DITA Open Toolkit User Guide and Reference

October 2006, DITA Open Toolkit Release 1.3

DITA Open Toolkit is an open source, reference implementation of the OASIS DITA standard

Authoring and publishing information for this document.

Title	*DITA Open Toolkit User Guide and Reference*
Edition, release	Second edition, October 17, 2006. Based on release 1.3 of DITA Open Toolkit.
Publishing information	DITA Open Toolkit is an open-source, reference implementation of the OASIS DITA standard (currently DITA 1.1)
Authors	Anna van Raaphorst and Richard H. (Dick) Johnson, principals, VR Communications, Inc. (*http://www.vrcommunications.com*).
Description	This document is the definitive source of information about DITA Open Toolkit (OT). 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. In general, the *DITA Open Toolkit User Guide and Reference* is "vanilla DITA," using the bookmap DTD. By processing this document to a given target environment you can see output with few specializations or other special configurations applied. For the most part, we have found the default outputs adequate for our needs.
Acknowledgements	The authors would like to thank those members of the DITA community who have authored or contributed to one or more topics in this document: Kylene Bruski, Don Day, Anne Gentle, JoAnn Hackos, Jennifer Linton, Deborah Pickett, Michael Priestley, 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. Comments on the current document are always welcome. Other contributions might include new topics, additions or changes to the core vocabulary, use cases, and guidelines or best practices based on user experience with DITA and DITA Open Toolkit. Please contact the authors with your ideas (contact information is on (*http://www.vrcommunications.com*).

Contents

Release 1.3 information

This section contains information about release 1.3 of DITA Open Toolkit: system requirements, supported applications, new and enhanced features, upgrade impacts, and known problems.
This is a set of topics describing the new features in release 1.3.

Sections in this topic:

Release notes

OASIS DITA 1.1 support

Things to know about OASIS DITA 1.1 support in this release:

1. DITA-OT 1.3 provides preliminary processing support for the upcoming OASIS DITA 1.1 specification (see *http://wiki.oasis-open.org/dita/Roadmap_for_DITA_development*). Because the proposed OASIS DITA 1.1 DTDs and schemas are fully backwards compatible with the latest DITA 1.0.1 DTDs and schemas, the 1.3 Toolkit provides the proposed 1.1 materials as the default DTDs for processing. The XML Catalog resolution maps any references for DITA 1.0 doctypes to the 1.1 DTDs, for example. All processing ordinarily dependent on the 1.0 definition continues to work as usual, and any documents that make use of the newer 1.1-based elements or attributes will be supported with specific new processing function (such as base support for the new <data> element). *Documents created with the proposed OASIS DITA 1.1 DTDs are the only ones ever likely to have features that invoke the specific new 1.1-based processing support*.

   Important: Because this support is based on a yet-to-be-approved version of the proposed OASIS DITA 1.1 specification, if you choose to investigate any 1.1-based function, be aware that the 1.1 implementation in this version of the Toolkit is preliminary. Upon final approval of the DITA 1.1 standard, Toolkit developers will, of course, review the current Toolkit implementation to make certain that it conforms to the DITA 1.1 specification.

2. Related to the DITA 1.1 preliminary implementation, the bookmap updates for DITA 1.1 will be provided as override capabilities for the FO plug-in (Idiom's donation). Note that:
   - The FO demo transform code at the 1.2.2 level is still included in the DITA 1.3 package, but is now deprecated.
   - To get the FO updates for 1.3, download the FO plug-in at its next update.
   - The updated FO plug-in will be usable with FOP as well as with XEP.

Changes

This verson of the Toolkit provides some strategic updates that correct some long-overdue faults in the original implementation. Necessarily, there are some changes to note:

1. **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 has been renamed as `build_demo.xml`. The 1.3 `build.xml` file is now the top-level script for starting a transformation. If you

have created Ant tasks to work around the former `build.xml` architecture, you might need to revise them to take advantage of the 1.3 functionality.

2. **Change to command-line invocations:** The "Ant refactoring" exercise for this release has changed some previously documented Ant calls for running demos. This change enables better use of the Ant modules for power users who need to integrate the Toolkit into programming build environments such as Eclipse, but the change affects some documentation. This is a permanent change that should remain stable from now on. Wherever you see an older instruction like "C:\dita-ot>ant all", you now need to indicate the component that contains the demos, so you would type "C:\dita-ot>ant all -f build_demo.xml".
3. **Separation of demo targets from formal component targets:** Another effect of the Ant refactoring is that the internal programming targets will now be 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".
4. **Classpath update to enable catalog resolver:** Release 1.3 includes 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 evironment settings or in a batch file that sets up the shell environment.
5. **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 are now included in root directory of the Toolkit. Use the one that applies to your environment.
6. **Two installation options:** In Release 1.3, the Toolkit has two installation distributions. The smaller one is for updating existing installations or for reuse in embedded applications that already provide the other processing components. A new package with "fullpackage" in the name incorporates the essential processing modules to create a processing environment for new users and evaluators that requires nothing more than to unzip the file into an appropriate directory and then click on a "start" batch file. A new document in its root directory (an output of `>doc/EvaluateOT.dita`, “Evaluating the DITA Open Toolkit (fullpackage version)”) informs new users how to install and use the Toolkit.
7. **Other enhancements:** The public design discussions that fed into the final selection and architecture for this release are documented at the DITA Focus Area in a topic called “DITA OT 1.3 Issues tracking” (*http://dita.xml.org/node/1282*).

[7 Improvements]

1. Preliminary support for OASIS DITA 1.1
2. Support ICU in index sorting
3. Integrate with Eclipse
4. Refactor Ant script for easy override
5. Topicmerge reimplementation in JAVA
6. Enable XML Catalog Resolver
7. Full package distribution (was GUI/usability)

[21 SF Bugs Fixed]

1. SF Bug 1582506 DocBook cannot handle <author>
2. SF Bug 1548189 Sections should not jump to <h4> for Accessibility reasons
3. SF Bug 1548180 Spaces dropped from index terms
4. SF Bug 1548154 XHTML index links should go to the topic

5. SF Bug 1545038 CommandLineInvoker is unfriendly towards spaces
6. SF Bug 1541055 topicref @id incorrectly uses NMTOKEN type
7. SF Bug 1530443 dost.jar relies on the incorrect behavior of Xerces
8. SF Bug 1473029 Syntax code makes overrides difficult
9. SF Bug 1470101 Metadata in topics is left out of XHTML headers
10. SF Bug 1470077 Choicetable headers create attribute inside attribute
11. SF Bug 1470057 Step template creates attributes after creating tags
12. SF Bug 1465947 <topichead> without children the whole branch to disappear
13. SF Bug 1465941 Keywords defined in map are ignored if <topicref> contains t
14. SF Bug 1465866 Problems in catalog-dita.txt
15. SF Bug 1460447 <morerows> not well supported in pdf tranformation.
16. SF Bug 1457187 'copy-to' doesn't actually copy files
17. SF Bug 1454835 OT renders files referenced via conref only
18. SF Bug 1427808 Should be easier to modify link attributes in XHTML
19. SF Bug 1422182 @colname renaming needs to apply to @namest and @nameend
20. SF Bug 1417820 fo and docbook outputs can\'t handle deep topic dirs
21. SF Bug 1368997 PDF Vertical list of author redundancy

[1 SF Patch Added]

1. SF Patch 1503296 Refactor of HTMLHelp inifiles creation

[1 SF RFE Added]

1. SF RFE 1160960 Enh: Toolkit should work with both both *.dita and *.xml

Note: Bugs, patches, and RFEs listed above are documented on the SourceForge tracker pages:

- Bugs tracker:

  ```
  http://sourceforge.net/tracker/?group_id=132728&atid=725074
  ```

- Patches tracker:

  ```
  http://sourceforge.net/tracker/?group_id=132728&atid=725076
  ```

- RFE tracker:

  ```
  http://sourceforge.net/tracker/?group_id=132728&atid=725077
  ```

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 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 *Release notes* on page 11

The following tools are required to use DITA Open Toolkit at the 1.3 release level.

Java Development Kits (SDKs)	Sun 1.4.2. You can download the Sun JDK from *http://java.sun.com/j2se/1.4.2/download.html*.
	IBM 1.4.2. 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/*.
	Xalan-J 2.6. You can download Xalan from *http://archive.apache.org/dist/xml/xalan-j/*.

For more information about the required tools, see *Installing the required tools for the small package* on page 49.

For information about the optional tools, see *Installing the optional tools* on page 52.

Supported languages

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

Language	xml:lang value
Arabic	ar-eg
Belarusian	bg-bg
Bulgarian	be-by
Catalan	ca-es
Chinese (Simplified)	zh-cn
Chinese (Traditional)	zh-tw
Croatian	hr-hr
Czech	cs-cz
Danish	da-dk
Dutch	nl-nl
Dutch (Belgian)	nl-be
English (Canadian)	en-ca
English (UK)	en-gb
English (US)	en-us
Estonian	et-ee
Finnish	fi-fi
French	fr-fr
French (Belgian)	fr-be

Language	**xml:lang value**
French (Canadian)	fr-ca
French (Swiss)	fr-ch
German	de-de
German (Swiss)	de-ch
Greek	el-gr
Hebrew	he-il
Hungarian	hu-hu
Icelandic	is-is
Italian	it-it
Italian (Swiss)	it-ch
Japanese	ja-jp
Korean	ko-lr
Latvian	lv-lv
Lithuanian	lt-lt
Macedonian	mk-mk
Norwegian	no-no
Polish	pl-pl
Portuguese	pt-pt
Portuguese (Brazilian)	pt-br
Romanian	ro-ro
Russian	ru-ru
Serbian	sr-sp
Slovak	sk-sk
Slovenian	sl-si
Spanish	es-es
Swedish	sv-se
Thai	th-th
Turkish	tr-tr
Ukranian	uk-ua

What's new

This is a set of topics describing the new features in release 1.3.

OASIS DITA 1.1 support

Overview of OASIS DITA 1.1 support

The support for OASIS DITA 1.1 adds several new DTDs and schemas with many new elements and attributes. These include:

bookmap	Support for the DITA 1.1 bookmap specification.
common elements	Adds new elements for data, image scaling, and unknown.
glossary	Adds new elements for glossary entries.
indexing	Adds new elements index-see, index-see-also, and index-sort-as.

The following sections contain more detailed information for some of this new support.

New element <abstract>

The new <abstract> element can include complex markups besides the <shortdesc> element. You can put the <shortdesc> element inside the <abstract> element, together with many other elements. The following examples illustrate how you can use the <abstract> element.

If you use several <shortdesc> elements inside the <abstract> element, they will be concatenated when pulled for hover help. After you format the source files, the content inside the <abstract> element will be transformed into normal text.

Note: Because OASIS DITA 1.1 was not yet approved at the time DITA Open Toolkit 1.3 was released, the functionality described here should be considered preliminary until DITA 1.1 is approved, the 1.3 Toolkit assessed relative to DITA 1.1, and appropriate changes made to the Toolkit, if necessary.

Examples

Example 1

In DITA 1.0, you can only use the <shortdesc> element that cannot contain the <p> element.

```
 <shortdesc>This is a short description in DITA 1.0. It <b>cannot</b>
contain paragraphs.</shortdesc>
```

Example 2

This example illustrates how with 1.3 you can use different elements besides <shortdesc> inside the <abstract> element, and apply different styles to the text inside the <abstract> element.

```
    <abstract>
        <shortdesc>This is the short description</shortdesc>
        <ol>
            <li>This is a <i>list</i>.</li>
        </ol>
        <p>This is a <b>paragraph</b>.</p>
        <codeblock>Here are some codes.</codeblock>
        <filepath>This is the file path.</filepath>
    </abstract>
```

Example 3

This example illustrates how you can use both the <shortdesc> element and plain text inside the <abstract> element.

```
 <abstract><shortdesc>This topic is about short description.</shortdesc>.

Short description is very important, so read more.</abstract>
```

New element <data>

DITA 1.1 includes a new element, <data>. This element and the content inside it is ignored in the transformation process of DITA files.

Note: Because OASIS DITA 1.1 was not yet approved at the time DITA Open Toolkit 1.3 was released, the functionality described here should be considered preliminary until DITA 1.1 is approved, the 1.3 Toolkit assessed relative to DITA 1.1, and appropriate changes made to the Toolkit, if necessary.

As a DITA content author you can add the <data> element, and put content inside it. When you transform the DITA files to the output that you want, the transformation ignores the <data> element and any content inside it.

If you specialize the <data> element, and put information inside the specialized element, you can create a transform override to use the information.

New indexing elements (see, see also, sort as)

DITA 1.1 supports the following new indexing elements:

- <index-see>
- <index-see-also>
- <index-sort-as>

See and See also indexing elements

In DITA 1.0, you cannot specify the <see> and <see also> index entries by using the current <indexterm> element. The DITA 1.1 standard introduces the following new child elements for <indexterm> that support this functionality:

- index-see
- index-see-also

For example, you can add the kind of index entry shown below:

```
<indexterm>computer
   <index-see>monitor</index-see>
   <index-see-also>Illustration</index-see-also>
</indexterm>
```

Then, if you generate HTML Help, JavaHelp, and PDF output with the indexing function enabled, the index entries will look like the following:

```

computer 43
        See monitor
        See also Illustration

```

In HTMLHelp and JavaHelp, the output will contain hyperlinks to the "see" and "see also" entries. In PDF output, the output is not hyperlinked.

See and See also indexing is ignored in XHTML. In PDF, you must enable indexing using the FO plug-in provided by Idiom.

Sort order indexing elements

With the DITA 1.1 standard, you can specify a sort phrase and sort index entries under the sort phrase. This feature allows you to sort an index entry in a way different from the default. One example would be to disregard insignificant leading text, such as punctuation or words like "the" or "a".

Another example would be to sort <data> under the letter D rather than the character "<", or to include such an entry under both the punctuation heading and also the letter D. In this case there would be index entry directives differentiated only by the sort order.

If you put the following content in the source file:

```
<indexterm>data<index-sort-as>key</index-sort-as></indexterm>
<indexterm>indextest<index-sort-as>abc</index-sort-as></indexterm>
```

the output should be:

```
indextest data
```

Another example might be a translation project in which a document needs to be translated into Japanese. Many of the index entries contain kanji, which need to be sorted in phonetic order. The translators, who can understand the language and see the entry in its context, can insert the `<index-sort-as>` elements into the `<indexterm>` elements as part of their localization work.

Supporting ICU in index sorting

With DITA Open Toolkit 1.3, you can get correctly sorted index output for different languages.

During normal transformation, the Toolkit looks for ICU (International Components Unicode) classes inside the `classpath` element. If the ICU is enabled, the Toolkit uses ICU's Collator class to do the comparing and sorting. If the ICU is not enabled, the Toolkit will use the JDK's Collator class to do the comparing and sorting.

Note: The ICU is included in the Toolkit 1.3 full installation package.

Extensible metadata attributes

As an aid to information architects, extensible metadata attributes are part of the proposed DITA 1.1 standard, and are supported by DITA Open Toolkit 1.3.

Note: Because OASIS DITA 1.1 was not yet approved at the time DITA Open Toolkit 1.3 was released, the functionality described here should be considered preliminary until DITA 1.1 is approved, the 1.3 Toolkit assessed relative to DITA 1.1, and appropriate changes made to the Toolkit, if necessary.

Example

- As a DITA information architect, you could:
  1. Define new attributes that the team needs, for example "proglanguage".
  2. Express each new attribute as a separate domain package, for example proglanguage.mod, with the new attribute specialized from the "props" attribute.
  3. Integrate the domain packages into the authoring DTDs or schemas:
     a. Redefine the "props" attribute entity to include the "proglanguage" attribute. Similarly, you can redefine element entities to integrate new domain elements.
     b. Add the new attribute domain to the list of domains in the domains attribute, preceded by an "a", for example, `domains="a(props proglanguage)"`.

- As a writer, you could:

  1. Add values to the new attributes of an element.
  2. Define values in the DITA filter file.
  3. Transform the DITA source files to remove or flag the content based on the new attributes, for example flagging all `proglanguage="Java"`

  After you do that, another writer could reuse the content.

  A specialization-unaware trademarking tool requires generalization of the contributed content. If the user runs all the content through the tool, the content is processed and filtered against the new attributes after the generalization. The new attributes are now collapsed into the "props" attribute.

  1. The generalization turns `proglanguage="Java"` into `props="proglanguage(Java)"`.
  2. The conditional processing transform recognizes the new form as equivalent to the old, and the instruction "`flag all proglanguage=java`" operates on either `props="proglanguage(Java)"` or `proglanguage="Java"`.

Graphic scaling improvement

DITA Open Toolkit 1.3 provides improved support for graphic scaling. This feature applies to HTML Help, XHTML, PDF, and FO. The feature is not supported in RTF output.

Note: Because OASIS DITA 1.1 was not yet approved at the time DITA Open Toolkit 1.3 was released, the functionality described here should be considered preliminary until DITA 1.1 is approved, the 1.3 Toolkit assessed relative to DITA 1.1, and appropriate changes made to the Toolkit, if necessary.

The DITA 1.1 standard specifies attributes that you can use to set the actual display size of the pictures in the `<image>` tag, such as "width" and "height" (for example, `<image height="80" width="60" href="a.jpg"/>`).

You can optionally specify the metric of the length in the height and width attributes fields, for example, `<image height="80pc" width="60pc" href="a.jpg"/>`. The metrics currently supported are: px, pc, pt, in, cm, mm, em. The default is px.

You can also use the scaling function in setting the actual display size of the output image, for example `<image height="80pc" width="60pc" href="a.jpg" scale="0.8"/>`. Scale="0.8" means the picture in the output will be displayed at 80% of the size that you specified by height and width. In this example the picture will be displayed at 64 pt in height and 48 pt in width.

Support for foreign content vocabulary (<unknown> element)

DITA 1.1 introduces the <unknown> element to incorporate existing standard vocabularies for special content, like MathML and SVG, as inline objects.

Note: Because OASIS DITA 1.1 was not yet approved at the time DITA Open Toolkit 1.3 was released, the functionality described here should be considered preliminary until DITA 1.1 is approved, the 1.3 Toolkit assessed relative to DITA 1.1, and appropriate changes made to the Toolkit, if necessary.

As a DITA content author you can add the <unknown> element, and put content inside it. When you transform the DITA files to the output you want, the transformation ignores the <data> element and any content inside it.

If you specialize the <data> element, and put information inside the specialized element, you can create a transform override that allows the information to appear correctly in the output.

Eclipse integration of DITA documentation plug-ins

You can use a template to develop documentation plug-ins with DITA in Eclipse PDE, and use DITA Open Toolkit 1.3 to build and pack the final plug-in. If you want to develop a documentation plug-in with DITA in

Eclipse, you cannot use the previous releases of the Toolkit in Eclipse to transform DITA to HTML. Although previous releases of the Toolkit support the feature to transform DITA files to an Eclipse documentation plug-in, they are not integrated with Eclipse. See *Using a DITA template for Eclipse document plug-ins* on page 86 for more information.

Topicmerge improvements

The topicmerge feature improves the Toolkit build speed and reduces the possibility of encountering an "out of memory" exception in the build process.

With the enhanced topicmerge feature, this situation will be less likely. An intermediate merged file keeps the structure information in the ditamap, and the structured Toc will be reflected in the output.

Writing a simple script will help you learn about the effects of this feature. DITA Open Toolkit 1.3, provides the TopicMerge module to help you implement this feature. You can use this module to generate the merged files. Here is an example.

sample.xml:

```
<project name="sample">
 <property name="dita.dir" value="${basedir}"/>
 <import file="${dita.dir}${file.separator}build.xml"/>

 <target name="premerge">
    <antcall target="preprocess">
    <param name="args.input" value="${input}"/>
  <param name="output.dir" value="${dita.dir}${file.separator}output"/>
  </antcall>
 </target>
 <target name="merge"  description="Merge topics" depends="premerge">
  <basename property="temp.base" file="${input}" suffix=".ditamap"/>
  <property name="temp.input"
value="${basedir}${file.separator}${dita.temp.dir}${file.separator}${temp.base}"/>
  <dirname property="temp.dir" file="${temp.input}"/>
  <pipeline message="topicmerge" module="TopicMerge"
   inputmap="${temp.dir}${file.separator}${temp.base}.ditamap"

extparam="output=${dita.dir}${file.separator}output${file.separator}${temp.base}_merged.xml;

      style=${dita.dir}${file.separator}xsl${file.separator}pretty.xsl" />
 </target>
</project>
```

To run the script: `ant -f sample.xml merge -Dinput="C:\DITA-OT1.3\test.ditamap"`.

Note: The path for -Dinput must be an absolute path

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 is 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.

If there is no language set in the map, DITA Open Toolkit uses the first language found in a given topic. If more than one language is found , the user is warned that the sort will be based on the first one found, along with the name of that language.

