop-0.20.5\lib \avalon-framework-cvs-20020806.jar

Installing the JavaHelp processor on Windows

1. Enter the URL: http://java.sun.com/products/javahelp/download_binary.html
2. From the Sun Developer Network page, scroll to find the heading JavaHelp 2.0_02 (Zip).
3. Select Download.
4. From the Sun Developer Network page, accept the license agreement and scroll to the heading "Platform - JavaHelp API 2.0_02 FCS"
5. Select javahelp-2_0_02.zip, 6.49 MB.
   The File Download window opens.
6. Click **Save** to unzip the javahelp-2.0_02.zip file and save it to the C:\ directory as javahelp.
7. Set the **JHHOME** environment variable to C:\javahelp\jh2.0

### Installing FOP on Linux

2. From the FOP page, in the Name column, select "fop-0.20.5-bin.tar.gz".
3. Save and extract the package file into a Linux home directory.
4. Set the **CLASSPATH** environment variable for the following jar files:
   ```
   build/fop.jar
   lib/batik.jar
   lib/avalon-framework-cvs-20020806.jar
   ```
   ```
   export CLASSPATH=${fop_dir}/build/fop.jar:${fop_dir}/lib/batik.jar:
   ${fop_dir}/lib/avalon-framework-cvs-20020806.jar:${CLASSPATH}
   ```

### Installing the JavaHelp processor on Linux

2. From the Sun Developer Network page, scroll to find the heading **JavaHelp 2.0_02 (Zip)**.
3. Select **Download**.
4. From the Sun Developer Network page, accept the license agreement and scroll to the heading "Platform - JavaHelp API 2.0_02 FCS".
5. Select **javahelp-2.0_02.zip**, 6.49 MB.
   *The File Download window opens.*
6. Click **Save** to unzip the javahelp-2.0_02.zip file and save it to a Linux home directory.
7. Add the **JHHOME** environment variable: `export JHHOME=${javahelp_dir}`

### DITAblogs (installing and upgrading)

**Authoring tools**

We recommend using a free, free-trial, or inexpensive DITA-aware authoring tool to create your first demo DITA documents; the experience you gain in a simple environment will help you make the intelligent purchase of a more sophisticated editor later on.

We deliberately chose to use an authoring tool that was free (at the time), since we thought many of our readers would be learning about the DITA Open Toolkit as part of an educational or pilot project where cost might be an issue. We also wanted to edit in "raw" XML ourselves so we would understand that technology well. A couple of months into the project we began exploring more advanced DITA-aware authoring tools that would provide us with additional functionality.

Most of our topics were written using Altova XMLSpy, which is "DITA-aware" (that is, it verifies that DITA source files are well-formed and valid). We had problems at first getting XMLSpy to use a catalog for the DITA DTDs, but that problem was solved (for more information, see **Configuring your authoring tool to use the Toolkit catalog** on page 65).
Because our authoring tool didn't have "plausible preview," we did frequent builds to check the output.
DITA Open Toolkit plug-ins

This section contains information about DITA Open Toolkit plug-ins.

Sections in this topic:

About DITA Open Toolkit plug-ins on page 63
Installing the plug-ins on page 64
DITAblogs (plug-ins) on page 64
For more information (plug-ins) on page 64

About DITA Open Toolkit plug-ins

You can extend or enhance the product capabilities provided by DITA Open Toolkit by installing Toolkit plug-ins. Once installed, a plug-in becomes part of the Toolkit environment and can be used to add new specializations or to define new targets for output processing.

Download sites for Toolkit plug-ins

You can download DITA Open Toolkit plug-ins from the following websites:

- Yahoo! dita-users group site: http://groups.yahoo.com/group/dita-users/files/Demos/
- SourceForge DITA Open Toolkit site: http://sourceforge.net/project/showfiles.php?group_id=132728

Plug-ins you can install

Key plug-ins are listed in the following table.

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiref0.8</td>
<td><strong>The API reference specialization</strong> provides a general-purpose basis for documenting callable programming libraries.</td>
</tr>
<tr>
<td>docbook2dita 1.0</td>
<td><strong>DocBook to DITA Transform</strong> This plug-in provides a transform from DocBook to DITA and some proof-of-concept examples of interoperability. In order to use this plug-in you need to install both DocBook 4.4 DTD and DocBook-xsl 1.69.1.</td>
</tr>
<tr>
<td>FrameMaker Adapter 0.8</td>
<td><strong>The FrameMaker Adapter</strong> (sometimes called the Mekon FrameMaker Adapter or Moldflow FrameMaker Adapter) adds the fmxml output target that allows you to take an existing DITA project and produce a file that can be used as input to FrameMaker 7.x. You can then apply an unstructured FrameMaker template to this content and publish it using FrameMaker functionality. You cannot, however, use FrameMaker functionality to make further changes to the DITA project. The adapter was not designed to be used for authoring DITA content. <strong>Note:</strong> This is not the Adobe FrameMaker Application Pack for DITA.</td>
</tr>
<tr>
<td>Idiom FO 1.1</td>
<td><strong>The Idiom FO plug-in</strong> uses the RenderX XEP processor (in place of the base-level Apache FOP processor) to enhance Toolkit support for PDF processing. The target name is pdf2 (the base-level target name is pdf.)</td>
</tr>
<tr>
<td>javaapiref0.8</td>
<td><strong>The Java API reference</strong> specialization provides a basis for documenting Java class libraries.</td>
</tr>
<tr>
<td>Music 1.0</td>
<td><strong>The Music specialization</strong> is a set of sample files that demonstrates how to create a new specialization plug-in. It should be placed in the demo directory inside the DITA Open Toolkit.</td>
</tr>
</tbody>
</table>
## The Thesaurus (also called Taxonomy) specialization

The Thesaurus (also called Taxonomy) specialization defines formal subjects and the relationships between them, so you can classify your content. This specialization conforms closely to the SKOS (Simple Knowledge Organization System) standard and generates RDF.

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesaurus 0.8.1</td>
<td>The Thesaurus (also called Taxonomy) specialization defines formal subjects and the relationships between them, so you can classify your content. This specialization conforms closely to the SKOS (Simple Knowledge Organization System) standard and generates RDF.</td>
</tr>
</tbody>
</table>

## Installing the plug-ins

Follow these general steps to install a plug-in to work with DITA Open Toolkit.

1. Download the plug-in .zip file from one of the websites described in About DITA Open Toolkit plug-ins on page 63.
2. Unzip the .zip file and read the installation documentation it contains.
3. Copy the plug-in files to a subdirectory in either the demo or plugins subdirectory in the ditaot root directory.
4. Run the command `ant -f integrator.xml` to add the plug-in to the Toolkit.

### Installing the Idiom FO plug-in

To install the Idiom FO plug-in, see the installation instructions provided by Idiom.

### DITAblogs (plug-ins)

We used the Idiom FO plug-in to produce the PDF2 output for this document.

## For more information (plug-ins)

You can download DITA Open Toolkit plug-ins from the following websites:

Setting up your working environment

This section contains information on how to configure your DITA editor and set up your source file directory.

Sections in this topic:

- Configuring your authoring tool to use the Toolkit catalog on page 65
- Configuring your authoring tool to use DTD or schema URLs on page 69
- Configuring the Eclipse XMLBuddy editor for DITA on page 70
- Setting up your source and output file directories on page 71
- DITAblogs (setting up) on page 73
- For more information (setting up) on page 74

Configuring your authoring tool to use the Toolkit catalog

"DITA-aware" authoring tools check source files for well-formedness and validate them against DTDs. However, you may need to configure your editor to enable it to do these tasks correctly. Some editors have a built-in copy of the DTDs; others require that you provide a catalog that can be used to look up DTD definitions specified in your source file DOCTYPE declaration. Some versions of Altova XMLSpy may require a catalog.

XMLSpy performs a catalog lookup on the PUBLIC identifier in the DOCTYPE declaration in DITA source files. If it does not find a match in its catalog, XMLSpy tries to open the disk file specified as the last URI in the DOCTYPE declaration. If the URI does not point to the relevant DTD, XMLSpy is unable to validate the DITA source file.

By modifying XMLSpy's CustomCatalog.xml file, you can move your source files to a new location without having to worry about incorrect URIs in your DITA source files.

Follow these steps to modify C:\Program Files\Altova\XMLSpy2006\CustomCatalog.xml.

1. Save the CustomCatalog.xml stub file under a new name (for example, CustomCatalogOLD.xml).
2. Open the CustomCatalog.xml stub file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v5 beta 1 U (http://www.xmlspy.com) by Vladislav Gavrielov (Altova) -->
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd"/>
</catalog>
```

3. Keeping the CustomCatalog.xml stub file open, also open catalog-dita.xml in the Toolkit home directory.

```xml
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog" prefer="public">
    <group xml:base="dtd">
        <!-- DITA concepts -->
        <public publicId="-//IBM//DTD DITA Concept//EN" url="dita132/concept.dtd"></public>
        <public publicId="-//IBM//ELEMENTS DITA Concept//EN" url="dita132/concept.mod"></public>
        <!-- DITA databases -->
        <public publicId="-//IBM//DTD DITA Composite//EN" url="dita132/ditabase.dtd"></public>
        <public publicId="-//IBM//DTD DITA Reference//EN" url="dita132/reference.dtd"></public>
        <public publicId="-//IBM//ELEMENTS DITA Reference//EN" url="dita132/reference.mod"></public>
    </group>
</catalog>
```

...
4. In catalog-dita.xml, change the relative path names to absolute path names.

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <!-- XMLSPY custom XML catalog for DITA DTDs -->
  <public publicId="-//IBM//DTD DITA Concept//EN" uri="C:/ditaot/dtd/dita132/concept.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Concept//EN" uri="C:/ditaot/dtd/dita132/concept.mod"></public>
  <public publicId="-//IBM//DTD DITA Composite//EN" uri="C:/ditaot/dtd/dita132/ditabase.dtd"></public>
  <public publicId="-//IBM//DTD DITA Reference//EN" uri="C:/ditaot/dtd/dita132/reference.dtd"></public>
  . . .
</catalog>
```

5. Modify the CustomCatalog.xml file and paste the new information into it.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v5 beta 1 U (http://www.xmlspy.com) by Vladislav Gavrielov (Altova) -->
  <!-- XMLSPY custom XML catalog for DITA DTDs -->
  <public publicId="-//IBM//DTD DITA Concept//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/concept.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Concept//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/concept.mod"></public>
  <public publicId="-//IBM//DTD DITA Composite//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/ditabase.dtd"></public>
  <public publicId="-//IBM//DTD DITA Reference//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/reference.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Reference//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/reference.mod"></public>
  . . .
</catalog>
```

6. Save the CustomCatalog.xml file.

The following is a complete sample CustomCatalog.xml created using this procedure.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v5 beta 1 U (http://www.xmlspy.com) by Vladislav Gavrielov (Altova) -->
  <!-- XMLSPY custom XML catalog for DITA 1.1 DTDs -->
  <public publicId="-//IBM//DTD DITA Concept//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/concept.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Concept//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/concept.mod"></public>
  <public publicId="-//IBM//DTD DITA Composite//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/ditabase.dtd"></public>
  <public publicId="-//IBM//DTD DITA Reference//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/reference.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Reference//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/reference.mod"></public>
  <public publicId="-//IBM//DTD DITA Task//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/task.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Task//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/task.mod"></public>
  <public publicId="-//IBM//DTD DITA Topic//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/topic.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Topic//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/topic.mod"></public>
  <public publicId="-//IBM//ENTITIES DITA Topic Class//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/topic_class.ent"></public>
  <public publicId="-//IBM//ENTITIES DITA Topic Definitions//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/topic_defn.ent"></public>
  <public publicId="-//IBM//DTD DITA Map//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/map.dtd"></public>
  <public publicId="-//IBM//ELEMENTS DITA Map//EN" uri="C:/DITA-OT1.3.1-full/dtd/dita132/map.mod"></public>
```

66 | DITA Open Toolkit | Setting up your working environment
For more detailed information about creating an XMLSpy catalog for DITA DTDs, see http://www.altova.com/manual2006/xmlspy/spypprofessional/index.html?validate.htm.

Configuring your authoring tool to use DTD or schema URLs

Some editors and XML tools require online URL access to the DITA DTD or schema definitions in order to do validation. You can code the DOCTYPE statement in a DITA source file to point to URL DTD definitions. For example, here is the beginning of a new DITA concept in the XMLMind editor using a concept DTD URL reference:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" 
"http://docs.oasis-open.org/dita/v1.0.1/dtd/concept.dtd">
<concept id="concept1">
<title>This is a title.</title>
</concept>
</conbody>
<p/>
</conbody>
</concept>
```

Here is the complete list of the V1.0.1 DITA DTD and schema URLs:

- http://docs.oasis-open.org/dita/v1.0.1/dtd/concept.dtd
- http://docs.oasis-open.org/dita/v1.0.1/dtd/task.dtd
- http://docs.oasis-open.org/dita/v1.0.1/dtd/reference.dtd
- http://docs.oasis-open.org/dita/v1.0.1/dtd/ditabase.dtd
- http://docs.oasis-open.org/dita/v1.0.1/dtd/map.dtd
- http://docs.oasis-open.org/dita/v1.0.1/schema/concept.xsd
- http://docs.oasis-open.org/dita/v1.0.1/schema/task.xsd
- http://docs.oasis-open.org/dita/v1.0.1/schema/reference.xsd
- http://docs.oasis-open.org/dita/v1.0.1/schema/ditabase.xsd
- http://docs.oasis-open.org/dita/v1.0.1/schema/map.xsd

The Committee Draft V1.1 DITA DTDs and Schemas list is:

- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/bookmap.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/concept.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/ditabase.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/glossary.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/map.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/reference.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/task.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/dtd/topic.dtd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/bookmap.xsd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/concept.xsd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/ditabase.xsd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/glossary.xsd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/map.xsd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/reference.xsd
- http://docs.oasis-open.org/dita/v1.1/CD01/schema/task.xsd
Configuring the Eclipse XMLBuddy editor for DITA

Before you can configure XMLBuddy, you have to install it in Eclipse. You can download both the free and product versions of XMLBuddy from www.xmlbuddy.com. To install it in Eclipse, you need to copy the XMLBuddy product directory (for example, com.objfac.xmleditor_2.0_72) to the plugins directory of your Eclipse distribution.

XMLBuddy is an editor that can be installed in Eclipse that will allow you to create and edit DITA topics and maps. XMLBuddy has the capability to author, validate against the DITA DTDs, and check DITA topics and maps for well-formedness.

Once you have the XMLBuddy plug-in installed, it can be used to edit files with filetypes .xml. You need to follow the next steps so that XMLBuddy can be used to create and edit DITA topics and maps.

1. To add file associations to Eclipse so that XMLBuddy can be used for files of types .dita and .ditamap, select Windows > Preferences > General > Editors > File Associations > Add to add both the .dita and .ditamap content types. Also add XMLBuddy as an associated editor for both the new content types. The File Associations dialog will look like this:

![File Associations dialog](image)

2. To enable XMLBuddy to do code assist for DITA topics and maps, and to validate these files against the DITA DTDs, you need to define the location of the DTDs. You do this by selecting Windows > Preferences > XMLBuddy > XML > Validation > DTD > New.

   Here is example of the dialog used to add a new DTD definition to XMLBuddy:
3. Make entries for each of the DITA topic and map types by filling in the DTD information:

After you have completed these steps, you can get help on using XMLBuddy by selecting Help > Help Contents > XMLBuddy User Guide.

Setting up your source and output file directories

In general, it is a good idea to store the DITA files you create separately from DITA Open Toolkit, because:

- It is easier to create, back up, and migrate DITA source files if they are all together in a separate master directory.
- It is easier to migrate to a new version of the Toolkit if you don't have to separate out and migrate your source files at the same time.
- You are less likely to accidentally change or erase Toolkit files if they are not mixed in with the source files you work with every day.

Note: The most likely reasons you might have to modify files in the ditaot directory are (1) to create a specialization and (2) in processing reuse, to customize the output through XSLT changes.
A few simple entries in your Ant build scripts allow this separation.

Follow these steps to set up a directory environment and copy to the source files the garage and grocery shopping sample files that come with DITA Open Toolkit.

1. Create four new directories in your C: root directory (on Windows) or /home/userid (on Linux).

   In this and other examples in this document, we assume the directories are: C:/DITASAMPLE_GARAGE_SOURCE, C:/DITASAMPLE_GARAGE_OUTPUT, C:/DITASAMPLE_GROCERY_SOURCE, and C:/DITASAMPLE_GROCERY_OUTPUT. We recommend building frequently, and it is easier to find and check the output files if they are in close proximity to the source files in your directory structure.

   The samples files are in http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html.

2. Copy the garage sample files to the DITASAMPLE_GARAGE_SOURCE directory.

3. Copy the grocery shopping sample files to the DITASAMPLE_GROCERY_SOURCE directory.

Your directory structures should look like the following figures:

```
    DITASAMPLE_GARAGE_OUTPUT
    DITASAMPLE_GARAGE_SOURCE
        ant_scripts
        concepts
        ditaval_files
        images
        tasks
```

```
    DITASAMPLE_GROCERY_OUTPUT
    DITASAMPLE_GROCERY_SOURCE
        completed
            ant_scripts
            concepts
            reference
            tasks
            topics
        template
            concepts
            reference
            tasks
            topics
        working
            ant_scripts
            concepts
            reference
            tasks
            topics
```

For information about setting up Ant scripts to find your DITA source files in this new directory structure, see Processing (building) and publishing DITA documents on page 75.

For information about creating demo files of your own, see Creating DITA topics on page 131 and Creating DITA maps on page 141.
How we set up our working environment

We structured the DITA Open Toolkit User Guide using the bookmark specialization. A single master map in the root directory contains the "chapters," and the map and files for each chapter are in separate subdirectories. Supporting files for the project (including Ant scripts, CSS files, and ditaval files) are also in separate subdirectories of the root directory.

At first we stored the root directory for the source files within the Toolkit directory structure, but when release 1.2.2 made it possible to detach source files from Toolkit files, we moved our entire source project outside the ditaot structure. Similarly we created a separate directory for the build output files. We found it much easier to back up our source files, which we do as often as twice a day. Moving to a new version of the Toolkit becomes easier, as well.

Our directory structure looks like this:

```
- DITA-OT1.3.1-full
- DITAOTUG_OUTPUT
- DITAOTUG_SOURCE
  - accessing
  - ant_scripts
  - build_files
  - core_vocabulary
  - css_files
  - customizing
  - distributing
  - ditaval_files
  - evaluating
  - faqs
  - gettinginformation
  - gettingstarted
  - images
  - installing
  - introduction
  - linking
  - localizing
  - managing
  - maps
  - migratingcontent
  - plugins
  - processing
  - project
  - relationship_tables
  - release_current
  - release_history
  - reusing
  - samples
  - settingup
  - specializing
  - topics
  - troubleshooting
```
## For more information (setting up)

<table>
<thead>
<tr>
<th>Name, description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support information for <strong>Altova XMLSpy</strong></td>
<td><a href="http://www.altova.com/support_center.html">http://www.altova.com/support_center.html</a></td>
</tr>
<tr>
<td>General information about configuring and using <strong>XML catalogs</strong></td>
<td><a href="http://forrest.apache.org/docs_0_70/catalog.html">http://forrest.apache.org/docs_0_70/catalog.html</a></td>
</tr>
</tbody>
</table>
Processing (building) and publishing DITA documents

This section contains information on how to process (build) and publish DITA documents.

Sections in this topic:

- Processing overview on page 80
- About Ant on page 75
- About Ant scripts on page 75
- Ant processing parameters on page 82
- About the garage sample on page 86
- Processing to the key Toolkit output targets on page 87
- Processing to other Toolkit output targets on page 98
- DITAblogs (processing) on page 107
- For more information (processing) on page 112

About Ant

Ant is a Java-based, open source tool provided by the Apache Foundation to automatically implement a sequence of build actions defined in an Ant build script. The Ant functionality is similar to the more well-known UNIX make and Windows nmake build tools; however, instead of using shell-based commands, like make, Ant uses Java classes.

The configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular task interface. Ant can be used for both software and document builds.

DITA Open Toolkit provides Java code and a set of XSLT transform scripts for producing different types of output, for example XHTML, Eclipse help, JavaHelp, and PDF. Ant build scripts build DITA output by controlling the execution of the DITA Open Toolkit Java code and the XSLT transform scripts.

Ant must be installed in your DITA processing environment for DITA Open Toolkit to function.

About Ant scripts

An Ant script is an XML build file, containing a single project and a single or multiple targets, each of which consists of a group of tasks that you want Ant to perform. A task is an XML element that Ant can execute to produce a result. Ant comes with a large number of built-in tasks; you can also add tasks of your own.

DITA Open Toolkit makes use of two kinds of Ant scripts:

- **System scripts**: System-level scripts handle DITA source file processing and transformation into published output. They are an integral part of DITA Open Toolkit and should never be modified by users. The files are located in the ditatoot root directory.

- **User scripts**: User-level processing scripts are created and modified by users. They provide to the system scripts (which do the actual processing) information about the names and locations of the DITA source files, where to put the processed target files, and values for specific processing parameters. DITA Open Toolkit contains a number of sample user-level processing files that you can view to gain understanding of the build process, and modify for your own use.
Main system scripts in DITA Open Toolkit

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build.xml</td>
<td>Initializes the Toolkit and builds various DITA targets.</td>
</tr>
<tr>
<td>build_demo.xml</td>
<td>Builds the Toolkit demos.</td>
</tr>
<tr>
<td>buildPackage.xml</td>
<td>Build source and binary packages for DITA Open Toolkit.</td>
</tr>
<tr>
<td>catalog-dita_template.xml and</td>
<td>Contains information that directs the Toolkit to the names and locations</td>
</tr>
<tr>
<td>catalog-dita.xml</td>
<td>of the DTD files. The template file creates the non-template file</td>
</tr>
<tr>
<td></td>
<td>dynamically during every build.</td>
</tr>
<tr>
<td>integrator.xml</td>
<td>Adds plug-ins to the build.</td>
</tr>
</tbody>
</table>

Sample Ant scripts

Sample Ant scripts are located at http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html. One of the sample scripts processes the garage sample source files with the topics displayed in a hierarchy. The other processes the garage sample with the topics displayed as a sequence. One of the targets in the hierarchy script does conditional processing.

The following section shows the script that publishes topics in a hierarchy. In Windows you would publish to XHTML by opening the Command Prompt, navigating to the ant_scripts directory, and entering the command:

```
ant -f garage_hierarchy_all.xml xhtml
```

For more information about processing (building) to XHTML targets, see Processing to XHTML targets on page 87.

Sample user script

Here is the annotated "hierarchy" script for the garage sample. This script is part of the DITA Open Toolkit User Guide package.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) Copyright 2006-2007 VR Communications, Inc. All rights reserved. -->
<!-- Ant script for the garage sample processed as a hierarchy of topics (garage_hierarchy_all.xml) -->
<!-- Depending on the argument passed to the script at runtime, this script processed to one (example: dita2xhtml), multiple, or all output targets. -->
<!-- This is a "user script," meant to be modified. However, be cautious in modifying the environment initialization section. -->

<!-- ENVIRONMENT INITIALIZATION SECTION -->

<!-- The project name is garage. The default output target is xhtml. -->
<project name="garage" default="xhtml">

<!-- This message is echoed to the screen at the beginning of the build process -->
<echo>Building garage sample as a hierarchy of topics.</echo>

<!-- To process using a different map, you need to change the following statement -->
<echo>Building garage sample as a hierarchy of topics.</echo>

<!-- Allow the build process access to environment variables -->
<property environment="env"/>

<!-- Set toolkit directory from environment variable -->
<property name="toolkit_dir" value="${env.DITA_DIR}"/>

<!-- Note: basedir may be set in a higher-level script that invokes this one. Some plug-ins depend on its pointing to the Toolkit directory. -->
```
In particular, the Idiom plug-in, which is required for PDF2 processing, requires it. -->

<property name="basedir" value="${env.DITA_DIR}"/>

<!-- Define the location of the project (source file) directory.
In this case, the project directory is up two directory levels from the location of this file. -->
<property name="projdir" value="${ant.file}/../.."/>

<!-- Define the location of the output directory.
In this case, it is a directory directly above the project (source file) directory. -->
<property name="outdir" value="${projdir}/..\DITASAMPLE_GARAGE_OUTPUT\"/>
<property name="args.logdir" value="${outdir}"/>

<!-- Set the name of the (master) ditamap -->

<!-- To process using a different map, you need to change the following value -->

<property name="MAP_file" value="hierarchy.ditamap"/>

<!-- Set the extension string for the dita files (files can also be .xml) -->
<property name="EXTNAME" value=".dita"/>

<!-- Define the path to the Java code -->
<path id="dost.class.path">
<pathelement location="${toolkit_dir}\lib\dost.jar"/>
</path>

<!-- Define the Ant targets and initialize the DITA processing environment -->
taskdef name="integrate" classname="org.dita.dost.platform.IntegratorTask">
<classpath refid="dost.class.path"/>
</taskdef>

<!-- Create the default XHTML output and debug cross-reference -->
target name="xhtml" depends="integrate, dita2xhtml">
</target>

<!-- Names of the output types this script is capable of producing -->
The target names are somewhat arbitrary. The following are traditional,
and they allude to the source and target (for example, dita2xhtml). -->
target name="alltargets" depends="integrate, dita2xhtml, dita2filtered, dita2htmlhelp, dita2pdf2,
dita2eclipsecontent, dita2eclipsehelp, dita2docbook, dita2javahelp, dita2trollof, dita2wordrtfl">
</target>

<!-- Handy abbreviation (all) for alltargets -->
target name="all" depends="alltargets"/>

<!-- Create xhtml output (unfiltered) -->
target name="dita2xhtml">
<ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
<property name="args.input" value="${projdir}/${MAP_file}"/>
<property name="output.dir" value="${outdir}/hierarchy/xhtml"/>
<property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
<property name="transtype" value="xhtml"/>
<property name="args.draft" value="yes"/>
<property name="args.indexshow" value="yes"/>
<!-- Create xhtml output (filtered) -->
<!-- The output file directory is structured so the filtered output files will not "step on" the existing files. They will be placed one level deeper than the existing hierarchy files. -->
<target name="dita2filtered">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/hierarchy/filtered/xhtml"/>
    <property name="dita.temp.dir" value="${outdir}/hierarchy/filtered//temp"/>
    <property name="transtype" value="xhtml"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
  </ant>
</target>

<!-- Specify the ditaval file to be used for filtering -->
<!-- To turn on filtering, you first need to change the following ditaval file to include the filter. See the ditaval file for instructions. Then run the Ant script with the dita2filtered target name.
Example (using the batch to call the Ant script): runbuild dita2filtered hierarchy -->
<property name="dita.input.valfile" value="${projdir}/ditaval_files/garage_filtering.ditaval"/>
</ant>
</target>

<!-- Create htmlhelp output -->
<target name="dita2htmlhelp">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/hierarchy/htmlhelp"/>
    <property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
    <property name="transtype" value="htmlhelp"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
  </ant>
</target>

<!-- Create idiom fo pdf2 output -->
<target name="dita2pdf2">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/hierarchy/pdf2"/>
    <property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
    <property name="transtype" value="pdf2"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
  </ant>
</target>

<!-- Create eclipsecontent output -->
<target name="dita2eclipsecontent">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/hierarchy/eclipsecontent"/>
    <property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
    <property name="transtype" value="eclipsecontent"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
  </ant>
</target>

<!-- Create eclipsehelp output -->
<target name="dita2eclipsehelp">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/hierarchy/eclipsehelp"/>
    <property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
    <property name="transtype" value="eclipsehelp"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
  </ant>
</target>
<!-- Create docbook output -->
target name="dita2docbook">
<ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
<property name="args.input" value="${projdir}/${MAP_file}"/>
<property name="output.dir" value="${outdir}/hierarchy/docbook"/>
<property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
<property name="transtype" value="docbook"/>
<property name="args.draft" value="yes"/>
<property name="args.indexshow" value="yes"/>
<property name="dita.extname" value="${EXTNAME}"/>
</ant>
</target>

<!-- Create javahelp output -->
target name="dita2javahelp">
<property name="jarfile" value="${outdir}/hierarchy/javahelp/garage.jar"/>
<ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
<property name="args.input" value="${projdir}/${MAP_file}"/>
<property name="output.dir" value="${outdir}/hierarchy/javahelp"/>
<property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
<property name="transtype" value="javahelp"/>
<property name="args.draft" value="yes"/>
<property name="args.indexshow" value="yes"/>
<property name="dita.extname" value="${EXTNAME}"/>
</ant>
<delete file="${jarfile}"/>
<jar destfile="${jarfile}" basedir="${outdir}/hierarchy/javahelp" includes="***"/>
</target>

<!-- Create troff output -->
target name="dita2troff">
<ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
<property name="args.input" value="${projdir}/${MAP_file}"/>
<property name="output.dir" value="${outdir}/hierarchy/troff"/>
<property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
<property name="transtype" value="troff"/>
<property name="args.draft" value="yes"/>
<property name="args.indexshow" value="yes"/>
<property name="dita.extname" value="${EXTNAME}"/>
</ant>
</target>

<!-- Create wordrtf output -->
target name="dita2wordrtf">
<ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
<property name="args.input" value="${projdir}/${MAP_file}"/>
<property name="output.dir" value="${outdir}/hierarchy/wordrtf"/>
<property name="dita.temp.dir" value="${outdir}/hierarchy/temp"/>
<property name="transtype" value="wordrtf"/>
<property name="args.draft" value="yes"/>
<property name="args.indexshow" value="yes"/>
<property name="dita.extname" value="${EXTNAME}"/>
</ant>
</target>
</project>
Processing overview

Processing is producing output files from a set of DITA source files.

DITA Open Toolkit, working with other tools like the Java Development Kit (JDK) and Ant, provides a method to process DITA documents.

The following figure provides an overview of the processing and publishing of DITA documents using DITA Open Toolkit:

- In step 1, the Ant build tool is initialized (either through an optional batch script or directly from the command line), and arguments that will be passed from the Ant script to Ant are validated.
- In step 2, the Ant build tool calls the Toolkit, which produces the rendered temporary files. Input to this step is the .dita and .ditamap source files, and the DITA project DTDs or schemas.
- In step 3, the XSLT processor (SAXON or Xalan) produces the published output files. Input to this step are the XSLT stylesheets for the project, and the temporary files produced in step 2.

The following figures show the annotated console log from processing the garage sample files to XHTML.

**Stage 1 - Initialization and validation of arguments**

Processing begins by initializing the Toolkit runtime environment and by echoing and validating the input arguments.
Stage 2 - Creating the merged files in the temp directory

This stage, carried out primarily by the Toolkit Java code, creates files in the temp directory where conrefs and related links have been resolved.

[pipeline] Using XERCES.
Generate file list...
Debug input files...
Debug and filter input files...
Copy image files...
Copy html files...
Copy flag files...
Copy generated files...
[copy] Copying 1 file to C:\DITASAMPLE_GARAGE_OUTPUT
Move index entries...
Resolve conref in input files...
Pull the navtitle and topicmeta from topics to ditamap...
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
[xslt] Loading stylesheet C:\DITA-OT1.3.1-full\xsl\preprocess
[copy] Copying 1 file to C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy
Find and generate related link information...
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
[xslt] Loading stylesheet C:\DITA-OT1.3.1-full\xsl\preprocess
Move the related link information to topics...
Pull metadata for link and xref element...
[xslt] Transforming into C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
[xslt] Loading stylesheet C:\DITA-OT1.3.1-full\xsl\preprocess
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
[xslt] Processing C:\DITASAMPLE_GARAGE_OUTPUT\hierarchy\temp
**Stage 3 - Create specified output type from temporary files**

During the final stage, carried out mostly by the appropriate XSLT stylesheets, the merged temporary files are transformed into the correct output format. Also, any other files, such as CSS files, are copied to the output directory. Other applications, such as the HTML Help compiler or the FO renderer are invoked.

<table>
<thead>
<tr>
<th>Build HTML TOC file...</th>
</tr>
</thead>
<tbody>
<tr>
<td>[xsl] Transforming into C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Loading stylesheet C:\DITA-OT1.3.1-full\xsl\map2htmtoc.xsl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Copy CSS files...</th>
</tr>
</thead>
<tbody>
<tr>
<td>[xsl] Transforming into C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Loading stylesheet C:\DITA-OT1.3.1-full\xsl\dita2xhtml.xsl</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
<tr>
<td>[xsl] Processing C:\DitaSample_Garage_OUTPUT\hierarchy \temp\toc.html</td>
</tr>
</tbody>
</table>

Processing ended.

**BUILD SUCCESSFUL**

Total time: 13 seconds

**Ant processing parameters**

The parameters are listed in alphabetical order. The names of required parameters are marked with an asterisk (*).

For examples of how these parameters are used in an Ant build script, see *About Ant scripts* on page 75.

<table>
<thead>
<tr>
<th>Parameter (*Required), Target</th>
<th>Definition, Usage</th>
<th>Valid values, Default, Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>args.artlbl</td>
<td>Adds annotation to images showing the filename of the image. Useful for pre-publishing editing.</td>
<td>Valid: yes or no, Default: no</td>
</tr>
<tr>
<td>args.copycss</td>
<td>Whether to copy user-specified CSS file(s) to the directory specified {args.outdir}{args.csspath}.</td>
<td>Valid: yes or no</td>
</tr>
<tr>
<td>Parameter (*Required), Target</td>
<td>Definition, Usage</td>
<td>Valid values, Default, Examples</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td>Name of user-specified CSS file. Local or remote (web) file.</td>
<td>Default: no</td>
</tr>
<tr>
<td>args.css</td>
<td>If ${args.csspath} is a URL, ${args.css} must be a filepath relative to the URL.</td>
<td></td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td>Path to user-specified CSS file.</td>
<td>Default: no path</td>
</tr>
<tr>
<td>args.csspath</td>
<td>Notes:</td>
<td>Example: <a href="http://www.ibm.com/css">http://www.ibm.com/css</a></td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td>• If ${args.csspath} is a URL, it must start with http:// or https://.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Local absolute paths are not supported for args.csspath.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Use &quot;/&quot; as the path separator, and do not append a &quot;/&quot; trailing separator (for example, use css/myscssfiles rather than css/myscssfiles/).</td>
<td></td>
</tr>
<tr>
<td>args.cssroot</td>
<td>Root directory of user-specified CSS file.</td>
<td></td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td>If this parameter is set, ${args.css} must be a filepath relative to args.cssroot.</td>
<td></td>
</tr>
<tr>
<td>args.dita.locale</td>
<td>Locale used for sorting index terms.</td>
<td>Default (If &quot;xml-lang&quot; is not specified): en-us</td>
</tr>
<tr>
<td>These targets only: htmlhelp and javahelp</td>
<td>If no locale is specified, the first occurrence of &quot;xml-lang&quot; is used as the default locale.</td>
<td></td>
</tr>
<tr>
<td>args.draft</td>
<td>Include draft and required cleanup content (that is, items identified as left to do before publishing).</td>
<td>Valid: yes or no</td>
</tr>
<tr>
<td>All targets</td>
<td>Default: no</td>
<td></td>
</tr>
<tr>
<td>args.eclipsecontent.toc</td>
<td>Root file name of the output Eclipse content toc file.</td>
<td>Default: name of the source ditamap file</td>
</tr>
<tr>
<td>Target: eclipsecontent only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>args.eclipsehelp.toc</td>
<td>Root file name of the output Eclipse help toc file.</td>
<td>Default: name of the source ditamap file</td>
</tr>
<tr>
<td>Targets: eclipsehelp only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>args.eclipse.provider</td>
<td>Provider name of the Eclipse help output.</td>
<td>Default: DITA</td>
</tr>
<tr>
<td>Targets: eclipsehelp only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>args.eclipse.version</td>
<td>Version number of the Eclipse help output.</td>
<td>Default: 1.0</td>
</tr>
<tr>
<td>Targets: eclipsehelp only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>args.fo.img.ext</td>
<td>Extension name of the image files in the PDF output.</td>
<td></td>
</tr>
<tr>
<td>Target: pdf only</td>
<td>A given target set can contain only one or the other extension (not both). Image files with the</td>
<td></td>
</tr>
<tr>
<td>Parameter (*Required), Target</td>
<td>Definition, Usage</td>
<td>Valid values, Default, Examples</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------</td>
<td>-------------------------------</td>
</tr>
</tbody>
</table>
| **args.fo.output.rel.links**  | Whether links will appear in the output files. | Valid: yes or no  
Default: no |
| **Target:** pdf only          |                   |                               |
| **args.fo.userconfig**       | Name of the configuration file for FOP processing. | Valid: yes or no  
Default: no |
| **Target:** pdf only          |                   |                               |
| **args.ftr**                  | URI of the file containing XHTML to be placed in the body running-footer area of the output file.  
The file must be well-formed XML. | Example: `<property name="args.ftr" value="file:/C:/sandbox/myfr.xml"/>` |
| **These targets only:** eclipsehelp, htmihelp, javahelp, or xhtml |                   |                               |
| **args.hdf**                  | URI of the file containing XHTML to be placed in the header area of the output file.  
The file must be well-formed XML. | Example: `<property name="args.hdf" value="file:/C:/sandbox/myhdf.xml"/>` |
| **These targets only:** eclipsehelp, htmihelp, javahelp, or xhtml |                   |                               |
| **args.hdr**                  | URI of the file containing XHTML to be placed in the body running-header area of the output file.  
The file must be well-formed XML. | Example: `<property name="args.hdr" value="file:/C:/sandbox/myhdr.xml"/>` |
| **These targets only:** eclipsehelp, htmihelp, javahelp, or xhtml |                   |                               |
| **args.htmlhelp.includefile** | File to be included in the output. |                               |
| **Target:** htmlhelp only     |                   |                               |
| **args.indexshow**            | Whether indexterm entries should display in the output text. Makes it possible to see what has been indexed in a pre-publishing review. | Valid: yes or no  
Default: no |
| **These targets only:** eclipsehelp, htmihelp, javahelp, or xhtml |                   |                               |
| **args.input**                | Path and name of the input file.  
Use the same case as the filename. | Example: `<property name="args.input" value="mastermap.ditamap"/>` |
| **All targets**               |                   |                               |
| **args.javahelp.map**         | Root file name of the output JavaHelp map file. | Default: name of the input ditamap file |
| **Target:** javahelp only     |                   |                               |
| **args.javahelp.toc**         | Root file name of the output JavaHelp toc file. | Default: name of the input ditamap file |
| **Target:** javahelp only     |                   |                               |
| **args.outext**               | Output file extension name for generated XHTML files.  
In most browser environments, either html or htm is acceptable. | Valid: html or htm  
Default: html |
<p>| <strong>These targets only:</strong> eclipsehelp, htmihelp, javahelp, or xhtml |                   |                               |</p>
<table>
<thead>
<tr>
<th>Parameter (*Required), Target</th>
<th>Definition, Usage</th>
<th>Valid values, Default, Examples</th>
</tr>
</thead>
</table>
| **args.logdir** | Directory used to store generated Ant log files. If you generate several outputs in a single build, the following rules apply:  
  - If you specified a common logdir for all transformations, it will be used as the log directory.  
  - If you did not specify a common logdir for all transformations:  
    - If all individual transforms have the same output directory, it will be used as the log directory.  
    - If all individual transforms do not have the same output directory, basedir will be used as the log directory. | Default: output directory specified by output.dir |
| **args.xhtml.toc** | Root file name of the output XHTML toc file. | Default: index |
| **args.xsl** | xsl transform file that will replace the default file:  
  - For transtype="docbook", dita2docbook.xsl will be replaced.  
  - For transtype="eclipsehelp" or transtype="xhtml": dita2xhtml.xsl.  
  - For transtype="html" or transtype="javahelp": dita2html.xsl.  
  - For transtype="pdf": dita2fo-shell.xsl.  
  - For transtype="rtf": dita2rtfImpl.xsl. | Example: `<property name="args.input" value="mastermap.ditamap"/>` |
| **basedir** | Path of the working directory for transformations. Notes:  
  - If basedir is a relative path, it will be set relative to the current directory.  
  - For Ant scripts, the default is the path set in the Ant build file.  
  - For the Java command line, the default is the current directory. | Example: `<project name="proj1" default="all" basedir="C:\ditaot"/>` |
| **clean.temp** | Whether to clean the temp directory before each build. | Valid: yes or no  
Default: no |
| **dita.dir** | Absolute path of the Toolkit’s home directory. |  |
### About the garage sample

The garage sample, which is downloadable from [http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html](http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html), is a set of DITA source files containing concepts and tasks related to organizing and doing tasks in a garage. The sample map files allow the topics to be published as either a hierarchy or a sequence. The sample also includes Ant scripts that allow you to publish to all target environments.