If the ICU (International Components for Unicode) is available (that is, the icu4.jar file is in your classpath), it is used for sorting. Otherwise, the default Java sort is used.

XML catalog improvements

In previous releases of DITA Open Toolkit, a simple XML catalog resolver was enabled. You 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 is enabled so that references to DTDs in DITA files do not need to be updated each time you change the file paths on your workstation or use another workstation.

With this enhanced feature, when a developer makes a new specialization, the developer only needs to update the mapping between the new DTD file's system ID (location relative to the catalog file) and public ID (the ID assigned by the developer in the head of the DITA or xml file which identifies the corresponding dtd file) in the catalog file (catalog-dita_template.xml), for example `<public publicId="-//IBM//DTD DITA ABC//EN" uri="dtd/abc.dtd"></public>`.

This enhanced feature does not change the normal behavior of the Toolkit.

Support for multiple file extensions in one DITA map

DITA Open Toolkit supports 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. This made file reuse difficult, because you had to change the file extensions manually.

In the 1.3 release, you can include both .xml and .dita as the file extensions in a single DITA map.

If you include both .xml and .dita files in one DITA map, and specify `/ditaext:.dita` in the Java command, the `.xml` files are transformed to `.dita` and put in the `temp` directory together with the `.dita` files. If you specify `/ditaext:.xml` in the Java command, all the `.dita` files are transformed to `.xml` in the `temp` directory.

The default process option is to change all files into .xml files.

Note:

- Do not include files with the same root name but different extensions in the same directory, because this might cause unexpected problems.
- Error messages and warning messages in the console might not reflect the real extension. For example, if there is incorrect usage in file `a.dita`, the warning message in the console might refer to file `a.xml`, because file `a.dita` was changed into file `a.xml` in the `temp` directory.
- If you use other file extensions together with `.dita` and `.xml` in one ditamap, for example, `.dit`, you run the risk of transformation failure.

Ant refactoring

In DITA Open Toolkit 1.3, the system-level Ant scripts have been refactored so that developers can easily find the targets they need and create their own extensions.

Developers can still use the Java command line as before. Ant build script `conductor.xml` is marked as deprecated, but it can still be used. Script `build.xml` is the renamed version of `conductor.xml`, and developers can also continue to use it, as well.

New users should use build scripts like `build_dita2*.xml`.

The original demo ant script `build.xml` has been renamed to `build_demo.xml`. The command `ant -f build_demo.xml` should now be used to verify the functions of the Toolkit.

Command-line help

In DITA OT 1.3, the command line help function has been enhanced to improve usability. You can display the version of the Toolkit and the usage of the command line by using the following commands:

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

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

You can display a brief description of the supported parameters in the command line window when you type a specific command. For example, if you type `java -jar lib/dost.jar -h`, you will get the following result:

```
D:\DITA-OT1.3_fullpackage_bin\DITA-OT1.3>java -jar lib/dost.jar -h

java -jar lib/dost.jar [mandatory parameters] [options]
Mandatory parameters:
  /i:{args.input}        specify the input file
  /transtype:{transtype} specify the transformation type
Options:
  -help, -h              print this message
  -version               print the version information and exit
  /basedir:{basedir}     specify the working directory
  /ditadir:{dita.dir}    specify the toolkit's home directory
  /outdir:{output.dir}   specify the output directory
  /tempdir:{dita.temp.dir} specify the temporary directory
  /logdir:{args.logdir}  specify the log directory
  /ditaext:{dita.extname} specify the dita file extension
  ...
```

Release history

This section contains feature and bug information about releases 1.0-1.2+ of DITA Open Toolkit.

Sections in this topic:

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 Microsoft® 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 appeared in HTML Help and JavaHelp.
9. **Mapref 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 supported 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 supported conditional processing and conref resolution.

4. **Map-driven processing**

   The map-driven processor generated 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.

Introduction

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

Sections in this topic:

About Darwin Information Typing Architecture (DITA)

DITA 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.

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

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 (TC).

About DITA Open Toolkit (DITA OT)

DITA Open Toolkit is an implementation of the OASIS DITA Technical Committee's specification for DITA DTDs and schemas. The Toolkit, which can be used in the Windows, UNIX/Linux, and Mac OS operating environments, transforms DITA content (maps and topics) into deliverable formats.

DITA Open Toolkit supports the following publishing environments:

- DocBook
- Eclipse content
- Eclipse help
- HTML Help
- JavaHelp
- PDF
- troff
- Word RTF
- XHTML

About this document

DITA Open Toolkit User Guide and Reference (this document) is the definitive source of information about DITA Open Toolkit (OT), 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:

Document contents on page 28
Target audience on page 29
Prerequisites on page 29
How and why this document was produced on page 29

Document contents

Chapter (major section)	**Contents**
Release 1.3 information on page 11	Information about the current release of DITA Open Toolkit: system requirements, supported applications, and known problems.
Release history on page 23	Summary information about prior releases of DITA Open Toolkit.
Evaluating DITA and DITA Open Toolkit on page 39	How to determine what it would take to create your DITA and DITA Open Toolkit authoring and production system.
Installing and upgrading DITA Open Toolkit on page 45	How to install and upgrade DITA Open Toolkit on Windows, Linux, and Mac OS.
Setting up your working environment on page 57	How to configure your DITA editor and set up your source file directory.
Processing (building) and publishing DITA documents on page 65	How to process (build) and publish DITA documents.
Troubleshooting the build process on page 105	How to troubleshoot the build process.
Creating DITA topics on page 123	How to create DITA topics (base topics, concepts, tasks, and reference topics).
Creating DITA maps on page 131	How to create DITA maps to define content structure.
Linking content on page 135	How to link DITA topics using cross-references (xrefs), related links, and relationship tables.
Managing your content on page 141	How to manage your content using backup, source control and content management.
Reuse concepts and techniques on page 145	How to reuse content (using conrefs), information design (using specializations), and processing code.
Expanding and customizing access to your information on page 151	How to expand access to your information through indexing, the use of metadata, and filtering (conditional processing).
Customizing your published output on page 159	How to customize your published output.
Localizing (translating) your DITA content on page 163	Information about translating the content in your DITA projects.
Distributing your published content on page 167	How to distribute your published content.
Migrating legacy content to DITA on page 169	How to migrate legacy content to DITA.

Chapter (major section)	Contents
Sample files on page 171	Information about the DITA sample source files that come with DITA Open Toolkit.
Frequently asked questions (FAQs) on page 173	Frequently asked questions about DITA and DITA Open Toolkit (OT).
DITA core vocabulary on page 181	Information about the DITA core vocabulary, a list of the terms in the vocabulary, and links to related terms and other information.

Target audience

An audience is a target group of users.

This document was written for both beginning and advanced users currently implimenting or planning to impliment 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, 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 13.

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. 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 "production notes" sections that are part of each chapter. For example, for our suggestions about troubleshooting the build process, see *Production notes (troubleshooting)* on page 118.

Getting started

This section contains introductory information about DITA, DITA Open Tooklit (OT), this document, and where to get more information about DITA and the Toolkit.

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 39 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 13.
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* on page 45.
4. Install DITA Open Toolkit and its prerequisite tools.

   For an overview of the installation process, see *Installation* on page 45. 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 57.
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 65.
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 123 and *Creating DITA maps* on page 131.
8. When you feel comfortable with the basics, expand your DITA skill by reading introductory DITA texts, working through DITA tutorials, and taking courses on using DITA.

   For additional sources of information about DITA and DITA Open Toolkit, see *Getting information (general)* on page 33.
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 efficient and effective expansion plan.
10. As you gain skill and tackle more complex DITA projects, use this document and others you collect along the way for guidance and reference.

Getting information (general)

This section contains sources of information about DITA and DITA Open Toolkit.

Sections in this topic:

General Information and Frequently Asked Questions (FAQs) about DITA and DITA Open Toolkit

Description	Information source
dita.xml.org: Official DITA community gathering place and information resource for the DITA OASIS Standard 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.	*http://dita.xml.org*
Cover Pages: Articles, presentations, and facts about DITA Hosted by OASIS.	*http://xml.coverpages.org/dita.html*
IBM site containing a collection of information about DITA	*http://www-128.ibm.com/developer-works/xml/library/x-dita1/*

Tools, downloads

Description	Information source
Download site for the Toolkit and related/prerequisite software	*https://sourceforge.net/projects/dita-ot*
alphaWorks IBM TaskModeler: An Eclipse-based tool for rapidly creating and analyzing models of human activity for DITA and user experience design	*http://www.alphaworks.ibm.com/tech/taskmodeler*
IBM download site: Collection of DITA downloads	*http://www-128.ibm.com/developerworks/xml/library/x-dita6/x-dita_downloads.html*

Training, education

Description	Information source
Introduction to DITA: *A User Guide to the Darwin Information Typing Architecture* Book (level: beginner). Authors: Jennifer Linton and Kylene Bruski. Publisher: Comtech Services, Colorado, 2006. Task-oriented approach to learning DITA. Authored using DITA markup and methodology in a variety of XML editors. Published using the Blast Radius and Mekon FrameMaker plugin in the DITA 1.2 Open Toolkit.	*http://www.comtech-serv.com*
Introduction to the Darwin Information Typing Architecture 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.	*http://www-128.ibm.com/developerworks/xml/library/x-dita1/*
Design patterns for information architecture with DITA map domains Article (level: intermediate). Authors: Eric Hennum, Don Day, John Hunt, and Dave Schell. Publisher: IBM, updated September 2005. Explains the design technique for creating a DITA map domain.	*http://www-128.ibm.com/developerworks/xml/library/x-dita7/*
An XML-based information architecture for learning content Part 1: A DITA specialization design. Article (level: intermediate). Authors: Robert Bernard and John P. Hunt. Publisher: IBM, August 2005. How topic-based DITA XML can provide the basis for developing an information architecture for single-sourced XML learning content.	*http://www-128.ibm.com/developerworks/xml/library/x-dita9b/*
An XML-based information architecture for learning content Part 2: A DITA content pilot. Article (level: intermediate). Authors: John P. Hunt and Robert Bernard. Publisher: IBM, August 2005. How topic-based DITA XML can provide the basis for developing an information architecture for single-sourced XML learning content.	*http://www-128.ibm.com/developerworks/xml/library/x-dita9a/*
Developing Quality Technical Information: A Handbook for Writors and Editors	*http://www-128.ibm.com/developerworks/xml/library/x-dita9a/*

Description	Information source
Authors: Gretchen Hargis et al. Publisher: IBM, 2nd edition, 2004. How to apply quality characteristics that make technical information easy to use, easy to understand, and easy to find.	

Consulting

Description	Information source
List of DITA consultants	*http://dita.xml.org/taxonomy/term/48*

DITA and DITA Open Toolkit standards and product releases

Description	Information source
Roadmap for DITA development Lists of proposed and committed features for future releases.	*http://wiki.oasis-open.org/dita/Roadmap_for_DITA_development*
Current DITA Open Toolkit release Bugs, support requests, patches, feature requests.	*http://dita-ot.sourceforge.net/*
OASIS DITA Open Toolkit project home	*http://dita-ot.sourceforge.net/*
OASIS DITA Open Toolkit 1.3 features and issues tracking	*http://dita.xml.org/node/1282*
DITA Open Toolkit developers mailing list	*https://lists.sourceforge.net/lists/listinfo/dita-ot-developer*
DITA 1.1 proposed features	*http://wiki.oasis-open.org/dita/DITA_1.1_Features*
DITA 1.1 work items	*http://wiki.oasis-open.org/dita/List_of_DITA_1.1_Work_Items*
OASIS DITA standards committee website	*http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita*
OASIS DITA Language Specification version 1.0 OASIS Standard, 09 May 2005.	*http://docs.oasis-open.org/dita/v1.0/dita-v1.0-spec-os-LanguageSpecification.pdf*
OASIS DITA Architectural Specification version 1.0	*http://docs.oasis-open.org/dita/v1.0/dita-v1.0-spec-os-ArchitecturalSpecification.pdf*

Description	**Information source**
OASIS Standard, 09 May 2005.	

Usage Information

Description	**Information source**
Apache Derby documentation 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.	For an overview of the documentation, see *http://db.apache.org/derby/manuals/index.html*. For the DITA source files, see *http://db.apache.org/derby/manuals/dita.html*.

User groups, forums

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

Conferences

Note: The following DITA-related conferences were or will be held in 2006. Some tentative dates are also given for 2007. For the latest information about these conferences, see the referenced websites.

Description	**Information source**
XML 2006 Date: December 5-7. Location: Boston. Audience: Those who want to use XML and related technologies.	*http://2005.xmlconference.org/*
DITA Europe 2006 Date: November 2-3. Location: Frankfurt. Audience: Those who are interested in implementing XML DITA or learning more about this new standard for topic-based authoring. Conference organizer: Center for Information Development Management.	*http://www.infomanagementcenter.com/DITAeurope/*
Tri-XML 2006 Date: July 27-29. Location: Raleigh, NC. Focus: Presentations on real-world use of XML. Conference organizer: Tri-XML, an interest group for the Research	*http://www.trixml.org//confindex.shtml*

Description	Information source
Triangle area, NC USA that focuses on educating members in XML-related technologies and potential uses.	
X-Pubs 2006 Date: June 20-21. Location:London. Focus: Seeks to critique the established business justification for the adoption of both XML and DITA in organizational publishing.	*http://www.x-pubs.com/index.php*
XTech 2006 Date: May 16-19. Location: Amsterdam. XTech 2007: Paris. Audience: Developers, information designers and managers working with web and standards-based technologies.	*http://xtech06.usefulinc.com/content/about*
Content Management Strategies 2006 Date: April 3-5. Location: San Francisco, CA. CMS 07: March 26-28, Boston, MA. Conference organizer: Center for Information Development Management (CIDM).	*http://www.infomanagementcenter.com/index.htm*
DITA 2006 Date: March 23-25. Location: Raleigh, NC. DITA 2007 (West Coast), February, California. DITA 2007 (East Coast), fall, Florida (tentative). Conference organizer: Bright Path Solutions.	*http://www.travelthepath.com/dita2006.html*

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.

In a general sense, "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

Sections in this topic:

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, or Mac OS)
- The prerequisite processing components: Java Development Kit (JDK), Ant build tool, and an XSLT engine 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 DITA Open Toolkit.

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 tool 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

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

Type	Description
Translation/localization	Producing documentation that needs to be translated into a number of 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.
Working with internal or business partners	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.

Type	**Description**
Using a library system or content management system	Tips on architecting, organizing, creating, and processing DITA information when using a library system or CMS.
Using a controlled vocabulary, taxonomy, or ontology	How use of a controlled vocabulary (ontology, taxonomy) relates to architecting, organizing, creating, and processing DITA content.
Modeling content	Using content modeling tools.
Migrating legacy content	Issues, tools, and techniques related to migration from books to topic-based information, or from unstructured to structured. A related topic could be migrating to new tools (e.g., from unstructured to structured FrameMaker, or from Microsoft Word to Arbortext Editor).
Migrating to a new tool	For example, from unstructured to structured FrameMaker, or from Microsoft Word to Arbortext Editor.
Prototyping	How and when to prototype. Use of tools to facilitate prototyping.
Lone writer scenario	Small organization where a single person needs to have all skills (architecture, communication, and technical) and wear all hats.

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

Type	**Description**
RSS	Using RSS to distribute your published output.
RDF/OWL	Using RDF/OWL to improve access to your published information. Using tools like SPARQL to query RDF information.

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, and Mac OS.

Sections in this topic:

Installation

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
- XML editor XML catalogs

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 13 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 full package comes with all required tools except the Jaca 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 the DITA authoring tools

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.

Some well-known DITA-aware editors are Adobe FrameMaker, Altova XMLSpy, Arbortext Editor, justsystems XMetaL (DITA edition), Pixware XMLmind, Stylus Studio, SyncRO Soft <Oxygen/>, and Syntext Serna.

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.

DITA Tookit distributions

Distribution overview

Starting with Release 1.3, DITA Open Toolkit is available in several distribution formats for download. They are:

- Full package distributuion
- Small package distribution
- Source distribution

These distributions are all available for download from *http://sourceforge.net/projects/dita-ot*.

Full package distribution

This distribution contains the Toolkit and most of the basic tools required for doing Toolkit builds. This distribution makes it easy to download and install a complete environment. 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

The only other software required to do DITA processing with this package is the Java J2SE SDK. You may optionally want to install the Microsoft HTML Help compiler and the JavaHelp processor.

Small package distribution

This distribution contains only DITA Toolkit. All the required and optional processors and tools must be installed separately to create a functioning build environment. 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 Toolkit prerequisites are still the same as those for release 1.2.2. The small distribution is also 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 (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.

Installing the Java SDK

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_12 SDK** (that is, .12 or the latest version).
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_12".
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_10`.
8. Set the *JAVA_HOME* environment variable to `C:\j2sdk1.4.2_12`.

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_12 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_12".
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 Tookit full package distribution

To install the full package distribution:

- Download the full package from *http://sourceforge.net/projects/dita-ot*.
- 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.

Evaluating the DITA Open Toolkit (fullpackage version)

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` 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` 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` 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 65 for more information on processing.

Demo targets

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

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>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 69 to learn more about processing options.

Installing the DITA Tookit small package distribution

To install the small package distribution:

1. Download the small package from *http://sourceforge.net/projects/dita-ot*.
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
   ```

Installing the required tools for the small package

The following tools are required for you to create and process (build) DITA documents with the small package. Each tool must be at a certain version level to allow it to work with the other tools in the set. For information about the required levels, see *System requirements and supported applications* on page 13.

Ant

Ant is an Apache, Java-based processing (build) tool.

An XSLT processor

You can install either SAXON or Xalan (or both) to process XSLT stylesheets (a set of which are part of the Toolkit).

Installing 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 13). 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.

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.

If this prompt displays,	type the following command
D:\	`C:`
H:\	`C:`
C:\My Documents\\...	`cd \`

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 54.

Installing 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 13

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/*
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

1. Enter the URL:*http://archive.apache.org/dist/xml/xalan-j/*
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`

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 54.

Installing the optional tools

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

For more information, see *http://www.eclipse.org/*.

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

For more information, see *http://www.eclipse.org/*.

(Optional) 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
   ```

(Optional) Installing HTML Help on Windows

1. Enter the URL: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwmicrosofthtmlhelpdownloads.asp*
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.

(Optional) Installing JavaHelp 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`.