**Note:** The garage sample that is part of the *DITA Open Toolkit User Guide* is *not* exactly the same set of files as the garage sample that is part of the Toolkit package.

Before you begin to use the garage sample files, we recommend creating two directories in your root directory called DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT (Windows examples would be C:/DITASAMPLE_GARAGE_SOURCE and C:/DITASAMPLE_GARAGE_OUTPUT) and then copying the garage sample files to DITASAMPLE_GARAGE_SOURCE. Your directory structure should then look like this:

```plaintext
+ DITASAMPLE_GARAGE_OUTPUT
  + DITASAMPLE_GARAGE_SOURCE
    + ant_scripts
    + concepts
    + ditaval_files
    + images
    + tasks
```
We recommend that you process your first test case to the XHTML target (dita2xhtml), which generally provides predictably positive results. The dita2htmlhelp, dita2pdf2, and the Eclipse targets (dita2eclipsecontent and dita2eclipsehelp) are also very reliable. The other targets are, in our experience, somewhat problematic, and may require special coding to get acceptable results.

One of the garage sample targets (dita2filtered, which produces XHTML output) illustrates conditional processing using filtering. The output is a hierarchically organized file with four of the garage topics (those dealing with "oil" or "snow") excluded.

**Processing to the key Toolkit output targets**

Sections in this topic:

- Processing to XHTML targets on page 87
- Processing to HTML Help targets on page 89
- Processing to PDF2 targets on page 90
- Processing to Eclipse targets on page 92

**Processing to XHTML targets**

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT directories in the root directory of your C: drive (or your /home directory in Linux).

2. If you have not already done so, copy the garage sample files into the DITASAMPLE_GARAGE_SOURCE directory.

You should have a directory structure that looks like this:

```
DITASAMPLE_GARAGE_OUTPUT
DITASAMPLE_GARAGE_SOURCE
   ant_scripts
   concepts
   ditaval_files
   images
   tasks
```

3. View and edit, if necessary for your specific working environment, DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml.

These are the significant Ant parameters and how they are set:

- **args.input** is set to use the hierarchy ditamap: hierarchy.ditamap in the DITASAMPLE_GARAGE_SOURCE directory.
- **output.dir** is set to send the output to DITASAMPLE_GARAGE_OUTPUT.
- **transtype** is set to xhtml.

4. In the Command Prompt window, move to DITASAMPLE_GARAGE_SOURCE/ant_scripts and invoke the above Ant script.

   For example: ant -f garage_hierarchy_all.xml dita2xhtml
Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2xhtml. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.

Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the XHTML file has processed successfully, go to the DITASAMPLE_GARAGE_OUTPUT directory. Your directory structure should look like this:

```
<table>
<thead>
<tr>
<th>DITASAMPLE_GARAGE_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

The xhtml subdirectory should contain these directories and files:

```
| concepts |
| images |
| tasks |
| commonltr.css |
| commonrtl.css |
| index.html |
```

6. Open the file index.html in your browser to view the XHTML output. The browser window should look something like this:

- **Garage Tasks**
  - Changing the oil in your car
  - Organizing the workbench and tools
  - Shovelling snow
  - Taking out the garbage
  - Spray painting
  - Washing the car

- **Garage Concepts**
  - Lawnmower
  - Oil
  - Paint
  - Shelving
  - Snow shovel
  - Tool box
  - Tools
  - Water hose
  - Wheelbarrow
  - Workbench
  - Windshield washer fluid
Processing to HTML Help targets

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT directories in the root directory of your C: drive (or your /home directory in Linux).

2. If you have not already done so, copy the garage sample files into the DITASAMPLE_GARAGE_SOURCE directory.

You should have a directory structure that looks like this:

```
  | DITASAMPLE_GARAGE_OUTPUT
  | DITASAMPLE_GARAGE_SOURCE
  |   | ant_scripts
  |   | concepts
  |   | ditaval_files
  |   | images
  |   | tasks
```

3. View and edit, if necessary for your specific working environment, DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml dita2htmlhelp.

These are the significant Ant parameters and how they are set:

- `args.input` is set to use the hierarchy ditamap: hierarchy.ditamap in the DITASAMPLE_GARAGE_SOURCE directory.
- `output.dir` is set to send the output to DITASAMPLE_GARAGE_OUTPUT.
- `transtype` is set to htmlhelp.

4. In the Command Prompt window, go to MY_DITA_SOURCE/samples/garage/ant_scripts and invoke the above Ant script. For example: ant -f garage_hierarchy_htmlhelp.xml

Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2htmlhelp. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.

Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the HTML Help file has processed successfully, move to the MY_DITA_OUTPUT directory.

Your directory structure should look like this:

```
  | hierarchy
  | htmlhelp
  |   | concepts
  |   | images
  |   | tasks
```

The htmlhelp directory should contain these directories and files:
6. Open the file hierarchy.chm in your browser to view the HTML Help output. The window should look something like this:

![HTML Help Output](image)

**Processing to PDF2 targets**

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document. Before generating PDF2 output, you must have the Idiom FO plugin processor installed.

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT directories in the root directory of your C: drive (or your /home directory in Linux).
2. If you have not already done so, copy the garage sample files into the DITASAMPLE_GARAGE_SOURCE directory.

You should have a directory structure that looks like this:

```
   + DITASAMPLE_GARAGE_OUTPUT
     + DITASAMPLE_GARAGE_SOURCE
       - ant_scripts
       - concepts
       - ditaival_files
       - images
       - tasks
```

3. View and edit, if necessary for your specific working environment, DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml.

These are the significant Ant parameters and how they are set:

- **args.input** is set to use the hierarchy ditamap: hierarchy.ditamap in the DITASAMPLE_GARAGE_SOURCE directory.
- **output.dir** is set to send the output to DITASAMPLE_GARAGE_OUTPUT.
- **transtype** is set to pdf2.

4. In the Command Prompt window, move to MY_DITA_SOURCE/samples/garage/ant_scripts and invoke the above Ant script.

For example: `ant -f garage_hierarchy_all.xml dita2pdf2`

Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2pdf2. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.

Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the PDF2 file has processed successfully, go to the MY_DITA_OUTPUT directory.

Your directory structure should look like this:

```
   + pdf2
     + Configuration
       - OpenTopic
         - cfg
           - common
           - artwork
         - Customization
         - OpenTopic
           - common
           - artwork
     - images
```

The pdf2 subdirectory should contain these directories and files:

```
Configuration
Customization
images
```

The `hierarchy.pdf` file should be present in the pdf2 subdirectory.
6. Open the file hierarchy.pdf in a PDF reader to view the PDF2 output. The window should look something like this:

![Adobe Acrobat Professional - [hierarchy.pdf]](image)

### Processing to Eclipse targets

Eclipse is an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools, and runtimes for building, deploying, and managing software across the lifecycle.

In addition to providing a development platform, Eclipse can also be used to create XHTML-based information centers (infocenters). This allows users to access Eclipse help files over the Internet or an intranet by displaying the files in a Web browser.

Sections in this topic:

- [Processing to Eclipse targets (overview)](page) on page 92
- [Processing to Eclipse help targets](page) on page 93
- [Running DITA builds in Eclipse](page) on page 97

### Processing to Eclipse targets (overview)

Eclipse Help is one of the DITA Open Toolkit target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plug-in. This allows the output to be displayed as Eclipse help or in an Eclipse Infocenter.

Eclipse content is one of the DITA target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plug-in. This allows the output to be displayed as Eclipse help or in an Eclipse Infocenter.
In contrast to Eclipse help output, which produces XHTML files to be directly displayed by a browser, Eclipse content output produces `.dita` files that are rendered into `.html` files on-the-fly by the Eclipse help processor.

The Toolkit can process to either Eclipse help or Eclipse content targets. In either case the output is a set of files in an output directory that can be installed in an Eclipse distribution as a plug-in. One of the files generated by the Toolkit output processing in the output directory is `plugin.xml`. This file is used by Eclipse to install the rest of the files in the output directory as an Eclipse help plug-in. The figure below shows a sample `plugin.xml` file generated by the Toolkit.

```xml
<?xml version="1.0" encoding="utf-8"?>
<plugin name="DITA Open Toolkit User Guide" id="ditaug_top" version="1.0" provider-name="DITA">
  <extension point="org.eclipse.help.toc"><toc file="DITAOTUG.xml" primary="true"/></extension>
  <extension point="org.eclipse.help.index">
    <index file="index.xml"/>
  </extension>
</plugin>
```

The name attribute contains the title Eclipse will display for the set of help files. The id and version attributes of the plug-in element in this file are combined together to specify the directory where Eclipse expects to find the rest of the plug-in files. In this case that directory would be `ditaug_top_1.0`. The name attribute is generated from the title attribute of the DITA map file. The id attribute is generated from the id attribute of the DITA map file. For the above `plugin.xml` file, the DITA map file contained:

```xml
<bookmap title="DITA Open Toolkit User Guide" id="ditaug_top">
  <list/>
</bookmap>
```

Note: A bookmap is a specialization of map.

**Processing to Eclipse help targets**

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create `DITASAMPLE_GARAGE_SOURCE` and `DITASAMPLE_GARAGE_OUTPUT` directories in the root directory of your `C:` drive (or your `/home` directory in Linux).

2. If you have not already done so, copy the garage sample files into the `DITASAMPLE_GARAGE_SOURCE` directory.

You should have a directory structure that looks like this:

```
- DITASAMPLE_GARAGE_OUTPUT
  - DITASAMPLE_GARAGE_SOURCE
    - ant_scripts
    - concepts
    - ditaval_files
    - images
    - tasks
```

3. View and edit, if necessary for your specific working environment, `DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml`.

These are the significant Ant parameters and how they are set:

- `args.input` is set to use the hierarchy ditamap: `hierarchy.ditamap` in the `DITASAMPLE_GARAGE_SOURCE` directory.
• **output.dir** is set to send the output to DITASAMPLE_GARAGE_OUTPUT.

• **transtype** is set to eclipsehelp.

4. In the Command Prompt window, move to DITASAMPLE_GARAGE_SOURCE/ant_scripts and invoke the above Ant script.
   For example: ant -f garage_hierarchy_all.xml dita2eclipsehelp
   Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2eclipsehelp. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.
   Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the Eclipse help file has processed successfully, go to the DITASAMPLE_GARAGE_OUTPUT directory. Your directory structure should look like this:

```
| eclipsehelp |
| concepts |
| images |
| tasks |
```

The htmlhelp subdirectory should contain these directories and files:

```
| concepts |
| images |
| tasks |
| commonctrl.css |
| commonctrl.css |
| hierarchy.xml |
| index.xml |
| plugin.xml |
```

6. Install the files in the output directory as an Eclipse plug-in. See the Eclipse documentation for how to do this.

7. From within Eclipse, enter **Help > Help Contents**

The following figure shows this document, the garage sample, and the grocery shopping sample displayed in an Eclipse Infocenter.
The following figure shows the title page of this document displayed in an Eclipse Infocenter.

The following figure shows the index tab of an Eclipse Infocenter.
The following figure shows the search tab of an Eclipse Infocenter.

The following figure shows the bookmark tab of an Eclipse Infocenter.
Running DITA builds in Eclipse

You must be using the full package distribution of DITA Toolkit 1.3.1 (or later) to use this procedure.

This topic describes how to incorporate an existing set of DITA Open Toolkit source and Ant build files into an Eclipse project so that the DITA source files can be edited and the DITA Toolkit Ant builds can be run within Eclipse. Eclipse provides various XML editors and also supports execution of Ant build scripts.

1. Create a new Eclipse general project. Begin by selecting File > New > Project > General project. Then select Next and give the project a name. In our example we call it DITAEclipse2.

2. Import the DITA source and build files into your Eclipse project. To add the existing DITA source and build files to the new project, right-click on the project in the Package Explorer and select Import > File System > Next, and use the Import Dialog to locate and select the files and directories to be added to the project.

3. Make any needed modifications to the Ant build script. After you have imported the Ant build script used to process your DITA files, check to make sure that the basedir and other properties in the build script are correct for the location of the files in your Eclipse project.

4. Create a project folder to reference the Toolkit lib directory. In order to run your Ant build script under Eclipse, you need to add the appropriate Toolkit .jar files to the classpath used by Eclipse when it runs the Ant script. Right-click on the project and select New > Folder > Advanced and check the "link to folder in the file system" box. Then select Browse and select the lib directory in the Toolkit. This directory contains all the necessary .jar files required for the Toolkit processing. Now right-click your project Ant build script and select Run As > 2 Ant Build... This will open the Modify attributes and launch dialog. Select the Classpath tab and use the dialog to add the jar files in the project lib folder and the lib folder itself to the CLASSPATH for the script.

The result should look something like this:
5. Now select **Apply > Run** to run your DITA build. The build output (and any error messages) will appear in the **console** window at the bottom of the Eclipse window.

**Processing to other Toolkit output targets**

Sections in this topic:

- *Processing to DocBook targets* on page 98
- *Processing to JavaHelp targets* on page 100
- *Processing from the Java command line* on page 102
- *Processing to PDF targets* on page 104
- *Processing to troff targets* on page 104
- *Processing to Word RTF targets* on page 105

**Processing to DocBook targets**

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

**Note:** In our experience, DocBook support by the DITA Open Toolkit can sometimes be problematic. According to IBM, "there is no guarantee of fault-free transforms."

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create **DITASAMPLE_GARAGE_SOURCE** and **DITASAMPLE_GARAGE_OUTPUT** directories in the root directory of your C: drive (or your /home directory in Linux).

2. If you have not already done so, copy the garage sample files into the **DITASAMPLE_GARAGE_SOURCE** directory.

You should have a directory structure that looks like this:
3. View and edit, if necessary for your specific working environment, DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml.

These are the significant Ant parameters and how they are set:

- **args.input** is set to use the hierarchy ditamap: hierarchy.ditamap in the DITASAMPLE_GARAGE_SOURCE directory.
- **output.dir** is set to send the output to DITASAMPLE_GARAGE_OUTPUT.
- **transtype** is set to docbook.

4. In the Command Prompt window, move to DITASAMPLE_GARAGE_SOURCE/ant_scripts and invoke the above Ant script. For example: `ant -f garage_hierarchy_all.xml dita2docbook`

Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (`runbuild.bat`): `runbuild dita2docbook`. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.

Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as `runbuild.bat`.

5. After the DocBook file has processed successfully, go to the DITASAMPLE_GARAGE_OUTPUT directory. Your directory structure should look like this:

```
- docbook
  - images
```

The docbook subdirectory should contain these directories and files:

```
- images
  - hierarchy.xml
```

6. Open the file hierarchy.xml to view the DocBook output.

   In a plain text editor like Microsoft WordPad, the file should look something like this:
Processing to JavaHelp targets

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document. To view JavaHelp output, you must have the JavaHelp processor installed. See Installing the JavaHelp processor on Windows on page 60 for information on how to do this.

Note: DITA Open Toolkit support for JavaHelp is adequate. However, in our experience, published JavaHelp results are less than adequate due to limitations of the Sun JavaHelp processor.

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT directories in the root directory of your C: drive (or your /home directory in Linux).

2. If you have not already done so, copy the garage sample files into the DITASAMPLE_GARAGE_SOURCE directory.

You should have a directory structure that looks like this:
3. View and edit, if necessary for your specific working environment, DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml.

These are the significant Ant parameters and how they are set:

- **args.input** is set to use the hierarchy ditamap: hierarchy.ditamap in the DITASAMPLE_GARAGE_SOURCE directory.
- **output.dir** is set to send the output to DITASAMPLE_GARAGE_OUTPUT.
- **transtype** is set to javahelp.

4. In the Command Prompt window, move to DITASAMPLE_GARAGE_SOURCE/ant_scripts and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_all.xml dita2javahelp` Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2javahelp. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.

Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the JavaHelp file has processed successfully, go to the DITASAMPLE_GARAGE_OUTPUT directory. Your directory structure should look like this:

The javahelp subdirectory should contain these directories and files:

6. Using the JavaHelp Viewer, open the file hierarchy_helpset.hs to view the JavaHelp output. You can start the Viewer by entering the command `java -jar %JHHOME%\demos\bin\hsviewer.jar` from the command line. The window should look something like this:
Processing from the Java command line

Under certain circumstances it may be desirable to run the Toolkit build code by invoking the Java JVM from
the command line instead of by invoking Ant. The Toolkit provides a way to do this by supporting a set of
command-line parameters. When invoking the Toolkit build code from the Java command line, the inputs to
Ant are stored in a temporary file by the Toolkit Java code, and then Ant is invoked to carry out the build.

Note: When using the Java command line, you still must have Ant installed.

Running a Java command line example

1. Go to the DITA Open Toolkit installation directory.
2. On the command line, enter the following command:
   ```
   java -jar lib/dost.jar /i:samples/sequence.ditamap /outdir:out /transtype:xhtml
   ```
   This example creates a properties file, and then calls Ant using this properties file to build the garage sample
   sequence.ditamap file and produce XHTML output to the out directory.

Note:
- In this example, the character slash preceded by a space is the separator for each parameter.
- The internally generated properties file is saved in the $ARGS.logdir directory. The following command
  shows an example of using this properties file with Ant:
  ```
  ant -f conductor.xml -propertyfile $ARGS.logdir/property.temp
  ```
**Supported parameters**

Parameters supplied on the Java command line are equivalent to similar parameters used in Ant build scripts. The Ant build script parameters are described in *Ant processing parameters* on page 82. The following table lists the parameters you can provide on the Java command line and the equivalent parameter that can be supplied in an Ant build script.

<table>
<thead>
<tr>
<th>Java parameter</th>
<th>Equivalent Ant script parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>/artlbl</td>
<td>args.artlbl</td>
</tr>
<tr>
<td>/basedir</td>
<td>basedir</td>
</tr>
<tr>
<td>/cleantemp</td>
<td>clean.temp</td>
</tr>
<tr>
<td>/copycss</td>
<td>args.copycss</td>
</tr>
<tr>
<td>/css</td>
<td>args.css</td>
</tr>
<tr>
<td>/cssroot</td>
<td>args.cssroot</td>
</tr>
<tr>
<td>/csspath</td>
<td>args.csspath</td>
</tr>
<tr>
<td>/ditalocale</td>
<td>args.dita.locale</td>
</tr>
<tr>
<td>/draft</td>
<td>args.draft</td>
</tr>
<tr>
<td>/ditadir</td>
<td>dita.dir</td>
</tr>
<tr>
<td>/ditaext</td>
<td>dita.extname</td>
</tr>
<tr>
<td>/eclipsecontenttoc</td>
<td>args.eclipsecontent.toc</td>
</tr>
<tr>
<td>/eclipsehelptoc</td>
<td>args.eclipsehelp.toc</td>
</tr>
<tr>
<td>/filter</td>
<td>args.input.valfile</td>
</tr>
<tr>
<td>/fouserconfig</td>
<td>args.fo.userconfig</td>
</tr>
<tr>
<td>/foimgext</td>
<td>args.fo.img.ext</td>
</tr>
<tr>
<td>/fooutputrellinks</td>
<td>args.fo.output.rel.links</td>
</tr>
<tr>
<td>/ftr</td>
<td>args.ftr</td>
</tr>
<tr>
<td>/hdf</td>
<td>args.hdf</td>
</tr>
<tr>
<td>/hdr</td>
<td>args.hdr</td>
</tr>
<tr>
<td>/htmlhelpincludefile</td>
<td>args.htmlhelp.includefile</td>
</tr>
<tr>
<td>/i</td>
<td>args.input</td>
</tr>
<tr>
<td>/id</td>
<td>dita.inputdirname</td>
</tr>
<tr>
<td>/if</td>
<td>dita.input</td>
</tr>
<tr>
<td>/indexshow</td>
<td>args.indexshow</td>
</tr>
<tr>
<td>/javahelpmap</td>
<td>args.javahelp.map</td>
</tr>
<tr>
<td>/javahelptoc</td>
<td>args.javahelp.toc</td>
</tr>
<tr>
<td>/logdir</td>
<td>args.logdir</td>
</tr>
<tr>
<td>/outdir</td>
<td>output.dir</td>
</tr>
<tr>
<td>/outtext</td>
<td>args.outtext</td>
</tr>
<tr>
<td>/provider</td>
<td>args.eclipse.provider</td>
</tr>
</tbody>
</table>
### Processing to PDF targets

The DITA Open Toolkit PDF transform, which was part of the 1.2.2 release, has been deprecated in favor of the PDF2 transform, which makes use of a plug-in from Idiom Technologies, Inc.

The PDF transform made use of FOP 0.20. By default, the PDF2 transform makes use of RenderX XEP.

For more information about the PDF2 transform, see *Processing to PDF2 targets* on page 90

### Processing to troff targets

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

⚠️ **Note:** In our experience, troff support by the DITA Open Toolkit can sometimes be problematic. According to IBM, "there is no guarantee of fault-free transforms."

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. If you have not already done so, create `DITASAMPLE_GARAGE_SOURCE` and `DITASAMPLE_GARAGE_OUTPUT` directories in the root directory of your `C:` drive (or your `/home` directory in Linux).

2. If you have not already done so, copy the garage sample files into the `DITASAMPLE_GARAGE_SOURCE` directory.

   You should have a directory structure that looks like this:

   ![Directory Structure](image)

3. View and edit, if necessary for your specific working environment, `DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml`.

   These are the significant Ant parameters and how they are set:

   - **args.input** is set to use the hierarchy ditamap: hierarchy.ditamap in the `DITASAMPLE_GARAGE_SOURCE` directory.
   - **output.dir** is set to send the output to `DITASAMPLE_GARAGE_OUTPUT`.
   - **transtype** is set to troff.

4. In the Command Prompt window, move to `DITASAMPLE_GARAGE_SOURCE/ant_scripts` and invoke the above Ant script.
For example: ant -f garage_hierarchy_all.xml dita2troff
Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2troff. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.
Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the troff file has processed successfully, go to the DITASAMPLE_GARAGE_OUTPUT directory. Your directory structure should look like this:

   ![Directory Structure Diagram]

The troff subdirectory should contain these directories and files:

   - concepts
   - images
   - tasks

6. Open the file concepts/garageconcepts.cli to view the troff output.
The file should look something like this:

```plaintext
.ad 1
.11 72
.ce 1000
\fBGarage Concepts\fR
.ce 0
.sp 2
A well-stocked garage can be the envy of the neighborhood.
.sp 2
   Lawnmower
.br
.sp 2
   Oil
.br
```

**Processing to Word RTF targets**

This topic assumes you have already installed DITA Open Toolkit and its prerequisite products, and have verified your installation, as described in the installation chapter of this document.

**Note:** In our experience, Word RTF support by the DITA Open Toolkit can sometimes be problematic. According to IBM, "there is no guarantee of fault-free transforms."

In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.
1. If you have not already done so, create DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT directories in the root directory of your C: drive (or your /home directory in Linux).

2. If you have not already done so, copy the garage sample files into the DITASAMPLE_GARAGE_SOURCE directory.

You should have a directory structure that looks like this:

3. View and edit, if necessary for your specific working environment, DITASAMPLE_GARAGE_SOURCE/ant_scripts/garage_hierarchy_all.xml.

These are the significant Ant parameters and how they are set:

- **args.input** is set to use the hierarchy ditamap: hierarchy.ditamap in the DITASAMPLE_GARAGE_SOURCE directory.
- **output.dir** is set to send the output to DITASAMPLE_GARAGE_OUTPUT.
- **transstype** is set to wordrtf.

4. In the Command Prompt window, move to DITASAMPLE_GARAGE_SOURCE/ant_scripts and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_all.xml dita2wordrtf`

   Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2wordrtf. The batch script is in the DITASAMPLE_GARAGE_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GARAGE_OUTPUT directory during processing.

   Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.

5. After the Word RTF file has processed successfully, go to the DITASAMPLE_GARAGE_OUTPUT directory.
   Your directory structure should look like this:

   The wordrtf subdirectory should contain these directories and files:

6. Open the file hierarchy.rtf in Microsoft Word to view the Word RTF output.
   The file should look something like this:
Garage Tasks

When you go into the garage, be prepared to get your hands dirty!

Changing the oil in your car

Organizing the workbench and tools

Shovelling snow

DITAblogs (processing)

Ant script used to build this document

To produce this document we used an Ant build script containing targets for building all transformation types (often called "transetypes" or "targets") supported by DITA Open Toolkit. The location of the Toolkit, the location of the source files, and the location of the output files are variables that can be changed. Here is our script:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) Copyright VR Communications, Inc. All rights reserved. -->
<!-- Ant script for the DITA Open Toolkit User Guide (DITAOTUG) -->
<!-- Depending on the argument passed to the script at runtime, this script processed to one (example: dita2xhtml), multiple, or all output targets. -->
<!-- This is a "user script," meant to be modified. However, be cautious in modifying the environment initialization section. -->

<!-- ENVIRONMENT INITIALIZATION SECTION -->

<!-- The project name is toolkitug. The default output target is xhtml. -->
<project name="toolkitug" default="xhtml">
<!-- This message is echoed to the screen at the beginning of the build process. -->
<echo>Building DITA Open Toolkit User Guide.</echo>
<!-- Allow the build process access to environment variables -->
<property environment="env"/>
<!-- Set toolkit directory from environment variable -->
<property name="toolkit_dir" value="${env.DITA_DIR}"/>
<!-- Note: basedir may be set in a higher-level script that invokes this one. Some plug-ins depend on its pointing to the Toolkit directory. In particular, the Idiom plug-in, which is required for PDF2 processing, requires it. -->
<property name="basedir" value="${env.DITA_DIR}"/>
<!-- Define the location of the project (source file) directory. In this case, the project directory is up two directory levels from the location of this file. -->
<property name="projdir" value="${basedir}/DITAOTUG"/>
<!-- Define the location of the output directory. In this case, it is a directory directly above the project (source file) directory. -->
<property name="outdir" value="${projdir}/DITAOTUG_OUTPUT"/>
<property name="PHPExe" value="C:/wamp/php/php.exe"/>
<property name="args.logdir" value="${args.logdir}"/>
<property name="ditaval_nonpdf" value="${ditaval_nonpdf}"/>
<!-- Define the path to the Java code -->
</project>
```
<!-- Define the Ant targets and initialize the processing environment -->
<taskdef name="integrate" classname="org.dita.dost.platform.IntegratorTask">
<classpath refid="dost.class.path"/>
</taskdef>

<!-- Create the default XHTML output and debug cross-reference -->
<target name="xhtml" depends="integrate, ditadebug, ditalinks, dita2xhtml">
</target>
<target name="integrate">
<integrate ditadir="${toolkit_dir}"/>
</target>

<!-- BEGIN INSTANCE PROCESSING SECTION -->

<!-- Names of the output types this script is capable of producing -->
<!-- The target names are somewhat arbitrary. The following are traditional, and they allude to the source and target (for example, dita2xhtml). -->
<target name="alltargets" depends="integrate, dita2xhtml, dita2htmlhelp, dita2roff, dita2docbook, dita2javahelp, dita2wordrtf, dita2eclipsecontent, dita2eclipsehelp, dita2pdf2"/>

<!-- Handy abbreviation (all) for alltargets -->
<target name="all" depends="allreports, alltargets"/>

<!-- Create all reports -->
<target name="allreports" depends="ditadebug, ditalinks, ditaauthors, ditakeys"/>

<!-- Create the ditamap debug cross-reference -->
<target name="ditadebug">
<echo>Building debugging cross-reference file ditadebug.txt</echo>
<mkdir dir="${outdir}/debug_files"/>
<exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditadebug.txt">
<arg value="${projdir}/project/tools/ditadebug.php"/>
<arg value="${MAP_file}"/>
</exec>
</target>

<!-- Create plausible preview -->
<target name="preview">
<echo>Building XHTML plausible preview</echo>
<mkdir dir="${outdir}/debug_files"/>
<exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditapreview.txt">
<arg value="${projdir}/project/tools/ditapreview.php"/>
<arg value="${MAP_file}"/>
<arg value="L"/>
</exec>
</target>

<!-- Create the ditamap URL check -->
<target name="ditalinks">
<echo>Building URL check file ditalinks.txt</echo>
<mkdir dir="${outdir}/debug_files"/>
<exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditalinks.txt">
<arg value="${projdir}/project/tools/ditalinks.php"/>
<arg value="${MAP_file}"/>
<arg value="L"/>
</exec>
</target>

<!-- Create the ditamap author and copyright statistics -->
<target name="ditaauthors">
<echo>Building author/copyright file ditaauth.txt</echo>
<mkdir dir="${outdir}/debug_files"/>
<exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditaauthors.txt">
<arg value="${projdir}/project/tools/ditaauthors.php"/>
<arg value="${MAP_file}"/>
</exec>
</target>
<!-- Create the ditamap ID listing -->
<target name="ditaidsls">
  <echo message="Building ID list file ditaidsls.txt"></echo>
  <mkdir dir="${outdir}/debug_files"/>
  <exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditaidsls.txt">
    <arg value="${projdir}/project/tools/ditaidsls.php"/>
    <arg value="${MAP_file}"/>
  </exec>
</target>

<!-- Create the ditamap keyword metadata statistics -->
<target name="ditakeys">
  <echo message="Building keyword metadata file ditakeys.txt"></echo>
  <mkdir dir="${outdir}/debug_files"/>
  <exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditakeys.txt">
    <arg value="${projdir}/project/tools/ditakeys.php"/>
    <arg value="${MAP_file}"/>
  </exec>
</target>

<!-- Create xhtml output -->
<target name="dita2xhtml">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/xhtml"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="xhtml"/>
    <property name="args.draft" value="yes"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
    <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
  </ant>
  <copy todir="${outdir}/xhtml">
    <fileset dir="${projdir}/build_files"/>
  </copy>
</target>

<!-- Create htmlhelp output -->
<target name="dita2htmlhelp">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/htmlhelp"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="htmlhelp"/>
    <property name="args.draft" value="yes"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
    <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
  </ant>
</target>

<!-- Create tocjs output -->
<target name="dita2tocjs">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/tocjs" clean="no"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="tocjs"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
    <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
  </ant>
</target>

<!-- Copy index.html to the output directory -->
<copy todir="${outdir}/xhtml">
  <fileset dir="${projdir}/build_files"/>
</copy>

<!-- Create the ditamap ID listing -->
<target name="ditaidsls">
  <echo message="Building ID list file ditaidsls.txt"></echo>
  <mkdir dir="${outdir}/debug_files"/>
  <exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditaidsls.txt">
    <arg value="${projdir}/project/tools/ditaidsls.php"/>
    <arg value="${MAP_file}"/>
  </exec>
</target>

<!-- Create the ditamap keyword metadata statistics -->
<target name="ditakeys">
  <echo message="Building keyword metadata file ditakeys.txt"></echo>
  <mkdir dir="${outdir}/debug_files"/>
  <exec executable="${PHPexe}" dir="${projdir}" output="${outdir}/debug_files/ditakeys.txt">
    <arg value="${projdir}/project/tools/ditakeys.php"/>
    <arg value="${MAP_file}"/>
  </exec>
</target>

<!-- Create xhtml output -->
<target name="dita2xhtml">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/xhtml"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="xhtml"/>
    <property name="args.draft" value="yes"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
    <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
  </ant>
  <copy todir="${outdir}/xhtml">
    <fileset dir="${projdir}/build_files"/>
  </copy>
</target>

<!-- Create htmlhelp output -->
<target name="dita2htmlhelp">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/htmlhelp"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="htmlhelp"/>
    <property name="args.draft" value="yes"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <property name="dita.extname" value="${EXTNAME}"/>
    <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
  </ant>
</target>

<!-- Create tocjs output -->
<target name="dita2tocjs">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/tocjs" clean="no"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="tocjs"/>
    <property name="clean.temp" value="no"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
In the Windows operating environment we built to all trametypes to make sure they would build cleanly. On Linux, we tested XHTML output only.

We use a Windows batch file (runbuild.bat) to initiate the build. The batch file calls the DITA Ant logger that types a summary of build progress on the console (rather than a more verbose set of messages) and uses a log file to capture the details. Here is our batch file:

```
ant -Dbasedir=%DITA_DIR% -f ant_scripts\DITAOTUG_all.xml -logger org.dita.dost.log.DITAOTBuildLogger %1
```
This method gives us enough information on the console screen to tell whether or not the build was successful and whether any errors occurred that need immediate attention.

**For more information (processing)**

We found the O'Reilly Java series book, *Ant: The Definitive Guide*, by Steve Holzner, useful in helping us work through processing problems as we produced this document.
Troubleshooting the build process

Processing (building) a DITA document results in one of three possible outcomes:

- The build was successful. You got a BUILD SUCCESSFUL message from Ant.
- You got a BUILD SUCCESSFUL message from Ant, but error messages were generated that you need to fix, or your output is not what you expect.
- The build failed. You got a BUILD FAILED message from Ant.

This section helps you deal with the second and third cases, where you need to debug processing problems. It describes tools and mechanisms available to you in the Toolkit itself, other tools available for the Ant/Java environment, and various strategies you can apply to find and fix processing errors quickly.

Sections in this topic:

- Capturing and using the log on page 113
- Error messages generated during processing on page 115
- Troubleshooting CLASSPATH and environment variables setup on page 123
- Debugging, reporting, and file generation tools on page 123
- DITAblogs (troubleshooting) on page 126
- For more information (troubleshooting) on page 129

Capturing and using the log

When a Toolkit Ant build script runs, it sends processing messages to the log. By default the messages appear only on the console.

By adding the parameter `-logger org.dita.dost.log.DITAOTBuildLogger` to the invocation of Ant, you can also write the messages to a disk file. You can also specify the location for the log file by setting the `args.logdir` Ant processing parameter.

In order to troubleshoot a build problem, it is useful to capture the Ant build output in a log file and to control the type of output Ant puts in the log. Here is an example of invoking Ant and capturing the Ant output in the file `antoutput.log`. The `-quiet` Ant command-line option is specified to eliminate non-error messages from the log.

```
ant -f ant\sample_xhtml.xml -quiet -l antoutput.log
```

The current version of the log file for each output target is placed in the document's base output directory. The prior version of the log file is replaced with each new build. Here is a set of log files for HTML Help, PDF2, and XHTML builds of this document.
Here is the beginning of the xhtml log file.

---

```plaintext
Processing started...
Initialize log directory and file name...
Validate and init input arguments...
[echo] ******************************************************
[echo] * basedir = C:\DITA-OT1.3.1-full\%
[echo] * dita.dir = C:\DITA-OT1.3.1-full
[echo] * input = C:\DITAOTUG\SOURCE\ant_scripts\DITAOTUG_d
[echo] * transtype = xhtml
[echo] * tempdir = C:\DITAOTUG\SOURCE\ant_scripts\DITAOTUG_xhtml
[echo] * outputdir = C:\DITAOTUG\SOURCE\ant_scripts\DITAOTUG_xhtml
[echo] * extname = .dita
[echo] * clean.temp = $(clean.temp)
[echo] * xslt.parser = XALAN
[echo] ******************************************************
Preprocessing started...
Clean temp directory...
[delete] Deleting 334 files from C:\DITAOTUG\OUTPUT\temp
[delete] Deleted 25 directories from C:\DITAOTUG\OUTPUT\temp
[pipeline] Using XERCES.
Generate file list...
Debug input files...
Debug and filter input files...
Debug and filter input files...
Copy image files...
[copy] Copying 1 file to C:\DITAOTUG\OUTPUT\xhtml
Copy html files...
Copy flag files...
Copy generated files...
[copy] Copying 1 file to C:\DITAOTUG\OUTPUT
Move index entries...
Resolve conref in input files...
[xslt] Transforming into C:\DITAOTUG\OUTPUT\temp
[xslt] Processing C:\DITAOTUG\OUTPUT\temp\accessing\about
[xslt] Loading stylesheet C:\DITA-OT1.3.1-full\xsl\preproc
[xslt] Processing C:\DITAOTUG\OUTPUT\temp\accessing\about
[xslt] Processing C:\DITAOTUG\OUTPUT\temp\accessing\about
```

---
Error messages generated during processing

Sections in this topic:

* Error messages overview on page 115
  * Messages generated by the Toolkit on page 116
  * Messages generated from other sources on page 123

Error messages overview

Messages in a DITA Toolkit log that begin with DOT are produced by the Toolkit software. Messages produced by other tools (for example, Java JDK or XML parser) are also generated. The Toolkit messages are of three types:

1. Messages beginning with DOTA from the Ant build scripts, for example, DOTA001F.
2. Messages beginning with DOTJ from the Toolkit Java code in `lib/dost.jar`, for example, DOTJ008F.
3. Messages beginning with DOTX from the Toolkit XSLT transforms in the `xsl` directory, for example, DTX009W.

Messages are accompanied by one or more lines of text, with the message as the last line. Each message has a message number, a type (or severity), message text, and a suggested user action to correct the problem. Here is an example of a message:

```
BUILD FAILED
C:\sandbox\ant\messages_xhtml.xml:18: The following error occurred while executing this line:
C:\sandbox\conductor.xml:101: The following error occurred while executing this line:
C:\sandbox\conductor.xml:113: [DOTA002F][FATAL] Invalid input. Provide valid args.input and dita.input
```

In this case the message (number DOTA002F) indicates a fatal error (type FATAL) found in the Ant build scripts. The message text is "Invalid input" and the recommended action is "Provide valid ...". The traceback shows the error occurred in line 101 of `conductor.xml`, which was invoked by line 18 of `messages_xhtml.xml`.

Here is another sample message DTX040I of type INFO from an XSLT transform:

```
[xslt] (File = C:/sandbox/doc/ditaug/concepts/access.dita, Element = draft-comment:1)
[xslt] [DOTX040I][INFO]: Draft comment area found.
If the output is only used as a draft, you do not need to do anything.
If you are producing production-level output, you should not use the /DRAFT option.
```

The Meaning of Message Type (severity)

Each Toolkit error message includes a message type which indicates the severity of the error.

<table>
<thead>
<tr>
<th>Level of severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Information about processing, processing continues.</td>
</tr>
<tr>
<td>WARN</td>
<td>A possible problem was noted, processing continues.</td>
</tr>
<tr>
<td>ERROR</td>
<td>A problem was found, processing continues.</td>
</tr>
<tr>
<td>FATAL</td>
<td>A problem was found, processing stops.</td>
</tr>
</tbody>
</table>
## Messages generated by the Toolkit

### Ant messages

<table>
<thead>
<tr>
<th>Message number</th>
<th>Type</th>
<th>Message text</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOTA001F</td>
<td>FATAL</td>
<td>Invalid transformation type.</td>
<td>Please provide the correct transformation types using Ant parameter 'transtype': xhtml, eclipsehelp, eclipsecontent, javahelp, htmlhelp, pdf, pdf2, troff, docbook, wordrtf.</td>
</tr>
<tr>
<td>DOTA002F</td>
<td>FATAL</td>
<td>Invalid input.</td>
<td>Please provide the correct input.</td>
</tr>
<tr>
<td>DOTA003F</td>
<td>FATAL</td>
<td>Can't find the user specified stylesheet '%1'.</td>
<td>Please provide the correct stylesheet using Ant parameter 'args.xsl'.</td>
</tr>
<tr>
<td>DOTA004F</td>
<td>FATAL</td>
<td>Invalid dita extension '%1'.</td>
<td>Please input the correct dita extension using Ant parameter 'dita.extname': dita, .dita, xml, .xml.</td>
</tr>
<tr>
<td>DOTA005W</td>
<td>WARN</td>
<td>Input parameter 'dita.input' and 'dita.input.dirname' are deprecated.</td>
<td>Please use 'args.input' instead.</td>
</tr>
<tr>
<td>DOTA006W</td>
<td>WARN</td>
<td>Local absolute 'csspath' is not supported, use the default instead.</td>
<td>Please use relative or URL like csspath.</td>
</tr>
</tbody>
</table>

### Java messages

<table>
<thead>
<tr>
<th>Message number</th>
<th>Type</th>
<th>Message text</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOTJ001F</td>
<td>FATAL</td>
<td>Input argument error: no ':' found in the parameter '%1'.</td>
<td>Please add a colon character ':' between the name and value of the parameter '%1'. For detail information, please refer to User Guide.</td>
</tr>
<tr>
<td>DOTJ002F</td>
<td>FATAL</td>
<td>Unsupported parameter '%1'.</td>
<td>Please refer to User Guide for supported parameters.</td>
</tr>
<tr>
<td>DOTJ003F</td>
<td>FATAL</td>
<td>Param value can't be null for the parameter '%1'.</td>
<td>Please provide a value for the parameter '%1'.</td>
</tr>
<tr>
<td>DOTJ004F</td>
<td>FATAL</td>
<td>Can't create temp directory '%1'.</td>
<td>Please check if you have the right to create the directory '%1'.</td>
</tr>
<tr>
<td>DOTJ005F</td>
<td>FATAL</td>
<td>Failed to create new instance for '%1'.</td>
<td>Please ensure that '%1' exists and you have right to access it.</td>
</tr>
<tr>
<td>DOTJ006F</td>
<td>FATAL</td>
<td>Invalid value '%1' for attribute 'extparam' of AntInvoker.</td>
<td>Please use correct way to call AntInvoker, e.g. extparam=&quot;maplinks=XXXX;other=YYYY&quot;.</td>
</tr>
<tr>
<td>DOTJ007E</td>
<td>ERROR</td>
<td>Duplicate condition in filter file for rule '%1'.</td>
<td>Check to make sure that there is none duplicate rule specified.</td>
</tr>
<tr>
<td>DOTJ008E</td>
<td>ERROR</td>
<td>MapIndexReader is not used in correct way. matchList is null.</td>
<td>Please check the code to see whether method setMatch(String matchPattern) is called before read(String filename).</td>
</tr>
<tr>
<td>Message number</td>
<td>Type</td>
<td>Message text</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOTJ009E</td>
<td>ERROR</td>
<td>Cannot overwrite file '%1' with file '%2'. The modified result may not be consumed by the following steps.</td>
<td>Check to see whether the file is locked by other application during the transformation process.</td>
</tr>
<tr>
<td>DOTJ010E</td>
<td>ERROR</td>
<td>Can't find %1 in the extparam for index generation.</td>
<td>Please specify %1 in the extparam.</td>
</tr>
<tr>
<td>DOTJ011E</td>
<td>ERROR</td>
<td>Failed to load the input file '%1' for index generation due to below exception, and no index information generated.</td>
<td>Please ensure '%1' exists and you have right to access it.</td>
</tr>
<tr>
<td>DOTJ012F</td>
<td>FATAL</td>
<td>Failed to parse the input file '%1' due to below exception.</td>
<td>Please correct the input base on the exception message.</td>
</tr>
<tr>
<td>DOTJ013E</td>
<td>ERROR</td>
<td>Failed to parse the referenced file '%1' due to below exception.</td>
<td>Please correct the reference base on the exception message.</td>
</tr>
<tr>
<td>DOTJ014W</td>
<td>WARN</td>
<td>The indexterm element does not have any content. Setting the term to ***.</td>
<td>Please add content to the indexterm.</td>
</tr>
<tr>
<td>DOTJ015F</td>
<td>FATAL</td>
<td>Log directory can't be null.</td>
<td>Please specify the correct log directory using ant parameter 'args.logdir'.</td>
</tr>
<tr>
<td>DOTJ016F</td>
<td>FATAL</td>
<td>Failed to create log directory '%1'.</td>
<td>Please specify the correct log directory using ant parameter 'args.logdir'.</td>
</tr>
<tr>
<td>DOTJ017F</td>
<td>FATAL</td>
<td>Failed to init log filename with input file.</td>
<td>Please specify input file using ant parameter 'args.input'.</td>
</tr>
<tr>
<td>DOTJ018I</td>
<td>INFO</td>
<td>Log file '%1' was generated successfully at directory '%2'.</td>
<td>You can find the detailed messages from the transformation process in this log file.</td>
</tr>
<tr>
<td>DOTJ019E</td>
<td>ERROR</td>
<td>Failed to generated log file.</td>
<td>Please correct the errors.</td>
</tr>
<tr>
<td>DOTJ020W</td>
<td>WARN</td>
<td>Plugin '%1' is required by plugin '%2'. Plugin '%2' cannot be loaded because of missing plugin '%1'.</td>
<td>Check and see whether all of the plugins required are installed in toolkit.</td>
</tr>
<tr>
<td>DOTJ021W</td>
<td>WARN</td>
<td>File '%1' was excluded from the 'dita.list' file since it is invalid and all its content has been filtered out by the ditaval file.</td>
<td>Please check the file '%1' and the ditaval file to see if this is the intended result.</td>
</tr>
<tr>
<td>DOTJ022F</td>
<td>FATAL</td>
<td>Failed to parse the input file '%1' due to all of its content has been filtered out.</td>
<td>Please check the input file '%1' and the ditaval file, and ensure that the input is valid.</td>
</tr>
<tr>
<td>DOTJ023E</td>
<td>ERROR</td>
<td>Failed to get the image file specified '%1' in RTF generation</td>
<td>Check whether the image exists and copied to output directory before the final step of RTF generation.</td>
</tr>
<tr>
<td>DOTJ024W</td>
<td>WARN</td>
<td>Extension name of picture file '%1' not supported.</td>
<td>Please convert your picture to JPEG or GIF style.</td>
</tr>
<tr>
<td>DOTJ025E</td>
<td>ERROR</td>
<td>The input of topic merge cannot be found.</td>
<td>Please check whether the input message for topic merge is absolute path.</td>
</tr>
<tr>
<td>DOTJ026E</td>
<td>ERROR</td>
<td>The output of topic merge is null.</td>
<td>Please check whether you have set output for topic merge correctly.</td>
</tr>
<tr>
<td>DOTJ027W</td>
<td>WARN</td>
<td>The file '%1' contains space in its file name. This file might not be transformed.</td>
<td>Remove the space in the file name and restart the transformation.</td>
</tr>
</tbody>
</table>
### XSLT messages

<table>
<thead>
<tr>
<th>Message number</th>
<th>Type</th>
<th>Message text</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOTX001W</td>
<td>WARN</td>
<td>No string named '%1' was found for language '%2'. Use the default language '%3'.</td>
<td>Add the mapping between default language and specific language for the string '%1'.</td>
</tr>
<tr>
<td>DOTX002W</td>
<td>WARN</td>
<td>The title attribute in ditamap is required for Eclipse output.</td>
<td>Add a title attribute to map element in ditamap file.</td>
</tr>
<tr>
<td>DOTX003I</td>
<td>INFO</td>
<td>The anchorref attribute should either point to another dita map, or to an Eclipse XML file. The value '%1' does not point to either.</td>
<td>Change the anchorref referring to ditamap or dita topic file.</td>
</tr>
<tr>
<td>DOTX004I</td>
<td>INFO</td>
<td>Found a navref that does not point to anything, the navref element should either point to another dita map, or to an Eclipse XML file.</td>
<td>Change the navref referring to ditamap or dita topic file.</td>
</tr>
<tr>
<td>DOTX005E</td>
<td>ERROR</td>
<td>Unable to find navigation title, using href instead: '%1'. If the topic is not accessible at build time, provide the navigation title in the map, and set the format or scope attributes to indicate why it is not accessible. Note: '%1' might be changed to use default dita topic file extension name '.dita' or '.xml'.</td>
<td>Provide the navigation title in the map or topic files.</td>
</tr>
<tr>
<td>DOTX006E</td>
<td>ERROR</td>
<td>Unknown file extension in href: '%1'. If this is a link to a non-DITA resource, set the format attribute to match the resource (for example, 'txt', 'pdf', or 'html'). If it's a link to a DITA resource, the file extension must be 'dita' or 'xml'.</td>
<td>Set the format attribute and specify the format of the file if href link doesn't point to dita topic file. Otherwise, change the file extension name to 'dita' or 'xml'.</td>
</tr>
<tr>
<td>DOTX007I</td>
<td>INFO</td>
<td>Only DITA topics, HTML files, and images may be included in your compiled CHM file. The reference to '%1' will be ignored.</td>
<td>To remove this message, you can set the toc=&quot;no&quot; attribute on your topicref.</td>
</tr>
<tr>
<td>DOTX008W</td>
<td>WARN</td>
<td>File '%1' does not exist.</td>
<td>Append the file '%1' into the source or change the href link to existing file. Note: '%1' might be changed to use standard dita topic file extension name '.dita' or '.xml'.</td>
</tr>
<tr>
<td>DOTX009W</td>
<td>WARN</td>
<td>Could not retrieve a title from '%1'. Using '%2' instead.</td>
<td>Add a title in the target topic file.</td>
</tr>
<tr>
<td>DOTX010E</td>
<td>ERROR</td>
<td>Unable to find target for conref=&quot;%1&quot;. Check to make sure that the target element is available, and that it is a 'xxxx' element. Note: '%1' might be changed to use default dita topic file extension name '.dita' or '.xml'.</td>
<td>Check to make sure the target of conref is correct.</td>
</tr>
<tr>
<td>DOTX011W</td>
<td>WARN</td>
<td>There is more than one possible target for conref=&quot;%1&quot;. Only the first will be used. Remove the duplicate ID from one of the targets. Note: '%1' might be changed to use default dita topic file extension name '.dita' or '.xml'.</td>
<td>Remove the duplicated id in the conref target file.</td>
</tr>
<tr>
<td>Message number</td>
<td>Type</td>
<td>Message text</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DOTX012W</td>
<td>WARN</td>
<td>When you conref another topic or an item in another topic, the domains attribute of the target topic must be equal to or a subset of the current topic's domains attribute.</td>
<td>Put your target under an appropriate domain. You can see the messages guide for more help.</td>
</tr>
<tr>
<td>DOTX013E</td>
<td>ERROR</td>
<td>A element with a conref attribute indirectly includes itself, which is not possible. Please fix the target of the conref attribute. The conref attribute points to '%1'. Note: '%1' might be changed to use default dita topic file extension name '.dita' or '.xml'.</td>
<td>Resolve the circle conref.</td>
</tr>
<tr>
<td>DOTX014E</td>
<td>ERROR</td>
<td>The element must provide the id of the target topicref you want to reuse. For example, mymap.ditamap#mytopicrefid.</td>
<td>Put an id into the target position of the content which will be reused.</td>
</tr>
<tr>
<td>DOTX015E</td>
<td>ERROR</td>
<td>Incorrectly formed conref attribute: '%1'. Make sure the syntax is correct and try again. Note: '%1' might be changed to use default dita topic file extension name '.dita' or '.xml'.</td>
<td>Change the conref attribute to conform the correct syntax.</td>
</tr>
<tr>
<td>DOTX016W</td>
<td>WARN</td>
<td>An href value appears to point to a DITA file, but the format attribute is inherited a value of '%1'. If the target '%2's is a DITA file, set the format attribute to &quot;dita&quot;. If it does not point to a DITA file, set the format attribute locally. Note: '%2' might be changed to use standard dita topic file extension name '.dita' or '.xml'.</td>
<td>Directly set the format attribute to correct value instead of inheriting from ancestor.</td>
</tr>
<tr>
<td>DOTX017E</td>
<td>ERROR</td>
<td>Found a topic reference with an empty HREF attribute. The attribute should point to a file or other valid address.</td>
<td>Remove the empty href link or put in some content.</td>
</tr>
<tr>
<td>DOTX018I</td>
<td>INFO</td>
<td>The type attribute on a topicref element does not match the target topic. The type attribute was set to '%1', but the topicref points to a more specific '%2' topic. This may cause your links to sort incorrectly in the output. Note that the type attribute is inherited in maps, so the value '%1' may come from an ancestor topicref.</td>
<td>Check and make the type of topicref match with the actual type of topic.</td>
</tr>
<tr>
<td>DOTX019W</td>
<td>WARN</td>
<td>The type attribute on a topicref element does not match the target topic. The type attribute was set to '%1', but the topicref points to a more specific '%2' topic. This may cause your links to sort incorrectly in the output. Note that the type attribute is inherited in maps, so the value '%1' may come from an ancestor topicref.</td>
<td>Check and make the type of topicref match with the actual type of topic.</td>
</tr>
<tr>
<td>DOTX020E</td>
<td>ERROR</td>
<td>Missing navtitle attribute for: '%1' When you set scope=&quot;peer&quot; you must provide a local navigation title, since the target is not accessible at build time.</td>
<td>Provide a local navigation title in ditamap.</td>
</tr>
<tr>
<td>DOTX021E</td>
<td>ERROR</td>
<td>Missing navtitle attribute for: '%1' When you set format to a non-DITA resource type, you must provide a local navigation title, since the target is not accessible to DITA-aware transforms.</td>
<td>Provide a local navigation title in ditamap.</td>
</tr>
<tr>
<td>Message number</td>
<td>Type</td>
<td>Message text</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOTX022W</td>
<td>WARN</td>
<td>Unable to retrieve navtitle from target: ‘%1’. Using linktext under topicmeta of this topicref instead of the navigation title.</td>
<td>Make sure the topicref type matches the target, and that the file name and topic id are correct.</td>
</tr>
<tr>
<td>DOTX023W</td>
<td>WARN</td>
<td>Unable to retrieve navtitle from target: ‘%1’.</td>
<td>Make sure the topicref type matches the target, and that the file name and topic id are correct.</td>
</tr>
<tr>
<td>DOTX024E</td>
<td>ERROR</td>
<td>Missing linktext and navtitle for: ‘%1’. When you set scope=“peer” you must use the navtitle attribute, since the target is not accessible at build time.</td>
<td>Provide a navtitle attribute for topicref.</td>
</tr>
<tr>
<td>DOTX025E</td>
<td>ERROR</td>
<td>Missing linktext and navtitle for: ‘%1’. When you set format to a non-DITA resource type, you must provide a navtitle attribute, since the target is not accessible to DITA-aware transforms.</td>
<td>Provide a navtitle attribute for topicref.</td>
</tr>
<tr>
<td>DOTX026W</td>
<td>WARN</td>
<td>Unable to retrieve linktext from target: ‘%1’. Using navtitle instead of linktext under topicmeta.</td>
<td>Make sure the topicref type matches the target, and that the file name and topic id are correct.</td>
</tr>
<tr>
<td>DOTX027W</td>
<td>WARN</td>
<td>Unable to retrieve linktext from target: ‘%1’.</td>
<td>Make sure the topicref type matches the target, and that the file name and topic id are correct.</td>
</tr>
<tr>
<td>DOTX028E</td>
<td>ERROR</td>
<td>Link or Xref must contain an unempty href attribute.</td>
<td>Add an unempty href attribute for link or xref element.</td>
</tr>
<tr>
<td>DOTX029I</td>
<td>INFO</td>
<td>The type attribute on a %1 element does not match the target %2. The type attribute was set to %3, but the %1 points to a more specific %4 element. This may cause your links to sort incorrectly in the output, and link text may not be properly retrieved. Note that the type attribute is inherited in maps, so the value ’%3’ may come from an ancestor element.</td>
<td>Check and make the type of element match with the actual type of target.</td>
</tr>
<tr>
<td>DOTX030W</td>
<td>WARN</td>
<td>The type attribute on a %1 element does not match the target %2. The type attribute was set to %3, but the %1 points to a more specific %4 element. This may cause your links to sort incorrectly in the output, and link text may not be properly retrieved. Note that the type attribute is inherited in maps, so the value ’%3’ may come from an ancestor element.</td>
<td>Check and make the type of element match with the actual type of target.</td>
</tr>
<tr>
<td>DOTX031E</td>
<td>ERROR</td>
<td>The file %1 is not available to resolve link information. Either the file could not be found, or a DITAVAL file was used to remove the file’s contents. Be aware that the path information above may not match the link in your topic.</td>
<td>Check to make sure the file exists and DITAVAL file isn’t used to remove the contents of the file.</td>
</tr>
<tr>
<td>DOTX032E</td>
<td>ERROR</td>
<td>Unable to retrieve link text from target: ‘%1’.</td>
<td>Make sure the link type matches the target, the ids for topic and element are correct, and that the target has a title. If the target is not accessible at build time,</td>
</tr>
<tr>
<td>Message number</td>
<td>Type</td>
<td>Message text</td>
<td>Action</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOTX033E</td>
<td>ERROR</td>
<td>Unable to find the target to determine the list item number.</td>
<td>Make sure the link type matches the target, and that the ids for topic and element are correct.</td>
</tr>
<tr>
<td>DOTX034E</td>
<td>ERROR</td>
<td>You are cross-referencing a list item in an unordered list. The process could not automatically generate cross-reference text, since the list item is not numbered.</td>
<td>You need to provide cross-reference text within the xref element, or you need to change the target to something the process can support. Using the id of the target as cross-reference text for now.</td>
</tr>
<tr>
<td>DOTX035E</td>
<td>ERROR</td>
<td>Unable to find the target to determine the footnote number.</td>
<td>Make sure the link type matches the target, and that the ids for topic and element are correct.</td>
</tr>
<tr>
<td>DOTX036E</td>
<td>ERROR</td>
<td>Unable to find the dlentry target to determine the dlterm text.</td>
<td>Make sure the link type matches the target, and that the ids for topic and element are correct.</td>
</tr>
<tr>
<td>DOTX037W</td>
<td>WARN</td>
<td>Topic contains no title; using &quot;****&quot;.</td>
<td>Add a title to your topic.</td>
</tr>
<tr>
<td>DOTX038I</td>
<td>INFO</td>
<td>The LONGDESCREF attribute on tag &quot;%1&quot; will be ignored. Accessibility needs to be handled another way.</td>
<td>To make the object accessible, you may need to add text before or after the element. You may also be able to handle it with a &lt;param&gt; element inside the object.</td>
</tr>
<tr>
<td>DOTX039W</td>
<td>WARN</td>
<td>Required cleanup area found.</td>
<td>If the output is only used as a draft, you do not need to do anything. If you are producing production-level output, you should not use the /DRAFT option.</td>
</tr>
<tr>
<td>DOTX040I</td>
<td>INFO</td>
<td>Draft comment area found.</td>
<td>If the output is only used as a draft, you do not need to do anything. If you are producing production-level output, you should not use the /DRAFT option.</td>
</tr>
<tr>
<td>DOTX041W</td>
<td>WARN</td>
<td>More than one title element in a section. Using the first one for the section's title.</td>
<td>Make sure that the title you wish to appear as a title is the first one. Then remove other title elements, or change them to a more appropriate element. As a last resort you could simply make them bold phrases.</td>
</tr>
<tr>
<td>DOTX042I</td>
<td>INFO</td>
<td>Flagging attribute found on &quot;%1&quot; attribute. Inline phrases cannot be flagged.</td>
<td>If it is important to flag this piece of information, try placing a flag on the block element that contains your phrase. If you just want to have an image next to the image, you may place an image directly into the document.</td>
</tr>
<tr>
<td>DOTX043I</td>
<td>INFO</td>
<td>The link to &quot;%1&quot; may appear more than once in &quot;%2&quot;.</td>
<td>If you do not want to occurrences of a link to the same href, remove one of the links or define the same attributes on both link elements. Note that links generated from</td>
</tr>
<tr>
<td>Message number</td>
<td>Type</td>
<td>Message text</td>
<td>Action</td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOTX044E</td>
<td>ERROR</td>
<td>Area element has no cross-reference HREF attribute. The area requires a cross-reference with an HREF attribute.</td>
<td>Add a href attribute to the xref element.</td>
</tr>
<tr>
<td>DOTX045W</td>
<td>WARN</td>
<td>Area element contains a cross-reference that is missing link text. The area recommends a cross-reference that contains link text; either from the referenced topic's title, or from the content of the cross-reference. Because there was no cross-reference content; the HREF attribute value is being used.</td>
<td>Add link text to the xref element.</td>
</tr>
<tr>
<td>DOTX046W</td>
<td>WARN</td>
<td>Area shape should be: default, blank (no value), rect, circle, or poly. This value is not recognized: '%1'. It was passed as-is through to the area element in the XHTML.</td>
<td>Correct the shape value.</td>
</tr>
<tr>
<td>DOTX047W</td>
<td>WARN</td>
<td>Area coordinates are blank. Coordinate points for the shape need to be specified.</td>
<td>Correct the coords value.</td>
</tr>
<tr>
<td>DOTX048I</td>
<td>INFO</td>
<td>In order to include '%1' in your help file, you will need to recompile the CHM file locally. The automatically compiled CHM file will only contain formatted DITA files, not files that are already in HTML.</td>
<td>For local HTML files, you will need to recompile your help project after all of your files have been returned. For external web sites, you can set the scope attribute to &quot;external&quot; to avoid this message. If you are linking to an actual HTML file that will not be available, it cannot be included in the project. You should set the toc attribute to &quot;no&quot; on your topicref element.</td>
</tr>
<tr>
<td>DOTX049I</td>
<td>INFO</td>
<td>Topicref to non-dita files will be ignored in PDF or Word transformation.</td>
<td>If you want to include these files in PDF or Word output, you may need to change them to dita format.</td>
</tr>
<tr>
<td>DOTX050W</td>
<td>WARN</td>
<td>Cannot find id attribute in map element.</td>
<td>Default id &quot;org.sample.help.doc&quot; is used for plugin. If you want to use your own id, please specify it in id attribute of map.</td>
</tr>
<tr>
<td>DOTX051W</td>
<td>WARN</td>
<td>The '%1' to '%2' with @format='%3' can not be recognized, output it without creating a hyperlink.</td>
<td>Set @format='dita' for %1 to DITA topic, set @format='html' for HTML files, other format is not supported.</td>
</tr>
<tr>
<td>DOTX052W</td>
<td>WARN</td>
<td>No string named '%1' was found. Using original value.</td>
<td>Add translation for the string '%1' in the default language, then add mapping between default language and other languages for it.</td>
</tr>
</tbody>
</table>
Messages generated from other sources

Other Java messages

<table>
<thead>
<tr>
<th>Message text</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>[pipeline] [Error] :13:39: Element type &quot;...&quot; must be declared.</td>
<td>An error has occurred parsing a DTD.</td>
</tr>
<tr>
<td>[pipeline] [Error] :14:13: The content of element type &quot;...&quot; must match &quot;...&quot;.</td>
<td>An error has occurred parsing a DTD.</td>
</tr>
<tr>
<td>BUILD FAILED C:\sandbox\ant\dotug_xhtml.xml:24: The following error occurred while executing this line:</td>
<td></td>
</tr>
<tr>
<td>C:\sandbox\conductor.xml:101: The following error occurred while executing this line: java.lang.OutOfMemoryError</td>
<td>Java does not have enough memory allocated to run the build. Change ANT_OPTS to a larger value, for example, ANT_OPTS=-Xmx256M. (The default value is 16M.)</td>
</tr>
<tr>
<td>Unable to instantiate specified logger class org.dita.log.DITAOTBuildLogger ...</td>
<td>Check that your CLASSPATH variable contains dost.jar.</td>
</tr>
<tr>
<td>Can't find resource\messages.xml</td>
<td>Check that your CLASSPATH variable contains dost.jar.</td>
</tr>
</tbody>
</table>

Troubleshooting CLASSPATH and environment variables setup

If you see a "can't find resource\messages.xml" error message when you try to build using Ant, remove other items from your CLASSPATH variable one-by-one until you find the culprit. Your environment variable setup can be the source of the problem because sometimes other applications or .jar files can override CLASSPATH settings. To assist in troubleshooting the CLASSPATH, create a simple run.bat file that contains only SAXON, Ant, and the DITA Open Toolkit dost.jar paths. For example:

```
set CLASSPATH=C:\saxon\saxon.jar;C:\ant\apache-ant-1.6.5\lib\dost.jar;.
ant demo.faq.
```

>Note: The dot at the end of the set CLASSPATH command ensures that your current directory is included in the CLASSPATH. By requesting a small build like "ant demo.faq" rather than "ant all" you can save time if the build is successful.

By setting the CLASSPATH just for one session and running the batch file within the ditaot directory, you can pinpoint if the problem is your CLASSPATH. Once you get a BUILD SUCCESSFUL message, add in CLASSPATH entries one by one until you find the entry that conflicts with DITA Open Toolkit.

Debugging, reporting, and file generation tools

Sections in this topic:

- About the tools on page 124
- Using the tools on page 124
About the tools

These tools are written in the PHP programming language, and are meant to be invoked either from a command line prompt or from an Ant build script. They all take a single argument, which is the name of a .ditamap file. The tools process all files in the hierarchy below the level of the invoked map.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ditaauthors.php</td>
<td>Returns a set of unique author and copyright strings, with counts for each.</td>
</tr>
<tr>
<td>ditadebug.php</td>
<td>Produces several debugging and informational reports, most of them in a format that can be imported into a spreadsheet or a database. The reports include:</td>
</tr>
<tr>
<td></td>
<td>• A list of any incorrect references found in the source files</td>
</tr>
<tr>
<td></td>
<td>• A list of directories containing files used by the map</td>
</tr>
<tr>
<td></td>
<td>• For each file in the map, its name, type, author, size, date last modified, and title</td>
</tr>
<tr>
<td></td>
<td>• The total number of files by type</td>
</tr>
<tr>
<td></td>
<td>• A list of all references found</td>
</tr>
<tr>
<td>ditaedit.php</td>
<td>Can be used to search for and replace strings in all files in the DITA map.</td>
</tr>
<tr>
<td>ditaidls.php</td>
<td>Produces an alphabetical list of IDs for all files in the map, with duplicates marked with an asterisk.</td>
</tr>
<tr>
<td>ditakeys.php</td>
<td>Produces a list of all metadata keywords defined.</td>
</tr>
<tr>
<td>ditalinks.php</td>
<td>Tests all URLs referenced by files in the map for validity.</td>
</tr>
<tr>
<td>ditapreview.php</td>
<td>Does incremental builds on small numbers of recently written files.</td>
</tr>
</tbody>
</table>

Using the tools

Tools overview

Because the authors of this document needed debugging, reporting, and automatic file generation support not available in DITA Open Toolkit, they produced several tools of their own. These tools are now available as part of the Toolkit.

Message topic generator (ditamsg_generator.xsl)

An XSLT stylesheet (ditamsg_generator.xsl) was used to automate the creation of the Toolkit messages reference topic (Messages generated by the Toolkit on page 116). This stylesheet could be useful to other DITA users as an example of how to handle a similar task when documenting other software, especially when the product messages are stored in a valid XML file.

DITA debugging tools (ditadebug.php, ditaedit.php and ditalinks.php)

Three debugging tools were written to deal with the following kinds of problems that were encountered while writing this document.

Cross-referencing problems handled by ditadebug.php

- An error is generated during the build that indicates a file cannot be found or opened. The Toolkit message log tells you the name of the file it cannot find, but not which file is referring to it. This makes it difficult to find the source of the error.
- Some errors in the DITA source files do not produce any build error messages at all, but the output produced is incorrect. These kinds of problems are subtle and can be very difficult to troubleshoot.

Examples of the second case include the following:
• When running under Windows, a referenced filepath uses the wrong case of one of the elements, for example, ../Dir1/fn.ft instead of ../dir1/fn.ft.
• A cross-reference points to an ID in a file that does not exist.

The ditadebug.php tool also helps to answer the following kinds of questions:

• Which directories contain files used by a given ditamap?
• Which files in source directories are not used in a ditamap, and therefore could be erased?
• For a given file referenced in a ditamap, what type of file is it, who was its author, how big is it, when was it last changed, and what is it about?
• Which URLs are referenced in the files used in a ditamap?
• What are all the linked connections from one file to another used by a given ditamap?

String search and replace with ditaedit.php

The ditaedit.php tool can be used to search for strings in all files in a DITA map. It can also be used to replace strings in all files.

URL checking problems handled by ditalinks.php

The ditalinks.php tool checks external URLs and verifies that they exist.

⚠️ Note: The only kind of URL this tool cannot handle is one requiring login to a website. The tool reports these with the message "URL may not exist." You will need to verify these manually.

DITA reporting tools (ditaauthors.php, ditaids.php, and ditakeys.php)

While writing this document we found it useful to have several types of reporting information derived from the source files:

• Who are the authors and copyright holders?
• Which IDs were defined in the source file and were any of them duplicates?
• Which keywords were used in the source file <prolog> elements?

The ditaauthors.php, ditaids.php, and ditakeys.php tools were written to provide answers to these questions.

Build preview tool (ditapreview.php)

The ditapreview.php tool dynamically builds a DITA map file for only the most recently modified topics and then invokes Internet Explorer to display them.

Software prerequisite required

To run these tools you must have the PHP interpreter installed on your build machine. PHP is a free tool that can be downloaded from http://www.php.net.

Example of how to invoke the tools

You can include any of these tools as part of an Ant build script. The following example shows how to run the debugging tool and write the output to a file.

```xml
<!-- create the ditamap debug cross-reference -->
<target name="debug">
  <echo>Building debugging file ditadebug.txt</echo>
  <mkdir dir="${outdir}/debug_files"/>
  <exec executable="${PHPdir}/php.exe" dir = "${projdir}" output="${outdir}/debug_files/ditadebug.txt">
    <arg value="${projdir}/project/tools/ditadebug.php"/>
    <arg value="${MAP_file}"/>
  </exec>
</target>
```
DITAblogs (troubleshooting)

Sections in this topic:

  Cross-reference debugging tool (ditadebug.php) on page 126
  Message topic generation tool (ditamsg_generator.xsl) on page 127

Cross-reference debugging tool (ditadebug.php)

Once we had large numbers of source files and directories to deal with, we ran into the following kinds of error situations that were difficult to resolve:

  • We had problems finding out the root cause of error messages in the Ant build log.
  • We lost track of which source files had references to other source files.
  • We often didn't know which URLs were linked to in the source files.
  • We wondered which source files were not actually being used.

Here is an example of an error message generated by the Toolkit that is caused by a bad href. Notice that the message tells you which file is referenced, but not the location of the original reference.

```
[pipeline] [DOTJ013E][ERROR] Failed to parse the referenced file 'installing
stalling_fo.dita' due to below exception.
Please correct the reference base on the exception message.
[pipeline] java.io.FileNotFoundException:
C:\DITAOT_UGRef_SOURCE\installing\nstalling_fo.dita
(The system cannot find the file specified)
```

Partly as a learning exercise, and partly to allow us to address these issues, we wrote a build tool that starts from a DITA map file and builds a set of cross-reference and error reports for the files used by the DITA map. This can be done because all the files that make up a DITA source tree have to be well-formed and valid XML files. Standard XML parsing libraries can be used to "walk" the set of source files included by the DITA map.

Our PHP script (ditadebug.php) uses the SimpleXML PHP library routines. We added an Ant target to our build script that allows us to run this tool every time we build the book. For the same error shown above, our tool produces the following error message:

```
Error, file C:\DITAOTUG_SOURCE\processing\..\installing\nstalling_fo.dita does not exist!
Bad reference: C:\DITAOTUG_SOURCE\processing\processing_pdf2.dita => ..\installing\nstalling_fo.dita
```

Now we know which file is missing and where the bad reference is! The PHP script is available as part of the DITA Open Toolkit documentation package.