(Optional) Installing FOP on Linux

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.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}
   ```

(Optional) Installing JavaHelp on Linux

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 a Linux home directory.
7. Add the *JHHOME* environment variable: `export JHHOME=${javahelp_dir}`.

Installing on Mac OS

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 13). 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 DITA Open Toolkit on Mac OS

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.

Directory	**Description**
root (ditaot)	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 67.
ant	Ant scripts that can be used as-is to process sample files or modified for your use.
css	Sample CSS (cascading style sheet) files.
demo	Specializations, plug-ins, and validators that demonstrate extensions to the base DITA language. Includes: • book: bookmap specialization • elementref: simple element reference description markup • enote: data object specialization • faq: faq (frequently asked questions) specialization • fo: plug-in files to produce PDF output • FrameMaker_adapter: plug-in to produce a structured FrameMaker (7.0+) input file • h2d: plug-in to convert XHTML to DITA topics • java: validators for DITA schemas Many of these directories have README files that provide information about how to use the specializations.
doc	DITA documentation: language reference and application notes.
dtd	Core DITA definitions in XML DTD format.
lib	Contains `dost.jar`, the executable jar file and other jar files it depends on.
plugins	DITA Open Toolkit plug-ins.
resource	Miscellaneous resource files, including the default (common) CSS files and error messages.
samples	Sample DITA source files and Ant scripts.
schema	Core DITA definitions in XML Schema format.
tools	Ant 1.6.5 processor.
xsl	Core and process-specific stylesheets. Includes: • common: stylesheets that can be used by any process (for example, internationalization) • docbook: stylesheets used in converting DITA source content into DocBook source • preprocess: code for conditional, conref, and link resolution • troff: stylesheets used in converting DITA source content into troff source • xslfo: code to support the processing of Formatting Objects (FO) output • xslhtml: code to support XHTML processing • xslrtf: code to support RTF processing

Production notes (installing and upgrading)

Authoring tools

We recommend using a free 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, 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 free and "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 57).

Because our authoring tool didn't have "plausible preview," we did frequent builds to check the output.

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

"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. One editor that needs a catalog is Altova XMLSpy.

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 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"/>
```

3. Keeping the `CustomCatalog.xml` stub file open, also open `ditaot/catalog-dita.xml`.

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

<group xml:base="dtd">

<public publicId="-//IBM//DTD DITA Concept//EN" uri="dita132/concept.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Concept//EN" uri="dita132/concept.mod"></public>

<public publicId="-//IBM//DTD DITA Composite//EN" uri="dita132/ditabase.dtd"></public>

<public publicId="-//IBM//DTD DITA Reference//EN" uri="dita132/reference.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Reference//EN"
uri="dita132/reference.mod"></public>
. . .
```

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

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

<!-- 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>
<public publicId="-//IBM//ELEMENTS DITA Reference//EN"
uri="C:/ditaot/dtd/dita132/reference.mod"></public>
. . .
```

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

```
<?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
[Modify below]
Catalog.xsd">

[Paste here]

</catalog>
```

6. Save the `CustomCatalog.xml` file.

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

```
<?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">

<!-- 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>
<public publicId="-//IBM//ELEMENTS DITA Reference//EN"
uri="C:/ditaot/dtd/dita132/reference.mod"></public>
```

```
<public publicId="-//IBM//DTD DITA Task//EN"
uri="C:/ditaot/dtd/dita132/task.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Task//EN"
uri="C:/ditaot/dtd/dita132/task.mod"></public>

<public publicId="-//IBM//DTD DITA Topic//EN"
uri="C:/ditaot/dtd/dita132/topic.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Topic//EN"
uri="C:/ditaot/dtd/dita132/topic.mod"></public>
<public publicId="-//IBM//ENTITIES DITA Topic Class//EN"
uri="C:/ditaot/dtd/dita132/topic_class.ent"></public>
<public publicId="-//IBM//ENTITIES DITA Topic Definitions//EN"
uri="C:/ditaot/dtd/dita132/topic_defn.ent"></public>

<public publicId="-//IBM//DTD DITA Map//EN"
uri="C:/ditaot/dtd/dita132/map.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Map//EN"
uri="C:/ditaot/dtd/dita132/map.mod"></public>
<public publicId="-//IBM//ENTITIES DITA Map Group Domain//EN"
uri="C:/ditaot/dtd/dita132/mapgroup.ent"></public>
<public publicId="-//IBM//ELEMENTS DITA Map Group Domain//EN"
uri="C:/ditaot/dtd/dita132/mapgroup.mod"></public>

<public publicId="-//IBM//ELEMENTS DITA Highlight Domain//EN"
uri="C:/ditaot/dtd/dita132/highlight-domain.mod"></public>
<public publicId="-//IBM//ENTITIES DITA Highlight Domain//EN"
uri="C:/ditaot/dtd/dita132/highlight-domain.ent"></public>

<public publicId="-//IBM//ELEMENTS DITA Programming Domain//EN"
uri="C:/ditaot/dtd/dita132/programming-domain.mod"></public>
<public publicId="-//IBM//ENTITIES DITA Programming Domain//EN"
uri="C:/ditaot/dtd/dita132/programming-domain.ent"></public>

<public publicId="-//IBM//ELEMENTS DITA Software Domain//EN"
uri="C:/ditaot/dtd/dita132/software-domain.mod"></public>
<public publicId="-//IBM//ENTITIES DITA Software Domain//EN"
uri="C:/ditaot/dtd/dita132/software-domain.ent"></public>

<public publicId="-//IBM//ELEMENTS DITA User Interface Domain//EN"
uri="C:/ditaot/dtd/dita132/ui-domain.mod"></public>
<public publicId="-//IBM//ENTITIES DITA User Interface Domain//EN"
uri="C:/ditaot/dtd/dita132/ui-domain.ent"></public>

<public publicId="-//IBM//ELEMENTS DITA Utilities Domain//EN"
uri="C:/ditaot/dtd/dita132/utilities-domain.mod"></public>
<public publicId="-//IBM//ENTITIES DITA Utilities Domain//EN"
uri="C:/ditaot/dtd/dita132/utilities-domain.ent"></public>

<public publicId="-//IBM//ELEMENTS DITA Metadata//EN"
uri="C:/ditaot/dtd/dita132/meta_xml.mod"></public>
<public publicId="-//IBM//ELEMENTS DITA CALS Tables//EN"
uri="C:/ditaot/dtd/dita132/tbl_xml.mod"></public>

<public publicId="-//OASIS//DTD DITA Concept//EN"
uri="C:/ditaot/dtd/concept.dtd"></public>
<public publicId="-//OASIS//ELEMENTS DITA Concept//EN"
uri="C:/ditaot/dtd/concept.mod"></public>

<public publicId="-//OASIS//DTD DITA Composite//EN"
uri="C:/ditaot/dtd/ditabase.dtd"></public>

<public publicId="-//OASIS//DTD DITA Reference//EN"
uri="C:/ditaot/dtd/reference.dtd"></public>
<public publicId="-//OASIS//ELEMENTS DITA Reference//EN"
uri="C:/ditaot/dtd/reference.mod"></public>

<public publicId="-//OASIS//DTD DITA Task//EN"
uri="C:/ditaot/dtd/task.dtd"></public>
```

```
<public publicId="-//OASIS//ELEMENTS DITA Task//EN"
uri="C:/ditaot/dtd/task.mod"></public>

<public publicId="-//OASIS//DTD DITA Topic//EN"
uri="C:/ditaot/dtd/topic.dtd"></public>
<public publicId="-//OASIS//ELEMENTS DITA Topic//EN"
uri="C:/ditaot/dtd/topic.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Topic Class//EN"
uri="C:/ditaot/dtd/topicAttr.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Topic Definitions//EN"
uri="C:/ditaot/dtd/topicDefn.ent"></public>

<public publicId="-//OASIS//DTD DITA Map//EN"
uri="C:/ditaot/dtd/map.dtd"></public>
<public publicId="-//OASIS//ELEMENTS DITA Map//EN"
uri="C:/ditaot/dtd/map.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Map Group Domain//EN"
uri="C:/ditaot/dtd/mapGroup.ent"></public>
<public publicId="-//OASIS//ELEMENTS DITA Map Group Domain//EN"
uri="C:/ditaot/dtd/mapGroup.mod"></public>

<public publicId="-//OASIS//ELEMENTS DITA Highlight Domain//EN"
uri="C:/ditaot/dtd/highlightDomain.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Highlight Domain//EN"
uri="C:/ditaot/dtd/highlightDomain.ent"></public>

<public publicId="-//OASIS//ELEMENTS DITA Programming Domain//EN"
uri="C:/ditaot/dtd/programmingDomain.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Programming Domain//EN"
uri="C:/ditaot/dtd/programmingDomain.ent"></public>

<public publicId="-//OASIS//ELEMENTS DITA Software Domain//EN"
uri="C:/ditaot/dtd/softwareDomain.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Software Domain//EN"
uri="C:/ditaot/dtd/softwareDomain.ent"></public>

<public publicId="-//OASIS//ELEMENTS DITA User Interface Domain//EN"
uri="C:/ditaot/dtd/uiDomain.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA User Interface Domain//EN"
uri="C:/ditaot/dtd/uiDomain.ent"></public>

<public publicId="-//OASIS//ELEMENTS DITA Utilities Domain//EN"
uri="C:/ditaot/dtd/utilitiesDomain.mod"></public>
<public publicId="-//OASIS//ENTITIES DITA Utilities Domain//EN"
uri="C:/ditaot/dtd/utilitiesDomain.ent"></public>

<public publicId="-//OASIS//ELEMENTS DITA Metadata//EN"
uri="C:/ditaot/dtd/metaDecl.mod"></public>
<public publicId="-//OASIS//ELEMENTS DITA CALS Tables//EN"
uri="C:/ditaot/dtd/tblDecl.mod"></public>
<public publicId="-//OASIS//ELEMENTS DITA Exchange Table Model//EN"
uri="C:/ditaot/dtd/tblDecl.mod"></public>

<public publicId="-//IBM//DTD DITA Element Reference//EN"
uri="C:/ditaot/demo/elementref/elementref_shell.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Element Reference//EN"
uri="C:/ditaot/demo/elementref/elementref.mod"></public>

<public publicId="-//IBM//DTD DITA FAQ//EN"
uri="C:/ditaot/demo/faq/faq_shell.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA FAQ//EN"
uri="C:/ditaot/demo/faq/faq.mod"></public>

<public publicId="-//IBM//DTD DITA eNote//EN"
uri="C:/ditaot/demo/enote/enote_shell.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA eNote//EN"
uri="C:/ditaot/demo/enote/enote.mod"></public>

<public publicId="-//IBM//DTD DITA BookMap//EN"
```

```
uri="C:/ditaot/demo/book/bookmap.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA BookMap//EN"
uri="C:/ditaot/demo/book/bookmap.mod"></public>

<public publicId="-//IBM//DTD DITA Book Information//EN"
uri="C:/ditaot/demo/book/bkinfo.dtd"></public>
<public publicId="-//IBM//ELEMENTS DITA Book Information//EN"
uri="C:/ditaot/demo/book/bkinfo.mod"></public>

</catalog>
```

For more detailed information about creating an XMLSpy catalog for DITA DTDs, see *http://www.altova.com/manual2006/xmlspy/spyprofessional/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 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>

  <conbody>
    <p/>
  </conbody>
</concept>
```

Here is the complete list of 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

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.

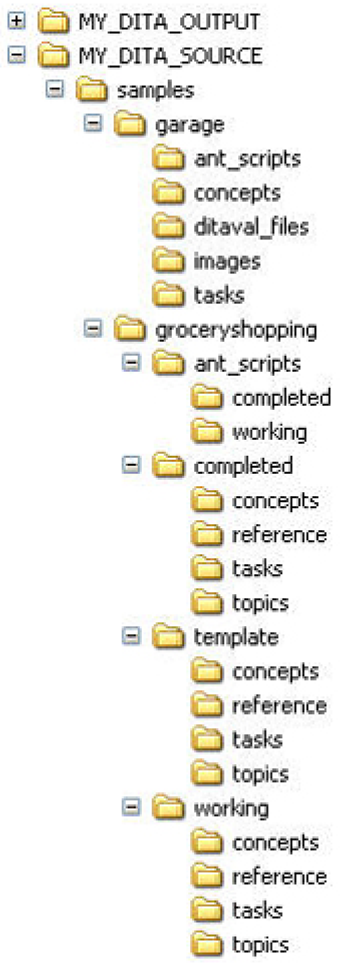
A few simple entries in your Ant build scripts allow this separation.

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

1. Create two 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 two directories are: `C:/MY_DITA_SOURCE` and `C:/MY_DITA_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.
2. Within `C:/MY_DITA_SOURCE` create a `samples` subdirectory.
3. Copy the garage sample files from `ditaot/samples` to the `samples` directory you just created.
4. Copy the grocery shopping sample files from `ditaot/samples` to the `samples` directory you just created.

Your directory structure should look like this:



```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
                completed
                working
            completed
                concepts
                reference
                tasks
                topics
            template
                concepts
                reference
                tasks
                topics
            working
                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 65.

For information about creating demo files of your own, see *Creating DITA topics* on page 123 and *Creating DITA maps* on page 131.

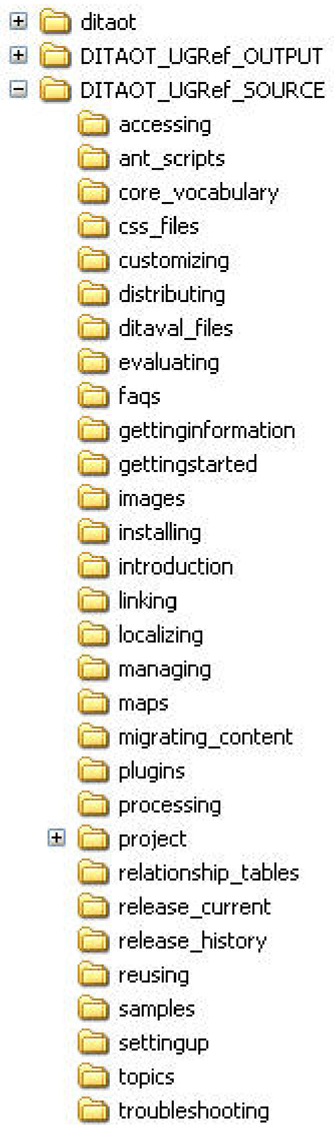
Production notes (setting up)

How we set up our working environment

We structured the *DITA Open Toolkit User Guide and Reference* using the bookmap 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:



```
⊞ ditaot
⊞ DITAOT_UGRef_OUTPUT
⊟ DITAOT_UGRef_SOURCE
     accessing
     ant_scripts
     core_vocabulary
     css_files
     customizing
     distributing
     ditaval_files
     evaluating
     faqs
     gettinginformation
     gettingstarted
     images
     installing
     introduction
     linking
     localizing
     managing
     maps
     migrating_content
     plugins
     processing
   ⊞ project
     relationship_tables
     release_current
     release_history
     reusing
     samples
     settingup
     topics
     troubleshooting
```

For more information (setting up)

Name, description	**Location**
Support information for **Altova XMLSpy**	*http://www.altova.com/support_center.html*
General information about configuring and using **XML catalogs**	*http://forrest.apache.org/docs_0_70/catalog.html*
Customizing XMetaL Author	*http://www.xmetal.com/en_us/_pdf/user_guides/XMetaL46_Customization_Guide.pdf*
XMLMind XML Editor and DITA	*http://www.xmlmind.com/xmleditor/_distrib/doc/dita/index.html*

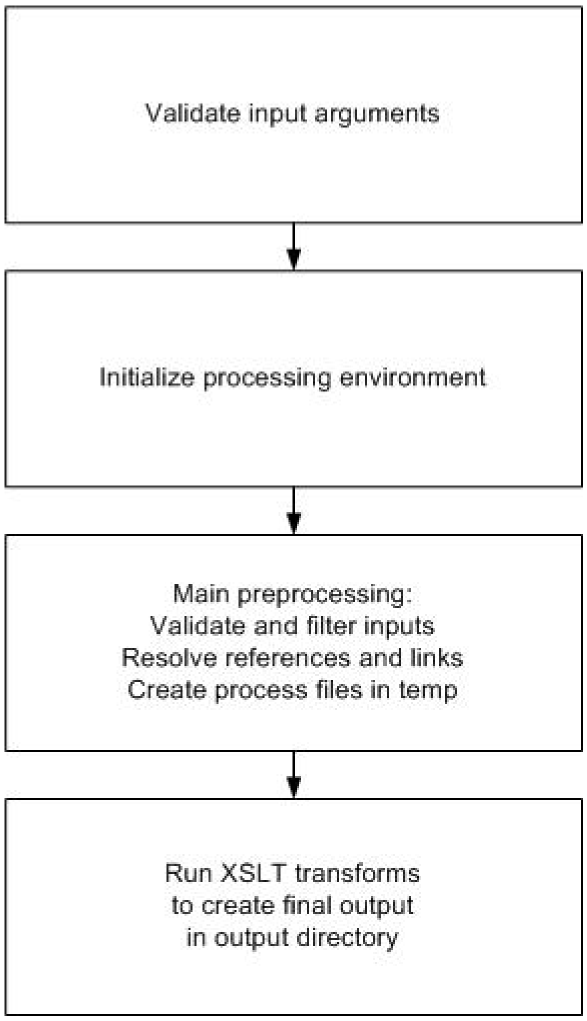
Processing (building) and publishing DITA documents

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

Sections in this topic:

Processing overview

The following diagram shows the major steps in processing to target output files using DITA Open Toolkit.


Validate input arguments
Initialize processing environment
Main preprocessing:
Validate and filter inputs
Resolve references and links
Create process files in temp
Run XSLT transforms
to create final output
in output directory

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 `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.

Main system scripts in DITA Open Toolkit

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

Creating user scripts in DITA Open Toolkit

Sample Ant scripts for all target publishing environment supported by DITA Open Toolkit are located in `ditaot/doc/ot-userguide/MY_DITA_SOURCE/samples/garage/ant_scripts` of the Toolkit source distribution. Most of these scripts process the garage sample source files with the topics displayed in a hierarchy. One processes the garage sample to XHTML with the topics displayed as a sequence. One filters out some of the topics using a ditaval file before publishing as XHTML in a hierarchical format.

The following section contains one of these sample scripts for XHTML targets. You would run this build script in Windows by opening the Command Prompt, navigating to the `ant_scripts directory`, and entering the command:

```
ant -f garage_hierarchy_xhtml.xml
```

For more information about processing (building) to XHTML targets, see *Processing to XHTML targets* on page 74.

Sample user script

Here is an annotated script for publishing the garage sample to XHTML in a hierarchical format. The lines in bold are the actual script statements; the other lines are annotations.

```

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

```
<!-- Copyright 2006 VR Communications, Inc. -->
<!-- All rights reserved. -->
<!-- SAMPLE: GARAGE -->
<!-- DITAMAP: HIERARCHY -->
<!-- TARGET: XHTML -->

<!-- SAMPLE ANT BUILD SCRIPT TO CREATE XHTML OUTPUT -->
<!-- from the provided sample ("garage"). -->

<!-- This is a "user script," meant to be -->
<!-- modified; however, be cautious in modifying the -->
<!-- environment initialization section. -->

<!-- -->
<!-- ENVIRONMENT INITIALIZATION SECTION -->
<!-- -->

<!-- Modify with caution. -->
<!-- garage_xhtml is an arbitrary name alluding to the -->
<!-- source and target trees. -->

<!-- default="all" means "build all the targets in the -->
<!-- 'depends' target list" (below) -->
<!-- (NOT all targets supported by DITA Open Toolkit). -->
<!-- basedir is the base directory for the Toolkit -->
<!-- executables (NOT the DITA source directory, -->
<!-- which is defined with projdir, below). -->
<!-- OK to modify. -->
<project name="garage_xhtml" default="all" basedir="C:/ditaot">


<!-- Location of DITA source files (projdir) and target -->
<!-- output files (outdir). -->
<!-- OK to modify. -->
<property name="projdir" value="C:/MY_DITA_SOURCE/samples/garage"/>
    <property name="outdir" value="C:/MY_DITA_OUTPUT/samples/garage"/>

<!-- Location of DITA Java classes. -->
<!-- DO NOT modify! -->
<path id="dost.class.path">
    <pathelement location="${basedir}/lib/dost.jar"/>
    </path>

<!-- Ant task to initialize the processing environment. -->
<!-- Defines a new Ant task called "integrate" that -->
<!-- executes the code defined in "classname". -->
<!-- DO NOT modify! -->
<taskdef name="integrate" classname="org.dita.dost.platform.IntegratorTask">
    <classpath refid="dost.class.path"/>
    </taskdef>

<!-- dita2xhtml is a somewhat arbitrary name that -->
<!-- alludes to the source and target. -->
<!-- It must match the target name in the instance -->
<!-- processing section below. -->
<!-- ${basedir} is the "basedir" defined above. -->
<!-- OK to modify. -->
<target name="all" depends="integrate, dita2xhtml"></target>
    <target name="integrate">
    <integrate ditadir="${basedir}"/>
    </target>

<!-- -->
<!-- INSTANCE PROCESSING SECTION -->
<!-- -->

<!-- Modify to process a particular source instance. -->
<!-- This sample builds only one target. -->
<!-- You could add other target sections -->
<!-- to build to other targets -->
<!-- (for example, PDF or HTML Help). -->
```

```
<!-- If you do that, you must also add appropriate names -->
<!-- (for example, dita2pdf or dita2htmlhelp) -->
<!-- to the "depends" list in the environment -->
<!-- initialization section above. -->

<!-- This section builds to a single target (xhtml). -->
<!-- Target name must match the target name in the -->
<!-- "depends" list in the environment initialization -->
<!-- section. -->
<target name="dita2xhtml">

<!-- The properties included below are input parameters. -->
<!-- They are listed and defined in the DITA OT User Guide -->
<!-- and Reference. -->
<ant antfile="${basedir}/conductor.xml" target="init">

<!-- projdir is defined in the environment initialization -->
<!-- section above. -->
<property name="args.input" value="${projdir}/hierarchy.ditamap"/>

<!-- outdir is defined in the environment initialization -->
<!-- section above. -->
<property name="output.dir" value="${outdir}/xhtml/hierarchy/unfiltered"/>

<!-- Name of the DITA temporary directory where files -->
<!-- are stored during processing. -->
<property name="dita.temp.dir" value="${outdir}/temp"/>

<!-- transformation type (target output type). -->
<property name="transtype" value="xhtml"/>