Here is a subset of a ditadebug-generated report for this document that illustrates the types of information it generates.

```
Starting from ditamap DITAOTUG.ditamap
dir: .
file: DITAOTUG.ditamap

6 unused files in directories used by this map:
C:\DITAOTUG_SOURCE\images\dita_finch_logo.jpg , *No DOCTYPE*
...
C:\DITAOTUG_SOURCE\ugxref.txt , *No DOCTYPE*

27 directories in this map:
C:\DITAOTUG_SOURCE
```
Message topic generation tool (ditamsg_generator.xsl)

Below is a listing for the XSLT stylesheet used to read the DITA Open Toolkit message repository resource/messages.xml file and convert it to a DITA reference topic (see Messages generated by the Toolkit on page 116). This reference topic has been generated multiple times during the production cycle of the DITA Open Toolkit User Guide, as the Toolkit moved to new point releases.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v4.2 -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" encoding="ISO-8859-1" indent="yes"
doctype-public="-//OASIS//DTD DITA Reference//EN"
doctype-system="http://docs.oasis-open.org/dita/v1.0.1/dtd/reference.dtd"/>
    <!-- Stylesheet to convert messages.xml to a DITA reference topic messages.dita -->
    <!-- Author: Dick Johnson 05/27/2006 -->
    <xsl:template match="/messages">
        <reference id="messages">
            <title>DITA Open Toolkit Messages</title>
            <refbody>
                <!-- put all the Ant messages in a simple table -->
                <section id="ant">
                    <title>Ant messages</title>
                    <p></p>
                    <simpletable>
                        <sthead>
                            <stentry>Message number</stentry>
                            <stentry>Type</stentry>
                            <stentry>Message text</stentry>
                            <stentry>Action</stentry>
                        </sthead>
                        <xsl:apply-templates select="message[substring(@id,1,4)='DOTA']"/>
                    </simpletable>
                </section>
            </refbody>
        </reference>
    </xsl:template>
</xsl:stylesheet>
```
<!-- put all the Java messages in a simple table -->
<section id="java">
<title>Java messages</title>
<p></p>
<simpletable>
<sthead>
<stentry>Message number</stentry>
<stentry>Type</stentry>
<stentry>Message text</stentry>
<stentry>Action</stentry>
</sthead>
<xsl:apply-templates select="message[substring(@id,1,4)='DOTJ']" />
</simpletable>
</section>

<!-- put all the XSLT messages in a simple table -->
<section id="xslt">
<title>XSLT messages</title>
<p></p>
<simpletable>
<sthead>
<stentry>Message number</stentry>
<stentry>Type</stentry>
<stentry>Message text</stentry>
<stentry>Action</stentry>
</sthead>
<xsl:apply-templates select="message[substring(@id,1,4)='DOTX']" />
</simpletable>
</section>

<!-- Reformat an individual message -->
<xsl:template match="message">
<strow>
<stentry>
<msgnum>
<xsl:apply-templates select="@id" /></msgnum>
</stentry>
<stentry>
<xsl:apply-templates select="@type" />
</stentry>
<stentry>
<msgph>
<xsl:apply-templates select="reason" /></msgph>
</stentry>
<stentry>
<xsl:apply-templates select="response" />
</stentry>
</strow>
</xsl:template>

<xsl:template match="description">
<p>
<td><xsl:value-of select="." /></td>
</p>
</xsl:template>
For more information (troubleshooting)

We found the O'Reilly Java series book, *Ant: The Definitive Guide*, by Steve Holzner, useful in helping us work through processing problems as we produced this document.
Creating DITA topics

This section contains information on how to create DITA topics (base topics, concepts, tasks, and reference topics).

This chapter and the next one are a short tutorial; you should work through the topics chapter (this one) before the maps chapter. The key concepts and tasks in this chapter are meant to be read and performed in the order shown below.

Sections in this topic:

- About the grocery shopping sample on page 131
- About topics on page 134
- Creating topics on page 134
- About concepts on page 135
- Creating concepts on page 135
- About tasks on page 136
- About reference information on page 137
- Creating reference topics on page 138
- Processing (building) a single topic on page 139
- DITAblogs (topics) on page 139
- For more information (topics) on page 140

About the grocery shopping sample

The grocery shopping sample, which is downloadable from http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html, is a simple DITA project that includes seven topics: an overview topic, two concepts, two tasks, and two reference topics. The project also includes a map that aggregates the files and links them meaningfully using a relationship table. Ant scripts that process to all DITA Open Toolkit targets are also provided.

Before you begin to use the grocery shopping sample files, we recommend creating two directories in your root directory called DITASAMPLE_GROCERY_SOURCE and DITASAMPLE_GROCERY_OUTPUT (Windows examples would be C:/DITASAMPLE_GROCERY_SOURCE and C:/DITASAMPLE_GROCERY_OUTPUT) and then copying into them both the grocery shopping sample files DITASAMPLE_GROCERY_SOURCE. Your directory structure should then look like this:
The grocery shopping sample assumes you are already familiar with the garage sample provided as both a project model as well as a tool to verify your DITA Open Toolkit installation, and that you have processed the garage sample as described in Processing (building) and publishing DITA documents on page 75.

Creating and processing the grocery shopping sample

You will be working in the DITASAMPLE_GROCERY_SOURCE directory, which contains a number of subdirectories.

Files in the template subdirectory provide you with a starting point for each file in your grocery shopping project. The first step in each task in this chapter is to copy a file from the template directory to the working directory. Then edit the "working" version of the file, as instructed.

If you need help along the way, you can use the files in the completed directory for reference.

The Ant scripts (in the ant_scripts directory) assume you will process (build) the "working" version of the files. Ant scripts that build the "completed" files are also provided.

If you follow the instructions in this chapter and the next ("Creating Maps") you will have your own working version of the sample, which you can modify to try and test DITA and Toolkit features not included in the sample files.

About topic-based authoring (structured writing)

A topic is a unit of information with a title and content, short enough to be specific to a single subject. A topic should be short enough to be easily readable, but long enough to make sense on its own.

A document may contain one topic or multiple topics, and a document type might support authoring one or many kinds of topics.

Regardless of where they occur, all topics have the same basic structure and capabilities. Books, PDF files, websites, and help sets, for example, can all be constructed from the same set of underlying topic content, although there may be some topics that are unique to a particular deliverable, and the organization of topics may differ to take advantage of the unique capabilities of each delivery mechanism.
Reference information is inherently topic-oriented, since it requires information to be modular and self-contained for the sake of retrievability.

**What is the origin of topic-based authoring?**

Topic-oriented authoring for conceptual and task information has its roots in Minimalism, an instructional design technique first espoused by John Carroll. The minimalist approach to information design focuses on identifying the smallest amount of instruction that allows for the successful completion of a task, or that provides basic knowledge of a concept. Readers have goals, and they want to achieve those goals as quickly as possible. Generally, readers don't want to read information just for the pleasure of reading. They are reading to learn to do something.

Some of the key principles of Minimalism are:

- Support actions. Let people act as they learn, and let them pursue the goals they want to accomplish.
- Document tasks, not tools or functions.
- Help readers anticipate and avoid errors.
- Let readers explore. They don't need to have explained what they can discover for themselves.

**When did topic-based authoring first become popular?**

Topic-based authoring first gained popularity with technical and professional writers when online help systems appeared in the mid-1990s. Writers of help systems quickly learned that they couldn't simply split existing books into help topics by making every heading level a new help page. Information architects (a term that originally referred to architects of online, but not printed information) needed to rethink the structure and content of help systems, and create a new set of standards for online information development. The result is topic-based authoring.

Today topic authoring is almost as popular with written information as with online help systems. The techniques used in topic authoring provide information developers with a way to create distinct modules of information that can stand alone for users. Each topic answers one question: "How do I ...?" "What is ...?" "What went wrong?" Each topic has a title to name its purpose and contains enough content for someone to begin and complete a task, grasp a basic concept, or look up critical reference information. Each topic has a carefully defined set of the basic content units that are required and accommodates other optional content. As information developers learn to author in topics and follow sound topic authoring guidelines consistently, they gain the ability to offer information written by many different experts that looks and feels the same to users.

**What are the benefits of topic-based authoring?**

Authoring in structured topics can decrease development costs and time to market, and provide increased value to customers:

- Structured topics contain only the information needed to understand one concept, perform one procedure, or look up one set of reference information.
- Structured, topic-based authoring promotes consistency in the presentation of similar information.
- Topics can be reviewed by subject-matter experts as soon as they are ready.
- Topics can be translated before entire volumes are complete, reducing the time to market for global customers.
- Assembling topics into multiple deliverables can be automated, reducing production time and costs.
- Consistently structured topics are easier to reuse in multiple deliverables.
- Structured topics may be combined in new ways to meet changes in product solutions, work structures, geographies, industries, or other customer configurations.
- Topics are easier to update immediately instead of waiting for the next release of an entire library of documents.
- Consistently structured topics help users build a firm mental model of the types of information being presented.
- Consistently structured topics help users navigate more quickly to the information they need.
Why does topic-based writing continue to be popular today?

By organizing content into topics, authors can achieve several goals simultaneously:

- Content is readable even when accessed from an index or search, not just when read in sequence as part of a chapter. Since most readers don't read information end-to-end, it's good information design to make sure each unit of information can be read on its own to give just-in-time help.
- Content can be organized differently for online and print purposes. Authors can create task flows and concept hierarchies for online orientation, and still have a print-friendly combined hierarchy that helps people who do want an organized reading flow.
- Content can be reused in different collections. Since the topic is written to make sense when accessed randomly (as by search), it should also make sense when included as part of different product deliverables, so authors can refactor information as needed, including just the topics that apply for each reuse scenario.
- In today's highly competitive, global business environment, topic-based writing is one important manifestation of the "faster/better/cheaper" paradigm. Content can be produced more quickly, to a higher level of overall quality, and for less cost.

About topics

The topic is the base DITA information type. Most DITA topics contain content; they can also be container topics that aggregate other topics.

Creating topics

In this topic you will create a simple DITA topic based on a template already provided. You will be working in the DITASAMPLE_GROCERY_SOURCE directory. This topic assumes you are familiar with the information in About the grocery shopping sample on page 131.

1. Go to the template/topics directory.
2. Copy the groceryshopping.dita file to the "working" directory (working/topics).
3. Using your authoring tool, edit the "working" version of the groceryshopping.dita file.
4. In the prolog section of the file, change the text of the author element text to your name.
5. Also in the prolog section, change the text of the copyrholder element to your company name.
6. Also in the prolog section, update the "revised" date of the critdates element.

Your groceryshopping.dita file should now look something like this:

```xml
<topic id="groceryshopping" xml:lang="en-us">
<title>Shopping for groceries</title>
<shortdesc>Tips on buying groceries.</shortdesc>
<prelog>
<author type="creator">Tom McIntyre</author>
<copyright>
<copyryear year="2007"/>
<copyrholder>Acme Company</copyrholder>
</copyright>
<critdates>
<created date="2006-August-07"/>
<revised modified="2007-March-18"/>
</critdates>
<meta data>
<keywords>
<keyword>grocery shopping</keyword>
</keywords>
</metadata>
```
7. Save the changed file.

## About concepts

A concept information type contains content of a conceptual nature.

## Creating concepts

In this topic you will create two concept topics based on templates already provided. You will be working in the DITASAMPLE_GROCERY_SOURCE directory. This topic assumes you are familiar with the information in About the grocery shopping sample on page 131.

1. Go to the template/concepts directory.
2. Copy the about_cannedgoods.dita file to the "working" directory (working/concepts).
3. Using your authoring tool, open the "working" version of about_cannedgoods.dita.
4. In the prolog section, change the text of the author element text to your name, the copyrholder element text to your company name, and the revised element text to the current date.
5. In the conbody section, add a paragraph element with some text about canned goods, and an unordered list with some reasons for buying canned goods.
   Here is a suggestion:
   ```
   <p>Canned goods are easy-to-use and easy-to-store staples in most kitchens. Common canned goods available in almost any grocery store are beans, canned vegetables, and canned fruits.</p>
   <p>You can save money on canned goods by:
   <ul>
   <li>buying from chain or discount grocery stores</li>
   <li>buying larger cans</li>
   </ul>
   </p>
   ```
6. Save the changed file.
7. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.
8. Go back to the template/concepts directory.
9. Copy the about_produce.dita file to the "working" directory (working/concepts).
10. Using your authoring tool, open the "working" version of about_produce.dita.
11. Edit the same prolog elements you did in the about_cannedgoods.dita file.
12. In the conbody section of about_produce.dita, add a paragraph element with some text about produce.
    Here is a suggestion:
    ```
    <p>One of the keys to good health is eating lots of produce.
    ```
It pays to buy fresh fruits and vegetables and serve them often.

You can save money by buying produce when it is in season.

13. Save the changed file.
14. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.

About tasks

A task information type is an information type for content that describes procedures or sets of steps a user follows in performing a task or using a product.

Creating tasks

In this topic you will create two task topics based on templates already provided. You will be working in the DITASAMPLE_GROCERY_SOURCE directory. This topic assumes you are familiar with the information in About the grocery shopping sample on page 131.

1. Go to the template/tasks directory.
2. Copy the buying_cannedgoods.dita file to the "working" directory (working/tasks).
3. Using your authoring tool, open the "working" version of buying_cannedgoods.dita.
4. In the prolog section, change the text of the author element text to your name, the copyrholder element text to your company name, and the revised element text to the current date.
5. In the taskbody section, add a series of steps to describe the process of buying a can of olives.
   Here is a suggestion:

   <context>Canned goods are usually stored on grocery store shelves by type of food—for example, all canned vegetables in one aisle and all canned fruits in another. Say you are looking for canned olives:</context>
   <steps>
   <step>
   <cmd>Find the olive display by reading the directional signs or asking a store clerk for help.</cmd>
   </step>
   <step>
   <cmd>Locate the type of olives you want to buy: green or black.</cmd>
   <info>If you're looking for ingredients for a green salad, green olives might be a better choice. If you're making enchiladas, look for cans of black olives.</info>
   </step>
   <step>
   <cmd>Check the sizes and prices to determine the best buy.</cmd>
   <info>For example, if you're planning to make enchiladas tonight and tacos on Friday, a larger can would probably be a better buy.</info>
   </step>
   <step>
   <cmd>Select a can and look it over carefully to be sure it has no dents that would cause the seal to be broken.</cmd>
   </step>
   </steps>
6. Save the changed file.
7. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.
8. Go back to the groceryshopping/template/tasks directory.
9. Using your authoring tool, open the "working" version of buying_cannedgoods.dita.
10. In the prolog section, change the text of the author element text to your name, the copyrholder element text to your company name, and the revised element text to the current date.
11. In the taskbody section, add a series of steps about how to choose and buy peaches.

Here is a suggestion:

```dita
<prereq>Do your produce shopping <i>after</i> you have selected your canned goods. 
Otherwise, the cans might bruise the fruits and vegetables!</prereq>
<context>Remember to look for local produce in season. 
The fruits and vegetables you buy will be fresher and cheaper! 
Say you’re shopping in August for peaches grown locally. 
When you get to the produce section of your grocery store:</context>
<steps>
  <step>
    <cmd>Get a plastic or paper bag to hold the peaches.</cmd>
  </step>
  <step>
    <cmd>Pick out the freshest peaches you can find, and put them gently into your bag.</cmd>
    <info>To avoid bruising, don’t put more than 6 peaches in each bag.</info>
  </step>
  <step>
    <cmd>Put the bag gently into your grocery cart.</cmd>
  </step>
</steps>
$postreq>When you check out, be sure the grocery clerk also handles your peaches carefully.</postreq>
```

12. Save the changed file.
13. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.

### About reference information

Reference information type is an information type for content that focuses on properties and relationships among a number of similar items.

Content in a DITA reference information type is used to record and present (often in a tabular format) reference (as contrasted with narrative) information. The information is presented to users in a way that facilitates quick lookup.
Creating reference topics

In this topic you will create two reference topics based on templates already provided. You will be working in the DITASAMPLE_GROCERY_SOURCE directory. This topic assumes you are familiar with the information in About the grocery shopping sample on page 131.

1. Go to the template/reference directory.
2. Copy the cannedgoods.dita file to the "working" directory (working/reference).
3. Using your authoring tool, open the "working" version of cannedgoods.dita.
4. In the prolog section, change the text of the author element text to your name, the copyrholder element text to your company name, and the revised element text to the current date.
5. In the refbody section, add a simple table showing product name, can size, and price for several canned goods products.
   Here is a suggestion:

   ```
   <section>
   <simpletable>
   <sthead>
   <stentry>Product</stentry>
   <stentry>Can size</stentry>
   <stentry>Price</stentry>
   </sthead>
   <strow>
   <stentry>Large black olives</stentry>
   <stentry>14 oz</stentry>
   <stentry>$2.39</stentry>
   </strow>
   <strow>
   <stentry>Small black olives</stentry>
   <stentry>6 oz</stentry>
   <stentry>$1.78</stentry>
   </strow>
   <strow>
   <stentry>Large green stuffed olives</stentry>
   <stentry>20 oz</stentry>
   <stentry>$4.56</stentry>
   </strow>
   <strow>
   <stentry>Small green plain olives</stentry>
   <stentry>8 oz</stentry>
   <stentry>$2.45</stentry>
   </strow>
   </simpletable>
   </section>
   ```
6. Save the changed file.
7. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.
8. Go back to the groceryshopping/template/reference directory.
9. Copy the produce.dita file to the "working" directory (working/reference).
10. Using your authoring tool, open the "working" version of produce.dita.
11. Edit the same prolog elements you did in the cannedgoods.dita file.
12. In the refbody section, add a simple table showing product name, can size, and price for several produce items.
Here is a suggestion:

```xml
<section>
  <simpletable>
    <sthead>
      <stentry>Item</stentry>
      <stentry>Type</stentry>
      <stentry>Price</stentry>
    </sthead>
    <strow>
      <stentry>Apple</stentry>
      <stentry>Fuji</stentry>
      <stentry>$.88/lb</stentry>
    </strow>
    <strow>
      <stentry>Apple</stentry>
      <stentry>Granny Smith</stentry>
      <stentry>$1.05/lb</stentry>
    </strow>
    <strow>
      <stentry>Pear</stentry>
      <stentry>Bartlett</stentry>
      <stentry>$.74/lb</stentry>
    </strow>
    <strow>
      <stentry>Orange</stentry>
      <stentry>Valencia</stentry>
      <stentry>$1.46/lb</stentry>
    </strow>
  </simpletable>
</section>
```

13. Save the changed file.
14. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.

**Processing (building) a single topic**

You can process (build) a single DITA topic by using its name in place of a ditamap's name in any of the Ant scripts.

**Note:** If you want to try processing a single file, you can modify one of the Ant scripts in the `groceryshopping/ant_scripts` directory to build one of the topics you created in this chapter. If you don't feel confident doing that yet, work through the maps chapter (following this one) first, where you'll learn more about processing DITA files with Ant.

**DITAblogs (topics)**

We used *Introduction to DITA: A User Guide to the Darwin Information Typing Architecture* by Jennifer Linton and Kylene Bruski to teach ourselves some of the basics of DITA. We recommend this book to others looking for a comprehensive, scenario-based approach to learning about DITA. For more information about this book, go to [http://www.comtech-serv.com](http://www.comtech-serv.com).
For more information (topics)

For a complete set of reference topics for DITA language elements, see the language reference document in the DITAOT/demo directory.
Creating DITA maps

This section contains information on how to create DITA maps to define content structure.

This chapter and the previous one are a short tutorial; you should work through the topics chapter before the maps chapter (this one). The key concepts and tasks in this chapter are meant to be read and performed in the order shown below.

Sections in this topic:

About maps on page 141
Creating maps on page 142
Processing (building) the grocery shopping sample on page 143
DITAblogs (maps) on page 144
For more information (maps) on page 145

About maps

A map is an aggregation of the topics in a DITA document, with the topics arranged as a list or a hierarchy. DITA documents can have multiple maps or sets of maps for a given document. For example, a software product available for both Windows and Linux might have two maps, each specifying the topics to include in that document version. As another example, a large, complex document might have a master map that included multiple submaps, specifying the topics to include in various "chapters" and "sections."

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- (c) Copyright 2006-2007 by VR Communications, Inc. All rights reserved. -->
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "../dtd/map.dtd">
<map id="customizing_map" title="Customizing your published output">
  <topicref href="customizing.dita"/>
  <topicref href="condproc.dita"/>
  <topicref href="aboutcondproc.dita"/>
  <topicref href="condproc_using.dita"/>
  <topicref href="addmeta.dita"/>
  <topicref href="filtcontent.dita"/>
  <topicref href="flagcontent.dita"/>
  <topicref href="showrev.dita"/>
</map>
```
Creating maps

In this topic you will create a map to aggregate the topics you created in the previous chapter. The map is based on a template already provided. The map file includes topicrefs to the topics you want to aggregate, process, and publish, and also a relationship table to link the included topics in a meaningful way. You will be working in the DITASAMPLE_GROCERY_SOURCE directories. This topic assumes you are familiar with the information in About the grocery shopping sample on page 131, and that you have created the topics according to the instructions in Creating DITA topics on page 131.

1. Go to the template directory.
2. Copy the groceryshopping_map.ditamap file to the working directory.
3. Using your authoring tool, open the "working" version of groceryshopping_map.ditamap.

Your working map file initially looks like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE map PUBLIC "-/OASIS//DTD DITA Map//EN" "../dtd/map.dtd">
<!-- This is a template file -->
<!-- The "groceryshopping" topic page is a mini-toc for the concept, task, and reference pages, which are displayed sequentially. You could display those pages in any order. -->
<map title="Grocery shopping">
  <topicref href="topics/groceryshopping.dita" type="topic">
  <!-- The concept, task, and reference topicrefs go here -->
  </topicref>
  <!-- The relationship table goes below -->
  <!-- The related concept, task, and reference files point to each other -->
</map>
```

4. Add nested topicrefs for your concept, reference and task files.

The topicref section of your file should look like this:

```xml
<topicref href="topics/groceryshopping.dita" type="topic">
  <topicref href="concepts/about_produce.dita" type="concept"/>
  <topicref href="concepts/about_cannedgoods.dita" type="concept"/>
  <topicref href="tasks/choosing_produce.dita" type="task"/>
  <topicref href="tasks/buying_cannedgoods.dita" type="task"/>
  <topicref href="reference/produce.dita" type="reference"/>
  <topicref href="reference/cannedgoods.dita" type="reference"/>
</topicref>
```

The concepts, tasks, and reference topics will all be nested within the groceryshopping topic. Notice how nesting is accomplished: the closing topicref tag for the groceryshopping topic appears below the topicref for cannedgoods.

5. Below the relationship table comment lines, add a relationship table linking your produce and canned goods topics together.

The relationship table section of your file should look like this:

```xml
<!-- Relationship table -->
<!-- The related concept, task, and reference files point to each other -->
<reltable>
  <relheader>
    <relcolspec type="concept"/>
    <relcolspec type="task"/>
    <relcolspec type="reference"/>
  </relheader>
</reltable>
```
Because we have concept, task, and reference information for each conceptual "topic" (the topics would be "canned goods" and "produce") in our document, we have chosen a three-column table that links all the topics about canned goods, and also links all the topics about produce. There are other ways to design a relationship table.

6. Save the changed file.
7. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.

**Processing (building) the grocery shopping sample**

In this topic you will process (build) the map you created in *Creating maps* on page 142. You will be working in the DITASAMPLE_GROCERY_SOURCE/antScripts directory. This topic assumes you will be building with the Ant scripts in the working subdirectory, but you can also build the completed files by using the Ant scripts in the completed subdirectory. This topic also assumes you are familiar with the information in *About the grocery shopping sample* on page 131, and that you have created the topics according to the instructions in *Creating DITA topics* on page 131. If you need more information about Ant or Ant scripts, see *About Ant* on page 75 or *About Ant scripts* on page 75.

1. Go to the antScripts/working directory.
2. Using your DITA authoring tool or a plain text editor, open the version of the Ant script you want to run. Process to the XHTML target environment.
3. Make sure the Ant script is set up correctly for your environment.
4. In the Command Prompt, navigate to the antScripts/working directory.
5. Invoke the Ant script for the XHTML target.
   For example: ant -f grocery_all.xml dita2xhtml
   Alternatively, if you are processing in the Windows environment, you can use the grocery shopping sample batch script (runbuild.bat): the command is runbuild dita2xhtml. The batch script is in the DITASAMPLE_GARAGE_SOURCE/working directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITASAMPLE_GROCERY_OUTPUT directory during processing.
   Alternatively, if you are processing in the Linux or UNIX environment, you can create your own batch script to perform the same functions as runbuild.bat.
6. Check the output directory to be sure the output files are correct.
7. Build the grocery shopping sample files to the HTML Help target.
8. Build the grocery shopping sample files to the PDF2 target.

Processing using multilevel maps

You can have multilevel maps in DITA projects. Multilevel maps help to organize larger projects (those 50-100 topics or larger).

The following figures show a complete example.

The figure below shows an annotated version of the top-level map ("master map"). Note that you need to include the format="ditamap" attribute statement in each reference to a lower-level map.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "../dtd/map.dtd">
<map title="Sample master map" id="master_map">
  <!-- Basic section -->
  <topicref href="basic/basic_map.ditamap" format="ditamap"/>
  <!-- Advanced section -->
  <topicref href="advanced/advanced_map.ditamap" format="ditamap"/>
  <!-- Glossary -->
  <topicref href="glossary/glossary_map.ditamap" format="ditamap"/>
</map>
```

The figure below shows one of the lower-level maps (for the "advanced" section of the document). Notice that the topics containing content are nested within a container ("landing-page") topic that contain no content. This convention is not required, but it may help to group topics with related content.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "../dtd/map.dtd">
<map title="Advanced section" id="advanced_map">
  <topicref href="advanced.dita">
    <topicref href="about_advanced.dita"/>
    <topicref href="about_topic1.dita"/>
    <topicref href="about_topic2.dita"/>
  </topicref>
</map>
```

DITAblogs (maps)

We used Introduction to DITA: A User Guide to the Darwin Information Typing Architecture by Jennifer Linton and Kylene Bruski to teach ourselves some of the basics of DITA. We recommend this book to others looking for a comprehensive, scenario-based approach to learning about DITA. For more information about this book, go to http://www.comtech-serv.com.
For more information (maps)

For a complete set of reference topics for DITA language elements, see the language reference document in the ditaot/demo directory.
About linking

In DITA, linking content involves various methods that connect topics to each other or to external references. In DITA, linking can be implemented through various elements, such as cross-reference (<xref>) and related links (<related-links>), and through relationship tables.

Linking using cross-references (xrefs)

The following example from this document links a simple table entry in the "Document contents" section of the "About this document" file to the landing page of the "Current release" section.

```xml
<strow>
 <stentry>
  <xref href="../release_current/release_current.dita">Release 1.3.1 information</xref>
 </stentry>
 <stentry>Information about release 1.3.1 of DITA Open Toolkit: system requirements, supported applications, new and enhanced features, upgrade impacts, and known problems.</stentry>
</strow>
```

The following example from this document uses xrefs to create a page table of contents for the "About this document" file.

```xml
<p>Sections in this topic:</p>
<sl>
 <sli>
  <xref href="#aboutditaotugref/contents">Document contents</xref>
 </sli>
 <sli>
  <xref href="#aboutditaotugref/audience">Audience</xref>
 </sli>
```
The following example from this document uses an xref to reference an external website.

```xml
<dl>
  <dt>Ant</dt>
  <dd>Ant 1.6.5. You can download Ant from <xref href="http://ant.apache.org/bindownload.cgi" format="html" scope="external"/></dd>
</dl>
```

## Linking using related links

The following example from the garage sample links the "Spray painting" task with the "paint" concept.

```xml
<related-links>
  <link href="../concepts/paint.dita" format="dita" type="concept">
    <linktext>Paint</linktext>
  </link>
</related-links>
```

## Linking using relationship tables

The following example shows the top section of the two-column relationship table for this document.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- (c) Copyright 2006-2007 by VR Communications, Inc. All rights reserved. -->
<!-- file is part of the DITA Open Toolkit project hosted on Sourceforge.net. the accompanying license.txt file for applicable licenses. -->
<map id="reltables_map" title="Relationship tables">
  <reltable>
```

```xml
</map>
```

```
</reltable>
The following example shows the three-column relationship table for the grocery shopping sample.

<!-- Relationship table -->
<!-- The related concept, task, and reference files point to each other -->
<reltable>
<relheader>
<relcolspec type="concept"/>
<relcolspec type="task"/>
<relcolspec type="reference"/>
</relheader>
<relrow>
<relcell>
<topicref href="concepts/about_produce.dita"/>
</relcell>
<relcell>
<topicref href="tasks/choosing_produce.dita"/>
</relcell>
<relcell>
<topicref href="reference/produce.dita"/>
</relcell>
</relrow>
How we decided what kind of linking to use

Some experienced DITA users employ only relationship tables (located in the master map or in a separate map file). The advantage of this approach is mostly for the producing organization: the links, which are recorded in only one place, can easily be searched and changed if updates to the project require a new linking paradigm. The disadvantage is mostly to the content users, who are given no information about why or under what circumstances they might want to consult the information in the related links.

We have chosen an approach that uses all three kinds of linking: cross-references (xrefs) where contextual information would seem to be helpful to the user ("I'll tell you why might might you want to click this link"), related links to point to external websites, and where the linked information is more along the lines of "I'll give you a few related topics you might want to consult if the titles sound interesting to you," we have used a single relationship table pointed to from the master map. The latter mostly contains "family" links within the core vocabulary section of the document.

To help ensure that our links continue to be accurate throughout the document production process, we run a set of debugging and link-checking tools we produced ourselves every time we do a major build. The tools check both internal and external (URL) links.

For more information (linking)

For a complete set of reference topics for DITA language elements (including the various linking elements), see the language reference document in the ditaot/demo directory of the DITA Open Toolkit.
Expanding and customizing access to your information

This section contains information on how to expand access to your information through indexing, the use of metadata, and filtering (conditional processing).

Sections in this topic:

- About indexing on page 151
- About metadata on page 151
- About RDF and DITA Open Toolkit on page 153
- DITAblogs (accessing) on page 153
- For more information (accessing) on page 154

About indexing

Indexing in DITA is accomplished with the `<indexterm>` tag, which can be nested.

Example

```
<indexterm>processing
<indexterm>to PDF targets</indexterm>
</indexterm>
```

The code produces the following two-level index entry:

processing
to PDF targets

About metadata

Metadata is semantic information about the information in a document, for example the name of the document's author, the date the document was created, the name of the product the information is describing, the target audience, and copyright information.

In DITA you can specify metadata at the topic or map level, with map-level metadata overriding topic entries.

Example

```
<metadata>
  <keywords>
    <keyword>Ant script</keyword>
    <indexterm>Ant scripts
    <indexterm>definition</indexterm>
    <indexterm>usage</indexterm>
  </keywords>
  <prodinfo>
    <prodname>DITA Open Toolkit</prodname>
    <vrmlist>
      <vrm version="1.3.1"/>
    </vrmlist>
  </prodinfo>
</metadata>
```
Providing metadata in DITA source files

The <prolog> section of a DITA source file can contain metadata about the source file including the author(s), date created, and keywords describing what the file is about. For instance, the source for this DITA topic contains the following metadata:

```
<prolog>
  <author type="creator">Anna van Raaphorst</author>
  <author type="contributor">Richard Johnson</author>
  <publisher>OASIS (Organization for the Advancement of Structured Information Standards)</publisher>
  <copyright>
    <copyryear year="2007"/>
    <copyrholder>VR Communications, Inc.</copyrholder>
  </copyright>
  <critdates>
    <created date="2006-June-10"/>
    <revised modified="2007-April-03"/>
  </critdates>
  <metadata>
    <keywords>
      <keyword>accessing information</keyword>
      <keyword>metadata</keyword>
      <indexterm>metadata</indexterm>
    </keywords>
    <prodinfo>
      <prodname>DITA Open Toolkit</prodname>
      <vrm version="1.3.1"/>
    </prodinfo>
  </metadata>
</prolog>
```

How the Toolkit processes metadata

In some cases, the output produced by a Toolkit build will contain content based on the metadata in the source file. For instance, when this source file is processed to XHTML, the output files will contain metadata in the Dublin Core format. Here is the metadata in the XHTML output for the source file above:

```
<head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
  <meta content="concept" name="DC.Type"/>
  <meta content="DITA.Open.Toolkit" name="DC.Title"/>
  <meta content="accessing information, metadata" name="DC.subject"/>
  <meta content="accessing information, metadata" name="keywords"/>
  <meta scheme="URI" name="DC.Relation" content="../accessing/accessing.html"/>
  <meta content="DITA.Open.Toolkit" name="prodname"/>
  <meta content="1.3.1" name="version"/>
  <meta content="Anna van Raaphorst" name="DC.Creator"/>
  <meta content="Richard Johnson" name="DC.Contributor"/>
  <meta content="OASIS (Organization for the Advancement of Structured Information Standards)" name="DC.Publisher"/>
  <meta content="VR Communications, Inc. 2007" type="primary" name="DC.Rights.Owner"/>
  <meta content="2006-June-10" name="DC.Date.Created"/>
  <meta content="2007-April-03" name="DC.Date.Modified"/>
  <meta content="XHTML" name="DC.Format"/>
  <meta content="aboutmetadata" name="DC.Identifier"/>
  <link href="/CSS/commonlttr.css" type="text/css" rel="stylesheet"/>
  <link href="/CSS/DITAOTUG_CSS.css" type="text/css" rel="stylesheet"/>
  <title>About metadata</title>
</head>
```
About RDF and DITA Open Toolkit

RDF is a W3C standard for describing information about a resource on the Web. RDF is meant to be read and understood by computers.

While it does not directly contain support for generating external or embedded RDF, DITA Open Toolkit does have some functionality that can be used to create RDF.

**Dublin Core**

The Dublin Core is a standard for metadata that is used to describe online information. The XHTML output produced by DITA Open Toolkit contains Dublin Core metatags generated from the various elements contained within the prolog, title, and short description elements in DITA source files. Further processing of the XHTML output can create RDF "triples" using these meta tags. (Functionality for that processing is not contained in the Toolkit.)

An RDF triple contains three pieces of information, a subject, a property type, and a value for the property.

For example, a `<title>` element might produce the following output in the generated XHTML:

```
<meta name="DC.Title" content="About metadata"/>
```

In this example the triple says the web page name is "About metadata".

**SKOS**

The Thesaurus (aka Taxonomy) plug-in can be installed with DITA Open Toolkit to provide a DITA specialization that can be used to identify and process content based on what the information is about by generating SKOS output. See About DITA Open Toolkit plug-ins on page 63 for more information about this plug-in.

DITAblogs (accessing)

We have included keywords and index entries for virtually all the topics in this documents. Several example are shown below.

```xml
<keywords>
  <keyword>XPath</keyword>
  <indexterm>XPath</indexterm>
</keywords>

<keywords>
  <keyword>localizing</keyword>
  <keyword>translating</keyword>
  <keyword>xml:lang attribute</keyword>
  <indexterm>localizing (translating)</indexterm>
  <indexterm>in XHTML output targets</indexterm>
</keywords>

<keywords>
  <keyword>evaluating DITA</keyword>
  <keyword>evaluating DITA Open Toolkit</keyword>
</keywords>
```
For more information (accessing)

The garage sample has coding to do filtering (conditional processing). See *About the garage sample* on page 86 for more information.
Reusing your content

Sections in this topic:

- About content reuse on page 155
- DITAblogs (content reuse) on page 155
- For more information (reuse) on page 156

About content reuse

Content reuse is the use of a single piece of content in multiple locations in a single document, or in multiple, related documents.

One example would be to use the same DITA topic in more than one map.

Another example would be to employ a conref to reuse a glossary or core vocabulary item in an "about" topic.

One of the major reasons users cite for moving to DITA is the ability to reuse content effectively.

DITAblogs (content reuse)

How we reused content

The DITA core vocabulary, which is a key feature of this document, has provided the authors many opportunities for content reuse. Most of the "about" topics in the book are "conref’ed" from the core vocabulary topic of the same name. For example, here is the core vocabulary content for the Ant topic:

```xml
<section id="ant_term">
  <p>Ant is a Java-based, open source tool provided by the Apache Foundation to automatically implement a sequence of build actions defined in an Ant build script. The Ant functionality is similar to the more well-known UNIX make and Windows nmake build tools; however, instead of using shell-based commands, like make, Ant uses Java classes.</p>
  <p>The configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular task interface.</p>
  <p>Ant can be used for both software and document builds.</p>
  <p>DITA Open Toolkit provides Java code and a set of XSLT transform scripts for producing different types of output, for example XHTML, Eclipse help, JavaHelp, and PDF. build scripts build DITA output by controlling the execution of the DITA Open Toolkit Java code and the XSLT transform scripts.</p>
  <p>Ant must be installed in your DITA processing environment for DITA Open Toolkit to function.</p>
</section>
```

Here is the conref in the About Ant topic:

```xml
<conbody>
  <section conref="../core_vocabulary/ant.dita#ant/ant_term"/>
</conbody>
```
For more information (reuse)

For more detailed discussions on content reuse, see *Introduction to DITA: Introduction to the Darwin Information Typing Architecture* by Jennifer Linton and Kylene Bruski. For information about the book, see [http://www.comtech-serv.com](http://www.comtech-serv.com).
Customizing your published output

This section contains information on how to customize your published output.

Sections in this topic:

- Conditional processing on page 157
- Using your own CSS (cascading style sheet) on page 162
- Customizing XHTML output on page 164
- Adding a product logo to a PDF2 cover page on page 167
- DITAblogs (customizing) on page 168
- For more information (customizing) on page 169

Conditional processing

Sections in this topic:

- About conditional processing on page 157
- Using conditional processing on page 157
- Filtering content on page 159
- Flagging content on page 160
- Showing revisions on page 161

About conditional processing

Conditional processing involves filtering or flagging content based on processing-time criteria, such as a target audience, platform, or product.

You can use metadata on elements to filter or flag content, and to show revised content. You can use attributes to provide information about what product, audience, or platform an element applies to, or what product revision it belongs to.

Say you have a need for two versions of your installation instructions, one for Windows and one for Linux. You can create a topic file with both sets of instructions (with each set properly labeled as either Windows or Linux), and then use a ditaval file to specify your processing rules (for example, whether to produce a Windows or Linux version of the document, or whether to produce a single output file with the content flagged appropriately with Windows and Linux icons).

You can exclude content based on its metadata. By default, all content is included. You can flag content based on metadata. By default, no content is flagged. You can show the revision information for your content. All revision information is hidden by default.

Using conditional processing

Attributes used for conditional processing

The following attributes for conditional processing are available on most DITA elements:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
<td>The product that is the subject of the content, or to which the content applies.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>platform</td>
<td>The platform (for example, Windows or UNIX) on which the product is deployed.</td>
</tr>
<tr>
<td>audience</td>
<td>The intended audience for the content.</td>
</tr>
<tr>
<td>rev</td>
<td>The revision or draft during which the content was added or changed. You can only flag revisions, not exclude them.</td>
</tr>
<tr>
<td>otherprops</td>
<td>Anything else.</td>
</tr>
</tbody>
</table>

Each attribute takes zero or more space-delimited string values. For example, you can use the product attribute to identify that an element applies to two particular products.

At processing time, specify the values you want to exclude and the values you want to flag.

**Setting conditions**

You can define what to do with these values in a .ditaval filtering file.

Specify the filter file at processing time by entering the /filter:{args.input.valfile} output option to name the .ditaval file that has these entries.

**Excluding information**

Specify the attribute and value you want to exclude. For example, exclude when audience="admin". For an element to be excluded, all the values in at least one attribute must be set to exclude. For example, if a paragraph applies to two audiences (audience="admin programmer") both values must be set to exclude in the filtering file before the paragraph is excluded.

You cannot exclude revisions, only choose whether or not to flag them.

**Flagging information**

Specify the attribute and value you want to flag. On output, the element will be flagged with the image you specify, or, in the case of revisions, with the method you specify. For an element to be flagged, at least one flagged value must be present. For example, if a paragraph applies to two audiences, either one set to flag will flag the element.

**Example**

Given this source:

```
<ul>
  <li audience="admin programmer">
    This is important for lots of reasons.
  </li>
  <li audience="programmer">
    This is only important to programmers.
  </li>
  <li audience="programmer" platform="unix">
    This is important only to UNIX programmers.
  </li>
  <li platform="unix">
    This applies on UNIX.
  </li>
</ul>
```

And given this filter file:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Action</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>audience</td>
<td>programmer</td>
<td>exclude</td>
<td></td>
</tr>
<tr>
<td>platform</td>
<td>unix</td>
<td>flag</td>
<td>../images/ngunix.png</td>
</tr>
</tbody>
</table>
You should get this output:

- This is important for lots of reasons.
- This applies on UNIX.

The first list item remains because it applies to an administrator and programmer audience. The second list item is removed because it applies to programmers only, and programmer-specific information has been excluded. The third list item is removed because it applies to programmers as well: the fact that it applies to the UNIX platform does not save it from exclusion. The fourth list item is flagged based on its platform attribute.

## Adding metadata to elements

Before using these attributes, work with your team to agree on a consistent set of values for use across your information set.

Use these attributes when you have content within a topic or a map that applies to only one or some of the products, platforms, or audiences for the topic or map as a whole.

1. Find the element for which you want to provide information.
   - If there are not attributes you need in the element, create a new element inside it, like a `<ph>`, which can hold the attributes instead.
2. Enter the attribute (for example, audience or platform).
3. Add a value or values to the attribute.
   - For example, `audience="novice-admin expert-user"` or `platform="wintel"`.
4. For *audience* or *product*, add an equivalent element to the prolog metadata.
   - You can provide more information about your audience or product values in the topic's prolog, inside the metadata element. To ensure consistency, you can define the elements in one place for the entire information set, and then reuse them in topics that require them by using the `conref` attribute.

Metadata attributes are inherited: they apply to any elements they contain, as well. In a table, metadata for a column applies to all the cells in the column.

## Filtering content

All the values in at least one attribute must be set to "exclude" for an element to be filtered out.

1. Create a DITA filter file in the directory where you want to add the file. Give the file a descriptive name, such as "audience-flag-build.ditaval".
2. Open the file, and type this content into it:
   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <val>
   </val>
   ```
3. Define the property you want to exclude:
   a) Find the attribute name. Type a new property to exclude by adding a new line `<prop att="" val="" action="exclude"/>`.
   b) Select the attribute ("audience", "platform", "product", "otherprops") on which you want to exclude.
       - For example, "audience".
   c) Type the value you want to exclude on.
       - For example, "programmer".
   - You can define more properties by following the steps above. The final .ditaval file with the example values should look like this:
     ```xml
     <?xml version="1.0" encoding="UTF-8"?>
     <val>
     <prop att="audience" val="programmer" action="exclude"/>
     </val>
     ```
4. When you publish output from the topic or map, specify the filter file you want by using the parameter 
/filter:{args.input.valfile} for the output options.

Content is excluded only when all the values in a single attribute are set to exclude. For example, if you have
a list item that applies to programmer and administrator audiences, and you exclude programmers, the list
item will remain in the output because it still applies to administrators.

Flagging content

Define a common set of images, and common alternative text for them, so that you can flag consistent content
across your information set. Generally flagging is supported for block-level elements such as paragraphs,
but not for phrase-level elements within a paragraph. This ensures that the images flagging the content are
easily scanned by the reader, instead of being buried in text.

At least one value in at least one attribute must be set to “flag” for the element to be flagged.

1. Create a DITA filter file in the directory where you want to add the file. Give the file a descriptive name,
such as “audience-flag-build.ditaval”.
2. Open the file, and type this content into it:

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

3. Define the property on which you want to flag:
   a) Type a new property for flag by adding a new line `<prop att="" val="" action="flag" img="" alt=""/>`.
   b) Select the attribute (“audience”, “platform”, “product”, “otherprops”) you want to flag on.
      For example, “audience”.
   c) Type the value you want to flag.
      For example, “programmer”.
   d) Enter the image you want to use to flag the content. Local absolute paths, or relative paths from the
      filter file both are supported.
      For example, “D:\resource\delta.gif”.
   e) Type the alternate text for the image that will be used by screen readers.
      For example, “sample alt text”.

   You can define more properties by following the steps above. The final .ditaval file with the example values
   should look like this:

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <val>
   <prop att="audience" val="programmer" action="flag" img="D:\resource\delta.gif" alt="sample alt text"/>
   </val>
   ```

4. When you publish output from the topic or map, specify the .ditaval file you want by using the parameter
/filter:{args.input.valfile} for the output options.

Content is flagged when any of the flagged values appear in any of the attributes. When flagged values are
found in a map, links generated from affected topicrefs will be flagged on output.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;p audience=&quot;admin programmer&quot;&gt;</code></td>
<td>When admin is set to flag, the paragraph will be preceded by the image you specified for admin. When both values are set to flag, both images will appear before the paragraph.</td>
</tr>
<tr>
<td><code>&lt;topicref platform=&quot;linux&quot; href=&quot;abc.dita&quot;&gt;</code></td>
<td>When linux is set to flag, any links to abc.dita derived from this topic reference will be flagged with the image you</td>
</tr>
</tbody>
</table>
Showing revisions

1. Create a DITA filter file in the directory to which you want to add the file. Give the file a descriptive name, such as "audience-flag-build.ditaval".

2. Open the file, and type this content into it:

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

3. Define the revision you want to show:
   a) Add a new property for flag by adding a new line `<revprop val="" style="" action="" date=""/>`.
   b) Type the name of the revision you want to show.
      For example, "rev1".
   c) Enter the date of the revision, or the date it started or ended.
      For example, "05-10-11".
   d) Define how the revision will be flagged in XHTML: either by images or colors.
      • Type the value of style if the revision will be flagged with color, either by using the RGB codes or color. For example, "#008040" or "Green". Type the value of action. For example, "noflag".
      • Enter the value of action if the revision will be flagged with image. For example, "flag".
      Use images instead of color to ensure that your output is flagged consistently and accessibly. Color works only for flagging phrase-level information, and cannot be distinguishable to color-blind readers, on monochrome displays, or once an online page is printed.
      If both style and action="flag" are used, the revision will be flagged by both of them.
   e) Enter the character to use when flagging the revision in PDF.
      For example, "A".

   - Note: Revision indication for PDF output will be specific to each implementation. See the documentation for your preferred FO/PDF output transform as to whether conditional markup and revisions are supported and how.

   You need to define a separate property for each revision you want to show. For example, if you want to show two revisions, you would need to define each separately.

   You can define more properties by following the steps above. The final .ditaval file with the example values should look like the following:

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <val>
   <revprop val="rev1" style="Green" action="noflag" date="2006-10-08" />
   <revprop val="rev2" action="flag" date="2007-10-16" />
   </val>
   ```

4. When you publish output from the topic or map, specify the filter file you want by using the parameter `/filter:{args.input.valfile}` for the output options.
Using your own CSS (cascading style sheet)

Default CSS behavior for XHTML processing
The Toolkit CSS stylesheet file resource/commonltr.css is copied to the output directory when you process to a target that creates XHTML output. All the generated XHTML output files include a link like the following that references the default CSS file:

```html
<link rel="stylesheet" type="text/css" href="../commonltr.css">
```

The generated XHTML pages reference classes defined in the default CSS file to control the styling of the XHTML page in a web browser.

Overriding the default CSS for a single DITA element
DITA provides an "outputclass" common attribute that can be used to explicitly set CSS classes for elements in the XHTML output. For example, if you want an entire section to be rendered as bold, you would code:

```xml
<section outputclass="caution" />
```

How to create your own CSS to override the default behavior
If you want to change the appearance of all the generated web pages, you can create your own CSS file that overrides part or all of the default CSS file. Your CSS will be included after the default CSS in all the generated pages.

For your override CSS to be used, you must set property values for the three Ant parameters in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>args.copycss</td>
<td>Whether to copy your CSS to the output directory.</td>
</tr>
<tr>
<td>args.css</td>
<td>Path to your CSS file.</td>
</tr>
<tr>
<td>args.csspath</td>
<td>Location of your CSS file in the output directory.</td>
</tr>
</tbody>
</table>

CSS override example
In this example we will make the background of all the generated web pages for the garage sample be the color aqua. We start by creating a new file garage.css. The file looks like this:

```css
/* garage CSS stylesheet */
body {
  font-family: verdana, arial, helvetica, sans-serif;
  font-size: 12px;
  background: Aqua;
}
```

Next we add some property definitions to our Ant build script as follows:

```xml
<property name="args.css" value="${projdir}/garage.css"/>
<property name="args.csspath" value="CSS"/>
<property name="args.copycss" value="yes"/>
```
When the Ant script is run our CSS will be copied to the CSS subdirectory in the output directory. In addition, the generated web pages will all contain the following lines:

```xml
<link rel="stylesheet" type="text/css" href="../CSS/commonltr.css"/>
<link rel="stylesheet" type="text/css" href="../CSS/garage.css"/>
```

This will cause all the web pages to have an aqua background color.

**Overriding XSLT stylesheet processing**

XSLT, a W3C standard, is a language for transforming XML into other formats. XSLT stylesheets are written in XML syntax.

In the final stage of processing, the DITA Open Toolkit runs XSLT stylesheet transforms (using either SAXON or Xalan) to produce output files. In certain cases, it is possible to override stylesheet processing to customize the output.

The following figure shows the XSLT stylesheet for the note element.

```xml
<xsl:template match="*" mode="process.note">
  <div class="note">
    <xsl:call-template name="ccmonattributes"/>
    <xsl:call-template name="setidname"/>
    <span class="notetitle">
      <xsl:call-template name="getString">
        <xsl:with-param name="stringName" select="'Note'"/>
      </xsl:call-template>
      <xsl:call-template name="getString">
        <xsl:with-param name="stringName" select="'ColonSymbol'"/>
      </xsl:call-template>
    </span><xsl:text>
    </div>
  </xsl:template>
```

The following XSLT stylesheets in the ditaot/xsl directory perform output transformation for various types of output (specified by setting the transtype). For those stylesheets marked with an asterisk (*) in the following table, you can override the default stylesheet with one you create.

<table>
<thead>
<tr>
<th>Transtype ('can be overridden')</th>
<th>XSLT stylesheet(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>*docbook</td>
<td>map2docbook.xsl, dita2docbook.xsl</td>
</tr>
<tr>
<td>eclipsecontent</td>
<td>map2eclipse.xsl, map2plugin-cp.xsl</td>
</tr>
<tr>
<td>eclipselhelp</td>
<td>map2eclipse.xsl</td>
</tr>
<tr>
<td>htmlhelp</td>
<td>map2hhc.xsl, map2hhp.xsl</td>
</tr>
<tr>
<td>*javahelp</td>
<td>dita2html.xsl</td>
</tr>
<tr>
<td>*pdf</td>
<td>dita2fo-shell.xsl</td>
</tr>
<tr>
<td>troff</td>
<td>dita2troff-step1-shell.xsl, dita2troff-step2-shell</td>
</tr>
<tr>
<td>*wordrtf</td>
<td>dita2rtf.xsl</td>
</tr>
<tr>
<td>*xhtml</td>
<td>dita2xhtml.xsl</td>
</tr>
</tbody>
</table>
How to override an XSLT stylesheet (generic instructions)

Follow these steps to override XSLT processing in a build:

1. In the ditaot/xsl directory, make a copy of the stylesheet you want to override and save it with its own unique name (don't replace the stylesheet that was originally included with the Toolkit).
   
   😡 Note: It is also possible to create a new stylesheet and use <xsl:import> to import the existing default stylesheet, and then make any changes you want to the existing targets.

2. In your new stylesheet, make any changes you want to the existing stylesheet code, and save it.
3. In your Ant build script, specify the "args.xsl" property with name of your new stylesheet.
4. Run your Ant build script.

XSLT override example

Follow these steps to modify the processing for a PDF target to remove the author list from the title page of the .pdf file. (The default XSLT stylesheet for PDF will add one author line to the title page for every <author> element in the <prolog> section of each file in the source tree.)

1. In the xsl directory of the Toolkit, make a copy of the file dita2fo-shell.xsl and save it as xditate2fo-shell.xsl.
2. Delete the following text in the file near line 134:

   ```xml
   <fo:block font-size="11pt" font-weight="bold" line-height="1.5">
     <xsl:text>
       [vertical list of authors]
     </xsl:text>
   </fo:block>
   <xsl:for-each select="//author">
     <fo:block font-size="11pt" font-weight="bold" line-height="1.5">
       [<xsl:value-of select="."/>
     </fo:block>
   </xsl:for-each>
   ```

3. Add the following line to the Ant target in your build script:

   ```xml
   <property name="args.xsl" value="${basedir}/xsl/xdita2fo-shell.xsl"/>
   ```

4. Run your Ant script again and verify that the .pdf output file does not have an author list on the first page.

Customizing XHTML output

Sections in this topic:

- Customizing XHTML output overview on page 164
- Including a header and footer in XHTML output on page 165
- Publishing XHTML output as a frameset on page 165

Customizing XHTML output overview

The Toolkit supports several <property> settings that can be used to customize the appearance of XHTML output. XHTML is generated when the "transtype" property has the value "xhtml". The following table shows which properties you can set that affect the appearance of your XHTML output:
Including a header and footer in XHTML output

Assume DITA source files are stored in C:/sandbox. In the sandbox directory are files myhdr.xml and myftr.xml. The files must be well-formed XML, so myftr.xml might look like this:

```xml
<p>DRAFT</p>
```

In the Ant script that builds the XHTML target, add properties for args.hdr and args.ftr. The target in the Ant script would look like this:

```xml
<target name="tk2xhtml">
  <ant antfile="${basedir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="doc/toolkit.ditamap"/>
    <property name="output.dir" value="out/toolkit/xhtml"/>
    <property name="transtype" value="xhtml"/>
    <property name="dita.extname" value=".dita"/>
    <property name="args.hdr" value="file:/C:/sandbox/myhdr.xml"/>
    <property name="args.ftr" value="file:/C:/sandbox/myftr.xml"/>
  </ant>
</target>
```

Publishing XHTML output as a frameset

By default, the Toolkit produces XHTML output in which the index.html page is a hierarchical list of document contents. When you click on a linked topic in the list, your browser opens a new page.

If you have a document of any size, it may be more user-friendly to create an XHTML document using framesets, in which the hierarchical contents are on the left, and the displayed topic is on the right. The following figure shows an example.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition, Usage</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>args.ftr</td>
<td>URI of the file containing XHTML to be placed in the body running-footer area of the output file. The file must be well-formed XML.</td>
<td>Example: <code>&lt;property name=&quot;args.ftr&quot; value=&quot;file:/C:/sandbox/myftr.xml&quot;/&gt;</code></td>
</tr>
<tr>
<td>args.hdf</td>
<td>URI of the file containing XHTML to be placed in the head area of the output file. The file must be well-formed XML, which means, for example, that you could include a JavaScript if it were labeled as CDATA.</td>
<td>Example: <code>&lt;property name=&quot;args.hdf&quot; value=&quot;file:/C:/sandbox/myhdf.xml&quot;/&gt;</code></td>
</tr>
<tr>
<td>args.hdr</td>
<td>URI of the file containing XHTML to be placed in the running-header area of the output file. The file must be well-formed XML.</td>
<td>Example: <code>&lt;property name=&quot;args.hdr&quot; value=&quot;file:/C:/sandbox/myhdr.xml&quot;/&gt;</code></td>
</tr>
</tbody>
</table>
With a few customizations to your DITA project, the Toolkit will produce frameset-based XHTML output automatically.

1. Put an XHTML file called index.html into a directory accessible by the Toolkit (ours is in a project directory called build_files).

   The file should look similar to the following.

   ```html
   <html>
   <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></meta>
   <title>DITA Open Toolkit User Guide</title>
   <!-- The following two lines cause the default styling for the navigation frame to be the same as the content frame. -->
   <link href="CSS/commonltr.css" type="text/css" rel="stylesheet" />
   <link href="CSS/DITAOTUG_CSS.css" type="text/css" rel="stylesheet" />
   </head>
   <!-- The left-hand navigation frame is 30% of the available space -->
   <!-- The right-hand content frame takes up the rest of the available space -->
   <frameset cols="30%,*">
   <!-- tocwin is the navigation frame -->
   <frame name="tocwin" src="toc.html"></frame>
   <!-- contentwin is the content frame -->
   <frame name="contentwin" src="DITAOTUG_bkinfo.html"></frame>
   </frameset>
   </html>
   
   The frame called "tocwin" is the left-hand frame for your table of contents. The source (src) file is toc.html.

   The frame called "contentwin" is the right-hand frame for your content pages. The source (src) file, DITAOTUG_bkinfo.html in this case, is the first page displayed in the content frame. For the DITA Open Toolkit User Guide (this document), it is the title page.
The TOC frame will take up 30% of the available space in the browser window, and the content frame will take up the rest.

2. Add code to your Ant script similar to the highlighted examples in the following figure

```xml
<target name="dita2xhtml">
  <ant antfile="${toolkit_dir}${file.separator}conductor.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/xhtml"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="xhtml"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="no"/>
    <property name="args.css" value="${projdir}/css_files/DITAOTUG_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <property name="args.xhtml.toc" value="toc"/>
    <property name="dita.extname" value="${EXTNAME}"/>
    <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
  </ant>

  <!-- Make the "index" file be toc.html -->
  <property name="args.xhtml.toc" value="toc"/>
  <property name="dita.extname" value="${EXTNAME}"/>
  <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
</target>

<!-- Copy index.html to the output directory -->
<copy todir="${outdir}/xhtml">
  <fileset dir="${projdir}/build_files"/>
</copy>
```

In the above code, the "args.xhtml.toc" parameter directs the Toolkit to substitute toc.html for the default index.html "index" page.

The above code also directs the Toolkit to copy the index.html file in the build_files directory to the output directory in a postprocessing step.

Because the TOC for this document is so long, we have customized the XHTML output to use framesets as described above.

## Adding a product logo to a PDF2 cover page

There are several ways to change the output to include a company or product logo image on the cover page of the PDF output file.

### Including an image reference in a bkinfo title

If you are using the bookmap/bkinfo specialization to produce a PDF book, you can place an image on the cover page by including an image in the title tag like this:

```xml
<title>
  <ph>First title phrase</ph>
  <image href="xxx.jpg" placement="break" />
  <ph>Another title phrase</ph>
</title>
```
Overriding the Idiom FO XSL stylesheet processing

It is also possible to modify or override the FO processing of the Idiom plugin. The way to do this is described in the file ./demo/fo/Customization/README.txt. To add an image to the cover page you can override cfg/fo/layout-masters.xsl by including these XML statements:

```xml
<fo:simple-page-master ...
<fo:region-body ... background-image="artwork:/foo.png" .../>
</fo:simple-page-master>
```

The image can also be included by overriding front-matter.xsl by adding an XSL statement like this:

```xml
<fo:external-graphic src="/path/to/images/my_logo_file.pdf"/>
```

DITAblogs (customizing)

Our experience with default CSS behavior

In the Ant scripts used to build the DITA Open Toolkit User Guide, we initially used only the default CSS stylesheet. However, the default CSS for XHTML does not have a filepath class defined, so all `<filepath>` elements are unstyled. For this reason we have added a custom CSS file that styles .filepath using the Courier font family. Here is what the CSS file looks like:

```css
/* Stylesheet overrides for DITA Open Toolkit User Guide */
/* DITAOT_UGRef_CSS.css */
/filepath { font-family: courier }
```

Note: We recently commented out this change, because we didn't like the way it looks in the XHTML output. However, this example is still a good illustration of how to make this kind of customization.

Garage sample files that illustrate filtering

One of the Ant scripts for the garage sample uses filtering to exclude topics having to do with oil and snow. The garage batch file that kicks off the processing (runbuild.bat) and the garage Ant script (garage_hierarchy_all.xml) are set up to do the filtering.

To run the build:

```bash
runbuild dita2filtered
```

Here is the section of the Ant script that references the .ditaval file:

```xml
<property name="dita.input.valfile" value="${projdir}/ditaval_files/garage_filtering.ditaval"/>
```
Here is the map file that uses the `<otherprops>` element to identify topics having to do with oil and snow:

```xml
<map title="Garage (hierarchy)"
<topicref href="concepts/garagetasks.dita" format="dita">
<topicref href="tasks/changingtheoil.dita" otherprops="oil" format="dita"/>
<topicref href="tasks/shovellingsnow.dita" otherprops="snow" format="dita"/>
<topicref href="tasks/takinggarbage.dita" format="dita"/>
<topicref href="tasks/spraypainting.dita" format="dita"/>
<topicref href="tasks/washingthecar.dita" format="dita"/>
</topicref>
<topicref href="concepts/garageconcepts.dita" format="dita">
<topicref href="concepts/lawnmower.dita" format="dita"/>
<topicref href="concepts/oil.dita" otherprops="oil" format="dita"/>
<topicref href="concepts/paint.dita" format="dita"/>
<topicref href="concepts/shelving.dita" format="dita"/>
<topicref href="concepts/snowshovel.dita" otherprops="snow" format="dita"/>
<topicref href="concepts/toolbox.dita" format="dita"/>
<topicref href="concepts/tools.dita" format="dita"/>
<topicref href="concepts/waterhose.dita" format="dita"/>
<topicref href="concepts/wheelbarrow.dita" format="dita"/>
<topicref href="concepts/workbench.dita" format="dita"/>
<topicref href="concepts/wwfluid.dita" format="dita"/>
</topicref>
</map>

Here is the `.ditaval` file, which is referenced in the Ant script, that excludes topics tagged as having to do with oil or snow:

```xml
<?xml version="1.0" encoding="utf-8"?>
<val>
<prop att="otherprops" val="oil" action="exclude"></prop>
<prop att="otherprops" val="snow" action="exclude"></prop>
</val>
```

---

**For more information (customizing)**

For more information about Ant parameters, see *Ant processing parameters* on page 82.
Specializing your information design

About specialization

What is specialization?

Specialization is the process by which new designs are created based on existing designs, allowing new kinds of content to be processed using existing processing rules. One of the key characteristics of DITA specialization is inheritance, which allows you to create new information types from existing ones. With inheritance you can use a class attribute to map an existing parent element to the specialized element you want to create.

Specialization allows you to define new kinds of information (new structural types or new domains of information), while reusing as much of existing design and code as possible, and minimizing or eliminating the costs of interchange, migration, and maintenance.

There are two kinds of specialization hierarchy: one for structural types (with topic or map at the root) and one for domains (with elements in topic or map at their root). Structural types define topic or map structures, such as concept or task or reference, which often apply across subject areas (for example, a user interface task and a programming task may both consist of a series of steps). Domains define markup for a particular information domain or subject area, such as programming, or hardware. Each of them represent an "is a" hierarchy, in object-oriented terms, with each structural type or domain being a subclass of its parent. For example, a specialization of task is still a task, and a specialization of the user interface domain is still part of the user interface domain.

Suppose a product group identifies three main types of reference topic: messages, utilities, and APIs. They also identify three domains: networking, programming, and database. By creating a specialized topic type for each kind of reference information, and creating a domain type for each kind of subject, the product architect can ensure that each type of topic has the appropriate structures and content. In addition, the specialized topics make XML-aware search more useful, because users can make fine-grained distinctions. For example, a user could search for xyz only in messages or only in APIs, as well as searching for xyz across reference topics in general.

Rules govern how to specialize safely: Each new information type must map to an existing one, and new information types must be more restrictive than the existing one in the content that they allow. With such specialization, new information types can use generic processing streams for translation, print, and web publishing. Although a product group can override or extend these processes, they get the full range of existing processes by default, without any extra work or maintenance.
Why specialization?

Specialization can have significant benefits for the development of new document architectures, for the following reasons.

- No need to reinvent the base vocabulary. Create a module in half a day with 10 lines versus six months with hundreds of lines; automatically pick up changes to the base.
- No impact from other designs that customize for different purposes. Avoid enormous, kitchen-sink vocabularies. Plug in the modules for your requirements.
- Interoperability at the base type. Guaranteed reversion from special to base.
- Reusable type hierarchies. Share understanding of information across groups, saving time and presenting a consistent picture to customers.
- Output tailored to customers and information. More specific search, filtering, and reuse that is designed for your customers and information, not just the common denominator.
- Consistency, both with base standards and within your information set.
- Learning support for new writers. Instead of learning standard markup plus specific ways to apply the markup, writers get specific markup with guidelines built in.
- Explicit support of different product architectural requirements. Requirements of different products and architectures can be supported and enforced, rather than suggested and monitored by editorial staff.

When to use specialization

Use specialization when you are dealing with new semantics (new, meaningful categories of information, either in the form of new structural types or new domains). The new semantics can be encoded as part of a specialization hierarchy that allows them to be transformed back to more general equivalents, and also ensures that the specialized content can be processed by existing transforms.

Implementing specialization

Sections in this topic:
- Creating a specialization on page 172
- Processing a specialization on page 172
- Specialization examples on page 173
- Limits of specialization on page 173

Creating a specialization

When choosing an element to specialize, look for a base element that:

- Has a more general meaning that also applies to your content
- Can accommodate the substructure of your content

Within the Toolkit dtd directory, create a DTD module in which the DTD elements derive from the elements of an existing DTD module.

Processing a specialization

If you do not modify the Toolkit processing, the Toolkit built-in generalization process automatically promotes your specialized element to the base element from which it derives, and processes it the same way it processes the base element.

If you want to modify the default processing, create a new XSLT script in the Toolkit xsl directory that imports the base XSLT script and provides special formatting for your specialized element.
In your Ant build script, add an "args.xsl" parameter to cause your new XSLT script to be used instead of the default.

**Specialization examples**


**Limits of specialization**

There are times when a new structural or domain type appears not to fit into the existing hierarchy, based on the semantics of the existing types and the restrictions of the specialization process. In these cases, consider the following options before abandoning the idea of specialization:

- **Specialize from generic elements.** For example, if you want to create a new kind of list but cannot usefully do so specializing from `<ul>`, `<ol>`, `<sl>`, or `<dl>`, you can create a new set of list elements by specializing nested `<ph>` elements. This new list structure will not be semantically tied to the other lists by ancestry, and so will require specialized processing to receive appropriate output styling. However, it will remain a valid DITA specialization, with the standard support for generalization, content referencing, conditional processing, and so forth. Always specialize from the semantically closest match whenever possible.

- **Create a customized subset document type.** Customized subset document types are not compliant with the DITA standard, and may not be supported by standards-compliant tools. However, they can help limit the quantity and mitigate the consequences of non-standard design in a customized implementation. Your customized document type can be transformed to a standard document type as part of the publishing pipeline. For example, if an authoring group requires additional metadata attributes, and finds authoring multiple metadata axes in one attribute (otherprops) unusable, the document type could be customized to add metadata attributes and then preprocessed to push those values into otherprops before feeding the documents into a standard publishing process. Customized document types are not compliant with the DITA standard and will not be supported by standards-compliant tools. However, a customized document type can help isolate and control the implications of non-standard design in a customized implementation.

**For more information (specialization)**

Localizing (translating) your DITA content

This section contains information about translating the content in your DITA projects.

Sections in this topic:

About localizing (translating) on page 175
DITA blogs (localizing) on page 181
For more information (localizing) on page 182

About localizing (translating)

Sections in this topic:

The value of DITA in localization on page 175
What the Toolkit provides for localization on page 175
Supported languages on page 176

The value of DITA in localization

Two of the key reasons organizations cite for using DITA are:

• To contain localization (translation) costs
• To respond quickly to customers who need product documentation translated into the primary language spoken in their country or region

What the Toolkit provides for localization

Localizing DITA content consists of two steps:

1. Translate the generated labels that appear in DITA output files.

   For example, the "Contents" label that appears above the list of contents on the first page of a PDF2 output file, as shown in the following figure.

   Another example would be the "Parent topic" and "Related topic" labels that appear in XHTML output files, as shown in the following figure.
The translation of these generated strings are controlled within the Toolkit build process.

2. Translate the DITA content.

An example of DITA content is the string "You can save money by buying produce when it is season." string in the figures above.

The translation of the DITA content is done (usually handled by translation centers or done by individual translators) outside the Toolkit build process.

Supported languages

DITA and DITA Open Toolkit support the languages listed in the following table.

<table>
<thead>
<tr>
<th>Language</th>
<th>xml:lang value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>ar-eg</td>
</tr>
<tr>
<td>Belarusian</td>
<td>bg-bg</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>be-by</td>
</tr>
<tr>
<td>Catalan</td>
<td>ca-es</td>
</tr>
<tr>
<td>Chinese (Simplified)</td>
<td>zh-cn</td>
</tr>
<tr>
<td>Chinese (Traditional)</td>
<td>zh-tw</td>
</tr>
<tr>
<td>Croatian</td>
<td>hr-hr</td>
</tr>
<tr>
<td>Czech</td>
<td>cs-cz</td>
</tr>
<tr>
<td>Danish</td>
<td>da-dk</td>
</tr>
<tr>
<td>Dutch</td>
<td>nl-nl</td>
</tr>
<tr>
<td>Dutch (Belgian)</td>
<td>nl-be</td>
</tr>
<tr>
<td>English (Canadian)</td>
<td>en-ca</td>
</tr>
<tr>
<td>Language</td>
<td>xml:lang value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>English (UK)</td>
<td>en-gb</td>
</tr>
<tr>
<td>English (US)</td>
<td>en-us</td>
</tr>
<tr>
<td>Estonian</td>
<td>et-ee</td>
</tr>
<tr>
<td>Finnish</td>
<td>fi-fi</td>
</tr>
<tr>
<td>French</td>
<td>fr-fr</td>
</tr>
<tr>
<td>French (Belgian)</td>
<td>fr-be</td>
</tr>
<tr>
<td>French (Canadian)</td>
<td>fr-ca</td>
</tr>
<tr>
<td>French (Swiss)</td>
<td>fr-ch</td>
</tr>
<tr>
<td>German</td>
<td>de-de</td>
</tr>
<tr>
<td>German (Swiss)</td>
<td>de-ch</td>
</tr>
<tr>
<td>Greek</td>
<td>el-gr</td>
</tr>
<tr>
<td>Hebrew</td>
<td>he-il</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hu-hu</td>
</tr>
<tr>
<td>Icelandic</td>
<td>is-is</td>
</tr>
<tr>
<td>Italian</td>
<td>it-it</td>
</tr>
<tr>
<td>Italian (Swiss)</td>
<td>it-ch</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja-jp</td>
</tr>
<tr>
<td>Korean</td>
<td>ko-kr</td>
</tr>
<tr>
<td>Latvian</td>
<td>lv-lv</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>lt-lt</td>
</tr>
<tr>
<td>Macedonian</td>
<td>mk-mk</td>
</tr>
<tr>
<td>Norwegian</td>
<td>no-no</td>
</tr>
<tr>
<td>Polish</td>
<td>pl-pl</td>
</tr>
<tr>
<td>Portuguese</td>
<td>pt-pt</td>
</tr>
<tr>
<td>Portuguese (Brazilian)</td>
<td>pt-br</td>
</tr>
<tr>
<td>Romanian</td>
<td>ro-ro</td>
</tr>
<tr>
<td>Russian</td>
<td>ru-ru</td>
</tr>
<tr>
<td>Serbian</td>
<td>sr-sp</td>
</tr>
<tr>
<td>Slovak</td>
<td>sk-sk</td>
</tr>
<tr>
<td>Slovenian</td>
<td>sl-si</td>
</tr>
<tr>
<td>Spanish</td>
<td>es-es</td>
</tr>
<tr>
<td>Swedish</td>
<td>sv-se</td>
</tr>
<tr>
<td>Thai</td>
<td>th-th</td>
</tr>
<tr>
<td>Turkish</td>
<td>tr-tr</td>
</tr>
<tr>
<td>Ukranian</td>
<td>uk-ua</td>
</tr>
</tbody>
</table>
Overview of how translation works for XHTML processing

For XHTML output, the unit of translation for the files included in a DITA map is the individual DITA topic file. In order to translate a topic to another language, it is necessary to translate text in the file and to cause the DITA Open Toolkit to translate any character strings it places in output, such as the string "Related links" in XHTML output.

xml:lang and translate universal element attributes

The language used in an element contained in a DITA topic is controlled by setting two universal attributes. They are:

- **xml:lang**: Specifies the language of the element content. When no xml:lang value is supplied, the default value of US English (en-us) is assumed.
- **translate**: Indicates whether the content of the element should be translated or not.

If a topic element does not specify the xml:lang attribute, it inherits the value from any ancestor element in the file. If no xml:lang attribute is specified, the default value en-us is assumed. The translate attribute controls whether the content of an element needs to be translated.

For more information regarding the XML standards related to the xml:lang attribute, see [http://www.w3.org/International/articles/language-tags/Overview.en.php](http://www.w3.org/International/articles/language-tags/Overview.en.php).

Steps to localize a DITA topic

To translate a topic to another language, first set the xml:lang attribute in the topic. Typically this is set on the root element. Setting the attribute causes the Toolkit processing to substitute localized values for any strings in inserts into the output. The final step is to translate text in the topic to the target language.

How DITA Open Toolkit processing handles localization for XHTML output

When it processes a topic, the Toolkit uses the xml:lang value in each topic element to determine how to do string substitution and to flow text (for example, left to right or right to left). If no xml:lang attribute is specified, it follows the ancestor chain of the current topic to determine xml:lang from any containing element. If none is found, the default language is used.

**Note:** Setting xml:lang in the .ditamap file embedding a topic has no effect on how the topic gets translated by the Toolkit.

Example

Here is an example of how this works. In the first case we have a concept that begins:

```
<concept id="about_produce_template" xml:lang="en-us">
```

This sets the language to US English for this topic. The XHTML output for the topic looks like the following figure, with the generated strings highlighted:
If we change the value of the xml:lang attribute to "de-de", the XHTML output will include generated strings in German, as shown in the following figure.

### About produce

Overview information about produce.

One of the keys to good health is eating lots of produce. It pays to buy fresh fruits and vegetables and serve them often.

You can save money by buying produce when it is in season.

**Parent topic:** Shopping for groceries

**Related tasks**

- Choosing produce

**Related reference**

- Available produce

---

Finally, if we change the value of the xml:lang attribute to "ar-eg", the XHTML output will include generated strings in Arabic with text flowed right to left, as shown in the following figure.

### About produce

Overview information about produce.

One of the keys to good health is eating lots of produce. It pays to buy fresh fruits and vegetables and serve them often.

You can save money by buying produce when it is in season.

**Übergeordnetes Thema:** Shopping for groceries

**Zugehörige Tasks**

- Choosing produce

**Zugehörige Verweise**

- Available produce

---
Localizing in PDF2 output targets

Sections in this topic:

- Overview of how translations works for PDF2 processing
- Setting the document.locale property
- PDF2 localization examples

Overview of how translation works for PDF2 processing

For PDF2 output (using the Idiom plug-in), the locale is set globally for the entire PDF output file. In order to translate the PDF output to another language, it is necessary to translate text in the individual DITA topic files and to cause the DITA Open Toolkit to translate any character strings it places in the PDF output, such as the string "Contents".

Setting the document.locale property in ./demo/fo/build.xml

The property ./demo/fo/build.xml sets the locale for the PDF2 output file. When the plug-in is installed, it is hard-coded to "en_US". Several locales are supported. They are:

- de_DE
- en_US
- es_ES
- fr_FR
- it_IT
- ja_JP
- zh_CN

For example, setting the locale like the following example will produce German strings in the PDF output file:

<property name="document.locale" value="de_DE" />
PDF2 localization examples

The following figure shows a page from a PDF2 output file where the (default) locale of US English was specified:

If we change the locale to German, the same page looks like this:

---

DITAblogs (localizing)

How we made this document "localization-ready"

Currently there are no plans to translate this document into languages other than US English. However, it would be well positioned to do so for the following key reasons:

- The document was written using DITA, an XML vocabulary well supported by popular translation tools.
- We have put xml:lang attributes in all of our root elements.
- The core vocabulary set could provide a translation center with basic information about the Toolkit and related products.
- Most of the "about" topics have been "conref'ed" from the core vocabulary, so that content would need to be translated only once.

Preparing your DITA files to be localization-ready

Even if you have no immediate plans to translate your DITA project files into languages other than US English, you can make your projects localization-ready by:
• Including xml:lang statements in your DITA source files.
• Creating well-architected projects that make effective use of content reuse.
• Always including a glossary, core vocabulary, or controlled vocabulary in your DITA projects.
• Investigating translation technologies as you convert your legacy content to DITA, even if you have no immediate plans to translate, to be sure your DITA projects will operate effectively with popular localization applications.
• Ensuring that your content management system integrates well with popular translation technologies.