<!-- The system default extname is .xml. -->
<!-- The following statement changes -->
<!-- the default to .dita. -->
<!-- If you use other extensions -->
<!-- (including .ditamap and .xml, -->
<!-- but also extensions like .jpg and .gif), -->
<!-- you must specify the format attribute in -->
<!-- your source files (for example, format="xml"). -->
<property name="dita.extname" value=".dita"/>
    </ant>
    </target>
    </project>
```

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 67.

Parameter (*Required), Target	Definition, Usage	Valid values, Default, Examples
`args.artlbl` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	Adds annotation to images showing the filename of the image. Useful for pre-publishing editing.	Valid: yes or no Default: no
`args.copycss`	Whether to copy user-specified CSS file(s) to the directory	Valid: yes or no

Parameter (*Required), Target	**Definition, Usage**	**Valid values, Default, Examples**
These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	specified `{args.outdir}${args.csspath}`.	Default: no
`args.css` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	Name of user-specified CSS file. Local or remote (web) file. If `${args.csspath}` is a URL, `${args.css}` must be a filepath relative to the URL.	
`args.csspath` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	Path to user-specified CSS file. Notes: • If `${args.csspath}` is a URL, it must start with `http://` or `https://`. • Local absolute paths are not supported for `args.csspath`. • Use "/" as the path separator, and do not append a "/" trailing separator (for example, use `css/mycssfiles` rather than `css/mycssfiles/`).	Default: no path Example: `http://www.ibm.com/css`
`args.cssroot` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	Root directory of user-specified CSS file. If this parameter is set, `${args.css}` must be a filepath relative to `args.cssroot`.	
`args.dita.locale` These targets only: htmlhelp and javahelp	Locale used for sorting indexterms. If no locale is specified, the first occurrence of "xml-lang" is used as the default locale.	Default (If "xml-lang" is not specified): en-us
`args.draft` All targets	Include draft and required cleanup content (that is, items identified as left to do before publishing).	Valid: yes or no Default: no
`args.eclipsecontent.toc` Target: eclipsecontent only	Root file name of the output Eclipse content `toc` file.	Default: name of the source ditamap file
`args.eclipsehelp.toc` Targets: eclipsehelp only	Root file name of the output Eclipse help `toc` file.	Default: name of the source ditamap file
`args.eclipse.provider`	Provider name of the Eclipse help output.	Default: DITA

Parameter (*Required), Target	Definition, Usage	Valid values, Default, Examples
Targets: eclipsehelp only		
`args.eclipse.version` Targets: eclipsehelp only	Version number of the Eclipse help output.	Default: 1.0
`args.fo.img.ext` Target: pdf only	Extension name of the image files in the PDF output. A given target set can contain only one or the other extension (not both). Image files with the non-specified extension will be renamed during the build process.	
`args.fo.output.rel.links` Target: pdf only	Whether links will appear in the output files.	Valid: yes or no Default: no
`args.fo.userconfig` Target: pdf only	Name of the configuration file for FOP processing.	Valid: yes or no Default: no
`args.ftr` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	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/myftr.xml"/>`
`args.hdf` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	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"/>`.
`args.hdr` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	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"/>`
`args.htmlhelp.includefile` Target: htmlhelp only	File to be included in the output.	
`args.indexshow` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	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

Parameter (*Required), Target	**Definition, Usage**	**Valid values, Default, Examples**
`*args.input` All targets	Path and name of the input file. Use the same case as the filename.	Example: `<property name="args.input" value="mastermap.ditamap"/>`
`args.javahelp.map` Target: javahelp only	Root file name of the output JavaHelp map file.	Default: name of the input ditamap file
`args.javahelp.toc` Target: javahelp only	Root file name of the output JavaHelp toc file.	Default: name of the input ditamap file
`args.outext` These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml	Output file extension name for generated XHTML files. In most browser environments, either html or htm is acceptable.	Valid: html or htm Default: html
`args.logdir` All targets	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` Target: xhtml only	Root file name of the output XHTML toc file.	Default: index
`args.xsl` All targets except Eclipse content and troff	xsl transform file that will replace the default file: • For transtype="docbook", dita2docbook.xsl will be replaced.	Example: `<property name="args.input" value="mastermap.ditamap"/>`

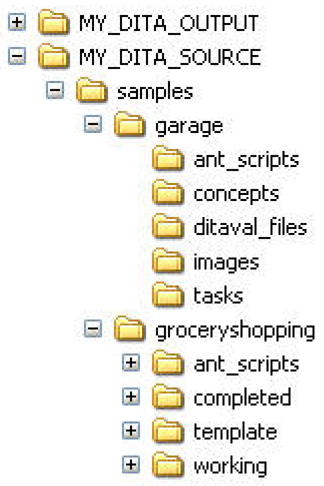
Parameter (*Required), Target	Definition, Usage	Valid values, Default, Examples
	• For transtype="eclipsehelp" or transtype="xhtml": dita2xhtml.xsl. • For transtype="html" or transtype="javahelp": dita2html.xsl. • For transtype="pdf": dita2fo-shell.xsl. • For transtype="rtf": dita2rtfImpl.xsl.	
`basedir` All targets	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` All targets	Whether to clean the `temp` directory before each build.	Valid: yes or no Default: no
`dita.dir` All targets	Absolute path of the Toolkit's home directory.	
`dita.extname` All targets	File extension of the DITA source files. If you use extensions other than the default or the one you specify with this processing option (including .ditamap, but also extensions like .jpg and .gif) you must specify the format attribute (for example, format="pdf") in your source file references. If you don't, you will get an error message.	Default: .xml in release 1.2; .dita in release 1.3 Example: <property name="dita.extname" value=".dita"/>
`dita.input.valfile` All targets	Name of the ditaval file that contains filter/flagging/revision information.	
`dita.temp.dir` All targets	Directory for the temporary files generated during the build.	Default: temp

Parameter (*Required), Target	Definition, Usage	Valid values, Default, Examples
***output.dir** All targets	Path of the output directory.	Example: <property name="output.dir" value="${projdir}/out/xhtml"/>
***transtype** All targets	Type of output to be produced.	Valid: docbook, eclipsecontent, eclipsehelp, htmlhelp, javahelp, pdf, troff, wordrtf, or xhtml Example: <property name="transtype" value="htmlhelp">

About the garage sample

The garage sample, which is located in the `ditaot/doc/ot-usergide/MY_DITA_SOURCE/samples` directory of the Toolkit source distribution, 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 to allow you to publish to all supported target environments.

Before you begin to use the sample files (which include both the garage sample and the grocery shopping sample), we recommend creating two directories in your root directory called `MY_DITA_SOURCE` and `MY_DITA_OUTPUT` (Windows examples would be `C:/MY_DITA_SOURCE` and `C:/MY_DITA_OUTPUT`) and then copying both the garage and grocery shopping sample files from the `ditaot/samples` directory to `MY_DITA_SOURCE`. Your directory structure should then look like this:



The garage sample includes Ant scripts that process to all supported target environments. A filtering (conditional processing) script is also included: `garage_filtering_xhtml.xml`. This script filters out (excludes) all files having to do with oil or snow, which are tagged with the "otherprops" attribute. Running this script produces a hierarchically organized output file with four of the topics excluded.

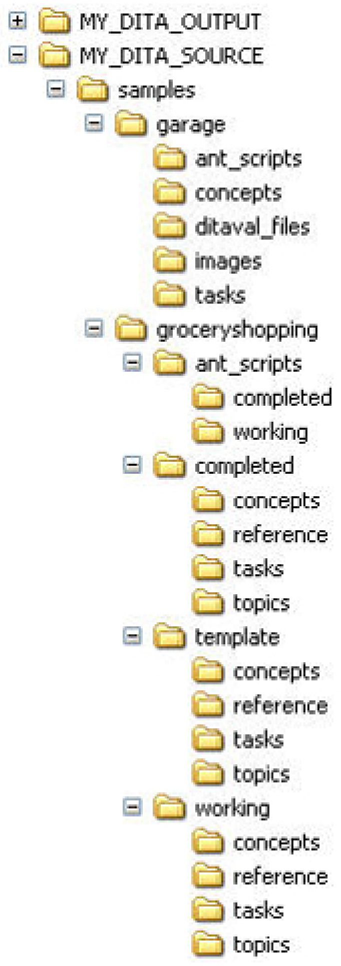
Processing to XHTML targets

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:

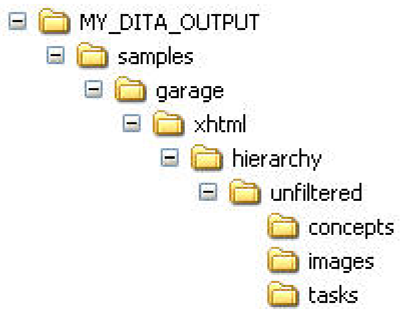
```
+ MY_DITA_OUTPUT
- MY_DITA_SOURCE
    - samples
        - garage
              ant_scripts
              concepts
              ditaval_files
              images
              tasks
        - groceryshopping
            - ant_scripts
                  completed
                  working
            - completed
                  concepts
                  reference
                  tasks
                  topics
            - template
                  concepts
                  reference
                  tasks
                  topics
            - working
                  concepts
                  reference
                  tasks
                  topics
```

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

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

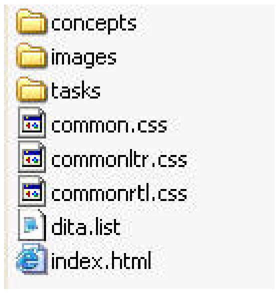
   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to xhtml.

4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_xhtml.xml`
5. After the XHTML file has processed successfully, go to the `MY_DITA_OUTPUT` directory.
   Your directory structure should look like this:



```
MY_DITA_OUTPUT
  samples
    garage
      xhtml
        hierarchy
          unfiltered
            concepts
            images
            tasks
```

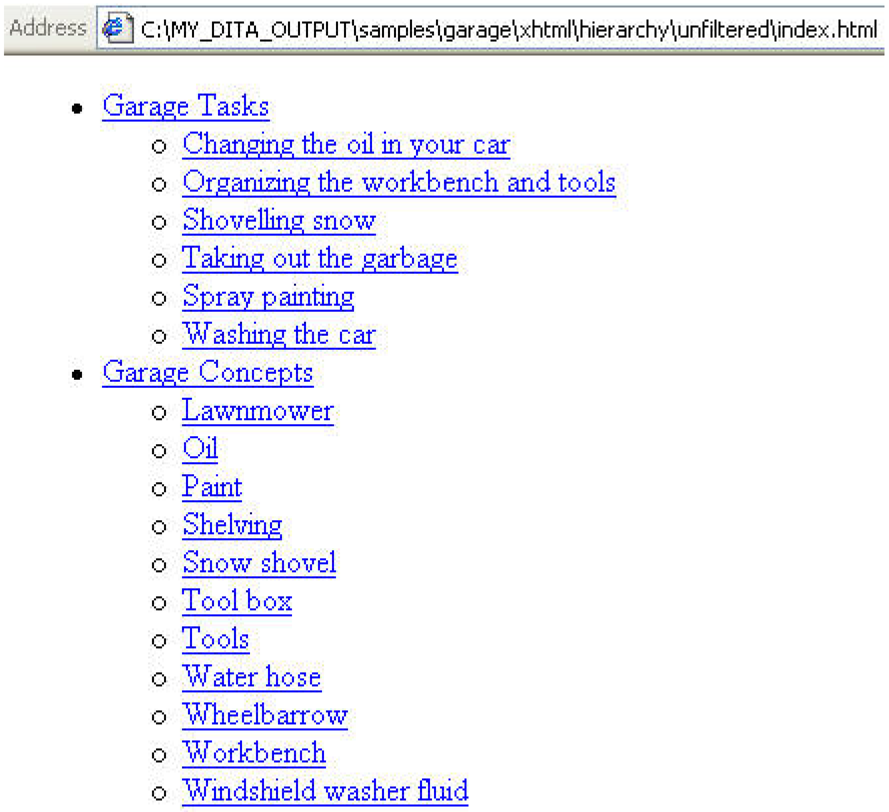
The `unfiltered` subdirectory should contain these directories and files:



```
concepts
images
tasks
common.css
commonltr.css
commonrtl.css
dita.list
index.html
```

6. **Open the file `index.html` in your browser to view the XHTML output.**

   The browser window should look something like this:



C:\MY_DITA_OUTPUT\samples\garage\xhtml\hierarchy\unfiltered\index.html

- 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

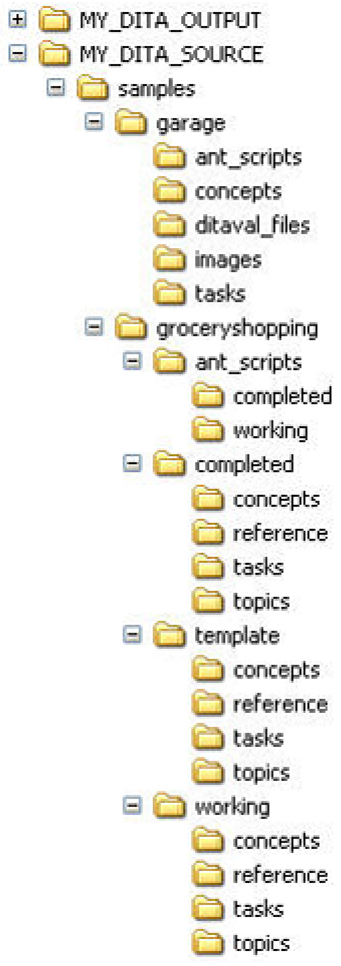
Processing to HTML Help targets

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 sample files into the `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:

```
+ MY_DITA_OUTPUT
- MY_DITA_SOURCE
    - samples
        - garage
              ant_scripts
              concepts
              ditaval_files
              images
              tasks
        - groceryshopping
            - ant_scripts
                  completed
                  working
            - completed
                  concepts
                  reference
                  tasks
                  topics
            - template
                  concepts
                  reference
                  tasks
                  topics
            - working
                  concepts
                  reference
                  tasks
                  topics
```

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

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

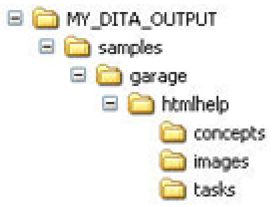
   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to htmlhelp.

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

For example: `ant -f garage_hierarchy_htmlhelp.xml`

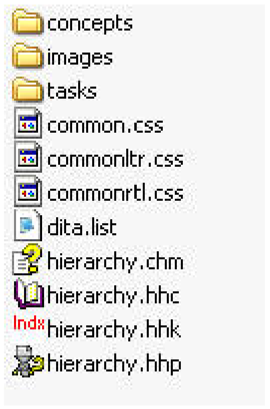
5. After the HTML Help file has processed successfully, move to the `MY_DITA_OUTPUT` directory.

   Your directory structure should look like this:



```
MY_DITA_OUTPUT
  samples
    garage
      htmlhelp
        concepts
        images
        tasks
```

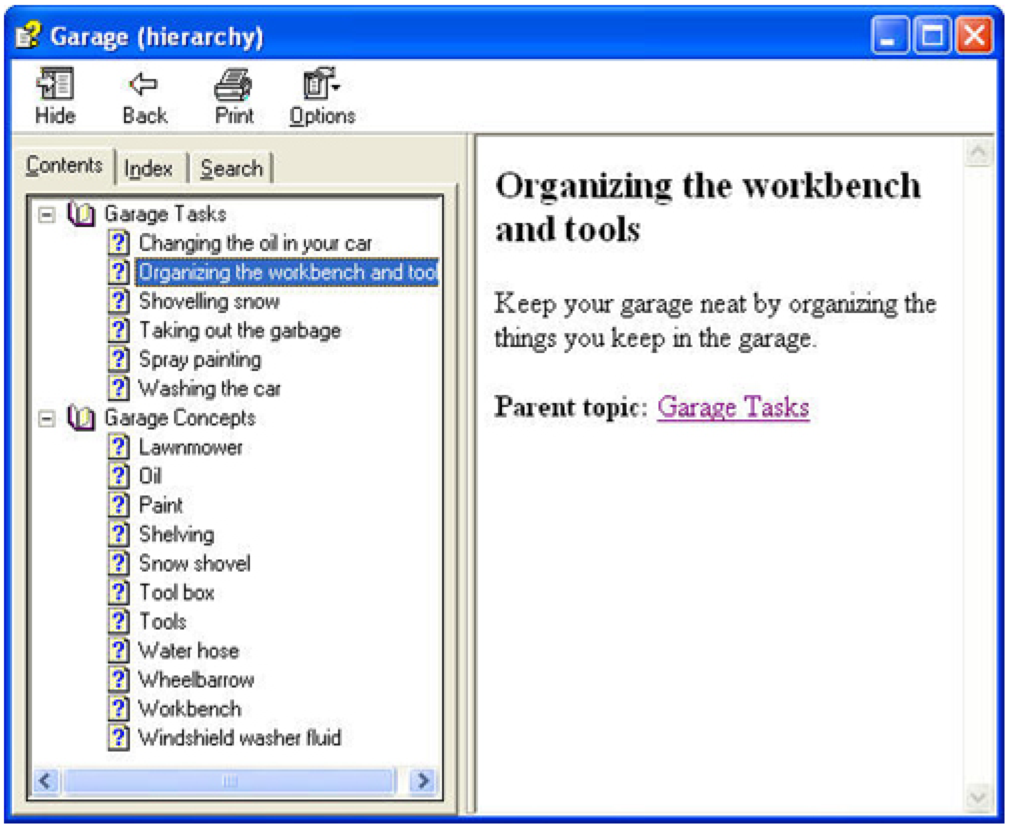
The `htmlhelp` directory should contain these directories and files:



```
concepts
images
tasks
common.css
commonltr.css
commonrtl.css
dita.list
hierarchy.chm
hierarchy.hhc
hierarchy.hhk
hierarchy.hhp
```

6. Open the file `hierarchy.chm` in your browser to view the HTML Help output.

   The window should look something like this:



Garage Tasks
- Changing the oil in your car
- Organizing the workbench and tool
- 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

Organizing the workbench and tools

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

Parent topic: Garage Tasks

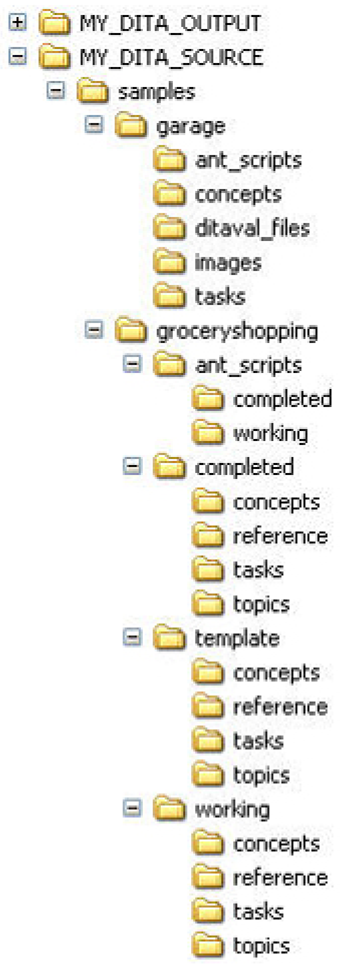
Processing to PDF2 targets

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. See *Installing the Idiom FO plug-in* for information on how to do this.

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:

```
⊞ MY_DITA_OUTPUT
⊟ MY_DITA_SOURCE
    ⊟ samples
        ⊟ garage
              ant_scripts
              concepts
              ditaval_files
              images
              tasks
        ⊟ groceryshopping
            ⊟ ant_scripts
                  completed
                  working
            ⊟ completed
                  concepts
                  reference
                  tasks
                  topics
            ⊟ template
                  concepts
                  reference
                  tasks
                  topics
            ⊟ working
                  concepts
                  reference
                  tasks
                  topics
```

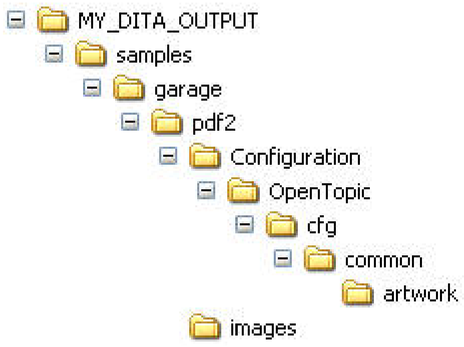
3. View and edit, if necessary for your specific working environment, `MY_DITA_SOURCE/ant_scripts/garage_hierarchy_pdf2.xml`.

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

   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to pdf2.

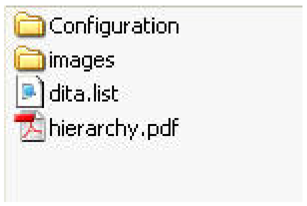
4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_pdf2.xml`
5. After the PDF2 file has processed successfully, go to the `MY_DITA_OUTPUT` directory.

   Your directory structure should look like this:

   ```
   MY_DITA_OUTPUT
     samples
       garage
         pdf2
           Configuration
             OpenTopic
               cfg
                 common
                   artwork
           images
   ```

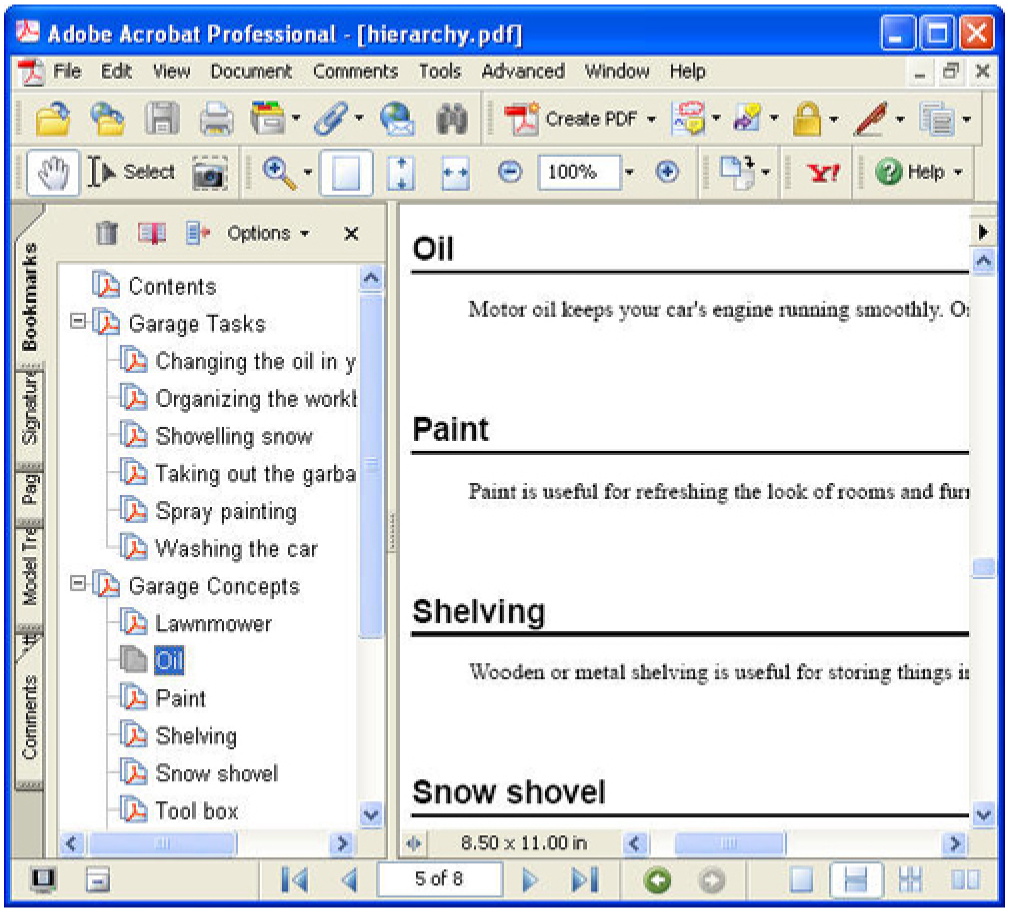
   The `pdf2` subdirectory should contain these directories and files:

   ```
   Configuration
   images
   dita.list
   hierarchy.pdf
   ```

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]
File Edit View Document Comments Tools Advanced Window Help
Create PDF
Select
100%
Help
Options
Bookmarks
Signatures
Pages
Model Tree
Comments
8.50 x 11.00 in
5 of 8

Contents
Garage Tasks
Changing the oil in y
Organizing the workb
Shovelling snow
Taking out the garba
Spray painting
Washing the car
Garage Concepts
Lawnmower
Oil
Paint
Shelving
Snow shovel
Tool box

Oil

Motor oil keeps your car's engine running smoothly. O

Paint

Paint is useful for refreshing the look of rooms and fur

Shelving

Wooden or metal shelving is useful for storing things i

Snow shovel

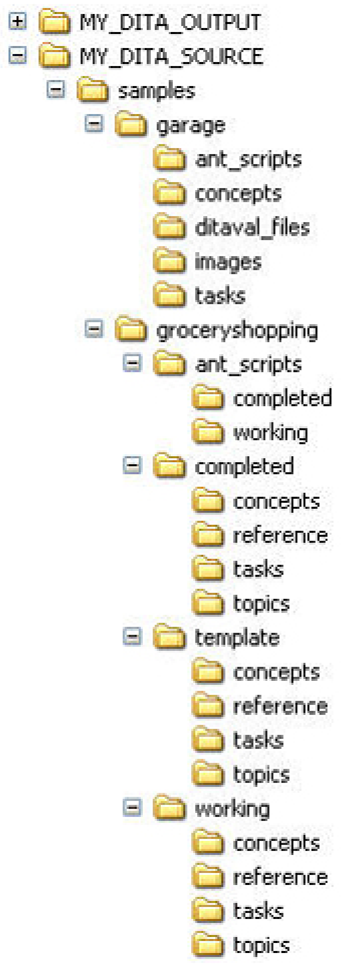
Processing to DocBook targets

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.

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:



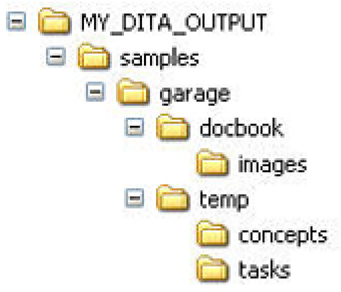
```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
                completed
                working
            completed
                concepts
                reference
                tasks
                topics
            template
                concepts
                reference
                tasks
                topics
            working
                concepts
                reference
                tasks
                topics
```

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

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

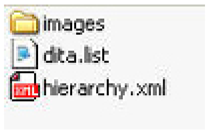
   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to docbook.

4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_docbook.xml`
5. After the DocBook file has processed successfully, go to the `MY_DITA_OUTPUT` directory.
   Your directory structure should look like this:



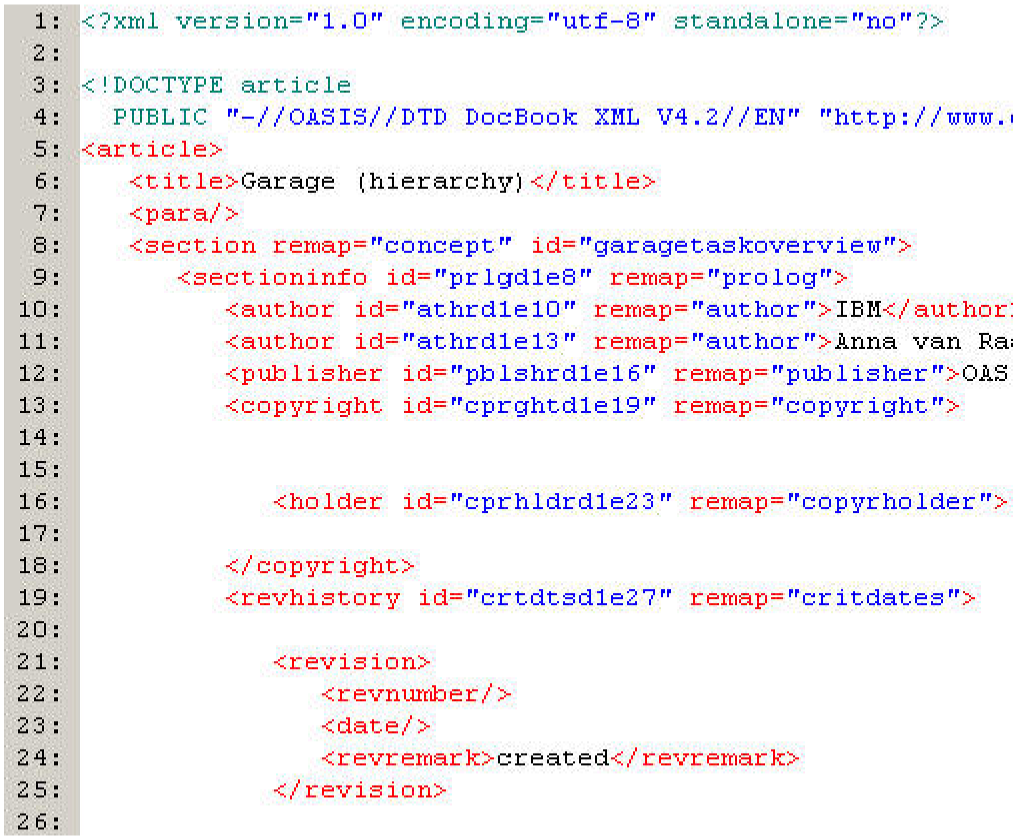
```
MY_DITA_OUTPUT
    samples
        garage
            docbook
                images
            temp
                concepts
                tasks
```

   The `docbook` subdirectory should contain these directories and files:



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

   In Arbortext Editor, the file should look something like this:



```
<?xml version="1.0" encoding="utf-8" standalone="no"?>

<!DOCTYPE article
  PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN" "http://www.
<article>
   <title>Garage (hierarchy)</title>
   <para/>
   <section remap="concept" id="garagetaskoverview">
      <sectioninfo id="prlgd1e8" remap="prolog">
         <author id="athrd1e10" remap="author">IBM</author
         <author id="athrd1e13" remap="author">Anna van Ra
         <publisher id="pblshrd1e16" remap="publisher">OAS
         <copyright id="cprghtd1e19" remap="copyright">


            <holder id="cprhldrd1e23" remap="copyrholder">

         </copyright>
         <revhistory id="crtdtsd1e27" remap="critdates">

            <revision>
               <revnumber/>
               <date/>
               <revremark>created</revremark>
            </revision>

```

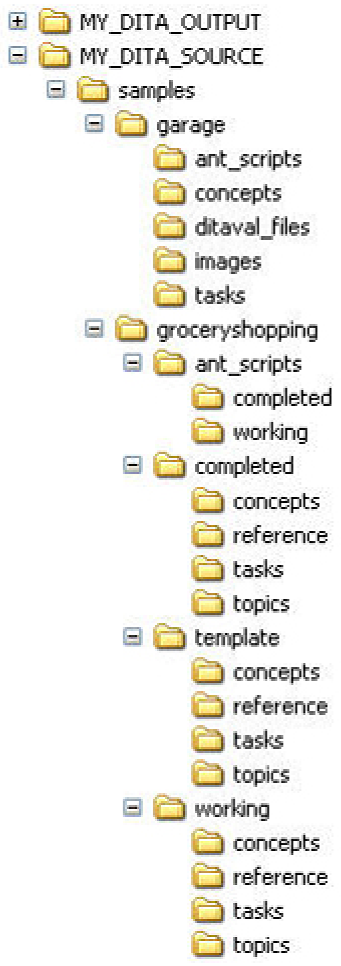
Processing to Eclipse content targets

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:



```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
                completed
                working
            completed
                concepts
                reference
                tasks
                topics
            template
                concepts
                reference
                tasks
                topics
            working
                concepts
                reference
                tasks
                topics
```

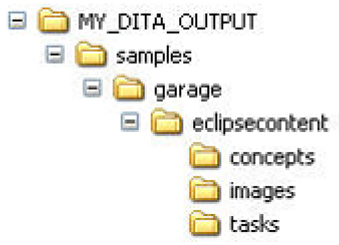
3. View and edit, if necessary for your specific working environment, `MY_DITA_SOURCE/ant_scripts/garage_hierarchy_eclipsecontent.xml`.

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

   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to eclipsecontent.

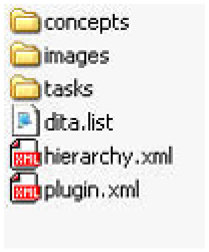
4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_eclipsecontent.xml`
5. After the Eclipse content file has processed successfully, go to the `MY_DITA_OUTPUT` directory.

   Your directory structure should look like this:



```
MY_DITA_OUTPUT
    samples
        garage
            eclipsecontent
                concepts
                images
                tasks
```

   The `eclipsecontent` subdirectory should contain these directories and files:



concepts
images
tasks
dita.list
hierarchy.xml
plugin.xml

6. Open the file `hierarchy.xml` in Eclipse to view the Eclipse content output.

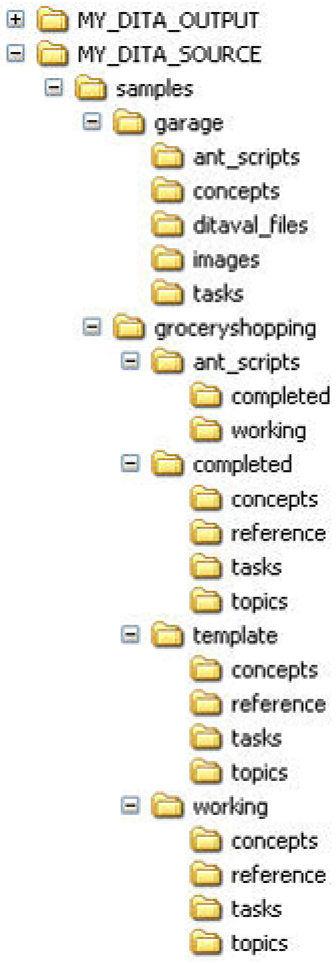
Processing to Eclipse help targets

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:



```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
                completed
                working
            completed
                concepts
                reference
                tasks
                topics
            template
                concepts
                reference
                tasks
                topics
            working
                concepts
                reference
                tasks
                topics
```

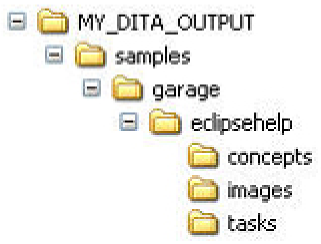
3. View and edit, if necessary for your specific working environment, `MY_DITA_SOURCE/ant_scripts/garage_hierarchy_eclipsehelp.xml`.

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

   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to eclipsehelp.

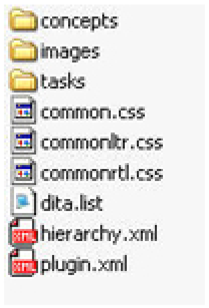
4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_eclipsehelp.xml`
5. After the Eclipse help file has processed successfully, go to the `MY_DITA_OUTPUT` directory.

   Your directory structure should look like this:

   ```
   MY_DITA_OUTPUT
     samples
       garage
         eclipsehelp
           concepts
           images
           tasks
   ```

   The `htmlhelp` subdirectory should contain these directories and files:

   ```
   concepts
   images
   tasks
   common.css
   commonltr.css
   commonrtl.css
   dita.list
   hierarchy.xml
   plugin.xml
   ```

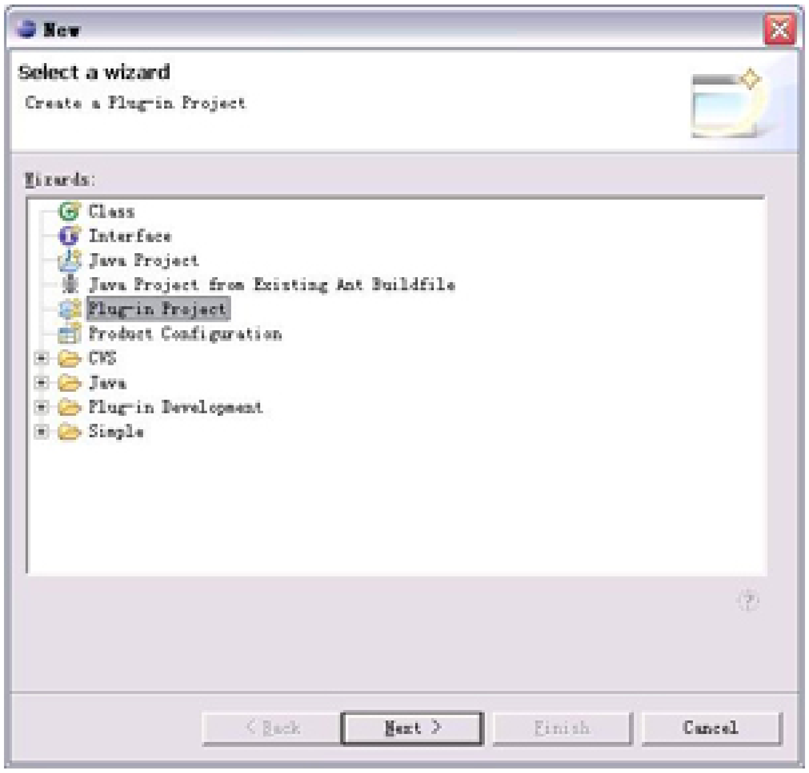
6. Open the file `hierarchy.xml` in Eclipse to view the Eclipse help output.

Using a DITA template for Eclipse document plug-ins

You can use a template to develop a documentation plug-in with DITA in Eclipse PDE. When you want to develop a documentation plug-in with DITA in Eclipse, you cannot use the previous releases of DITA OT in Eclipse to transform DITA to HTML.

Although previous releases of DITA OT supported transforming DITA files to an Eclipse documentation plug-in, they were not integrated with Eclipse. With DITA OT 1.3 integrated with WPT, you can develop document plug-ins with DITA in the Eclipse PDE and build and package the final plug-in by taking the following steps.

1. Create a new PDE project in Eclipse, and apply the DITA template to the project by following the wizard.


New

Select a wizard

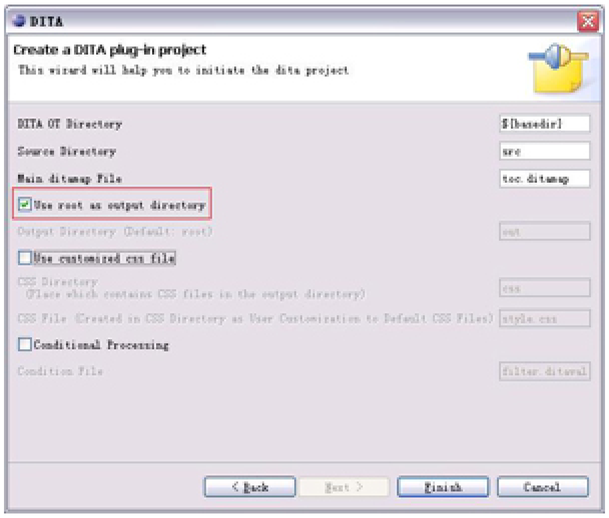
Create a Plug-in Project

Wizards:

- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- Product Configuration
- CVS
- Java
- Plug-in Development
- Simple

< Back | Next > | Finish | Cancel

2. Set the source directory, the main ditamap file, the output directory (the default value is the root directory of the project), css storage directory (used to contain common.css, commonltr.css, and commonrtl.css), user customized .css file name, and conditional processing ditaval file in the wizard. **Use root as output directory** is selected as the default.


DITA

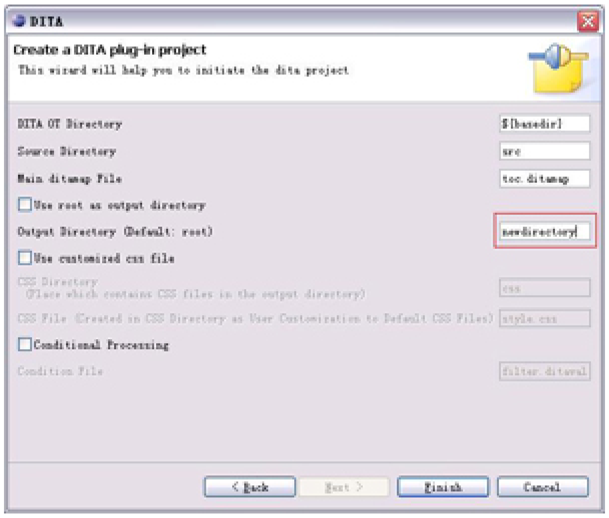
Create a DITA plug-in project

This wizard will help you to initiate the dita project

DITA OT Directory	${basedir}
Source Directory	src
Main ditamap File	toc.ditamap
[✓] Use root as output directory	
Output Directory (Default: root)	out
[ ] Use customized css file	
CSS Directory (Place which contains CSS files in the output directory)	css
CSS File (Created in CSS Directory as User Customization to Default CSS Files)	style.css
[ ] Conditional Processing	
Condition File	filter.ditaval

< Back | Next > | Finish | Cancel

You can also clear **Use root as the output directory** and specify another output directory.


DITA

Create a DITA plug-in project

This wizard will help you to initiate the dita project

DITA OT Directory: ${basedir}

Source Directory: src

Main ditamap File: toc.ditamap

☐ Use root as output directory

Output Directory (Default: root): newdirectory

☐ Use customized css file

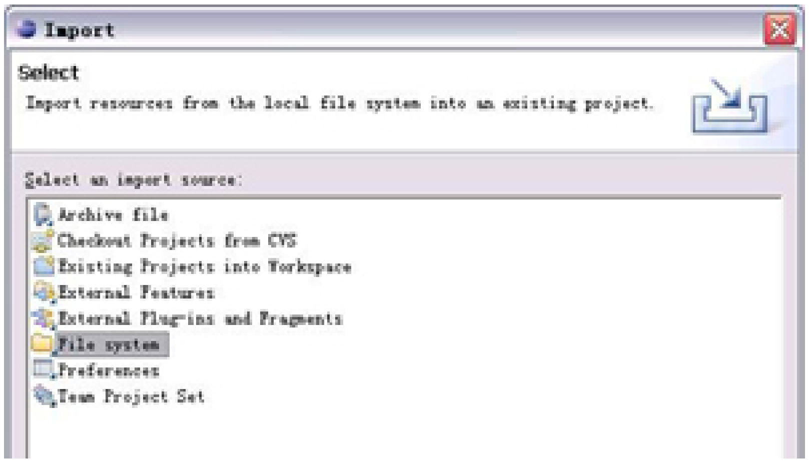
CSS Directory (Place which contains CSS files in the output directory): css

CSS File (Created in CSS Directory as User Customization to Default CSS Files): style.css

☐ Conditional Processing

Condition File: filter.ditaval

3. Create DITA files in the source directory and a ditamap to include the topic files that you created.
4. Import the DITA files into the `src` directory of the DITA plug-in project you just created.
   a) Right-click a directory where you want to put the imported files and select Import, and then File system.


Import

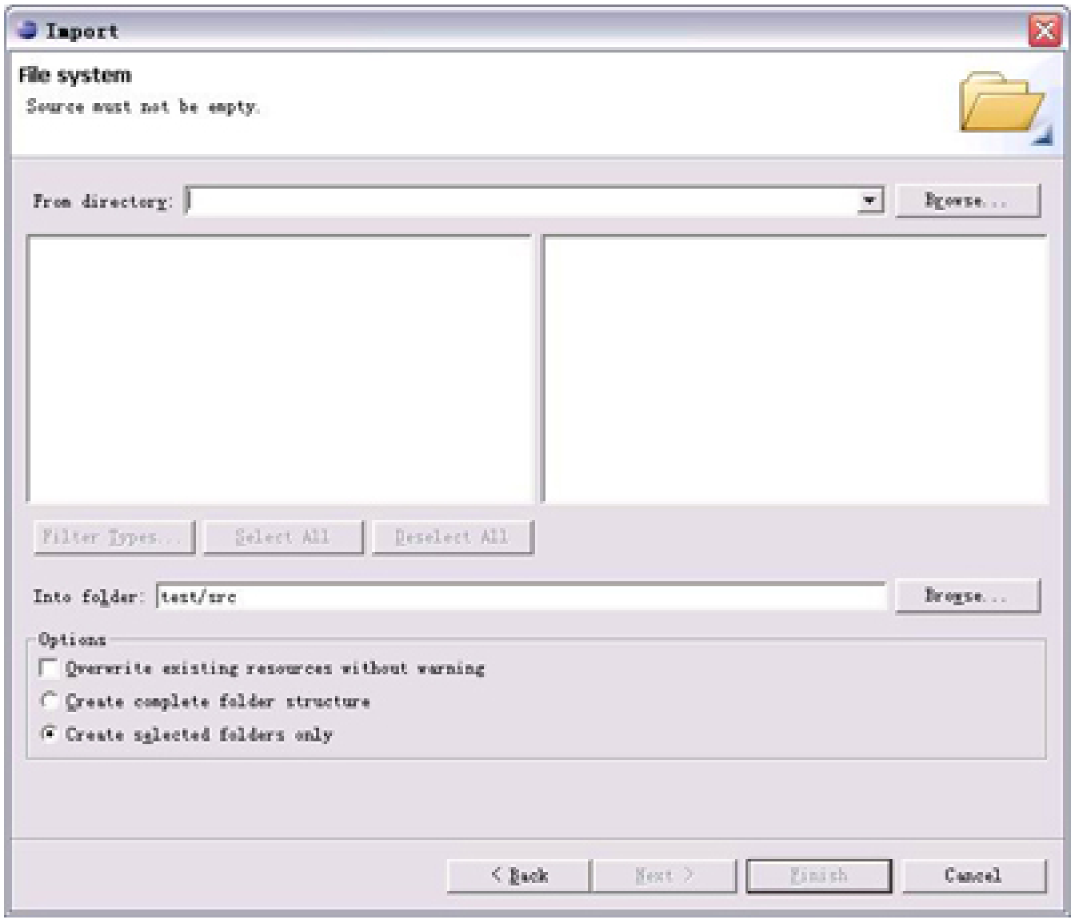
Select

Import resources from the local file system into an existing project.

Select an import source:

Archive file
Checkout Projects from CVS
Existing Projects into Workspace
External Features
External Plug-ins and Fragments
File system
Preferences
Team Project Set

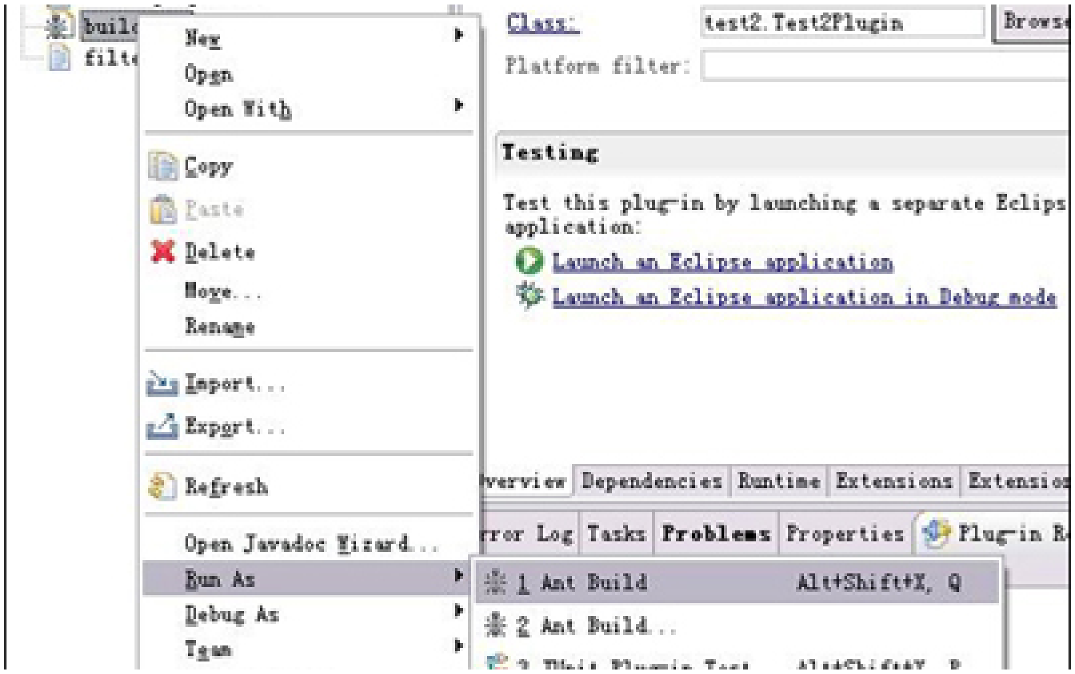
   b) Select the directory under which you put the DITA files.



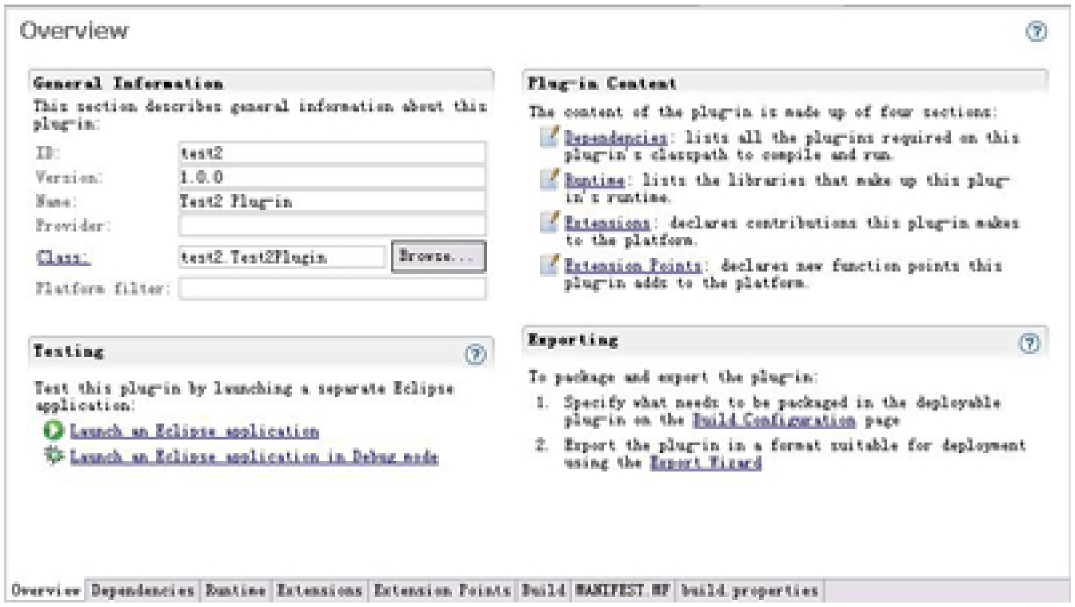
c) Click Finish after you selected the DITA files under the specified directory. The DITA files are then imported to your DITA project.

5. Right-click `build.xml`, select Run As, and then ANT Build.

   After the transformation, the output is in the output directory set in `build.xml`. Refresh the project after the build is successful.



6. **Edit the plug-in description of the property file MANITEST.MF in the plug-in editor after you run the Ant build successfully.**
   a) Click MANITEST.MF to go to the Overview page.



Overview

General Information

This section describes general information about this plug-in:

ID:	test2
Version:	1.0.0
Name:	Test2 Plug-in
Provider:	
Class:	test2.Test2Plugin
Platform filter:	

Plug-in Content

The content of the plug-in is made up of four sections:

- Dependencies: lists all the plug-ins required on this plug-in's classpath to compile and run.
- Runtime: lists the libraries that make up this plug-in's runtime.
- Extensions: declares contributions this plug-in makes to the platform.
- Extension Points: declares new function points this plug-in adds to the platform.

Testing

Test this plug-in by launching a separate Eclipse application:

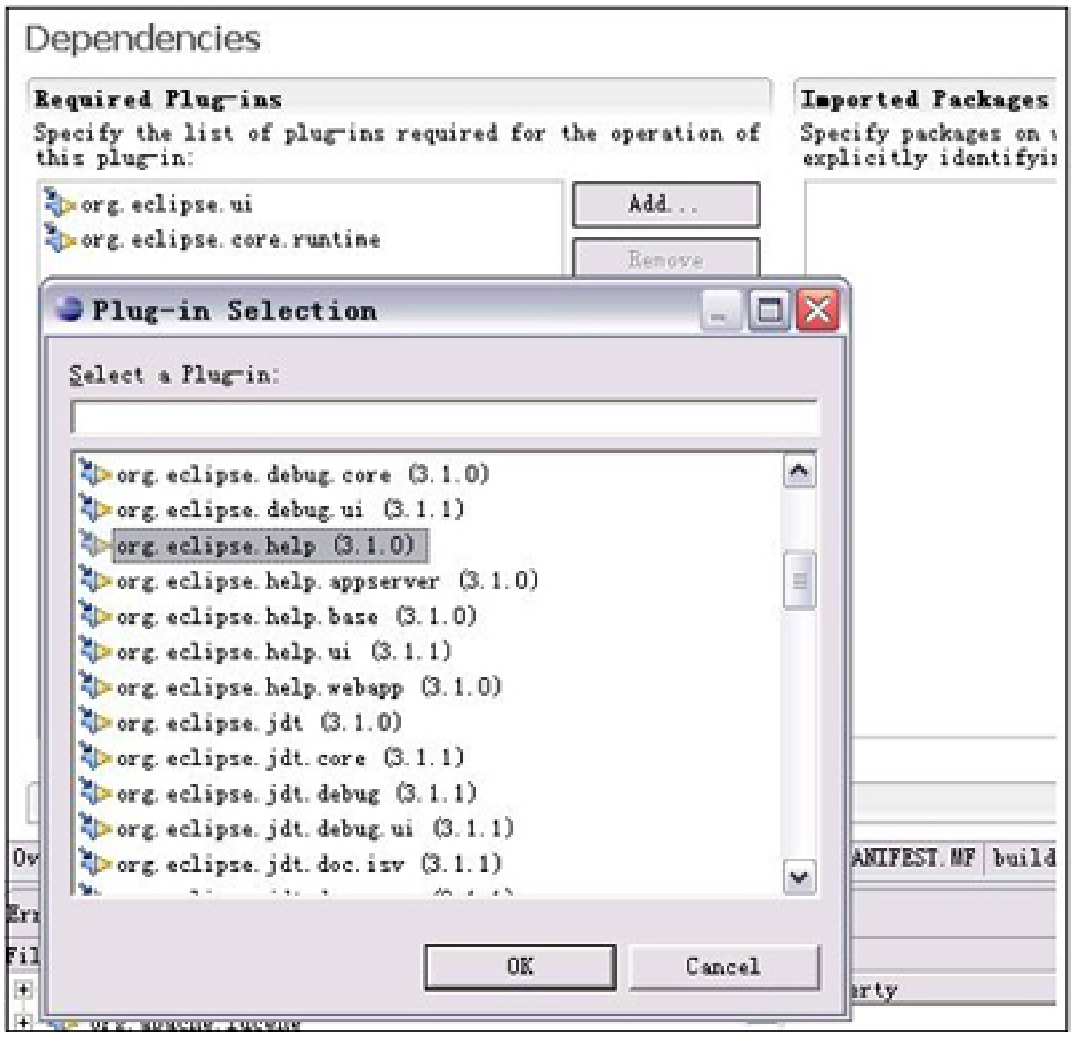
- Launch an Eclipse application
- Launch an Eclipse application in Debug mode

Exporting

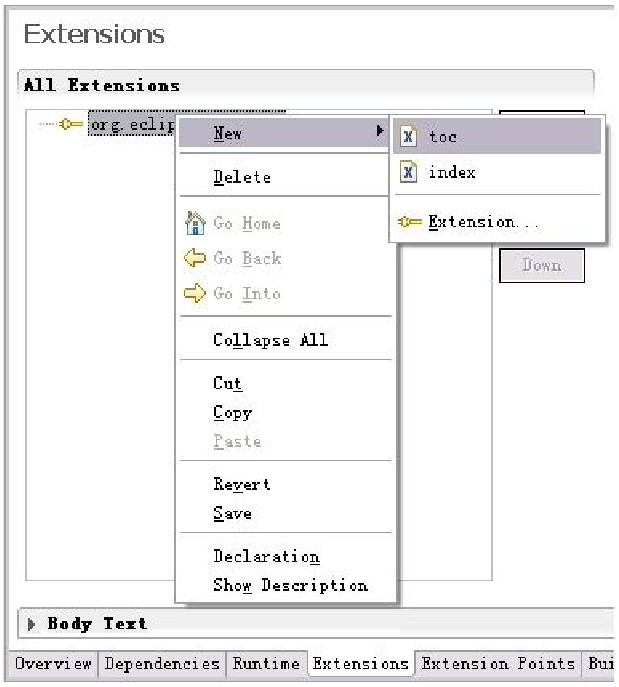
To package and export the plug-in:

1. Specify what needs to be packaged in the deployable plug-in on the Build Configuration page
2. Export the plug-in in a format suitable for deployment using the Export Wizard

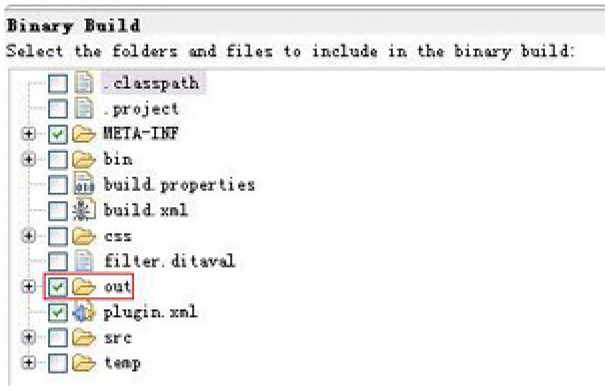
   b) Edit Dependencies to include `org.eclipse.help`


Dependencies
Required Plug-ins
Specify the list of plug-ins required for the operation of this plug-in:
org.eclipse.ui
org.eclipse.core.runtime
Add...
Remove
Imported Packages
Plug-in Selection
Select a Plug-in:
org.eclipse.debug.core (3.1.0)
org.eclipse.debug.ui (3.1.1)
org.eclipse.help (3.1.0)
org.eclipse.help.appserver (3.1.0)
org.eclipse.help.base (3.1.0)
org.eclipse.help.ui (3.1.1)
org.eclipse.help.webapp (3.1.0)
org.eclipse.jdt (3.1.0)
org.eclipse.jdt.core (3.1.1)
org.eclipse.jdt.debug (3.1.1)
org.eclipse.jdt.debug.ui (3.1.1)
org.eclipse.jdt.doc.isv (3.1.1)
OK
Cancel

c) Edit Extensions to add `org.eclipse.help.toc`; right-click the added `prgeclipse.help.toc`, and select New, and then toc.



d) Edit the Build Configuration to include the `out` directory or the directory you specified above.

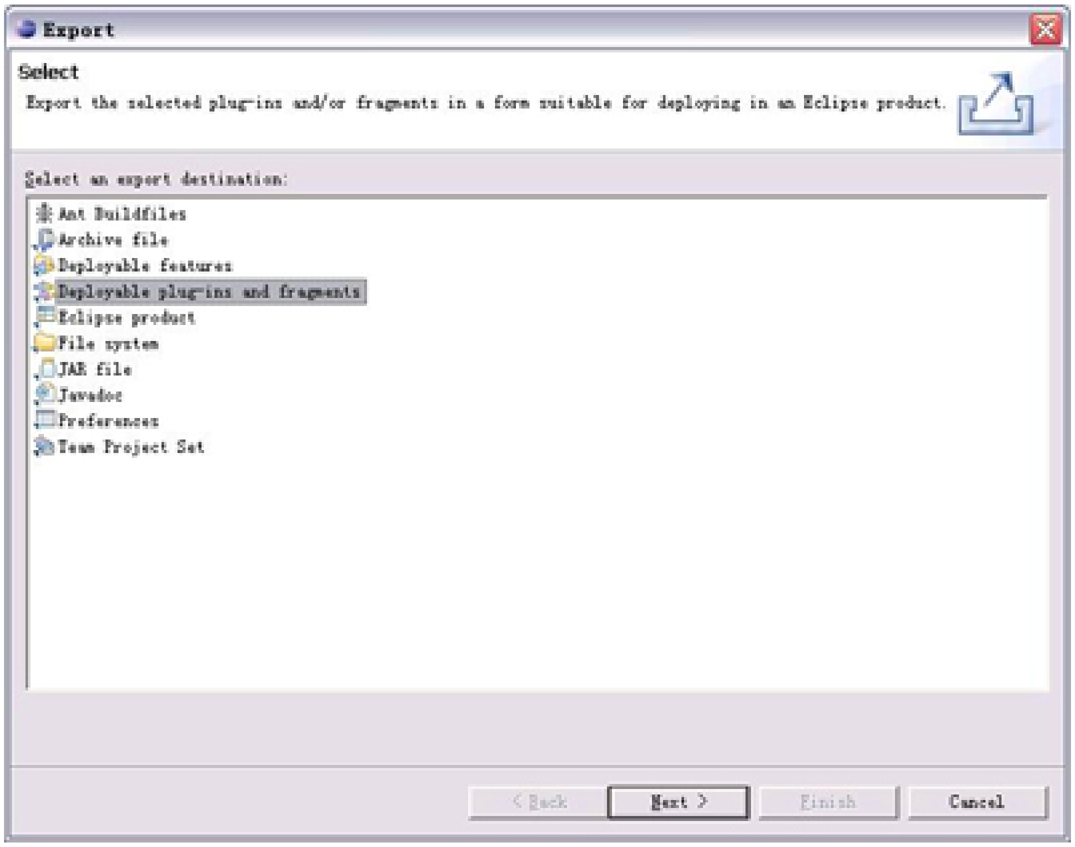


e) Save the changes you made to the property file MANITEST.MF.

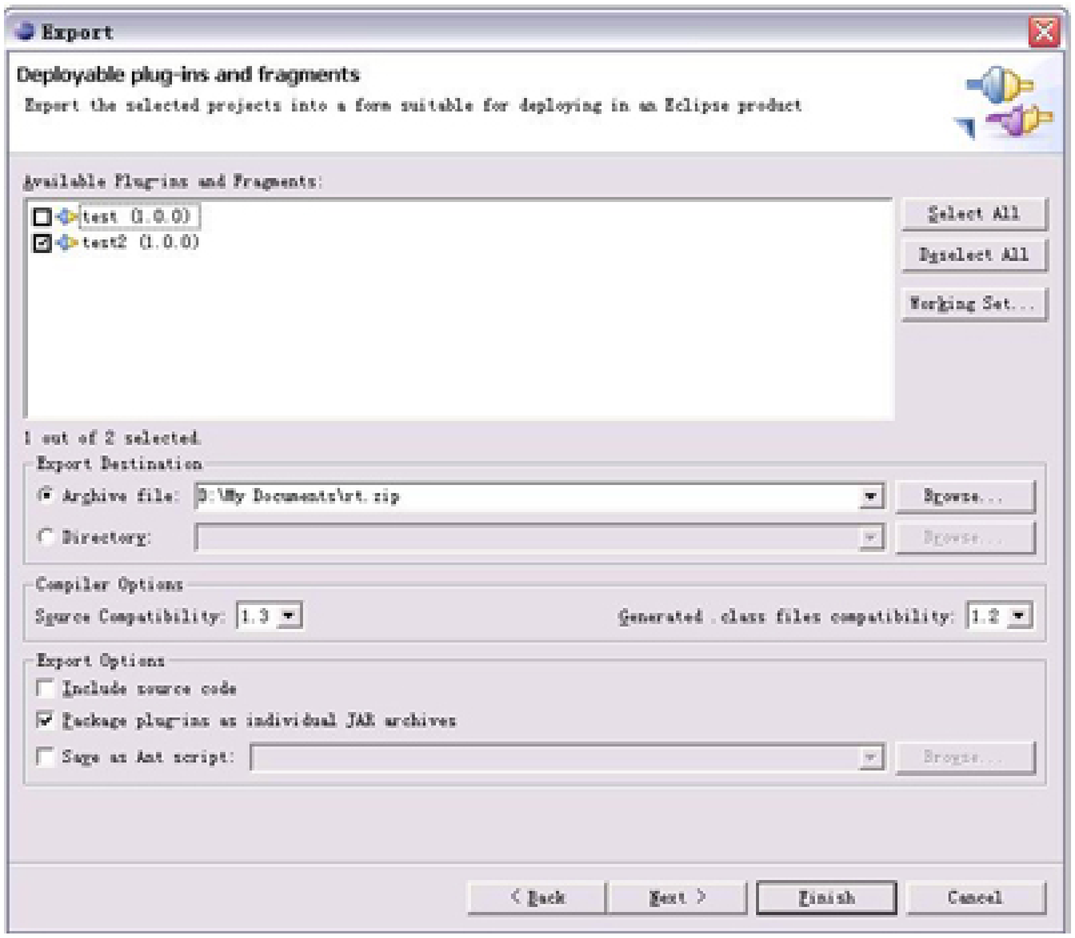
7. Export the output to a documentation plug-in.

Note: `build.xml` can be customized to meet the requirement of headless build.

a) Select **File ➤ Export** ; select "Deployable plug-ins and fragments" and click **Next**.


Export
Select
Export the selected plug-ins and/or fragments in a form suitable for deploying in an Eclipse product.
Select an export destination:
Ant Buildfiles
Archive file
Deployable features
Deployable plug-ins and fragments
Eclipse product
File system
JAR file
Javadoc
Preferences
Team Project Set
< Back
Next >
Finish
Cancel

b) Select the plug-in you want to export and specify a directory under which you want to put the plug-in package.


Export

Deployable plug-ins and fragments

Export the selected projects into a form suitable for deploying in an Eclipse product

Available Plug-ins and Fragments:

- ☐ test (1.0.0)
- ☑ test2 (1.0.0)

1 out of 2 selected.

Export Destination

- ● Archive file: D:\My Documents\rt.zip
- ○ Directory:

Compiler Options

Source Compatibility: 1.3 Generated .class files compatibility: 1.2

Export Options

- ☐ Include source code
- ☑ Package plug-ins as individual JAR archives
- ☐ Save as Ant script:

c) Click **Finish** to export the plug-in package.

Processing to Eclipse help targets using Eclipse

This topic assumes you are already familiar with the Eclipse environment and know how to develop Eclipse plug-ins.

This topic assumes you want to author DITA topics making up an Eclipse help plug-in using Eclipse to both edit and build the content.

1. Start Eclipse and create a new Eclipse plug-in development project.
2. Apply the DITA template to the new project. This causes the template wizard to start.
3. In the wizard, set:

   - the source directory
   - the DITA map file
   - the output directory
   - the CSS directory
   - the name of any customized CSS file
   - the name of any ditaval file

   The main DITA map will be created in the source directory and `build.xml` will be created in the root directory.

4. Edit one or more DITA topic files in the source directory.
5. Update the DITA map file to include the topics created.
6. Select `build.xml` and run it with Ant.
   The DITA output should now be in the output directory.
7. Edit the plug-in description in the plug-in editor, referring to the TOC files generated in the output directory.
8. Update the build property file to include all the output files.
9. Export the output files to a document plug-in.

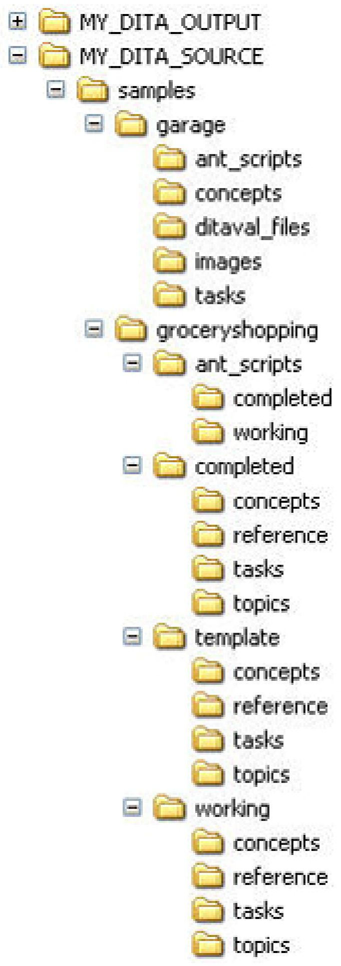
Processing to JavaHelp targets

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 *(Optional) Installing JavaHelp on Windows* on page 52 for information on how to do this.

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:



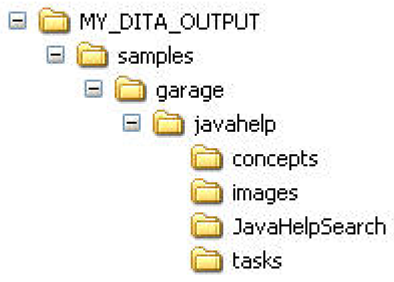
3. View and edit, if necessary for your specific working environment, `MY_DITA_SOURCE/ant_scripts/garage_hierarchy_javahelp.xml`.

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

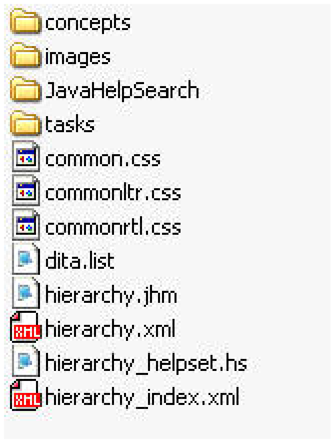
   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to javahelp.

4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_javahelp.xml`
5. After the JavaHelp file has processed successfully, go to the `MY_DITA_OUTPUT` directory.

   Your directory structure should look like this:

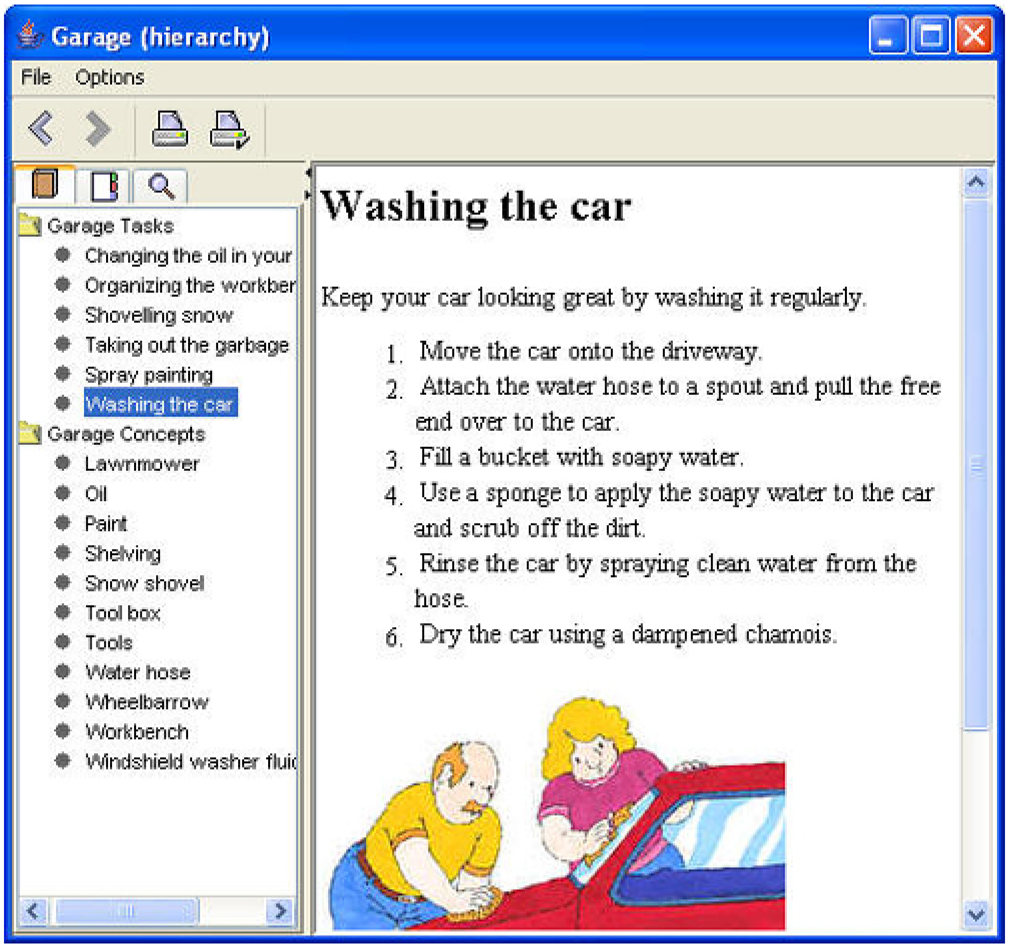


   The `javahelp` subdirectory should contain these directories and files:


concepts
images
JavaHelpSearch
tasks
common.css
commonltr.css
commonrtl.css
dita.list
hierarchy.jhm
hierarchy.xml
hierarchy_helpset.hs
hierarchy_index.xml

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:



Garage Tasks
- Changing the oil in your
- Organizing the workber
- 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 fluic

Washing the car

Keep your car looking great by washing it regularly.

1. Move the car onto the driveway.
2. Attach the water hose to a spout and pull the free end over to the car.
3. Fill a bucket with soapy water.
4. Use a sponge to apply the soapy water to the car and scrub off the dirt.
5. Rinse the car by spraying clean water from the hose.
6. Dry the car using a dampened chamois.

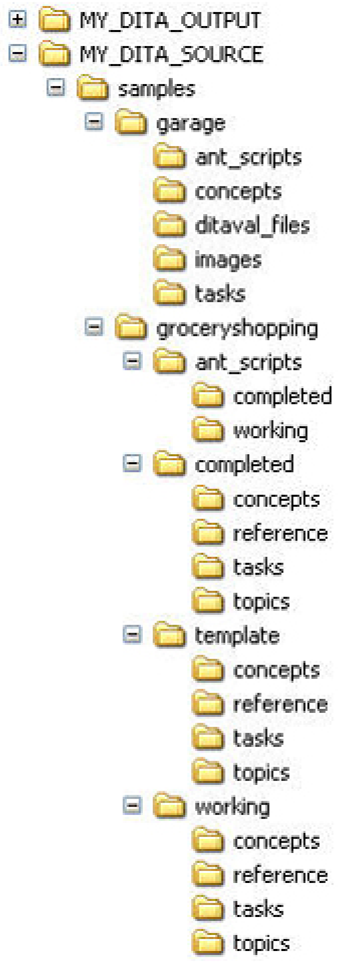
Processing to troff targets

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.

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:

   ```
   MY_DITA_OUTPUT
   MY_DITA_SOURCE
     samples
       garage
         ant_scripts
         concepts
         ditaval_files
         images
         tasks
       groceryshopping
         ant_scripts
           completed
           working
         completed
           concepts
           reference
           tasks
           topics
         template
           concepts
           reference
           tasks
           topics
         working
           concepts
           reference
           tasks
           topics
   ```
3. View and edit, if necessary for your specific working environment, `MY_DITA_SOURCE/ant_scripts/garage_hierarchy_troff.xml`.

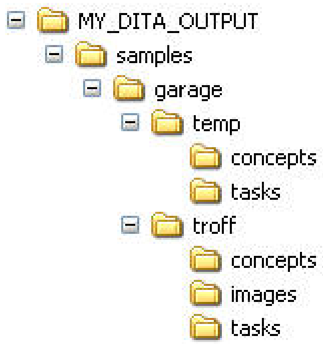
   These are the significant Ant parameters and how they are set:

   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to troff.
4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.

For example: `ant -f garage_hierarchy_troff.xml`

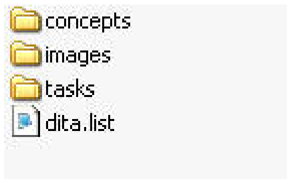
5. After the troff file has processed successfully, go to the `MY_DITA_OUTPUT` directory.

   Your directory structure should look like this:



```
MY_DITA_OUTPUT
    samples
        garage
            temp
                concepts
                tasks
            troff
                concepts
                images
                tasks
```

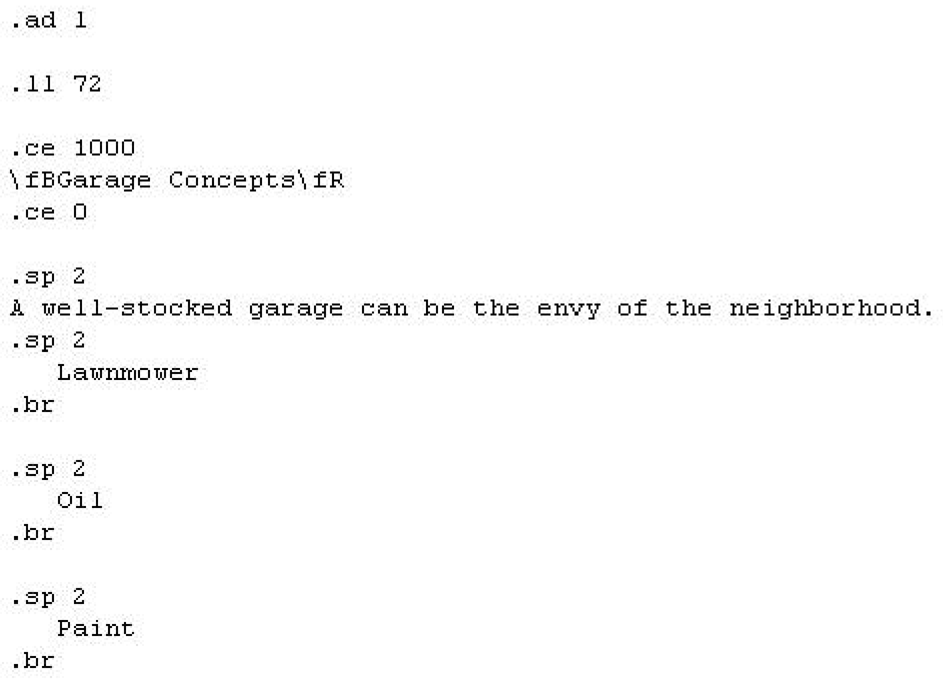
   The `troff` subdirectory should contain these directories and files:



```
concepts
images
tasks
dita.list
```

6. Open the file `concepts/garageconcepts.cli` to view the troff output.

   The file should look something like this:



```
.ad l

.ll 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

.sp 2
   Paint
.br
```

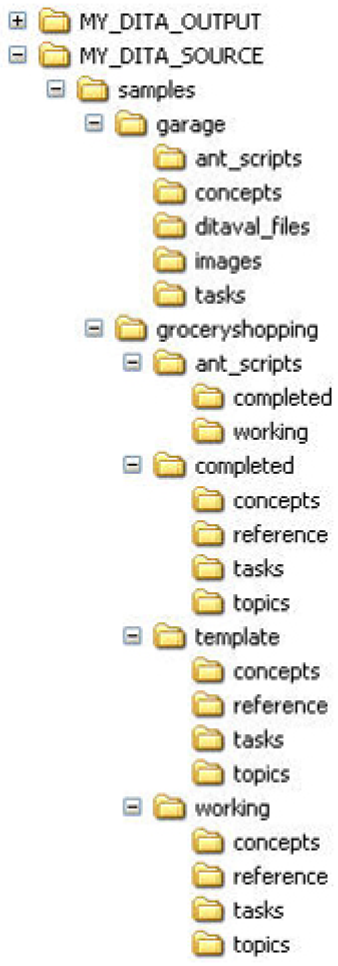
Processing to Word RTF targets

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.

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 `MY_DITA_OUTPUT` and `MY_DITA_SOURCE` 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 `MY_DITA_SOURCE` directory.

   You should have a directory structure that looks like this:



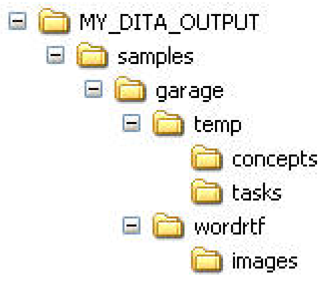
```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
                completed
                working
            completed
                concepts
                reference
                tasks
                topics
            template
                concepts
                reference
                tasks
                topics
            working
                concepts
                reference
                tasks
                topics
```

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

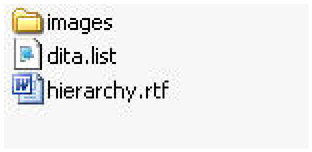
   These are the significant Ant parameters and how they are set:

   - **args.outdir** is set to send the output to `MY_DITA_OUTPUT`.
   - **args.input** is set to use the hierarchy ditamap.
   - **transtype** is set to wordrtf.

4. In the Command Prompt, move to `MY_DITA_SOURCE/samples/garage/ant_scripts` and invoke the above Ant script.
   For example: `ant -f garage_hierarchy_wordrtf.xml`
5. After the Word RTF file has processed successfully, go to the `MY_DITA_OUTPUT` directory.
   Your directory structure should look like this:

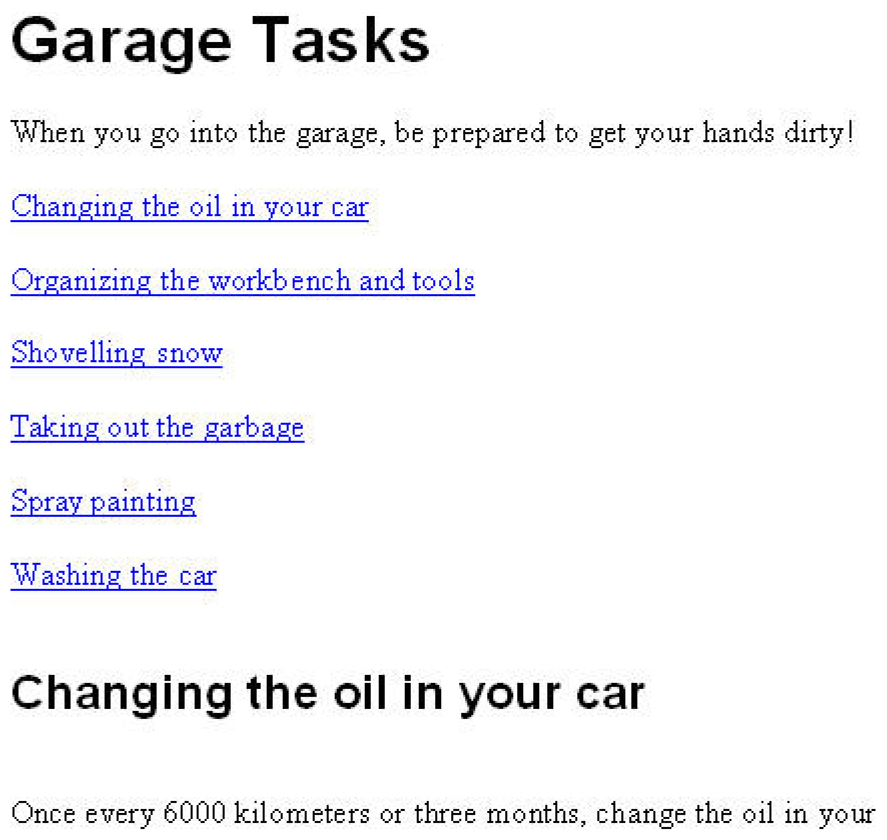


The `wordrtf` subdirectory should contain these directories and files:



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

Taking out the garbage

Spray painting

Washing the car

Changing the oil in your car

Once every 6000 kilometers or three months, change the oil in your

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 69. 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.

Java parameter	Equivalent Ant script parameter
/artlbl	args.artlbl
/basedir	basedir
/cleantemp	clean.temp
/copycss	args.copycss
/css	args.css
/cssroot	args.cssroot
/csspath	args.csspath
/ditalocale	args.dita.locale
/draft	args.draft
/ditadir	dita.dir
/ditaext	dita.extname
/eclipsecontenttoc	args.eclipsecontent.toc
/eclipsehelptoc	args.eclipsehelp.toc
/filter	args.input.valfile
/fouserconfig	args.fo.userconfig
/foimgext	args.fo.img.ext
/fooutputrellinks	args.fo.output.rel.links
/ftr	args.ftr

Java parameter	**Equivalent Ant script parameter**
/hdf	args.hdf
/hdr	args.hdr
/htmlhelpincludefile	args.htmlhelp.includefile
/i	args.input
/id	dita.input.dirname
/if	dita.input
/indexshow	args.indexshow
/javahelpmap	args.javahelp.map
/javahelptoc	args.javahelp.toc
/logdir	args.logdir
/outdir	args.outdir
/outext	args.outext
/provider	args.eclipse.provider
/tempdir	dita.temp.dir
/transtype	transtype
/version	args.eclipse.version
/xhtmltoc	args.xhtml.toc
/xsl	args.xsl

Production notes (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 "transtypes" 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. These lines at the beginning of the script show how this is done:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- (c) Copyright IBM Corp. 2004, 2005 All Rights Reserved. -->
<project name="toolkitug" default="all" basedir="C:/ditaot">

 <!-- Location of project working files -->
 <property name="projdir" value="C:/DITAOT_UGRef_SOURCE"/>
 <property name="outdir" value="C:/DITAOT_UGRef_OUTPUT"/>
 <property name="args.logdir" value="${outdir}"/>
```

In the Windows operating environment we built to all transtypes 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 -f ant_scripts\DITAOT_UGRef_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

This section contains information on how to troubleshoot 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.