For more information (localizing)

<table>
<thead>
<tr>
<th>Name, description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Localization Industry Standards Association (LISA) is an international forum</td>
<td><a href="http://www.lisa.org/">http://www.lisa.org/</a></td>
</tr>
<tr>
<td>for organizations doing business globally. It has published a set of localization</td>
<td></td>
</tr>
<tr>
<td>best practices that list the right and wrong ways to support international</td>
<td></td>
</tr>
<tr>
<td>customers, products, and services.</td>
<td></td>
</tr>
<tr>
<td>Globalization and Localization Association (GALA) is a non-profit international</td>
<td><a href="http://www.gala-global.org/">http://www.gala-global.org/</a></td>
</tr>
<tr>
<td>industry association for the translation, internationalization, localization, and</td>
<td></td>
</tr>
<tr>
<td>globalization industry. The association gives members a common forum to discuss</td>
<td></td>
</tr>
<tr>
<td>issues, create innovative solutions, promote the industry, and present a joint</td>
<td></td>
</tr>
<tr>
<td>voice within the industry and to the outside community.</td>
<td></td>
</tr>
<tr>
<td>ATA (American Translators Association) is a professional association founded to</td>
<td><a href="http://www.atanet.org/">http://www.atanet.org/</a></td>
</tr>
<tr>
<td>advance the translation and interpreting professions and foster the professional</td>
<td></td>
</tr>
<tr>
<td>development of individual translators and interpreters. Its members include</td>
<td></td>
</tr>
<tr>
<td>translators, interpreters, teachers, project managers, web and software</td>
<td></td>
</tr>
<tr>
<td>developers, language company owners, hospitals, universities, and government</td>
<td></td>
</tr>
<tr>
<td>agencies.</td>
<td></td>
</tr>
<tr>
<td>W3C internationalization activity The W3C Internationalization Activity has the</td>
<td><a href="http://www.w3c.org/International/">http://www.w3c.org/International/</a></td>
</tr>
<tr>
<td>goal of proposing and coordinating any techniques, conventions, guidelines and</td>
<td></td>
</tr>
<tr>
<td>activities within the W3C and together with other organizations that allow and</td>
<td></td>
</tr>
<tr>
<td>make it easy to use W3C technology worldwide, with different languages, scripts,</td>
<td></td>
</tr>
<tr>
<td>and cultures.</td>
<td></td>
</tr>
</tbody>
</table>
Managing your DITA source content

This section contains information on how to manage your content using backup, source control, and content management.

Sections in this topic:

- Backing up your source files on page 183
- About library or source control systems on page 183
- About content management systems on page 184
- DITA blogs (managing content) on page 185
- For more information (managing content) on page 185

Backing up your source files

Why it is important to back up your files

All the work you do editing and debugging the files in your DITA project ends up being stored as files on disk. If something happens to one or more of those disk files, you work may need to be re-created. Disk files can be lost for several reasons, including:

- You accidentally erase them. Because DITA projects may have hundreds or thousands of files, it may be relatively easy to do this when you don't have a library system or content management system.
- The hard drive in your computer fails. Hard drives are mechanical devices and will fail after a finite amount of time.
- The file system on your disk drive becomes corrupted.

Strategies for preserving your data

The basic strategy for preserving your data is to make sure it is stored on more than one hard drive. How you do this depends on your work environment:

- If you have a standalone desktop or laptop computer, you should make frequent backups to an external disk drive or USB device.
- If you are using a library or source control system, make sure your work gets checked in frequently. The source control system can serve as your backup system.
- If you are using a content management system, use it as your backup system.

It is easier to back up your DITA projects if they contain all your relevant project files (including your Ant scripts) and are located in a directory separate from your ditaot build directory.

About library or source control systems

Storing DITA files in library (source control) systems

In library or source control systems, DITA source files are stored on disk as ASCII text files, typically along with the source code files for the software project being documented. A source control system stores and tracks changes to files, and features "check in/out" of files, so users of the system are not making conflicting
updates. Multiple versions of source files are kept, and most systems have built-in capabilities to display differences between different versions of the same file.

Although having the code and documentation in the same system may seem useful, source control systems that were written with code in mind may not be the best place to keep content, mostly because these systems are typically not "content-aware."

A better solution would be either a content management system that integrates with the library system, or a separate content management system that has features appropriate for content.

Some popular library (source control) systems

The following table shows just a few of the many source control systems that can be used for storing DITA files.

<table>
<thead>
<tr>
<th>System name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVS (Concurrent Versions System)</td>
<td>Open source project. For more information, see <a href="http://www.cvsnt.org/wiki/">http://www.cvsnt.org/wiki/</a></td>
</tr>
<tr>
<td>Perforce</td>
<td>Commercial software configuration management system. You can download a free version for small projects. For more information, see <a href="http://www.perforce.com/">http://www.perforce.com/</a></td>
</tr>
<tr>
<td>Subversion</td>
<td>Open source project that is meant as a newer replacement for CVS. For more information, see <a href="http://subversion.tigris.org/">http://subversion.tigris.org/</a></td>
</tr>
</tbody>
</table>

About content management systems

If your DITA project involves large numbers of topics, many authors, or geographically distributed authoring and production teams, you may benefit from the features provided by a CMS, which can include:

- Workflow support
- Validation of topic links
- Support for the Semantic Web
- Localization (translation) support

To be truly effective, a CMS being used to store files for a DITA project must be aware of the tree-structured ("web") nature of the project, the content contained in the source files, and the relationships among the files and their content. The CMS must also be able to report on this kind of information: both meta information about the files, and syntactic and semantic information about the content. Ask your CMS vendor about the product's content analytical capabilities before deciding to buy.

It can also be useful for the CMS to include DITA Open Toolkit processing, and debugging and reporting aids that operate along with file processing. CMSs should also provide basic library (source control) functionality.

Popular CMSs include:

- Hosted (web-based)
  - Astoria
  - DocZone
  - XDocs
- Open-source
  - Alfresco
  - NetBeans
- Proprietary, server-based
• Documentum
• Ixiasoft
• Vasont

• Hosted DITA training site: ditausers.org (includes the DITA Storm editor)

**DITAblogs (managing content)**

In producing this document we processed and backed up our files daily or sometimes even more than once a day.

We did not use a library (source control) system or content management system, but now that the project has grown to 300+ files, we are investigating systems that will meet our needs with future editions of this document and other DITA projects.

**For more information (managing content)**

<table>
<thead>
<tr>
<th>Name, description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CM Professionals</strong>, the international content management community of practice, is a membership organization that fosters the sharing of content management information, practices, and strategies. CM Pros members are content management practitioners, both inside and outside organizations, who want to develop their expertise and share it with others.</td>
<td><a href="http://www.cmprofessionals.org/">http://www.cmprofessionals.org/</a></td>
</tr>
<tr>
<td><strong>CMS Review</strong> provides resources to help you choose a content management solution.</td>
<td><a href="http://www.cmsreview.com/">http://www.cmsreview.com/</a></td>
</tr>
<tr>
<td><strong>CMS Watch™</strong> provides independent evaluations of content management, records management, enterprise search, and portal solutions.</td>
<td><a href="http://www.cmswatch.com/">http://www.cmswatch.com/</a></td>
</tr>
<tr>
<td>The international association for <strong>Open Source Content Management</strong>, is an association connecting users and developers of open source content management solutions.</td>
<td><a href="http://www.oscom.org/">http://www.oscom.org/</a></td>
</tr>
</tbody>
</table>
Distributing your published content

Sections in this topic:

- About distributing published content on page 187
- Distributing published content using RSS on page 187
- Publishing content on a web server on page 188
- For more information (distributing) on page 188

About distributing published content

Distributing content means making your published DITA content available to your customers, potential customers, and other interested users.

Distributing published content using RSS

RSS is a family of web feed formats written in XML and used in content syndication by enabling applications like RSS readers to find out when new information is available on a website.

Since creation of output from DITA source files is automated by using Ant, it is possible to create or update RSS information about the output at the same time the output is produced. The RSS file can then be uploaded along with the output so its availability can be known to those subscribed to the RSS feed.

For example, an Ant build could create a file like this to announce the availability of a new version of this document:

```xml
<?xml version='1.0'?>
<rss version='2.0'>
<channel>
 <title>DITA OT User Guide</title>
 <copyright>Copyright (c) 2006-2007 by VR Communications, Inc. All rights reserved.</copyright>
 <link>http://www.mysite.com/ditaug.html</link>
 <description>DITA Open Toolkit User Guide 1.3.1</description>
 <language>en-us</language>
 <lastBuildDate>Mon, 16 Apr 2007 15:14:30 PST</lastBuildDate>
 <item>
  <title>DITA OT User Guide 1.3.1</title>
  <link>http://www.mysite.com/ditaug.pdf</link>
  <pubDate>Wed, 18 Apr 2007 11:35:45 PST</pubDate>
  <description>The latest version of DITA OT User Guide is now available.</description>
 </item>
</channel>
</rss>
```
Publishing content on a web server

All the application code required for building XHTML output from DITA source is written in the Java programming language. In the future, DITA Open Toolkit may support incremental builds, a build will only process changed files to produce output. Some DITA users are already talking about running the DITA build on a web server so they can dynamically create web pages from DITA source files.

For more information (distributing)

For an easy-to-understand tutorial on RSS, see http://www.mnot.net/rss/tutorial/.
Migrating legacy content to DITA

This section contains information about migrating legacy content to DITA.

Sections in this topic:

   About content migration on page 189
   DITAblogs (migrating content) on page 189
   For more information (migrating content) on page 189

About content migration

A number of applications are available to help you migrate legacy content to DITA. The degree of success you will experience in using these tools will depend a great deal on the current state of your legacy content. Content that is not already structured will take a great deal more work than content that has already been published as a set of topics, for example an online help system.

One of the more widely used migration packages is Adobe FrameMaker DITA Application Pack, a free plug-in for FrameMaker 7.2, which was released in September 2006.

Note: This is a plug-in for FrameMaker, not DITA Open Toolkit.

The Application Pack adds a plug-in to Framemaker that allows editing DITA topics and map in structured view. It is also possible to convert a DITA map (and all the other files it references) into a FrameMaker book. Output can be produced by using the usual Framemaker output processing or by running the Toolkit build process from within FrameMaker.

DITAblogs (migrating content)

This document is about 90% new material. The small amount of legacy content was migrated manually.

For more information (migrating content)

<table>
<thead>
<tr>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bright Path Solutions</td>
<td><a href="http://www.travelthepath.com/structure4.html">http://www.travelthepath.com/structure4.html</a></td>
</tr>
<tr>
<td>PTC: What you should know about DITA</td>
<td><a href="http://www.arbortext.com/resources/dita_q_a.htm">http://www.arbortext.com/resources/dita_q_a.htm</a></td>
</tr>
<tr>
<td>IBM: Migrating HTML to DITA</td>
<td><a href="http://www-128.ibm.com/developerworks/xml/library/x-dita8a/">http://www-128.ibm.com/developerworks/xml/library/x-dita8a/</a></td>
</tr>
</tbody>
</table>
Sample files

This section contains information about the DITA sample source files that come with DITA Open Toolkit.

The following sample files (garage and grocery shopping) are included in the package accompanying this document. Ant scripts are also included.

The garage sample is featured in the processing instructions in the processing chapter of this document (Processing (building) and publishing DITA documents on page 75

The grocery shopping sample is featured in the creating topics and maps chapters of this document (Creating DITA topics on page 131 and Creating DITA maps on page 141).

Garage sample

The garage sample, which is downloadable from http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html, is a set of DITA source files containing concepts and tasks related to organizing and doing tasks in a garage. The sample map files allow the topics to be published as either a hierarchy or a sequence. The sample also includes Ant scripts that allow you to publish to all target environments.

Note: The garage sample that is part of the DITA Open Toolkit User Guide is not exactly the same set of files as the garage sample that is part of the Toolkit package.

Before you begin to use the garage sample files, we recommend creating two directories in your root directory called DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT (Windows examples would be C:/DITASAMPLE_GARAGE_SOURCE and C:/DITASAMPLE_GARAGE_OUTPUT) and then copying the garage sample files to DITASAMPLE_GARAGE_SOURCE. Your directory structure should then look like this:

```
+ DITASAMPLE_GARAGE_OUTPUT
  + DITASAMPLE_GARAGE_SOURCE
    + ant_scripts
    + concepts
    + ditaaval_files
    + images
    + tasks
```

We recommend that you process your first test case to the XHTML target (dita2xhtml), which generally provides predictably positive results. The dita2htmlhelp, dita2pdf2, and the Eclipse targets (dita2eclipsecontent and dita2eclipsehelp) are also very reliable. The other targets are, in our experience, somewhat problematic, and may require special coding to get acceptable results.

One of the garage sample targets (dita2filtered, which produces XHTML output) illustrates conditional processing using filtering. The output is a hierarchically organized file with four of the garage topics (those dealing with "oil" or "snow") excluded.

Grocery shopping sample

The grocery shopping sample, which is downloadable from http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html, is a simple DITA project that includes seven topics: an overview topic, two concepts, two tasks, and two reference topics. The project also includes a map that aggregates the files and links them meaningfully using a relationship table. Ant scripts that process to all DITA Open Toolkit targets are also provided.

Before you begin to use the grocery shopping sample files, we recommend creating two directories in your root directory called DITASAMPLE_GROCERY_SOURCE and DITASAMPLE_GROCERY_OUTPUT (Windows
examples would be C:/DITASAMPLE_GROCERY_SOURCE and C:/DITASAMPLE_GROCERY_OUTPUT) and then copying into them both the grocery shopping sample files DITASAMPLE_GROCERY_SOURCE. Your directory structure should then look like this:

```
  DITASAMPLE_GROCERY_OUTPUT
  DITASAMPLE_GROCERY_SOURCE
    completed
      ant_scripts
      concepts
      reference
      tasks
      topics
    template
      concepts
      reference
      tasks
      topics
    working
      ant_scripts
      concepts
      reference
      tasks
      topics
```

The grocery shopping sample assumes you are already familiar with the garage sample provided as both a project model as well as a tool to verify your DITA Open Toolkit installation, and that you have processed the garage sample as described in *Processing (building) and publishing DITA documents* on page 75.
Frequently asked questions (FAQs)

This section contains frequently asked questions about DITA and DITA Open Toolkit.

Sections in this topic:

- Do I need to know XML to use DITA? on page 193
- Why is "Darwin" in the name of the DITA architecture and DITA Open Toolkit? on page 193
- How does DITA differ from DocBook? on page 194
- Should I set my root element to <dita> or <concept> (or <task> or <reference>)? on page 194
- How can I use conditional processing effectively? on page 195
- What steps should I follow to architect and write a large DITA document? on page 196

Do I need to know XML to use DITA?

**Contributing author:** Deborah Pickett

**Date:** June 9, 2006

XML's a funny beast—people imagine that it has all kinds of mystical powers, but it's little more than a set of rules for how to structure data. I'll try to work in a reference to the emperor's new clothes here.

If all you plan to do is write topics in DITA, then you need to know very little about XML. If you're going to use a newfangled WYSIWYG editor to hide the structure from you, then you might be able to get by knowing nothing at all about XML. If you stay in plaintext-land, then I'd list the following XML ideas as essential:

- Elements and attributes
- Angle brackets (< and >), ampersand (&), slashes, and quotation marks (" and ")
- Content models (the idea that elements contain other elements)

These would be nice, too:

- Doctypes
- Well-formedness vs. validity

Don't let anyone try to talk you into studying these in the short term:

- DTDs and schemas
- XSLT
- Namespaces

After that, it's probably just a matter of brushing up on DITA's ideas of content models.

The above doesn't hold if you are planning to write your own specializations or transformations.

If you want to understand the internals of the DITA Open Toolkit, then you are in for a bumpier ride, because you'll need to become acquainted with Ant, Java, XSLT, and how they all talk to each other.

Why is "Darwin" in the name of the DITA architecture and DITA Open Toolkit?

**Contributing author:** IBM

**Date:** August 8, 2006
The Darwin Information Typing Architecture name has the following meaning and significance:

- **Darwin** because it uses the principles of specialization and inheritance.
- **Information Typing** because it capitalizes on the semantics of topics (concept, task, and reference) and of content (messages, typed phrases, and semantic tables).
- **Architecture** because it provides vertical headroom (new applications) and edgewise extension (specialization into new types) for information.

The DITA architecture supports the proper construction of specialized DTDs from any higher-level DTD or schema. The base DTD is ditabase, which contains an archetype topic structure and three peer topics that are typed specializations from the base topic, concept, task, and reference. The principles of specialization and inheritance resemble the principle of variation in species proposed by Charles Darwin, so the name is reminiscent of the key extensibility mechanism inherent in the architecture.

---

**How does DITA differ from DocBook?**

**Contributing author:** IBM
**Date:** August 8, 2006

DocBook and DITA take fundamentally different approaches.

DocBook was originally designed for a single, continuous technical narrative, where the narrative might be an article, book, or multivolume length. Through transforms, DocBook can “chunk” this information into topics to provide support for websites and other information sets. Because the goal of the DocBook DTD is to handle all standard requirements for technical documentation, the usage model encourages customization to exclude elements that aren’t local requirements. The usage model supports but discourages local extensions because of the potential for unknown new elements to break tool support and interoperability.

By contrast, DITA was designed for discrete technical topics. DITA collects topics into information sets, potentially using filtering criteria. The core DITA information types are not intended to cover all requirements, but rather to provide a base for meeting new requirements through extension. Extension is encouraged, but new elements must be recognizable as specializations of existing elements. Through generalization, DITA provides for tool reuse and interoperability.

Each approach has its strengths. DocBook would be the likely choice for a technical narrative. DITA would be the likely choice for large, complex collections of topics or for applications that require both extensibility and interoperability. Technical communications groups might want to experiment with both packages to determine which approach is better suited for their processes and outputs.

---

**Should I set my root element to <dita> or <concept> (or <task> or <reference>)?**

**Contributing author:** Don Day
**Date:** 2006

Using <dita> as your root element (which is the default behavior for FrameMaker DITA Edition and XMetaL DITA Edition) provides the following benefits for DITA information planners:

- It allows writers to manage multiple topics in a single editing instance (which is not a problem because they can still cross-reference any contained topic individually).
- The nesting rules in this ditabase context allow any of concept, task, or reference child topics, which might be useful if you are creating a single overview task that starts with a concept and then nests two or three related tasks that are unique to the scope of this overview topic.
The key disadvantages of relying on the `<dita>` wrapper consistently is that the topics within it cannot be topicref'ed individually to be output as their own nodes in a map. This behavior is consistent for the occasional overview topic with dedicated content, but in general you would prefer individual topics for best ease of reuse. (This behavior is the default for XMLmind and Serna.)

DITA Open Toolkit is agnostic to any starting element, as long as it is either a map, a topic, a specialization of either, or a `<dita>` file. All outputs produce intermediate, fully-resolved topics in the temp directory. But whereas the HTML-based transforms that produce standalone result files work directly on these intermediate topics, the aggregation-based outputs (FO and RTF) do a subsequent topicmerge step into a single intermediate file as input to their respective final transforms. Again, as long as the process starts with anything that is ostensibly a map or topic, a "correct" output in terms of its own top-level structure will result.

A `<dita>` file always results in a single deliverable, whether processed by reference in a map or individually from the command line. When you process a `<dita>` file with peer topics to HTML, it will produce a single, well-formed HTML file in which the content of the peer topics show as a sequence of `<h1>` titles followed by their respective content. In other words, it is still a valid output, but the HTML deliverable is always a single output instance, following the input file structure.

One way to use this behavior to some advantage is to reference the contained topics via conref, liberating them for reuse in different sequences. For example, you can rearrange the sequences in this construction, and thereby change the order (or even the optionality of inclusion) of what the standalone `<dita>` file produces.

```xml
<dita>
  <topic conref="sometopic.dita#sometarget"><title/></topic>
  <topic conref="othertopic.dita#othertarget"><title/></topic>
  ...
  <topic conref="end.dita#last"><title/></topic>
</dita>
```

Obviously, to achieve this "aggregated maplet" advantage, you must have separated those topics as standalone chunks in the first place. The result will still be a single output instance, unlike true map topicrefs. But you can change the order as needed, and conditionalize the conref even. Your analysis of the information architecture will lead to deciding which of the possible arrangements is best for your situation. Of the possibilities I've mentioned, all are "preferred roots" if the design so requires. You as an author should request that your authoring tool of choice allows you to manage the DITA architecture as needed.

How can I use conditional processing effectively?

**Contributing author:** Deborah Pickett

**Date:** Aug 16, 2006

**Question:** If I have both audience and platform attributes in my topics, do I have to specify at least one value for each attribute in my `.ditaval` file to filter on both attributes?

**Answer:** If you don't specify any include/exclude properties for, say, platform, then all of the platform="" attributes in your topics will be ignored; they just don't feature in the filtering logic at all. This should make sense, because you haven't asked for any platform filtering to happen. Just as you haven't asked for any product filtering or otherprops filtering.

**Question:** I put attribute values for filtering into topic and text elements, not in the map files. Why do I sometimes get empty XML files created by the Toolkit?

**Answer:** What happens is you have a map with a topicref/@href to a file that has had its root element removed (hence is now an empty file, and not a valid DITA topic). The map reference becomes a dangling link, in just the same way as if you'd referenced a file that doesn't exist at all.

I know that it's tempting to want the topic itself to be aware of which platforms and which audience it's useful for, but the DITA filtering model is unclear about whether it supports this, and DITA-OT certainly doesn't.
There's an open RFE on SourceForge, but it's been unlooked-at because it's unclear what is the "correct" behaviour.) The way I squint at it, by putting the filtering attribute on the topicref rather than the topic, I'm not precluding some mad fool for including the topic outside its natural scope in their own map, for some reason I haven't thought of. In that respect, it's promoting more (though perhaps not better) reuse.

You *can* mark a topic as being applicable to certain audiences through using the prolog tag, but that's got nothing to do with filtering, and more to do with how your topic might get indexed in a CMS. There is likewise an element.

**Question:** The hope here is that all the information about a single concept, reference, task, or generic topic could reside in the same file to simplify updates for all output variations. Thus, if a change like the filename of the product's installation executable changes, the person doing the doc update sees the data for all variations of installation instructions.

**Answer:** I get what you're saying. I wouldn't say that you're wrong, but there's a slight culture clash here, because bits of what you describe above are considered by some to be the domain of the content management system rather than the author. But we're getting into philosophical areas here, and it might be best if someone from the DITA Technical Committee took over lest I put words in their mouth.

I will close, though, by expressing a suspicion that filtering isn't high up on DITA's list because DITA strives to solve the same problem by breaking up topics into pieces small enough that they can be shared using judicious conrefs and selected through careful use of maps. Extravagant use of filtering may be a sign that the author is either clinging to outmoded notions about sharing information in topics, or that DITA is the wrong hammer for their nail.

---

**What steps should I follow to architect and write a large DITA document?**

**Author:** Anna van Raaphorst

**Updated:** April 3, 2007

I followed the following general process to architect and edit *DITA Open Toolkit User Guide* (a document of over 300 topics and approximately 250 pages when built to PDF). The official writing team was only two people, but we got input from a number of other sources.

1. Do a high-level design of the document with the key topics (in this case, "chapters," because it's based on the bookmap demo specialization).
2. Do an assessment of any existing content and establish a migration plan.
3. Create an appropriate directory structure.
4. Create a top-level ditamap (and lower-level maps, if required).
5. Add sample section/topic names to the chapters. Establish preliminary guidelines for file names and IDs.
6. Decide on the metadata that will be included in each topic.
7. Decide on the graphics guidelines.
8. Establish a preliminary plan for linking the content.
9. Establish a preliminary plan for content reuse (conref'ing).
10. Set up the infrastructure and establish the procedures necessary for your team members to process both their own sections and the entire document.
11. Review your progress and preliminary decisions with your other team members. Make adjustments, if necessary.
12. Write sample topics and stub topics (all with metadata). Ask some of your team members to do this, as well.
13. Make additional adjustments, as appropriate. Keep your name on the list of writers, if only for a small section.
14. Assign topics to writers and ask them to write several additional sample topics. Meet with your team and make adjustments based on their feedback.
15. Establish key files as templates or samples. Better yet (if you have time), write a style guide with lots of examples.

16. Begin to edit some of your team’s topics.

17. After about 10-15% of the topics are in draft format, meet with your team to establish indexing guidelines. Adjust the guidelines up to about the 30%-complete level, and then write up the indexing guidelines and stick to them.

18. Continue to edit your team’s topics.

19. As information architect (or editor), process the entire document often, and look for problems. Meet with your writing team to establish an action plan for every issue. Reverse roles some of the time (writers edit, and editors write).

20. When your team has approximately 40% of the content written, meet to establish the final guidelines for linking (both topic-to-topic and external, if relevant).

21. Do regular TOC, indexing, and linking reviews.

22. Establish criteria for the alpha and beta versions of the document.

23. Meet with your team to review the publishing plan, including testing assignments.

24. Create the alpha version of the document, test, and fix bugs.

25. Create the beta version of the document, test, and fix bugs.

26. Publish and distribute.

---

**How can I add a special character (for example, trademark) to a DITA file?**

**Contributing author:** Deborah Pickett

**Date:** April 18, 2007

The `<tm>` element is recommended where you can use it. For example:

```
<tm trademark="DB2" tmtype="tm" tmclass="ibm">DITA</tm>
```

The tmtype for registered trademark is "reg", and for service mark it is "service".

The Unicode character entities for common symbols are:
• trademark: &amp;#2122;
• registered trademark: &amp;#00AE;
• service mark: &amp;#2120;
• copyright symbol: &amp;#00A9;

For more information about Unicode symbols, see http://www.unicode.org/charts/
Release 1.3.1 and 1.3 information

Release 1.3.1

Release 1.3.1 was a maintenance release in which 15 bugs were fixed. Release 1.3.1 was based on the prior major release (1.3).

Note: Release 1.3.1 is still based on the DITA 1.0 standard. As of this writing, DITA 1.1 is expected to be approved in 2Q07. The release that will bring the Toolkit up to the DITA 1.1 standard has not yet been announced.

Deprecated functionality

The following functionality is deprecated as of release 1.3.1:

- The "old" PDF output (in favor of PDF2).
- The "old" topicmerger function (of concern only to developers).
- The demo bookmark DTD (which is still being used in this document). A new, fully supported bookmark is part of the DITA 1.1 standard and will be supported in the next release of DITA Open Toolkit.

In our experience, several target outputs can be problematic (although they are not officially listed as deprecated): DocBook, JavaHelp, troff, and Word RTF.

In addition, any specializations and plug-ins in the demo directory are in the "use at your own risk" category.

Release 1.3

Release 1.3 was a major release to add new and improved functionality, fulfill new requirements, and fix bugs.

Strategic updates to DITA Open Toolkit

1. Toolkit distributions: In prior releases, the DITA Open Toolkit distribution contained only the Toolkit. In release 1.3.1, the Toolkit was available in three distributions:
   - Full package distribution, which contained all required tools except Java J2SE SDK
   - Small package distribution, which contained only the Toolkit
   - Source distribution, which contained the source and executable code for the Toolkit, and the source code for this document

2. Change to build.xml: To make the DITA processing environment more like other Ant-driven build environments, the build.xml file from the 1.2.2 version of the Toolkit was renamed as build_demo.xml. The build.xml file was the top-level script for starting a transformation.

3. Change to command-line invocations: The "Ant refactoring" exercise for this release changed some previously documented Ant calls for running demos. This change enabled better use of the Ant modules for power users who needed to integrate the Toolkit into programming build environments such as Eclipse.

4. Separation of demo targets from formal component targets: The internal programming targets are displayed when you type "ant -p". To see both those programming targets and the demos that are part of this component, type "C:\dita-ot\ant -p -f build_demo.xml". To run just one of the demos that you see in the resulting list, dita.faq, for example, type "C:\dita-ot\ant dita.faq -f build_demo.xml".

5. Classpath update to enable catalog resolver: Release 1.3 included the Apache catalog resolver for improved lookup of DTDs by any of the Toolkit components. The full package version of the Toolkit sets up these variables for each session. For the small package distribution of the Toolkit, you need to include lib and lib\resource\resolver.jar in your CLASSPATH. For example, if your CLASSPATH is:

C:\dita-ot\lib\dost.jar
you need to change it to:

C:\dita-ot\lib;C:\dita-ot\lib\dost.jar;C:\dita-ot\lib\resolver.jar

The full package can be used like a normal installation as long as you update the system variables either in the environment settings or in a batch file that sets up the shell environment.

6. License bundling: To reduce the duplication of builds on SourceForge in which the only difference was the license provided in each, both the Apache and CPL licenses were included in root directory of the Toolkit. Use the one that applies to your environment.

New and enhanced features

1. OASIS DITA 1.1 support

• New element <abstract>
  The <abstract> element can include complex markups besides the <shortdesc> element. The <shortdesc> element can be put inside the <abstract> element, together with many other elements.

• New element <data>
  The proposed DITA 1.1 standard includes a new element, <data>. This element and the content inside it is ignored in the transformation process of DITA files.

• The following new indexing elements have been proposed for DITA 1.1: <index-see>, <index-see-also>, and <index-sort-as>. Preliminary support was added in DITA Open Toolkit 1.3.

• As an aid to information architects, extensible metadata attributes have been proposed for DITA 1.1. Preliminary support was added in DITA Open Toolkit 1.3.

• DITA Open Toolkit 1.3 provided improved support for graphic scaling. This feature applied to HTML Help, XHTML, PDF, and FO, but was not supported in RTF output.

• The <unknown> element has been proposed for DITA 1.1 to incorporate existing standard vocabularies for special content, like MathML and SVG, as inline objects.

2. Eclipse integration of DITA documents

Capabilities were added to allow use of a template to develop documentation plug-ins with DITA in Eclipse PDE, and to use DITA Open Toolkit to build and pack the final plug-in.

3. Topicmerge improvements

The topicmerge feature improved the Toolkit build speed and reduced the possibility of encountering an "out of memory" exception in the build process.

4. Indexing improvements for localization

In prior releases of DITA Open Toolkit, index entries were sorted based on the Java JDK collator. For example, Swedish words beginning with umlauted a and o vowels appeared with the equivalent non-umlauted vowels, ignoring the linguistic significance in Swedish.

In release 1.3, the index sort was based on the paradigm of the language specified for the files. In the case of Swedish umlauted vowels, they now appear at the end of the alphabet, after Z.

5. XML catalog improvements

In prior releases of DITA Open Toolkit, a simple XML catalog resolver was enabled. Users did not need to update the references to DTDs in DITA files when the file paths were changed; however, this simple implementation could be redistributed because it did not support standard XML catalogs.

In DITA OT 1.3, a standard XML catalog resolver was enabled so that references to DTDs in DITA files do not need to be updated each time users change the file paths on their workstations or use another workstation.

6. Support for multiple file extensions in one DITA map
Release 1.3 introduced support for two different file extensions, .dita and .xml. Previous releases of DITA Open Toolkit did not support the transformation of DITA maps containing inconsistent file types, for example one DITA map containing both .dita and .xml files.

7. Ant refactoring

In DITA Open Toolkit 1.3, the system-level Ant scripts were refactored so developers could easily find the targets they needed, and create their own extensions.

8. Command-line help

In release 1.3, the command-line help function was enhanced to improve usability, allowing users to display the version of the Toolkit and the usage of the command line with the following commands:

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

Bug fixes: 21
Patches added: 1
RFEs added: 1
Release history

For information about the current release of DITA Open Toolkit, see Release 1.3.1 and 1.3 information on page 199.

Sections in this topic:
- Release 1.2 on page 203
- Release 1.1 on page 204
- Release 1.0 on page 205

Release 1.2.2
Release 1.2.2 was a maintenance release based on release 1.2 in which 24 bugs were fixed.
Also included were the following improvements:
- Chinese support was added for Word RTF.
- The plug-in architecture and dependency handling was improved.

Release 1.2.1
Release 1.2.1 was a maintenance release based on release 1.2 in which 12 bugs were fixed.
Also included were the following improvements:
- A problem with corrupted tables generated in Word RTF was fixed.
- Pictures are now merged into the Word RTF instead of linking to them.
- The lq element is supported in Word RTF.
- Generated text can be translated to different languages in Word RTF.
- In Word RTF, if no <choption> is specified, a head is generated in tables.

Release 1.2
Release 1.2 was a major release to add new and improved functionality, fulfill new requirements, and fix bugs.

New and enhanced features
1. Plug-in architecture
   New capabilities were added to help users download, install, and use plug-ins, and to help developers create new plug-ins for DITA Open Toolkit.

2. Word RTF transformation
   Capabilities were added to allow transformations from DITA source files to output in a Word RTF file.

3. HTML-to-DITA migration tool
   A tool was added that migrates HTML files to DITA files. This tool originated from the developerWorks publication of Robert D. Anderson’s how-to articles with the original h2d code.

4. Problem determination
   Logging was improved to capture additional status and transformation information, as well as warning, error, and fatal error messages both on-screen and in the log file.

5. Conditional processing documentation
   Information about conditional processing was added to the DITA Open Toolkit documentation set.
6. Language reference documentation
   The OASIS DITA standard language reference was added to the Toolkit documentation set.

7. DTD files
   The DTD files in DITA Open Toolkit were updated to the DITA 1.0.1 level.

Bug fixes: 19

Release 1.1.2.1
Release 1.1.2.1 fixed one bug: the build process failed with the "Ant all" parameter, which prevented users from running the installation verification tests.

Release 1.1.2
Release 1.1.2 was a maintenance release with 14 bug fixes, minor changes to some Ant parameters, support for additional Java parameters, and minor changes to the organization of the doc directory.

Release 1.1.1
Release 1.1.1 was a maintenance release with 11 bug fixes and a dost1.0.jar name change back to dost.jar.

Release 1.1
Release 1.1 was a major release to add new and improved functionality, fulfill new requirements, and fix bugs.

New and enhanced features in release 1.1

1. Support for OASIS DITA 1.0
   Support was added for the OASIS DITA 1.0 standard for DITA DTDs and schemas.

2. Transformation to troff
   Support was added for transformation to the troff document processing system.

3. XML catalog support
   Support was added for XML catalogs, which are logical structures containing mapping information between public IDs and URLs of DTD files. A catalog entry can be used to locate a unified resource identifier (URI) reference for a DTD file. An external entity's public identifier is used for mapping to the URI reference. The URI of any system identifier can be ignored.

4. Topicref referring to nested topic
   The href attribute of the topicref entity was extended to quote a nested topic in a DITA file.

5. Localization support
   Support was added to support DITA content in 20 languages, and translation of DITA keywords in the same 20 languages.

6. Accessibility support
   Accessibility partially applied to XHTML and PDF transformations.

7. Eclipse Content Provider support
   Release 1.1 supported the Eclipse Content Provider.

8. Index information in HTML Help and JavaHelp
   Index information now appears in HTML Help and JavaHelp.

9. Mref element
The mapref element (a specialization of the topicref element) was added to allow a reference to another ditamap file.

10. **TOC generation for Eclipse Help**
Tables of contents could be generated for Eclipse help.

11. **Helpsets supported in Java Help**
Support was added for helpsets in JavaHelp.

12. **Additional parameter support for Java commands**
Support was added for the following Java command parameters: /indexshot, /outext, /copycss, /xsl, and /tempdir.

13. **Additional parameter support for Ant scripts**
Support was added for the following Ant command parameters: .args.indexshow, args.outext, args.copycss, args.xsl, and dita.temp.dir.

**Bug fixes: 7**

**Release 1.0.2**
Release 1.0.2 was a maintenance release with 7 bug fixes and minor enhancements.

**Release 1.0.1**
Release 1.0.1 was a maintenance release with 11 bug fixes and minor enhancements.

**Release 1.0**
Release 1.0 was the initial release of the open-source version of the DITA Toolkit, which evolved from a developerWorks version.

**New and enhanced features in release 1.0**

1. **Java-based processing**
The Java-based processing architecture supports single-threaded execution throughout.

2. **Ant-based processing**
Release 1.0 featured Ant-based orchestration of the processing environment, from preprocessing through transformation to any required postprocessing.

3. **Conditional processing**
A preprocessor core supports conditional processing and conref resolution.

4. **Map-driven processing**
The map-driven processor generates links for transformed topics.

5. **New DITA-to-HTML transform**
A DITA-to-HTML transform that was designed for high-volume usage replaced the previous topic2html_Impl.xsl core transform.
DITA core vocabulary

This section contains information about the DITA core vocabulary, a list of the terms in the vocabulary, and links to related terms and other information.

About the DITA core vocabulary

The DITA core vocabulary is a set of terms related to the DITA standard and DITA Open Toolkit. It is intended to be used as a controlled or specialized metadata vocabulary for describing and documenting DITA, DITA Open Toolkit, and other resources related to DITA or the Toolkit.

Inspiration for this effort is the Dublin Core Metadata Initiative (DCMI), an organization dedicated to promoting the widespread adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable intelligent information discovery systems. The mission for the DCMI is to provide simple standards to facilitate the finding, sharing and management of information.

In the spirit of the DCMI effort, the key goals (and potential benefits) of the DITA core vocabulary are to:

• Facilitate the finding, sharing and management of information about DITA, DITA Open Toolkit, and related technologies
• Promote understand and widespread usage of DITA and the Toolkit
• Make the terms accessible and "reusable" locally in this document and by the greater DITA community by creating the vocabulary as a set of DITA topics and publishing them in the DITA Open Toolkit User Guide
• As much as possible, use DCMI metadata in documenting DITA core vocabulary topics

Without a common understanding of "what we are talking about" and how our dialog relates to resources and information that are part of DITA and the Toolkit, these goals are much more difficult to achieve. We invite the DITA community to participate in the effort to create, promote, and use the DITA core vocabulary.

Administrator or manager audience category

Administrators and managers are people responsible for the administration and management of DITA projects. This category is a target audience for this document. It includes application administrators, staff managers, and project and workflow managers.

Apache Ant

Ant is a Java-based, open source tool provided by the Apache Foundation to automatically implement a sequence of build actions defined in an Ant build script. The Ant functionality is similar to the more well-known UNIX make and Windows nmake build tools; however, instead of using shell-based commands, like make, Ant uses Java classes.

The configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular task interface. Ant can be used for both software and document builds.
DITA Open Toolkit provides Java code and a set of XSLT transform scripts for producing different types of output, for example XHTML, Eclipse help, JavaHelp, and PDF. Ant build scripts build DITA output by controlling the execution of the DITA Open Toolkit Java code and the XSLT transform scripts.

Ant must be installed in your DITA processing environment for DITA Open Toolkit to function.

### Ant processing parameter

An Ant processing parameter is a property set in the Ant build script.

Examples are "args.draft," "basedir," and "dita.dir."

### Ant script

An Ant script is an XML build file, containing a single project and a single or multiple targets, each of which consists of a group of tasks that you want Ant to perform. A task is an XML element that Ant can execute to produce a result. Ant comes with a large number of built-in tasks; you can also add tasks of your own.

DITA Open Toolkit makes use of two kinds of Ant scripts:

- **System scripts**
  - System-level scripts handle DITA source file processing and transformation into published output. They are an integral part of DITA Open Toolkit and should never be modified by users. The files are located in the `ditaot` root directory.

- **User scripts**
  - User-level processing scripts are created and modified by users. They provide to the system scripts (which do the actual processing) information about the names and locations of the DITA source files, where to put the processed target files, and values for specific processing parameters. DITA Open Toolkit contains a number of sample user-level processing files that you can view to gain understanding of the build process, and modify for your own use.

### Audience

An audience is a target group of users.

This document was written for both beginning and advanced users currently implementing or planning to implement DITA and DITA Open Toolkit to produce structured XML documents to be published through any of the supported channels.

Target audience categories and types for this document are:

- **Content specialists** (for example, information architect; content creator and editor; and graphic, interface, print-document, and website designer)
- **Technical specialists** (for example, application designer and developer, content manager, and database and system administrator)
- **Administrators and managers** (for example, application designer and developer, content manager, and database and system administrator)
- **Vendors** (for example, companies that want to embed the Toolkit in a software product)
Authoring tool (DITA)

A "DITA-aware" authoring tool is the software used to create DITA source content.

To be DITA-aware, an authoring tool must:

• Check for "well-formedness," that is, that the opening and closing tags match
  An example would be <note>Text</note>.

• Check for validity, that is, that all DTD rules are obeyed
  An example would be a check to see whether the tags appearing in the document are nested according to rules in the document's DTDs.

DITA-aware authoring tools should insist on well-formedness and validity. That is, the tool should prevent you from saving or warn you before saving a file with errors.

Popular DITA-aware authoring tools include:

• Altova XMLSpy
• Arbortext Editor
• JustSystems XMetaL
• Adobe FrameMaker 7.2 with the Application Pack for DITA
• SyncRO Soft <oXygen/>
• Pixware XMLmind
• Stylus Studio
• Syntext Serna
• XMLBuddy (built into Eclipse)
• Xopus (front end for DocZone content management system)

Best practice

A best practice is a guideline that applies to many similar cases, crosses organizational boundaries, and is agreed to by recognized experts in the relevant field.

Block element

A block element defines the structure of a block of text.

Many block elements in DITA have the same names as HTML tags.

Example(s)
<p>, <sl>, and <example>.

Body element (<body>)

A body element is a container for the main content of a DITA topic.
Build file

An Ant build file is an Ant file that connects the source files and production processes for one or more target publishing environments (for example, XHTML, Eclipse help, or HTML Help)

CMS (Content Management System)

If your DITA project involves large numbers of topics, many authors, or geographically distributed authoring and production teams, you may benefit from the features provided by a CMS, which can include:

- Workflow support
- Validation of topic links
- Support for the Semantic Web
- Localization (translation) support

To be truly effective, a CMS being used to store files for a DITA project must be aware of the tree-structured ("web") nature of the project, the content contained in the source files, and the relationships among the files and their content. The CMS must also be able to report on this kind of information: both meta information about the files, and syntactic and semantic information about the content. Ask your CMS vendor about the product's content analytical capabilities before deciding to buy.

It can also be useful for the CMS to include DITA Open Toolkit processing, and debugging and reporting aids that operate along with file processing. CMSs should also provide basic library (source control) functionality.

Popular CMSs include:

- Hosted (web-based)
  - Astoria
  - DocZone
  - XDocs
- Open-source
  - Alfresco
  - NetBeans
- Proprietary, server-based
  - Documentum
  - Ixiasoft
  - Vasont
- Hosted DITA training site: ditausers.org (includes the DITA Storm editor)

CSS (Cascading StyleSheet)

A CSS file specifies the look and feel of HTML and XHTML documents. DITA Open Toolkit provides default stylesheets; you can override the defaults by including a CSS file of your own in the build.
Choice table

In a task step, a choice table (<choicetable>) presents the user with two or more options (choices) to complete the step.

Example

```
<choicetable frame="none">
  <chhead>
    <choptionhd>If this prompt displays, </choptionhd>
    <chdeschd>type the following command</chdeschd>
  </chhead>
  <chrow>
    <choption>D:\</choption>
    <chdesc>
      <codeblock>C:\</codeblock>
    </chdesc>
  </chrow>
  <chrow>
    <choption>H:\</choption>
    <chdesc>
      <codeblock>C:\</codeblock>
    </chdesc>
  </chrow>
  <chrow>
    <choption>C:\My Documents\...</choption>
    <chdesc>
      <codeblock>cd \</codeblock>
    </chdesc>
  </chrow>
</choicetable>
```

To see how this table displays, go to Verifying the installation on Windows on page 56.

Collection-type attribute

A collection type is one of a group of attributes used to create relationships among "sibling" topics that share a common parent.

To specify a collection-type attribute for a group of topics that do not have a common parent (for example, topics listed in the same cell of a relationship table, use the topic group container element.

Example

```
<reltable>
  <relrow>
    <relcell>
      <topicgroup collection-type="family">
        <topicref href="/release_current/sysreqs.dita"/>
        <topicref href="/installing/installing_overview.dita"/>
      </topicgroup>
    </relcell>
  </relrow>
</reltable>
```

Command element (<cmd>)

In a task step, a command element (<cmd>) describes the action the user needs to take.
Example

```xml
<taskbody>
<context>
<p>Once every 6000 kilometers or three months, change the oil in your car. This will help keep the engine in good condition.</p>
<p>To change the oil:</p>
</context>
<steps>
<step>
<cmd>Remove the old oil filter.</cmd>
</step>
<step>
<cmd>Drain the old oil.</cmd>
</step>
</steps>
</taskbody>
```

Concept

A concept can mean:

- Conceptual, background, or descriptive information.
- A DITA core information type, which is used to document information of a conceptual nature.

Concept analysis

Concept analysis is analysis of the concept types required in a DITA document or group of documents.

Concept information type

A concept information type contains content of a conceptual nature.

Conditional processing (filtering and flagging content)

Conditional processing involves filtering or flagging content based on processing-time criteria, such as a target audience, platform, or product.

You can use metadata on elements to filter or flag content, and to show revised content. You can use attributes to provide information about what product, audience, or platform an element applies to, or what product revision it belongs to.

Say you have a need for two versions of your installation instructions, one for Windows and one for Linux. You can create a topic file with both sets of instructions (with each set properly labeled as either Windows or Linux), and then use a ditaval file to specify your processing rules (for example, whether to produce a Windows or Linux version of the document, or whether to produce a single output file with the content flagged appropriately with Windows and Linux icons).
You can exclude content based on its metadata. By default, all content is included. You can flag content based on metadata. By default, no content is flagged. You can show the revision information for your content. All revision information is hidden by default.

Content

Content is information (for example, the text and graphics that make up a news story appearing on a website) in a DITA file that will be published and delivered to an end user.

Content inventory

A content inventory is an inventory of documents (DITA and non-DITA) in an existing document set. The inventory is input to a documentation plan involving changes to the existing documents, new use of related documents, or the creation of new documents.

Content reference attribute

When a topic references a complete topic or smaller piece of content with the conref attribute, the referenced content gets dynamically copied into the referencing topic. If the referenced topic is changed and the document containing the referencing topic is rebuilt, the new version of the referenced topic is automatically replaced.

Content reuse

Content reuse is the use of a single piece of content in multiple location in a single document, or in multiple, related documents.

One example would be to use the same DITA topic in more than one map.

Another example would be to employ a conref to reuse a glossary or core vocabulary item in an "about" topic.

One of the major reasons users cite for moving to DITA is the ability to reuse content effectively.

Content specialist audience category

Content specialists are primarily responsible for the content of a DITA project.

This category is a target audience category for this document. It includes information architects; content creators and editors; and graphic, interface, print-document, and website designers.

Context element (<context>)

A context element contains information that helps users understand the background and purpose of a task.
Example

In this topic you will create a map to aggregate the topics you created in the previous chapter. The map is based on a template already provided. The map file includes topicrefs to the topics you want to aggregate, process, and publish, and also a relationship table to link the included topics a meaningful way.

You will be working in the directories.

This topic assumes you are familiar with the information in and that you have created the topics according to the instructions in .

You will be working in the directories.

This topic assumes you are familiar with the information in and that you have created the topics according to the instructions in .

Controlled vocabulary

A controlled vocabulary is a specified list of topic names or metadata elements and attributes to be included in a DITA document.

Cross-reference element (<xref>)

A cross-reference element identifies a term or phrase you want to cross reference to another piece of information. The element's hyperlink (<href>) attribute specifies the path and target file.

Example

Set the CLASSPATH environment variable for codeph:dost.jar for codeph>

Definition list (<dl>)

A definition list is a structure for listing terms, products, components, and so forth, along with their definitions. Examples of usage are glossary and product feature lists.

Example

You can get current information about bugs, patches, and change requests in the following locations:
Distributing content

Distributing content means making your published DITA content available to your customers, potential customers, and other interested users.

DITA Open Toolkit distributions

DITA Open Toolkit is available in the following distribution formats:

- Full package distribution
- Small package distribution
- Source distribution

These distributions are all available for download from http://sourceforge.net/projects/dita-ot.

Full package distribution

The full package distribution contains the Toolkit and most of the basic tools required for doing document builds. Included in the full package are:

- DITA Open Toolkit
- Ant build processor
- XML catalog resolver
- FOP processor for creating PDF outputs
- icu4j (ICU) globalization routines
- Xalan XSLT processor
- Shell scripts for setting the necessary runtime environment variables

To process DITA documents you must also download and install the Java J2SE SDK.

If one of your target output types is HTML Help, you will probably want to install the Microsoft HTML Help compiler. If you one of your target output types is JavaHelp, you will probably want to install the JavaHelp processor.

Small package distribution
The small package distribution contains only DITA Open Toolkit. You must separately install all the other required and optional processors to create a functioning build environment.

Release 1.2.2 and prior releases of the Toolkit were distributed only in this way.

You might want to download this distribution if you have a prior version of the Toolkit already installed, since the release 1.3 and 1.3.1 Toolkit prerequisites are still the same as those for release 1.2.2.

The small distribution is typically the one used to embed the Toolkit in other products.

**Source distribution**

The source distribution contains the source and executable code for the Toolkit (and it also contains the source code for this document). You might download this distribution if you need to modify Toolkit Java code or if you want a detailed look at how the Toolkit works.

---

**DITA (Darwin Information Typing Architecture)**

DITA (Darwin Information Typing Architecture) is an XML-based, end-to-end architecture for authoring, producing, and delivering information (often called *content*) as discrete, typed topics. Typical information delivered using the DITA architecture is technical or scientific in nature and published as online help, through product support portals, or as print-ready PDF files.

DITA originated and is extensively used in the IBM Corporation; in 2005 it was adopted as an Organization for the Advancement of Structured Information Standards (OASIS) standard. DITA is currently used in many organizations world-wide, and is supported by an ever-growing list of commercial and open-source tools. DITA is actively being extended and enhanced under the direction of the OASIS DITA Technical Committee.

The DITA architecture, along with appropriate tools, is used to:

- Create, manage, and publish XML-based, structured information in a wide variety of environments and platforms
- Facilitate information sharing and reuse, and collaborative writing projects
- Reduce writing, publishing, and translation costs
- Integrate more tightly with related technologies
DITA Open Toolkit (OT)

DITA Open Toolkit is a Java-based implementation of the OASIS DITA Technical Committee's specification for DITA DTDs and schemas. The Toolkit, which can be used in the Windows, Linux/UNIX, and Mac OS operating environments, transforms DITA content (maps and topics) into deliverable formats. The Toolkit uses Apache Ant for processing.

DITA Open Toolkit publishes to the following environments:

- Key targets
  - XHTML
  - HTML Help
  - PDF2 (processing using the Idiom plug-in and RenderX XEP)
  - Eclipse help

- Other targets
  - DocBook
  - Eclipse content
  - JavaHelp
  - PDF (processing using standard Toolkit functionality and Apache FOP)

**Note:** The DITA Open Toolkit PDF transform, which was part of the 1.2.2 release, has been deprecated in favor of the PDF2 transform, which makes use of a plug-in from Idiom Technologies, Inc.

- troff
- Word RTF
DITA Open Toolkit User Guide

*DITA Open Toolkit User Guide* (this document) is the definitive source of information about DITA Open Toolkit. It is also a product of the architecture and the recommended best practices, having been written entirely in DITA XML and produced using the principles and procedures described in the document.

DocBook

DocBook is a markup language for technical documentation, available in both SGML and XML forms, and publishable to a variety of formats. DocBook began in 1991 as a joint project between HaL Computer Systems and O'Reilly & Associates. In 1998 it moved to the SGML Open consortium, which subsequently became OASIS.

DocBook is one of the DITA target outputs.

**DOCTYPE declaration**

A Document Type Declaration, or DOCTYPE, associates a particular SGML or XML document with a Document Type Definition (DTD).

**Example**

```xml
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" 
  "../dtd/concept.dtd">
```

**DTD (Document Type Definition)**

A DTD is the definition of the allowable elements, attributes, and other document pieces of an XML document. The DITA DTDs are (base) topic, concept, task, reference, map, and bookmap. Every DITA file must reference its DTD in the DOCTYPE declaration.

DTDs are *not* written in XML syntax.

The following figure shows a snippet of the DITA Concept DTD.
**Domain element**

A domain element is an element associated with a particular subject area, for example bioengineering, financial services, or software programming.

**Examples of software domain elements:**

<codeblock>, <msgblock>, <varname> and <systemoutput>.

---

**Eclipse**

Eclipse is an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools, and runtimes for building, deploying, and managing software across the lifecycle.

In addition to providing a development platform, Eclipse can also be used to create XHTML-based information centers (infocenters). This allows users to access Eclipse help files over the Internet or an intranet by displaying the files in a Web browser.

---

**Eclipse content**

Eclipse content is one of the DITA target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plug-in. This allows the output to be displayed as Eclipse help or in an Eclipse Infocenter.

In contrast to Eclipse help output, which produces XHTML files to be directly displayed by a browser, Eclipse content output produces .dita files that are rendered into .html files on-the-fly by the Eclipse help processor.
Eclipse help

Eclipse Help is one of the DITA Open Toolkit target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plug-in. This allows the output to be displayed as Eclipse help or in an Eclipse Infocenter.

Editor

An editor can refer to a person responsible for creating guidelines for writing and publishing DITA documents, and editing DITA documents to ensure conformance to the guidelines.

Editor is of the target audience types for this document, in the content specialist category.

An editor can also mean an authoring tool used to create DITA source content.

Environment variable

An environment variable is a variable you must set for an application to function.

In Windows you set the variables in the Control Panel. In Linux or UNIX you set the variables in the shell profile. Variables you need to set for DITA Open Toolkit are: PATH, CLASSPATH, ANT_HOME, ANT_OPTS, JAVA_HOME, and JHHOME.

Example element (<example>)

An example element (<example>) describes or illustrates the expected or sample outcome of performing a task.

DITA example elements are used only in tasks.

Family linking

See

Collection-type attribute on page 211

Figure element (<fig>)

A figure element is a container that allows you to include an image and, optionally, its caption as content in a DITA file.
Filtering content

See Conditional processing (filtering and flagging content) on page 212.

FOP processor

FOP is an Apache tool that aggregates style and information during the DITA build process for PDF output. The FOP processor must be installed in your DITA processing environment for DITA Open Toolkit to generate PDF output, but it is not part of the Toolkit installation package.

Format attribute

The format attribute is the file type of a referenced file or other information source. The most common formats for DITA files are "dita," "xml," and "ditamap." You can also reference other formats like "pdf" and "html."

Example

```xml
<conbody>
  <p>Motor oil keeps your car's engine smoothly. Oil should be changed every 6000 kilometers.</p>
</conbody>
<related-links>
  <link href="../tasks/changingtheoil.dita" format="dita" type="task">
    <linktext>Changing the oil</linktext>
  </link>
</related-links>
```

Garage sample

The garage sample, which is downloadable from http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html, is a set of DITA source files containing concepts and tasks related to organizing and doing tasks in a garage. The sample map files allow the topics to be published as either a hierarchy or a sequence. The sample also includes Ant scripts that allow you to publish to all target environments.

Note: The garage sample that is part of the DITA Open Toolkit User Guide is not exactly the same set of files as the garage sample that is part of the Toolkit package.

Before you begin to use the garage sample files, we recommend creating two directories in your root directory called DITASAMPLE_GARAGE_SOURCE and DITASAMPLE_GARAGE_OUTPUT (Windows examples would be
C:/DITASAMPLE_GARAGE_SOURCE and C:/DITASAMPLE_GARAGE_OUTPUT) and then copying the garage sample files to DITASAMPLE_GARAGE_SOURCE. Your directory structure should then look like this:

We recommend that you process your first test case to the XHTML target (dita2xhtml), which generally provides predictably positive results. The dita2htmlhelp, dita2pdf2, and the Eclipse targets (dita2eclipsecontent and dita2eclipsehelp) are also very reliable. The other targets are, in our experience, somewhat problematic, and may require special coding to get acceptable results.

One of the garage sample targets (dita2filtered, which produces XHTML output) illustrates conditional processing using filtering. The output is a hierarchically organized file with four of the garage topics (those dealing with "oil" or "snow") excluded.

Grocery shopping sample

The grocery shopping sample, which is downloadable from http://dita-ot.sourceforge.net/SourceForgeFiles/doc/user_guide.html, is a simple DITA project that includes seven topics: an overview topic, two concepts, two tasks, and two reference topics. The project also includes a map that aggregates the files and links them meaningfully using a relationship table. Ant scripts that process to all DITA Open Toolkit targets are also provided.

Before you begin to use the grocery shopping sample files, we recommend creating two directories in your root directory called DITASAMPLE_GROCERY_SOURCE and DITASAMPLE_GROCERY_OUTPUT (Windows examples would be C:/DITASAMPLE_GROCERY_SOURCE and C:/DITASAMPLE_GROCERY_OUTPUT) and then copying into them both the grocery shopping sample files DITASAMPLE_GROCERY_SOURCE. Your directory structure should then look like this:
The grocery shopping sample assumes you are already familiar with the garage sample provided as both a project model as well as a tool to verify your DITA Open Toolkit installation, and that you have processed the garage sample as described in *Processing (building) and publishing DITA documents* on page 75.

### Graphic designer

A graphic designer is a person responsible for designing and creating the graphics in DITA documents. Graphic designer is one of the target audience types for this document, in the content specialist category.

### Guideline

A guideline is a recommendation about how to perform a task or set of tasks. Guidelines generally reflect an organization’s policies or “groundrules,” or they explain how “best results” (in a technology sense or for consistency of results) can be obtained.

### Hover help

Hover help is a form of context-sensitive help. Similar to What's This? help and balloon help, hover help displays a small pop-up window when the mouse pointer is over an element of the interface. A brief description of the interface element is displayed in the pop-up window.

### HTML Help

HTML help is a compiled help system. HTML Help is one of the DITA Open Toolkit target outputs. If you plan to publish HTML Help, the HTML Help compiler must be installed in your DITA processing environment.

### HTML Help compiler

A Microsoft product that generates (X)HTML Help files during the DITA build process. HTML Help is a compiled help system. HTML Help is one of the DITA Open Toolkit target outputs. The HTML Help compiler must be installed in your DITA processing environment for DITA Open Toolkit to function, but it is not part of the Toolkit installation package.

### ID attribute

An ID attribute is an identifier unique within a given topic that can be used to reference the topic.
IDs can contain letters, numbers, and underscores.

**Examples**

```xml
<concept id="framework">
<title>The DITA authoring/production framework</title>
</concept>

<conbody>
<section id="javahelp_term">

## Indexing content

Indexing in DITA is accomplished with the `<indexterm>` tag, which can be nested.

**Example**

```xml
<indexterm>processing
<indexterm>to PDF targets</indexterm>
</indexterm>
```

The code produces the following two-level index entry:

- processing
to PDF targets

## Information analysis

Information analysis is a task that is performed in the early stages of planning a structured documentation set.

An information analysis should include user, concept, task, and reference analyses. Output from the information analysis is a planning document listing planned concept, task, and reference topics organized by key topic.

## Information architect

An information architect is a person responsible for designing DITA documents, planning for content reuse, and creating DITA maps.

Information architect is one of the target audience types for this document, in the content specialist category.
Information developer

See

_Writer_ on page 245

Information element (<info>)

An information element (in a task step), describes additional information required to complete the step beyond the instruction in the command element.

Example

```xml
<step>
  <cmd>Save and extract the package file into a Linux home directory. </cmd>
</step>

<info>
  <note>You can extract all package files and toolkits either to your private home directory for exclusive usage or to the <filepath>/usr/local/share/</filepath> directory for sharing. </note>
</info>
```

Information type

Information typing is the architectural basis of topic-based authoring, and the practice of identifying types of topics that contain distinct kinds of information, such as concepts, tasks, and reference information. Topics that answer different kinds of questions can be categorized as different information types.

Classifying information by type helps authors:

- Design new information more easily and consistently
- Ensure that the right design gets used for the kind of information being presented (for example, retrieval-oriented structured like tables for reference information, and simple sequences of steps for task information)
- Focus on tasks, which is what most users are likely to be interested in
- Factor out supporting concepts and reference information into other topics, where they can be read if required and ignored if not
- Eliminate unimportant or redundant information, and identify common or reusable subjects

Information typing is part of the general authoring approach called structured writing, which is used across the technical authoring industry to improve information quality. It is based on extensive research and experience, for example Robert Horn's Information Mapping.

The core information types in DITA are concept, task, and reference topic.
**Inheritance**

In object-oriented programming, inheritance is a way to form new classes using classes that have already been defined. DITA and DITA Open Toolkit are structured around the principle of inheritance.

In DITA, child topics or elements inherit attributes from their parents. For example, metadata applied to a section of a DITA file will automatically be applied to topics contained in the section. Inheritance also plays an important role in DITA specialization, which allows you to extend a base topic to match your specific requirements by defining only the differences between it and its base topic; the bulk of the specialized definition is inherited.

**Java**

Java is a general-purpose, object-oriented programming language.

The DITA Open Toolkit and other tools associated with it (for example, Apache Ant and SAXON) are written in Java. The fact that Java is portable allows the Toolkit to run in virtually any operating environment (for example, Windows, Linux/UNIX, and Mac OS).

The following figure shows a Java code example (Toolkit code).

```java
/*
 * This file is part of the DITA Open Toolkit project hosted on Sourceforge.net. see the accompanying license.txt file for
 * applicable licenses.
 */

/*
 * (c) Copyright IBM Corp. 2004, 2005 All Rights Reserved.
 */
package org.dita.dofl.pipeline;

import org.dita.dofl.exception.DITAException;
import org.dita.dofl.module.AbstractPipelineModule;
import org.dita.dofl.module.ModuleFactory;

/**
 * PipelineFacade implement AbstractFacade and control the constructing and executing
 * of a module.
 *
 * @author Lian Li
 */
public class PipelineFacade implements AbstractFacade {

    /**
     * Automatically generated constructor: PipelineFacade
     */
    public PipelineFacade() {
    }
}
```

**Java Development Kit (JDK)**

The JDK is an IBM/Sun Microsystems product used by developers to write, compile, debug, and run Java applets and applications.
The JDK must be installed in your DITA processing environment for the DITA Open Toolkit to function, but it is not part of the Toolkit installation package.

**JavaHelp**

JavaHelp is a set of Java-based files that can be incorporated in applications, components, operating systems, applets, and devices.

JavaHelp is one of the DITA Open Toolkit target outputs. If you plan to produce JavaHelp output, the JavaHelp processor must be installed in your DITA processing environment for DITA Open Toolkit to function, but it is not part of the Toolkit installation package.

**JavaHelp processor**

The JavaHelp processor is a Sun Microsystems product used to generate JavaHelp files, which can be incorporated in applications, components, operating systems, applets, and devices.

JavaHelp is one of the DITA Open Toolkit target outputs. If you plan to produce JavaHelp output, the JavaHelp processor must be installed in your DITA processing environment for DITA Open Toolkit to function, but it is not part of the Toolkit installation package.

**Keyword element (<keyword>)**

A keyword element identifies a term or phrase in an abstract, title, subject heading, or general text that may be used in a special context, such as a glossary or search engine.

**Example**

```xml
<metadata>
<keywords>
<keyword>Ant script</keyword>
<indexterm>Ant scripts</indexterm>
<indexterm>definition</indexterm>
<indexterm>usage</indexterm>
</keywords>
</metadata>
```

**Linking attribute**

A linking attribute controls the direction of links between topic references and whether a topic can be linked to or from.

For example, if a topic has a linking attribute of "targetonly" it cannot link to other topics, but other topics can link to it. A "sourceonly" link allows a topic to link to other topics, but not the other way around.
**Linking content**

In DITA, linking content involves various methods that connect topics to each other or to external references. In DITA, linking can be implemented through various elements, such as cross-reference (<xref>) and related links (<related-links>), and through relationship tables.

**DITA Toolkit processing log**

When a Toolkit Ant build script runs, it sends processing messages to the log. By default the messages appear only on the console. By adding the parameter `-logger org.dita.dost.log.DITAOTBuildLogger` to the invocation of Ant, you can also write the messages to a disk file. You can also specify the location for the log file by setting the `args.logdir` Ant processing parameter.

**Map**

A map is an aggregation of the topics in a DITA document, with the topics arranged as a list or a hierarchy. DITA documents can have multiple maps or sets of maps for a given document. For example, a software product available for both Windows and Linux might have two maps, each specifying the topics to include in that document version. As another example, a large, complex document might have a master map that included multiple submaps, specifying the topics to include in various "chapters" and "sections."

**Example**

```xml
<?xml version="1.0" encoding="utf-8"?>
<map id="customizing_map" title="Customizing your published output">
  <topicref href="customizing.dita"/>
  <topicref href="condproc.dita"/>
  <topicref href="aboutcondproc.dita"/>
  <topicref href="condproc_using.dita"/>
  <topicref href="addmeta.dita"/>
  <topicref href="filtcontent.dita"/>
  <topicref href="flagcontent.dita"/>
  <topicref href="showrev.dita"/>
</map>
```
Metadata

Metadata is semantic information about the information in a document, for example the name of the document's author, the date the document was created, the name of the product the information is describing, the target audience, and copyright information.

In DITA you can specify metadata at the topic or map level, with map-level metadata overriding topic entries.

Example

```xml
<metadata>
  <keywords>
    <keyword>Ant script</keyword>
    <indexterm>Ant scripts</indexterm>
    <indexterm>definition</indexterm>
    <indexterm>usage</indexterm>
  </keywords>
  <prodinfo>
    <prodname>DITA Open Toolkit</prodname>
    <vrm list=
      vrm version="1.3.1"/>
  </prodinfo>
</metadata>
```

Navigation title (<navtitle>)

A navigation title is an alternative title for a topic, specified as an attribute on a topicref element.

Navigation titles are usually shorter than the full title, or they could be set, as in the following example, to allow the use of 'scope="peer"' in related links (see Related links element (<related-links>) on page 235).

Example

```xml
<concept id="audience">
  <title>Audience type</title>
  <titlealts>
    <navtitle>Audience</navtitle>
  </titlealts>
  <shortdesc/>
  <prolog>
```

OASIS (Organization for the Advancement of Structured Information Standards)

OASIS is a not-for-profit, global consortium that drives the development, convergence, and adoption of e-business standards.

DITA is an OASIS standard.
Ordered list (<ol>)

An ordered list is a list (typically numbered) in which the order of list items is significant (for example, steps in a procedure.)

Example

```xml
<conbody>
<ol>
  <li>garden dirt</li>
  <li>tools</li>
  <li>trash</li>
</ol>
</conbody>
```

OWL (Web Ontology Language)

OWL is a W3C standard for providing an exact description of things and their relationships. OWL is built on top of RDF. OWL is meant to be read and understood by computers.

PDF (Portable Document Format)

PDF is an open standard file format, proprietary to Adobe Systems, for representing two-dimensional documents in a device-independent and resolution-independent format. PDF files encode the exact look of a document in a device-independent way.

PDF (with the output processed by FOP) and PDF2 (using the Idiom plug-in, with the output processed by XEP) are DITA Open Toolkit target outputs.

Phrase element

A phrase element is used to describe words or phrases within a block or structure element.
Phrases include semantic elements used inline to mark items like user interface controls, keywords, index terms, and cross references. Phrases also include domain elements (associated with a particular subject area) and typographic elements (tags to mark words or phrases to be displayed in italic, bold, underscore, and so forth).

**Plug-in**

You can extend or enhance the product capabilities provided by DITA Open Toolkit by installing Toolkit plug-ins. Once installed, a plug-in becomes part of the Toolkit environment and can be used to add new specializations or to define new targets for output processing.

**Post-requirement element (**<postreq>**)**

In a task, a post-requirement element specifies something a user needs to do after completing the task.

**Example**

```
<taskbody>
<prereq>Do your produce shopping <i>after</i> you have bought your canned goods.
Otherwise, the cans might bruise the fruits and vegetables!</prereq>
<context>Remember to look for local produce in season.
The fruits and vegetables you buy will be fresher and cheaper!
Say you're shopping in August for peaches grown locally.
When you get to the produce section of your grocery store:</context>
<steps>
<step>
<cmd>Get a plastic or paper bag to hold the peaches.</cmd>
</step>
<step>
<cmd>Pick out the freshest peaches you can find, and put them gently into your bag.</cmd>
</step>
<info>To avoid bruising, don't put more than 6 peaches in each bag.</info>
</step>
<step>
<cmd>Put the bag gently into your grocery cart.</cmd>
</step>
</steps>
<postreq>When you check out, be sure the grocery clerk also handles your peaches carefully.</postreq>
</taskbody>
```

**Prerequisite element (**<prereq>**)**

In a task, a prerequisite element specifies something a user needs to know or do before beginning the task.

**Example**

```
<taskbody>
<prereq>Do your produce shopping <i>after</i> you have bought your canned goods.
Otherwise, the cans might bruise the fruits and vegetables!</prereq>
<context>Remember to look for local produce in season.
The fruits and vegetables you buy will be fresher and cheaper!
```
Say you’re shopping in August for peaches grown locally. When you get to the produce section of your grocery store:

<steps>
<step>
<cmd>Get a plastic or paper bag to hold the peaches.</cmd>
</step>
<step>
<cmd>Pick out the freshest peaches you can find, and put them gently into your bag.\</cmd>
</step>
<info>To avoid bruising, don’t put more than 6 peaches in each bag.\</info>
</step>
<step>
<cmd>Put the bag gently into your grocery cart.</cmd>
</step>
</steps>

(postreq>When you check out, be sure the grocery clerk also handles your peaches carefully.</postreq>
</taskbody>

Print-document designer

A print-document designer is a person responsible for designing DITA print documents.

Print-document designer is one of the target audience types for this document, in the content specialist category.

Processing (building)

Processing is producing output files from a set of DITA source files.

DITA Open Toolkit, working with other tools like the Java Development Kit (JDK) and Ant, provides a method to process DITA documents.

The following figure provides an overview of the processing and publishing of DITA documents using DITA Open Toolkit:

- In step 1, the Ant build tool is initialized (either through an optional batch script or directly from the command line), and arguments that will be passed from the Ant script to Ant are validated.
- In step 2, the Ant build tool calls the Toolkit, which produces the rendered temporary files. Input to this step is the .dita and .ditamap source files, and the DITA project DTDs or schemas.
- In step 3, the XSLT processor (SAXON or Xalan) produces the published output files. Input to this step are the XSLT stylesheets for the project, and the temporary files produced in step 2.
**Project manager**

A product manager is a person responsible for designing overall processes and procedures, and setting schedules for DITA projects.  

Product manager is one of the target audience types for this document, in the administrators and managers category.

**Prolog element (<prolog>)**

A prolog element is an element containing metadata for a topic (for example, author, creation date, and modification date).

**Example**

```xml
<prolog>
<author type="creator">Anna van Raaphorst</author>
<copyright>
<copyryear year="2007"/>
<copyrholder>VR Communications, Inc.</copyrholder>
</copyright>
<critdates>
<created date="2006-August-07"/>
<revised modified="2006-August-07"/>
</critdates>
<metadata>
<keywords>
```

![Diagram](image)
RDF (Resource Description Framework)

RDF is a W3C standard for describing information about a resource on the Web. RDF is meant to be read and understood by computers.

While it does not directly contain support for generating external or embedded RDF, DITA Open Toolkit does have some functionality that can be used to create RDF.

**Dublin Core**

The Dublin Core is a standard for metadata that is used to describe online information. The XHTML output produced by DITA Open Toolkit contains Dublin Core metatags generated from the various elements contained within the prolog, title, and short description elements in DITA source files. Further processing of the XHTML output can create RDF "triples" using these meta tags. (Functionality for that processing is not contained in the Toolkit.)

An RDF triple contains three pieces of information, a subject, a property type, and a value for the property. For example, a `<title>` element might produce the following output in the generated XHTML:

```xml
<meta name="DC.Title" content="About metadata"/>
```

In this example the triple says the web page name is "About metadata".

**SKOS**

The Thesaurus (aka Taxonomy) plug-in can be installed with DITA Open Toolkit to provide a DITA specialization that can be used to identify and process content based on what the information is about by generating SKOS output. See About DITA Open Toolkit plug-ins on page 63 for more information about this plug-in.

Reference analysis

A reference analysis is an analysis of the reference information required in a DITA document or group of DITA documents.

Reference information type

Reference information type is an information type for content that focuses on properties and relationships among a number of similar items.

Content in a DITA reference information type is used to record and present (often in a tabular format) reference (as contrasted with narrative) information. The information is presented to users in a way that facilitates quick lookup.
Related links element (**<related-links>**)

A related links element is a container for linking related topics to a given topic.

**Example**

```xml
<related-links>
<link href="audience.dita" scope="peer"/>
<link href="../evaluating/framework.dita" scope="local"/>
</related-links>
```

**Note:** If you use scope="peer" to work in this example instead of scope="local", you must set a navtitle in the target (audience.dita).

Relationship table

In a DITA map, a relationship table is a systematic structure for creating and maintaining relationships among DITA topics.

The relationships are displayed in a tabular format. Each row of the table represents a relationship, which is usually rendered as a link. Each cell lists one or more participants in the relationship. No relationships exist between the rows of the table.

**Example**

```xml
<!-- Relationship table -->
<!-- The related concept, task, and reference files point to each other -->
<reltable>
<relheader>
<relcolspec type="concept"/>
<relcolspec type="task"/>
<relcolspec type="reference"/>
</relheader>
<relrow>
<relcell>
<topicref href="concepts/about_produce.dita"/>
</relcell>
<relcell>
<topicref href="tasks/choosing_produce.dita"/>
</relcell>
<relcell>
<topicref href="reference/produce.dita"/>
</relcell>
</relrow>
<relrow>
<relcell>
<topicref href="concepts/about_cannedgoods.dita"/>
</relcell>
<relcell>
<topicref href="tasks/buying_cannedgoods.dita"/>
</relcell>
<relcell>
<topicref href="reference/cannedgoods.dita"/>
</relcell>
</relrow>
<reltable>
```
Result element (<result>)

In a task, a result element provides the user with information about what completing a task should have accomplished.

Example

```xml
<taskbody>
  <context>
    <p>Keep your car looking great by washing it regularly.</p>
  </context>
  <steps>
    <step>
      <cmd>Move the car onto the driveway.</cmd>
    </step>
    <step>
      <cmd>Attach the water hose to a spout and pull the free end over to the car.</cmd>
    </step>
    <step>
      <cmd>Fill a bucket with soapy water.</cmd>
    </step>
    <step>
      <cmd>Use a sponge to apply the soapy water to the car and scrub off the dirt.</cmd>
    </step>
    <step>
      <cmd>Rinse the car by spraying clean water from the hose.</cmd>
    </step>
    <step>
      <cmd>Dry the car using a dampened chamois.</cmd>
    </step>
  </steps>
  <result>
    <p>
      ![washing the car](../images/carwash.jpg)
    </p>
  </result>
</taskbody>
```

RSS (Really Simple Syndication)

RSS is a family of web feed formats written in XML and used in content syndication by enabling applications like RSS readers to find out when new information is available on a website.

Since creation of output from DITA source files is automated by using Ant, it is possible to create or update RSS information about the output at the same time the output is produced. The RSS file can then be uploaded along with the output so its availability can be known to those subscribed to the RSS feed.

SAXON XSLT processor

SAXON transforms DITA source files into rendered output using the DITA XSLT stylesheets.

You need to install either the SAXON XSLT processor or the Xalan XSLT processor in your DITA processing environment for DITA Open Toolkit to function, but neither processor is part of the Toolkit installation package.
Schema

A schema defines the structure of an XML document. Schemas define elements and attributes valid for a particular namespace.

Schemas are W3C standards, and are written in XML syntax.

DITA schemas exist, but currently are seldom used. (DTDs are more common.)

The following figure shows the schema for DITA Concept.

```xml
<!-- Base type: topic.class -->
<xs:complexType name="concept.class">
  <xs:sequence>
    <xs:group ref="title"/>
    <xs:group ref="titlealts" minOccurs="0"/>
    <xs:choice minOccurs="0">
      <xs:group ref="shortdesc"/>
      <xs:group ref="abstract"/>
    </xs:choice>
    <xs:group ref="prolog" minOccurs="0"/>
    <xs:group ref="conbody" minOccurs="0"/>
    <xs:group ref="related-links" minOccurs="0"/>
    <xs:group ref="concept-info-types" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="conref" type="xs:string"/>
  <xs:attributeGroup ref="select-atts"/>
  <xs:attribute ref="ditaarch:DITAArchVersion"/>
  <xs:attribute name="outputclass" type="xs:string"/>
  <xs:attributeGroup ref="localization-atts"/>
  <xs:attributeGroup ref="global-atts"/>
  <xs:attribute ref="class" default="- topic/topic concept/concept "/>
</xs:complexType>
```

Scope attribute

A scope attribute indicates the location of source topics relative to a DITA map.

Values are "peer" (for references to topics in the same directory as the map), "local" (for references not local but in the same document area), and "external" (for topics outside the document set).

Example

```xml
<p>Before you use DITA Open Toolkit, be sure your operating environment meets the system requirements described in</p>
<xref href="../release_current/sysreqs.dita" scope="local"/>

System requirements and supported applications</xref>.</p>
</section>
<section id="howproduced">
<title>How and why this document was produced</title>
<p>This document was produced as a collaborative effort</p>
```
Search title element (<searchtitle>)

A search title is an alternative title used in search result summaries.

Short description

A short description is text that briefly introduces and describes a topic. In DITA, short descriptions are tagged with the short description element (<shortdesc>).

Example

```xml
<concept id="debuggingtools_overview">
  <title>About the debugging, reporting, and file generation tools</title>
  <shortdesc>Information about tools and techniques you can use to debug your processing problems, get information about your source files, and generate DITA files automatically from XML-based source code.</shortdesc>
</concept>
```

The DITA short description is one device to achieve "progressive disclosure" in topic-based writing, in the sense that title expands to short description, which expands to the topic itself as a user drills into information. Consistent use of short description helps to provide this progressive user experience. The "short description" approach in writing topics could be compared with the “topic sentence” approach in writing paragraphs that is regularly taught in classes from elementary school through college.

The DITA short description has several functions (and reiterating the title is not one of them):

- To act as a sort of “thesis paragraph” for the topic, letting readers quickly assess whether it's worth continuing
- To act as a link hover help in related topics, letting uncertain readers get a sense of the target's content when the title isn't enough
- To act as a summary of the content in generated overviews, letting readers who are drilling down from higher-level content navigate successfully to the right answer through progressive disclosure instead of massive scanning

Simple list (<sl>)

A simple list is a list in which the order of list items is not significant. A simple list is similar to unordered list, but in a simple list the items are not marked with a bullet or other symbol.

The following simple list functions as a topic table of contents.
Example

```
<section>
<p>Sections in this topic:
<sl>
<sl1>
<xref href="#aboutditaotugref/contents">Document contents</xref>
</sl1>
<sl1>
<xref href="#aboutditaotugref/target_audience">Audience</xref>
</sl1>
<sl1>
<xref href="#aboutditaotugref/prerequisites">Prerequisites</xref>
</sl1>
<sl1>
<xref href="#aboutditaotugref/howproduced">How this document was produced</xref>
</sl1>
</sl>
</p>
</section>
```

**Simple table element** (<simpletable>)

A simple table is used to describe tabular information that does not require control over the display properties and layout.

**Example**

```
<simpletable>
<thead>
<stentry>Name, description</stentry>
<stentry>Location</stentry>
</thead>
<strow>
<stentry>
<p><b>The Localization Industry Standards Association (LISA)</b> is an international forum for organizations doing business globally. It has published a set of localization best practices that list the right and wrong ways to support international customers, products, and services.</p>
</stentry>
<stentry>
<xref href="http://www.lisa.org/" format="html" scope="external"/>
</stentry>
</strow>
...
</simpletable>
```

**SourceForge website**

SourceForge is a download location for DITA Open Toolkit, *DITA Open Toolkit User Guide*, and other DITA-related tools, documents, and samples.
Specialization (information design reuse)

Specialization is the process by which new designs are created based on existing designs, allowing new kinds of content to be processed using existing processing rules. One of the key characteristics of DITA specialization is inheritance, which allows you to create new information types from existing ones. With inheritance you can use a class attribute to map an existing parent element to the specialized element you want to create.

Specialization allows you to define new kinds of information (new structural types or new domains of information), while reusing as much of existing design and code as possible, and minimizing or eliminating the costs of interchange, migration, and maintenance.

There are two kinds of specialization hierarchy: one for structural types (with topic or map at the root) and one for domains (with elements in topic or map at their root). Structural types define topic or map structures, such as concept or task or reference, which often apply across subject areas (for example, a user interface task and a programming task may both consist of a series of steps). Domains define markup for a particular information domain or subject area, such as programming, or hardware. Each of them represent an "is a" hierarchy, in object-oriented terms, with each structural type or domain being a subclass of its parent. For example, a specialization of task is still a task, and a specialization of the user interface domain is still part of the user interface domain.

Suppose a product group identifies three main types of reference topic: messages, utilities, and APIs. They also identify three domains: networking, programming, and database. By creating a specialized topic type for each kind of reference information, and creating a domain type for each kind of subject, the product architect can ensure that each type of topic has the appropriate structures and content. In addition, the specialized topics make XML-aware search more useful, because users can make fine-grained distinctions. For example, a user could search for xyz only in messages or only in APIs, as well as searching for xyz across reference topics in general.

Rules govern how to specialize safely: Each new information type must map to an existing one, and new information types must be more restrictive than the existing one in the content that they allow. With such specialization, new information types can use generic processing streams for translation, print, and web publishing. Although a product group can override or extend these processes, they get the full range of existing processes by default, without any extra work or maintenance.

Step element (<step>)

A step element is a micro-task that a user needs to perform to complete a more high-level task. For example, three of the steps in making a peanut butter sandwich might be finding the peanut butter jar, opening it, and spreading the peanut butter on a piece of bread with a knife.

Example

```xml
<taskbody>
<context>If you are new to DITA and DITA Open Toolkit, we recommend that you follow these steps to get started.</context>
<steps>
  <step>
    <cmd>Read the topics in</cmd>
    <xref href="../evaluating/evaluating.dita" scope="local">
      Evaluating DITA and DITA Open Toolkit</xref>
    for suggestions on how to evaluate for use in your environment, and how to choose your initial pilot project.</step>
</steps>
<context>Be sure your system environment meets the requirements in</context>
```
Structure element

A structure element is a base element that you can use with every DITA information type. Examples are topic, title, short description, and body.

Structured writing

See

*Topic-based authoring (structured writing)* on page 243.

Stylesheet

A stylesheet is a mechanism for adding style (for example, fonts, colors and spacing) to web documents.

Staff manager

A staff manager is a person responsible for managing DITA projects and the personnel involved in project planning and implementation.

Manager is one of the target audience types for this document, in the administrators and managers category.

Table element (<table>)

A table element is a container element to define display properties and layout of tabular items.

Example

```xml
<table colsep="1" rowsep="1">
  <tgroup cols="2">
    <colspec colname="COLSPEC0" colwidth="40%"/>
    <colspec colname="COLSPEC1" colwidth="60%"/>
    <thead>
      <row>
        <entry colname="COLSPEC0" valign="top">Description</entry>
        <entry colname="COLSPEC1" valign="top">Information source</entry>
      </row>
    </thead>
    <tbody>
      <row>
        <p>
```
Technology specialist audience category

Technology specialists are persons responsible for the technology of DITA projects.

Technology specialist is a target audience category for this document. The category includes application designers and developers, content managers, and database and system administrators.

Task analysis

Task analysis is an analysis of the the user tasks required in a DITA document or group of DITA documents.

Task information type

A task information type is an information type for content that describes procedures or sets of steps a user follows in performing a task or using a product.
**Task Modeler**

Task Modeler is an IBM-produced, Eclipse-based software tool for modeling human activity as a hierarchy of task and related elements.

An information architect can use Task Modeler to design DITA maps. A usability practitioner can produce either classic HTA (Hierarchical Task Analysis) diagrams or RAG (Roles and Goals) diagrams.

**Topic information type**

The topic is the base DITA information type. Most DITA topics contain content; they can also be container topics that aggregate other topics.

**Topic-based authoring (structured writing)**

A topic is a unit of information with a title and content, short enough to be specific to a single subject. A topic should be short enough to be easily readable, but long enough to make sense on its own.

A document may contain one topic or multiple topics, and a document type might support authoring one or many kinds of topics.

Regardless of where they occur, all topics have the same basic structure and capabilities. Books, PDF files, websites, and help sets, for example, can all be constructed from the same set of underlying topic content, although there may be some topics that are unique to a particular deliverable, and the organization of topics may differ to take advantage of the unique capabilities of each delivery mechanism.

Reference information is inherently topic-oriented, since it requires information to be modular and self-contained for the sake of retrievability.

**troff**

troff is a document processing system developed by AT&T for the UNIX operating system. troff is the default format for UNIX documentation. Various macro packages handle different document styles, including macros for formatting research papers, man macros for creating UNIX man pages, and the ms macros for books, technical documentation, and reports.

troff is one of the DITA Open Toolkit target outputs.

**Typographic element**

A typographic element is an element to mark words or phrases to display in bold, italic, underlined, as a subscript or superscript, and in teletype (monospace, usually Courier) font.

**Example**

```<p>```
Unordered list (<ul>)

An unordered list is a list in which the order of list items is not significant. Items are usually marked with a bullet or other similar symbol.

Example

Once we had large numbers of source files and directories to deal with, we ran into the following kinds of error situations that were difficult to resolve:

- We had problems finding out the root cause of error messages in the Ant build log.
- We lost track of which source files had references to other source files.
- We often didn't know which URLs were linked to in the source files.
- We wondered which source files were not actually being used.

Vendor audience category

Vendors are persons or organizations creating software (for example, plug-ins) complementary to DITA or DITA Open Toolkit.

This category is a target audience category for this document.

Website designer

A website designer is a person responsible for the overall design and development of the websites and webpages on which DITA HTML-based documents are published.

Website designer is one of the target audience types for this document, in the content specialist category.

Word RTF (Rich Text Format)

RTF (Rich Text Format) is a proprietary document file format developed and owned by Microsoft for cross-platform document interchange. Most word processors are able to read and write RTF documents.

Word RTF is one of the DITA Open Toolkit target outputs.
**Writer**

A writer is a person responsible for creating DITA topics and implementing effective topic reuse.

Writer is one of the target audience types for this document, in the content specialist category.

---

**Xalan XSLT processor**

Xalan is a tool developed as part of the Apache XML project that transforms DITA source files into rendered output using the DITA XSLT stylesheets.

You need to install either the Xalan XSLT compiler or the SAXON XSLT processor in your DITA processing environment for DITA Open Toolkit to function, but neither processor is part of the Toolkit installation package.

---

**XEP processor**

XEP is an XSL FO and SVG processor that aggregates style and information during the DITA build process for PDF2 output.

The XEP processor must be installed in your DITA processing environment for DITA Open Toolkit to generate PDF2 output, but it is not part of the Toolkit installation package. You can obtain the XEP processor from RenderX, Inc.

---

**XHTML**

XHTML (Extensible HyperText Markup Language), is a markup language that has the same expressive possibilities as HTML, but a stricter syntax. XHTML is an application of XML. XHTML is a W3C standard.

XHTML is one of the DITA Open Toolkit target outputs.

---

**XML (eXtensible Markup Language)**

XML is a W3C-recommended general-purpose markup language that supports a wide variety of applications, and is extensible. XML is a simplified subset of Standard Generalized Markup Language (SGML). DITA is a “dialect” of XML.

The following figure shows a simple, annotated XML file.
The following figure shows the XML object model for the simple XML file.

```
<?xml version="1.0"?>
<!-- This is an example -->
<people>
  <person>
    <name>Sue Writer</name>
    <employment status="part time"/>
    <motto xmlns="http://www.w3.org/1999/xhtml">
      I
      <h:b>love</h:b> DITA!
    </motto>
  </person>
</people>
```

The following figure shows a simplified tree structure model for a typical DITA project.
XML declaration

An XML declaration is a processing instruction that identifies a document as XML. DITA documents must begin with an XML declaration.

Example

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

XPath

XPath traverses an XML document's tree structure to describe node sets in an XML document. XPath uses pattern-matching to find sets of elements (for example, `<note>Text</note>`).

XPath is a W3C standard and is used in XSLT.

The following figure shows XPath expressions in an XSLT document.

```xml
<xsl:template match="/*[contains(@class, ' map/map ')]">
  <xsl:param name="pathFromMapList"/>
  <xsl:if test=".//*[@class, ' map/topicref '][not(@toc='no')]">
    <UL><xsl:value-of select="$newline"/>
      <xsl:apply-templates select="*[contains(@class, ' map/topicref ')]"/>
      <xsl:with-param name="pathFromMapList" select="$pathFromMapList"/>
    </xsl:apply-templates>
    </UL><xsl:value-of select="$newline"/>
  </xsl:if>
</xsl:template>
```
XSLT (eXtensible Stylesheet Language Transformations)

XSLT, a W3C standard, is a language for transforming XML into other formats.

XSLT stylesheets are written in XML syntax.

In the final stage of processing, the DITA Open Toolkit runs XSLT stylesheet transforms (using either SAXON or Xalan) to produce output files. In certain cases, it is possible to override stylesheet processing to customize the output.

The following figure shows the XSLT stylesheet for the note element.

```xml
<xsl:template match="*" mode="process.note">
  <div class="note">
    <xsl:call-template name="commonattributes"/>
    <xsl:call-template name="setidname"/>
    <span class="notetitle">
      <xsl:call-template name="getString">
        <xsl:with-param name="stringName" select="'Note'"/>
      </xsl:call-template>
    </span>
    <xsl:call-template name="getstring">
      <xsl:with-param name="stringName" select="'ColonSymbol'"/>
    </xsl:call-template>
    <xsl:call-template name="flagit"/>
    <xsl:call-template name="revblock"/>
  </div>
</xsl:template>
```
Index

A

access and distribution use cases 43
accessing your information 151, 153, 154
   DITAblogs for 153
   more information about 154
administrator or manager audience category 207
Adobe FrameMaker 17, 189, 209
   migrating from unstructured to structured 189
Adobe FrameMaker DITA Application Pack 189
Alfresco 17, 210
Altova XMLSpy 17, 61, 209
Ant 54, 56, 58, 75, 82, 207
   installing on Linux 56
   installing on Mac OS 58
   installing on Windows 54
   processing parameters for 82
Ant parameters 82, 162, 208
   args.artlbl 82
   args.copycss 82, 162
   args.css 82, 162
   args.csspath 82, 162
   args.cssroot 82
   args.dita.locale 82
   args.draft 82
   args.eclipse.provider 82
   args.eclipse.version 82
   args.eclipsecontent.toc 82
   args.eclipsehelp.toc 82
   args.fo.img.ext 82
   args.fo.output.rel.links 82
   args.fo.userconfig 82
   args.ftr 82
   args.hdr 82
   args.html.includefile 82
   args.indexshow 82
   args.input 82
   args.logdir.toc 82
   args.outext.map 82
   args.xhtml.toc 82
   args.xsl 82
   basedir 82
   clean.temp 82
   dita.dir 82
   dita.extname 82
   dita.input.valfile 82
   dita.temp.dir 82
   output.dir 82
   transtype 82
Ant scripts 75, 107, 208
   production notes for 107
   system-level 75
   user-level 75
ANT_HOME environment variable 54, 56, 57, 220
ANT_OPTS environment variable 54, 56, 57, 220
Apache FOP processor 59, 221
   downloading 59
API reference specialization 63
Application Pack for DITA (FrameMaker) 17, 209
applications, supported 47
Arbortext Editor 17, 209
architecting a DITA document 196
args.artlbl Ant parameter 82
args.copycss Ant parameter 82, 162
args.css Ant parameter 82, 162
args.csspath Ant parameter 82, 162
args.cssroot Ant parameter 82
args.dita.locale Ant parameter 82
args.draft Ant parameter 82
args.eclipse.provider Ant parameter 82
args.eclipse.version Ant parameter 82
args.eclipsecontent.toc Ant parameter 82
args.eclipsehelp.toc Ant parameter 82
args.fo.img.ext Ant parameter 82
args.fo.output.rel.links Ant parameter 82
args.fo.userconfig Ant parameter 82
args.fr Ant parameter 82
args.hdr Ant parameter 82
args.html.includefile Ant parameter 82
args.indexshow Ant parameter 82
args.input Ant parameter 82
args.logdir.toc Ant parameter 82
args.outext.map Ant parameter 82
args.xhtml.toc Ant parameter 82
args.xsl Ant parameter 82
Astoria 17, 210
audience 208
audience attribute 157
audience categories
   administrator or manager 12, 207, 208
   content specialist 12, 208, 213
   technology specialist 12, 208, 242
   vendor 244
audience types
   editor 220
   graphic designer 223
   information architect 224
   information developer 225
   print-document designer 232
   project manager 233
   staff manager 241
   website designer 244
   writer 245
author and copyright reporting tool (ditaauthors.php) 124
authoring framework 41
authoring in an IDE use case 43
authoring tool 61, 65, 69, 70, 220
   configuring to use the catalog 65
   configuring to use URLs 69
   configuring XMLBuddy for DITA 70
authoring tools (DITA)
Adobe FrameMaker 17, 209
Altova XMLSpy 17, 209
Arbortext Editor 17, 209
JustSystems XMetaL 17, 209
Pixware XMLmind 17, 209
Stylus Studio 17, 209
SyncRO Soft 17, 209
XMLBuddy 17, 209
Xopus 209

automatic file generation tools 126
automatic message file generation tool 124
avalon-framework-cvs-20020806.jar 60

B
backing up DITA source files 183
basedir Ant parameter 82
basic end-to-end system maturity level 41
batik.jar 60, 61
best practices for producing DITA documents 209
block element 209
body element 209
book directory 58
bookmap DTD 199
build (processing) file 210
build.xml 75, 180
building (processing)
  Ant parameters 82
  DITA files 232
  overview diagram 80
  the garage sample 75
  to DocBook targets 98
  to Eclipse content targets 87, 92, 98
  to Eclipse help targets 87, 92, 93
  to HTML Help targets 87, 89
  to JavaHelp targets 98, 100
  to PDF targets 98, 104
  to PDF2 targets 87, 90
  to troff targets 98, 104
  to Word RTF targets 98, 105
  to XHTML targets 87
  troubleshooting 113
  using the Java command line 102
business partner use cases 43

C
Carroll, John 132, 243
cascading stylesheet (CSS) 162, 210
  overriding the default with your own 162
catalog-dita-template.xml 75
catalog-dita.xml 75
catalog, configuring your authoring tool to use 65
choice table 211
CLASS environment variable 220
CLASSPATH environment variable 54, 57, 60, 61, 220
clean.temp Ant parameter 82
CMS (content management system) 184
CMSs (content management systems) 184
  Alfresco 17, 210
  Astoria 17, 210

CMSs (content management systems) (continued)
  CVS 17
ditausers.org 210
Documentum 17, 210
DocZone 17, 210
Ixiasoft 210
Perforce 17
Subversion 17
Vasant 17, 210
XDoks 17, 210
collection-type attribute 211
collection for this document 2
command element 211
colophon for this document 2
common directory 58
commonlr.css Ant parameter 162
catalog-dita_template.xml 75
catalog-dita.xml 75
catalog, configuring your authoring tool to use 65
content management use cases 43
cross-reference debugging tool (ditadebug.php) 124, 126
cross-reference element 214
ditausers.org 210
Documentum 17, 210
DocZone 17, 210
Ixiasoft 210
Perforce 17
Subversion 17
Vasant 17, 210
XDoks 17, 210
content management system (CMS) 184
content management systems (CMSs)
  Alfresco 17, 210
  Astoria 17, 210
  CVS 17
ditausers.org 210
Documentum 17, 210
DocZone 17, 210
Ixiasoft 210
Perforce 17
Subversion 17
Vasant 17, 210
content management use cases 43
content reference attribute 213
content reference mechanism 155
content reuse 155, 213
  concepts and techniques 155
  DITAblogs for 155
  more information about 156
context element 213
dictionary vocabulary 207, 214
dictation tool 155, 156
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
dictation tool use cases 43
control element 211
copyright symbol 197
core vocabulary 207
creating
cross-reference debugging tool (ditadebug.php) 124, 126
cross-reference element 214
cross-reference linking 147
CSS (cascading stylesheet) 162, 168, 210
  overriding the default with your own 162, 168
customer success stories
  access and distribution 43
  business partner 43
  content management system 43
  controlled vocabulary 43
  industry 43
  internal 43
  library system 43
  localization 43
  lone writer scenario 43
  migrating to a new tool 43
  modeling content 43
  ontology 43
  problem/solution scenarios 43
  prototyping 43
  publishing environment 43
  taxonomy 43
  template for 44
  translation 43
customizing access to your information 151
customizing XHTML output 164
customizing your published output 157, 168, 169
  DITAblogs for 168
  more information about 169
CVS 17

D
Darwin Information Typing Architecture (DITA) 11, 216
Darwin, significance in the DITA name 193
debugging tools 123, 124, 126
definition list 214
demo functionality 199
demo maturity level 41
deprecated functionality 199
directories
  book 58
  common 58
  ditaot 58
  doc 58
  docbook 58
  dtd 58
  elementref 58
  enote 58
  faq 58
  fo 58
  FrameMaker_adapter 58
  h2d 58
  java 58
  lib 58
  output file 71
  plugins 58
  preprocess 58
  resource 58
  samples 58
  schema 58
  source file 71
  troff 58
  xsl 58
directories (continued)
  xsifo 58
  xslhtml 58
  xslrtf 58
distributing content 187, 188, 215
  more information about 188
distributing published content 187, 188
  on a web server 188
DITA (Darwin Information Typing Architecture) 11, 194, 216
  comparison to DocBook 194
DITA authoring tools 51
DITA controlled vocabulary 207
DITA core vocabulary 207
DITA editor 51
DITA lifecycle 15
DITA Open Toolkit 11, 33, 35, 47, 49, 52, 53, 215, 217
  distributions 49, 52, 53, 215
  full package distribution 49, 52, 215
  getting information about 35
  getting started with 33
  installing 47
  small package distribution 49, 53, 215
  source distribution 49, 215
DITA Open Toolkit User Guide 12, 49, 215, 218
  source files 49, 215
dita root element 194
DITA Storm editor 210
dita wrapper 194
dita.dir Ant parameter 82
dita.extname Ant parameter 82
dita.input.valfile Ant parameter 82
dita.temp.dir Ant parameter 82
ditaauthors.php author and copyright reporting tool 124
DITAblogs
  accessing your information 153
  Ant scripts 107
  content reuse concepts and techniques 155
  customizing your published output 168
  installing and upgrading 61
  linking 150
  localizing (translating) 181
  managing content 185
  maps 144
  migrating legacy content to DITA 189
  plug-ins 64
  processing (building) 107
  setting up a working environment 73
  topics 139
  troubleshooting 126
ditadebug.php cross-reference debugging tool 124, 126
ditaedit.php string search/replace tool 124
ditaids.php ID checking reporting tool 124
ditakeys.php keyword checking reporting tool 124
ditalinks.php URL checking debug tool 124
ditamaps 141, 142, 145
  creating 142
  more information about 145
ditamsg_generator.xsl message topic generator 124, 126
ditaot directory 58
ditapreview.php incremental build tool 124
ditatargets.xml 75
ditausers.org 210
ditaval file 157, 212
doc directory 58
DocBook 11, 218
docbook directory 58
DocBook targets, processing (building) to 98
DocBook to DITA transform 63
DOCTYPE declaration 218
Document Type Definition (DTD) 21, 218
Documentum 17, 210
DocZone 17, 210
domain element 219
DTD (Document Type Definition) 21, 218
dtd directory 58
DTDs, preserving during upgrade process 50
Dublin Core Metadata Initiative (DCMI) 207
Dublin Core metatags 153, 234
E
Eclipse 70, 219
Eclipse content 11, 219
Eclipse content processor, downloading 59
Eclipse content targets, processing (building) to 87, 92
Eclipse help 11, 220
Eclipse help processor, downloading 59
Eclipse help targets, processing (building) to 87, 92, 93
Eclipse Infocenter 219, 220
Eclipse targets, processing (overview) 92
Eclipse, authoring in 43
elementref directory 58
directory 58
enterprise system maturity level 41
enote directory 58
environment variables
ANT_HOME 54, 56, 57, 220
ANT_OPTS 54, 56, 57, 220
CLASSPATH 54, 56, 69, 70, 220
JAVA_HOME 51, 54, 57, 220
JHHOME 54, 57, 60, 61, 220
PATH 54, 69
setting on Linux 57
setting on Mac OS 58
setting on Windows 54
error messages
produced by the Java infrastructure 123
produced by the Toolkit 115, 116
produced during processing 115
evaluating DITA and DITA Open Toolkit 41
example element 220
expanding access to your information 151
eXtensible Markup Language (XML) 19, 245
eXtensible Stylesheet Language Transformations (XSLT) 22, 248
F
family linking 220
faq directory 58
FAQs (frequently asked questions) 193
figure element 220
file generation tools 123, 124, 126
filtering 221
filtering content 157, 159, 212
flagging content 160, 212
to directory 58
FO plug-in 63
footer, including in XHTML output 165
FOP processor 58, 59, 60, 61, 221
downloading 59
installing on Linux 61
installing on Mac OS 58
installing on Windows 60
fop.jar 60, 61
for more information
accessing your information 154
customizing your published output 169
distributing content 188
linking 150
localizing (translating) 182
managing content 185
maps 145
plug-ins 64
processing (building) 112
reuse concepts and techniques 156
setting up a working environment 74
specialization (information design reuse) 173
topics 140
troubleshooting 129
format attribute 221
Framemaker 7.2 Application Packs for DITA 63
FrameMaker Adapter 63
FrameMaker DITA Application Pack 189
FrameMaker editor 17, 189, 209
migrating from unstructured to structured 189
FrameMaker_adapter directory 58
frameset, publishing XHTML output as 165
framework (authoring and production) 41
frequently asked questions (FAQs) 195

G
garage sample 75, 86, 191, 221
DITA source files 191
processing (building) 75
getting information about DITA Open Toolkit 35
getting started with DITA Open Toolkit 33
graphic designer 223
grocery shopping sample 131, 141, 143, 191, 222
DITA source files 191
grocery shopping sample (continued)
processing (building) 143
guidelines for producing DITA documents 209, 223

H
h2d directory 58
header, including in XHTML output 165
Hierarchical Task Analysis (HTA) diagram 243
hover help 223
HTA (Hierarchical Task Analysis) diagram 243
HTML Help 11, 223
HTML Help compiler 59, 60, 223
downloading 59
installing on Windows 60
HTML Help targets, processing (building) to 87, 89

I
IBM Eclipse content processor, downloading 59
IBM Eclipse help processor, downloading 59
IBM Task Modeler
  DITA map editor 243
  Eclipse 3.0 243
  XMLBuddy 243
ID attribute 223
ID checking reporting tool (ditaids.php) 124
Idiom FO plug-in 63, 64
installing 64
Index term list plug-in 63
indexing content 151, 224
industry use cases 43
Infocenter (Eclipse) 219, 220
information analysis 224
information architect 224
information architecture 196
information center (Eclipse Infocenter) 219
information design reuse (specialization) 171, 172, 240
information developer 225
information development framework 41
information element 225
information type 225
information types
  concept 212
  reference 234
  task 242
  topic 243
inheritance 226
installing
  considerations 50
  DITA Open Toolkit 47
  DITAblogs for 61
  Idiom FO plug-in 64
  on Linux 50, 56, 59
  on Mac OS 50, 58, 59
  on Windows 50, 54, 59
  optional tools 59
  overview 49
  plug-ins 64
  things to consider before 50
  verifying the installation on Linux 58
  verifying the installation on Windows 56
installing on Linux
  Ant 56
  FOP 61
  JavaHelp 61
  JDK 51
  Saxon XSLT processor 57
  Xalan XSLT processor 57
installing on Mac OS 58
installing on Windows
  Ant 54
  FOP 60
  HTML Help 60
  JavaHelp 60
  JDK 51
  Saxon XSLT processor 54
  Xalan XSLT processor 54
integrator.xml 75
internal use cases 43
Ixiasoft 210

J
jar files
  avalon-framework-cvs-20020806.jar 60
  batik.jar 60, 61
  fop.jar 60, 61
Java 23, 226
Java API reference specialization 63
Java command line
  building output using 102
Java Development Kit (JDK) 51, 58, 226
installing on Linux 51
installing on Mac OS 58
installing on Windows 51
java directory 58
Java SDK, installing 51
JAVA_HOME environment variable 51, 54, 57, 220
JavaHelp 11, 227
JavaHelp processor 59, 60, 61, 227
downloading 59
installing on Linux 61
installing on Windows 60
JavaHelp targets, processing (building) to 98, 100
JDK (Java Development Kit) 51, 58, 226
installing on Linux 51
installing on Mac OS 58
installing on Windows 51
JHHOME environment variable 54, 57, 60, 61, 220
JustSystems XMetaL 17, 209

K
key concepts
  lifecycle 15
  processing 18
  project components 24
  publishing 18
keyword checking reporting tool (ditakeys.php) 124
keyword element 227
languages, supported 47
legacy content, migrating to DITA 189
legacy lifecycle 15
lib directory 58
library system use cases 43
library systems
  CVS 183
  Perforce 183
  Subversion 183
lifecycle 15, 16
  DITA 15
  technology trends 16
  traditional (legacy) 15
linking 147, 148, 150, 220
  DITAblogs for 150
    family 220
    more information about 150
    using related links 148
    using relationship tables 148
    using xrefs 147
linking attribute 227
linking content 228
Linux, installing on 50, 56, 59
lists
  definition 214
  ordered 230
  simple 238
  unordered 244
localization use cases 43
localizing (translating) 175, 178, 180, 181, 182
  DITAblogs for 181
    in PDF2 output targets 180
    in XHTML output targets 178
    more information about 182
log 228
log file (Ant processing), capturing and using 113
lone writer scenario use cases 43

M
Mac OS, installing on 50, 58, 59
managing content 183, 185
  DITAblogs for 185
    more information about 185
map 228
maps 141, 142, 144, 145
  creating 142
  DITAblogs 144
    more information about 145
maturity levels
  basic end-to-end system 41
  demo 41
  enterprise system 41
  pilot project 41
  prototype 41
message topic generator (ditamsg_generator.xsl) 124, 126
messages
  produced by the Java infrastructure 123
  produced by the Toolkit 115, 116
  produced during processing 115
metadata 151, 229
metadata attributes 157, 159, 160, 161
  adding 159
  filtering on 159
  flagging on 160
  showing revisions 161
Microsoft HTML Help compiler 59, 223
downloading 59
migrating legacy content to DITA 189
  DITAblogs for 189
    more information about 189
migrating to a new tool use cases 43
Minimalism 132, 243
modeling content use cases 43
multilevel maps, processing using 144
Music specialization 63

N
navigation title 229
NetBeans, authoring in 43

O
OASIS (Organization for the Advancement of Structured Information Standards) 229
object model (XML) 19
online DITA DTDs and schemas 69
ontology use cases 43
Open Toolkit (DITA) 217
ordered list 230
Organization for the Advancement of Structured Information Standards (OASIS) 229
otherprops attribute 157
output.dir Ant parameter 82
outputclass common attribute 162
outputs supported by DITA Open Toolkit
  DocBook 11, 218
  Eclipse content 11, 219
  Eclipse help 11, 220
  HTML Help 223
  JavaHelp 11, 227
  PDF (Portable Document Format) 11, 230
  troff 11, 243
  Word RTF (Rich Text Format) 11, 244
  XHTML 11, 245
OWL (Web Ontology Language) 153, 230

P
PATH environment variable 54, 57
PDF (Portable Document Format) 11, 230
PDF targets, processing (building) to 98, 104
PDF2 cover page, adding a product logo to 167
PDF2 targets, processing (building) to 87, 90
Perforce 17
PHP tool set for debugging, reporting, and file generation 123, 124, 126
phrase element 230
pilot project maturity level 41
Pixware XMLmind 17, 209
SAXON XSLT processor (continued)
installing on Windows 54
schema 237
schema directory 58
schemas 21
scope attribute 237
search title element 238
service mark symbol 197
setting environment variables
on Linux 57
on Windows 54
setting up a working environment 65, 71, 73, 74
DITAblogs for 73
more information about 74
source and output file directories 71
short description 238
showing revisions 161
Simple Knowledge Organization System (SKOS) 153, 234
simple list element 238
simple table element 239
skills required for a DITA/Toolkit system 41
SKOS (Simple Knowledge Organization System) 153, 234
source control systems
CVS 183
Perforce 183
Subversion 183
SourceForge website 239
specialization (information design reuse) 171, 172, 173, 240
more information about 173
specializations
API reference 63
elementref 58
to 58
Java API reference 63
Music 63
RDF 63
SKOS (Simple Knowledge Organization System)
specialization 63
Taxonomy 234
Thesaurus 63, 234
staff manager 241
step element 240
structure element 241
structured writing 132, 225, 243
stylesheet 22, 241, 248
stylesheet processing 163
Stylus Studio 17, 209
Subversion 17
Sun Microsystems JavaHelp processor 59, 227
downloading 59
supported applications 47
supported languages 47
SyncRO Soft 17, 209
system requirements 47
system-level Ant scripts
build.xml 75
catalog-dita_template.xml 75
catalog-dita.xml 75
conductor.xml 75
ditatargets.xml 75
integrator.xml 75
system-level Ant scripts (continued)
pretargets.xml 75

T

table element 241
targets supported by DITA Open Toolkit
DocBook 11, 218
Eclipse content 11, 219
Eclipse help 11, 220
HTML Help 223
JavaHelp 11, 227
PDF (Portable Document Format) 11, 230
troff 11, 243
Word RTF 11
Word RTF (Rich Text Format) 244
XHTML 11, 245
task 225
task analysis 242
task information type 242
Task Modeler (IBM) 243
task topic 131, 136
tasks, creating 136
Taxonomy specialization 63
taxonomy use cases 43
technology specialist audience category 242
template for use cases 44
Thesaurus specialization 63, 234
tm element 197
TOC Javascript plug-in 63
tool set for debugging, reporting, and file generation 123
Toolkit (DITA) 217
tools and techniques for troubleshooting the build process 124, 126
topic 131, 132, 134, 139, 140, 225, 243
creating 134
more information about 140
processing (building) a single 139
production notes 139
topic information type 243
topic-based authoring 132, 243
trademark symbol 197
traditional lifecycle 15
translating (localizing) 175, 178, 180
in PDF2 output targets 180
in XHTML output targets 178
translation use cases 43
trantype Ant parameter 82
troff 11, 243
troff directory 58
troff targets, processing (building) to 98, 104
troubleshooting CLASSPATH and environment variables setup 123
troubleshooting the build process 113, 124, 126, 129
more information about 129
tools and techniques for 124, 126
typographic element 243

U

unordered list 244
upgrading
  overview 50
  things to consider before 50
URL checking debug tool (ditalinks.php) 124
usage scenarios
  access and distribution 43
  business partner 43
  content management system 43
  controlled vocabulary 43
  industry 43
  internal 43
  library system 43
  localization 43
  lone writer scenario 43
  migrating to a new tool 43
  modeling content 43
  ontology 43
  problem/solution scenarios 43
  prototyping 43
  publishing environment 43
  taxonomy 43
  template for 44
  translation 43
use cases
  access and distribution 43
  business partner 43
  content management system 43
  controlled vocabulary 43
  industry 43
  internal 43
  library system 43
  localization 43
  lone writer scenario 43
  migrating to a new tool 43
  modeling content 43
  ontology 43
  problem/solution scenarios 43
  prototyping 43
  publishing environment 43
  taxonomy 43
  template for 44
  translation 43
user-level Ant scripts 75

W
Web Ontology Language (OWL) 153, 230
website designer 244
Windows, installing on 50, 54, 59
Word RTF (Rich Text Format) 11, 244
Word RTF targets, processing (building) to 98, 105
writer 245

X
Xalan XSLT processor 54, 57, 58, 245
  installing on Linux 57
  installing on Mac OS 58
  installing on Windows 54
XEP processor 59, 63, 245
downloading 59
XHTML 11, 245
XHTML output
  customizing 164
  including a header and footer in 165
  publishing as a frameset 165
XHTML output, customizing 164
XHTML targets, processing (building) to 87
XMetaL editor 17, 209
XML (eXtensible Markup Language) 19, 193, 245
  relationship to DITA 193
XML declaration 247
xml lang attribute 175, 178, 181
XML object model 19
XMLBuddy 17, 70, 209
  configuring for DITA 70
XMLmind editor 17, 209
XMLSpy editor 17, 61, 209
Xopus editor 209
XPath 22, 247
xref linking 147
xsl directory 58
XSL stylesheet for automatic message topic generation 124, 126
xslfo directory 58
xslhtml directory 58
xslrtf directory 58
XSLT (eXtensible Stylesheet Language Transformations) 22, 248
XSLT processing 163
XSLT processors
  SAXON 236
  Xalan 245
XSLT stylesheets
  preserving customized during upgrade process 50

V
validators (java) 58
Vasont 17, 210
vendor audience category 244
vendors 208
verifying the installation
  on Linux 58
  on Mac OS 58
  on Windows 56