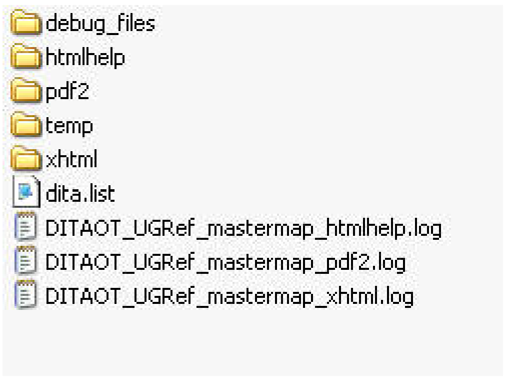
Sections in this topic:

Capturing and using the log

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.

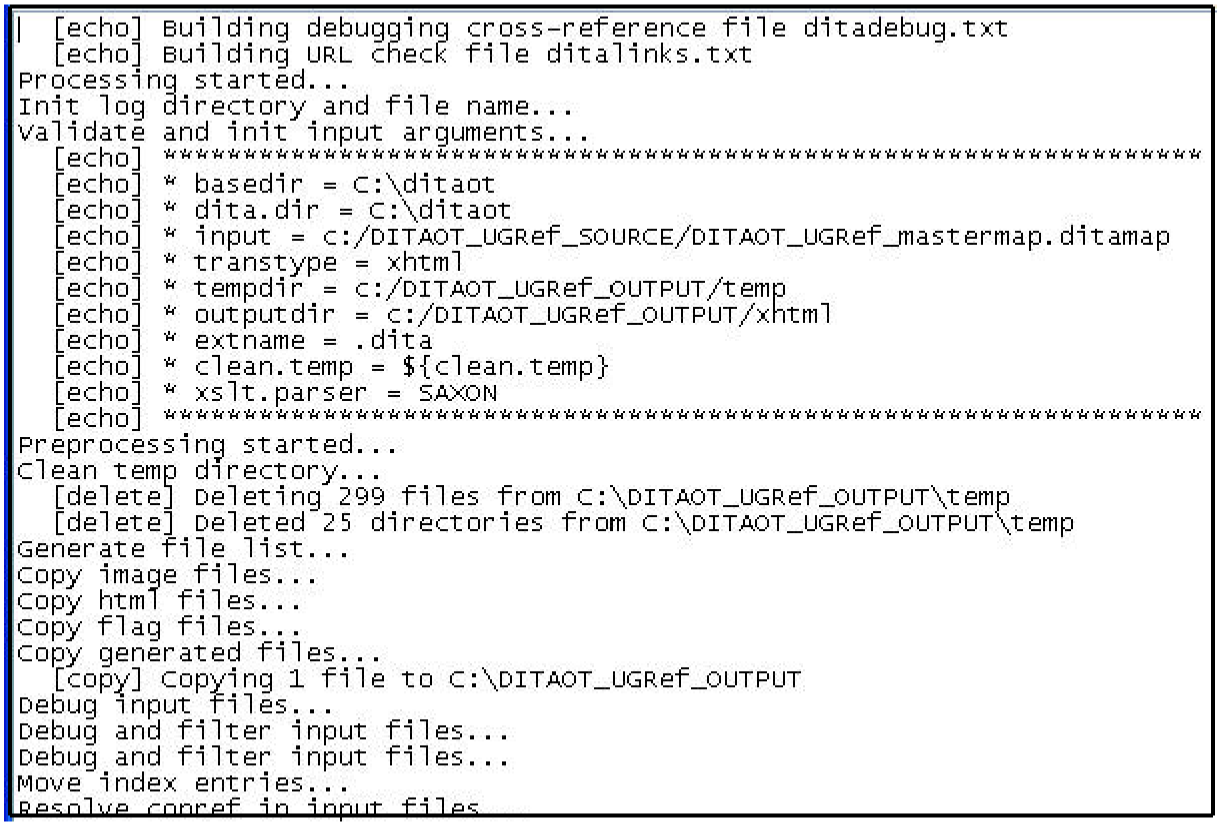
```
ant -f ant\sample_xhtml.xml -quiet -l antoutput.log
```

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.



debug_files
htmlhelp
pdf2
temp
xhtml
dita.list
DITAOT_UGRef_mastermap_htmlhelp.log
DITAOT_UGRef_mastermap_pdf2.log
DITAOT_UGRef_mastermap_xhtml.log

Here is the beginning of the xhtml log file.



```
   [echo] Building debugging cross-reference file ditadebug.txt
   [echo] Building URL check file ditalinks.txt
Processing started...
Init log directory and file name...
Validate and init input arguments...
   [echo] *****************************************************************
   [echo] * basedir = C:\ditaot
   [echo] * dita.dir = C:\ditaot
   [echo] * input = c:/DITAOT_UGRef_SOURCE/DITAOT_UGRef_mastermap.ditamap
   [echo] * transtype = xhtml
   [echo] * tempdir = c:/DITAOT_UGRef_OUTPUT/temp
   [echo] * outputdir = c:/DITAOT_UGRef_OUTPUT/xhtml
   [echo] * extname = .dita
   [echo] * clean.temp = ${clean.temp}
   [echo] * xslt.parser = SAXON
   [echo] *****************************************************************
Preprocessing started...
Clean temp directory...
   [delete] Deleting 299 files from C:\DITAOT_UGRef_OUTPUT\temp
   [delete] Deleted 25 directories from C:\DITAOT_UGRef_OUTPUT\temp
Generate file list...
Copy image files...
Copy html files...
Copy flag files...
Copy generated files...
   [copy] Copying 1 file to C:\DITAOT_UGRef_OUTPUT
Debug input files...
Debug and filter input files...
Debug and filter input files...
Move index entries...
Resolve conref in input files
```

DITA Open Toolkit error messages overview

For reference information about error messages generated by the Toolkit, see *Messages generated by the Toolkit* on page 107.

For reference information about other messages generated during Toolkit processing, see *Messages generated from other sources* on page 115

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 `lib/dost.jar`, for example, DOTJ008F.
3. Messages beginning with DOTX from the Toolkit XSLT transforms in the `xsl` directory, for example, DOTX009W.

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 DOTX040I of type INFO from an XSLT transform:

```
[xslt] file:/C:/sandbox/xsl/common/output-message.xsl:57:16: Warning!
[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.

Level of severity	Description
INFO	Information about processing, processing continues.
WARN	A possible problem was noted, processing continues.
ERROR	A problem was found, processing continues.
FATAL	A problem was found, processing stops.

Messages generated by the Toolkit

Ant messages

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

Java messages

Message number	Type	Message text	Action
DOTJ001F	FATAL	`Input argument error: no ':' found in the parameter '%1'.`	Please add a colon character ':' between the name and value of the parameter '%1'. For detail information, please refer to User Guide.
DOTJ002F	FATAL	`Unsupported parameter '%1'.`	Please refer to User Guide for supported parameters.
DOTJ003F	FATAL	`Param value can't be null for the parameter '%1'.`	Please provide a value for the parameter '%1'.
DOTJ004F	FATAL	`Can't create temp directory '%1'.`	Please check if you have the right to create the directory '%1'.
DOTJ005F	FATAL	`Failed to create new instance for '%1'.`	Please ensure that '%1' exists and you have right to access it.
DOTJ006F	FATAL	`Invalid value '%1' for attribute 'extparam' of AntInvoker.`	Please use correct way to call AntInvoker, e.g. extparam="maplinks=XXXX;other=YYYY".
DOTJ007E	ERROR	`Duplicate condition in filter file for rule '%1'.`	Check to make sure that there is none duplicate rule specified.
DOTJ008E	ERROR	`MapIndexReader is not used in correct way. matchList is null.`	Please check the code to see whether method setMatch(String matchPattern) is called before read(String filename).
DOTJ009E	ERROR	`Cannot overwrite file '%1' with file '%2'. The modified result may not be consumed by the following steps.`	Check to see whether the file is locked by other application during the transformation process.
DOTJ010E	ERROR	`Can't find %1 in the extparam for index generation.`	Please specify %1 in the extparam.
DOTJ011E	ERROR	`Failed to load the input file '%1' for index generation due to below exception, and no index information generated.`	Please ensure '%1' exists and you have right to access it.
DOTJ012F	FATAL	`Failed to parse the input file '%1' due to below exception.`	Please correct the input base on the exception message.
DOTJ013E	ERROR	`Failed to parse the referenced file '%1' due to below exception.`	Please correct the reference base on the exception message.
DOTJ014W	WARN	`The indexterm element does not have any content. Setting the term to ***.`	Please add content to the indexterm.
DOTJ015F	FATAL	`Log directory can't be null.`	Please specify the correct log directory using ant parameter 'args.logdir'.
DOTJ016F	FATAL	`Failed to create log directory '%1'.`	Please specify the correct log directory using ant parameter 'args.logdir'.

Message number	**Type**	**Message text**	**Action**
DOTJ017F	FATAL	`Failed to init log filename with input file.`	Please specify input file using ant parameter 'args.input'.
DOTJ018I	INFO	`Log file '%1' was generated successfully at directory '%2'.`	You can find the detailed messages from the transformation process in this log file.
DOTJ019E	ERROR	`Failed to generated log file.`	Please correct the errors.
DOTJ020W	WARN	`Plugin '%1' is required by plugin '%2'. Plugin '%2' cannot be loaded because of missing plugin '%1'.`	Check and see whether all of the plugins required are installed in toolkit.
DOTJ021W	WARN	`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.`	Please check the file '%1' and the ditaval file to see if this is the intended result.
DOTJ022F	FATAL	`Failed to parse the input file '%1' due to all of its content has been filtered out.`	Please check the input file '%1' and the ditaval file, and ensure that the input is valid.
DOTJ023E	ERROR	`Failed to get the image file specified '%1' in RTF generation`	Check whether the image exists and copied to output directory before the final step of RTF generation.
DOTJ024W	WARN	`Extension name of picture file '%1' not supported.`	Please convert your picture to JPEG or GIF style.
DOTJ025E	ERROR	`The input of topic merge cannot be found.`	Please check whether the input message for topic merge is absolute path.
DOTJ026E	ERROR	`The output of topic merge is null.`	Please check whether you have set output for topic merge correctly.

XSLT messages

Message number	**Type**	**Message text**	**Action**
DOTX001W	WARN	`No string named '%1' was found for language '%2'. Use the default language '%3'.`	Add the mapping between default language and specific language for the string '%1'.
DOTX002W	WARN	`The title attribute in ditamap is required for Eclipse output.`	Add a title attribute to map element in ditamap file.
DOTX003I	INFO	`The anchorref attribute should either point to another dita map, or to an Eclipse XML file. The value '%1' does not point to either.`	Change the anchorref referring to ditamap or dita topic file.
DOTX004I	INFO	`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.`	Change the navref referring to ditamap or dita topic file.

Message number	**Type**	**Message text**	**Action**
DOTX005E	ERROR	`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'.`	Provide the navigation title in the map or topic files.
DOTX006E	ERROR	`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'.`	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'.
DOTX007I	INFO	`Only DITA topics, HTML files, and images may be included in your compiled CHM file. The reference to "%1" will be ignored.`	To remove this message, you can set the toc="no" attribute on your topicref.
DOTX008W	WARN	`File '%1' does not exist.`	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'.
DOTX009W	WARN	`Could not retrieve a title from '%1'. Using '%2' instead.`	Add a title in the target topic file.
DOTX010E	ERROR	`Unable to find target for conref="%1". 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'.`	Check to make sure the target of conref is correct.
DOTX011W	WARN	`There is more than one possible target for conref="%1". 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'.`	Remove the duplicated id in the conref target file.
DOTX012W	WARN	`When you conref another topic or an item in another topic, the domains attribute of the target topic must be equal to or a subset of the current topic's domains attribute.`	Put your target under an appropriate domain. You can see the messages guide for more help.
DOTX013E	ERROR	`A element with a conref attribute indirectly includes itself, which is not possible. Please fix the target of the conref attribute. The conref`	Resolve the circle conref.

Message number	Type	Message text	Action
		`attribute points to '%1'. Note:'%1' might be changed to use default dita topic file extension name '.dita' or '.xml'.`	
DOTX014E	ERROR	`The element must provide the id of the target topicref you want to reuse. For example, mymap.ditamap#mytopicrefid.`	Put an id into the target position of the content which will be reused.
DOTX015E	ERROR	`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'.`	Change the conref attribute to conform the correct syntax.
DOTX016W	WARN	`An href value appears to point to a DITA file, but the format attribute is inherited a value of "%1". If the target '%2'is a DITA file, set the format attribute to "dita". 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'.`	Directly set the format attribute to correct value instead of inheriting from ancestor.
DOTX017E	ERROR	`Found a topic reference with an empty HREF attribute. The attribute should point to a file or other valid address.`	Remove the empty href link or put in some content.
DOTX018I	INFO	`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.`	Check and make the type of topicref match with the actual type of topic.
DOTX019W	WARN	`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.`	Check and make the type of topicref match with the actual type of topic.
DOTX020E	ERROR	`Missing navtitle attribute for: '%1' When you set scope="peer" you must provide a local navigation title, since the target is not accessible at build time.`	Provide a local navigation title in ditamap.

Message number	**Type**	**Message text**	**Action**
DOTX021E	ERROR	`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.`	Provide a local navigation title in ditamap.
DOTX022W	WARN	`Unable to retrieve navtitle from target: '%1'. Using linktext under topicmeta of this topicref instead of the navigation title.`	Make sure the topicref type matches the target, and that the file name and topic id are correct.
DOTX023W	WARN	`Unable to retrieve navtitle from target: '%1'.`	Make sure the topicref type matches the target, and that the file name and topic id are correct.
DOTX024E	ERROR	`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.`	Provide a navtitle attribute for topicref.
DOTX025E	ERROR	`Missing linktext and navtitle attribute 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.`	Provide a navtitle attribute for topicref.
DOTX026W	WARN	`Unable to retrieve linktext from target: '%1'. Using navtitle instead of linktext under topicmeta.`	Make sure the topicref type matches the target, and that the file name and topic id are correct.
DOTX027W	WARN	`Unable to retrieve linktext from target: '%1'.`	Make sure the topicref type matches the target, and that the file name and topic id are correct.
DOTX028E	ERROR	`Link or Xref must contain an unempty href attribute.`	Add an unempty href attribute for link or xref element.
DOTX029I	INFO	`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.`	Check and make the type of element match with the actual type of target.
DOTX030W	WARN	`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`	Check and make the type of element match with the actual type of target.

Message number	Type	Message text	Action
		`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.`	
DOTX031E	ERROR	`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.`	Check to make sure the file exists and DITAVAL file isn't used to remove the contents of the file.
DOTX032E	ERROR	`Unable to retrieve link text from target: '%1'.`	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, or does not have a title, give the link text as content of the xref element.
DOTX033E	ERROR	`Unable to find the target to determine the list item number.`	Make sure the link type matches the target, and that the ids for topic and element are correct.
DOTX034E	ERROR	`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.`	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.
DOTX035E	ERROR	`Unable to find the target to determine the footnote number.`	Make sure the link type matches the target, and that the ids for topic and element are correct.
DOTX036E	ERROR	`Unable to find the dlentry target to determine the dlterm text.`	Make sure the link type matches the target, and that the ids for topic and element are correct.
DOTX037W	WARN	`Topic contains no title; using "***".`	Add a title to your topic.
DOTX038I	INFO	`The LONGDESCREF attribute on tag '%1' will be ignored. Accessibility needs to be handled another way.`	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 <param> element inside the object.
DOTX039W	WARN	`Required cleanup 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.
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.
DOTX041W	WARN	`More than one title element in a section. Using the first one for the section's title.`	Make sure that the title you wish to appear as a title is the first one. Then remove other title elements, or change

Message number	Type	Message text	Action
			them to a more appropriate element. As a last resort you could simply make them bold phrases.
DOTX042I	INFO	`Flagging attribute found on '%1' attribute. Inline phrases cannot be flagged.`	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.
DOTX043I	INFO	`The link to '%1' may appear more than once in '%2'.`	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 a <reltable> in a DITA Map will have the role attribute set to friend.
DOTX044E	ERROR	`Area element has no cross-reference HREF attribute. The area requires a cross-reference with an HREF attribute.`	Add a href attribute to the xref element.
DOTX045W	WARN	`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.`	Add link text to the xref element.
DOTX046W	WARN	`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.`	Correct the shape value.
DOTX047W	WARN	`Area coordinates are blank. Coordinate points for the shape need to be specified.`	Correct the coords value.
DOTX048I	INFO	`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.`	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 "external" 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 "no" on your topicref element.
DOTX049I	INFO	`Topicref to non-dita files will be ignored in PDF or Word transformation.`	If you want to include these files in PDF or Word output, you may need to change them to dita format.

Message number	Type	Message text	Action
DOTX050W	WARN	`Cannot find id attribute in map element.`	Default id "org.sample.help.doc" is used for plugin. If you want to use your own id, please specify it in id attribute of map.
DOTX051W	WARN	`The %1 to '%2' with @format='%3' can not be recognized, output it without creating a hyperlink.`	Set @format='dita' for %1 to DITA topic, set @format='html' for HTML files, other format is not supported.
DOTX052W	WARN	`No string named '%1' was found. Using original value.`	Add translation for the string '%1' in the default language, then add mapping between default language and other languages for it.

Messages generated from other sources

Other Java messages

Message text	Action
`[xslt] : Warning! Failure reading file:... Cause: java.io.EOFException: no more input [xslt]`	May occur with Toolkit messages. Check for an invalid file path in the input.
`[pipeline] [Error] :13:39: Element type "..." must be declared.`	An error has occurred parsing a DTD.
`[pipeline] [Error] :14:13: The content of element type "..." must match "...".`	An error has occurred parsing a DTD.
`BUILD FAILED C:\sandbox\ant\dotug_xhtml.xml:24: The following error occurred while executing this line: C:\sandbox\conductor.xml:101: The following error occurred while executing this line: java.lang.OutOfMemoryError`	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.)
`Unable to instantiate specified logger class org.dita.log.DITAOTBuildLogger ...`	Check that your CLASSPATH variable contains dost.jar.
`Can't find resource\messages.xml`	Check that your CLASSPATH variable contains dost.jar.

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.

About the debugging, reporting, and file generation tools

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 107). 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.

For more information about these tools, see *Using the debugging and reporting tools* and *Production notes (troubleshooting)* on page 118.

Using the debugging and reporting tools

List and descriptions of the debugging and reporting tools

These tools (introduced in *About the debugging, reporting, and file generation tools* on page 116) 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. The tools all take a single argument, which is the name of a DITA map file. The tools process all files in the hierarchy below the level of the invoked DITA map.

Tool	Description
ditaauthors.php	Returns a set of unique author and copyright strings, with counts for each.
ditadebug.php	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: • A list of any incorrect references found in the source files • A list of directories containing files used by the map • For each file in the map, its name, type, author, size, date last modified, and title • The total number of files by type • A list of all references found
ditaedit.php	Can be used to search for and replace strings in all files in the DITA map..
ditaids.php	Produces an alphabetical list of IDs for all files in the map, with duplicates marked with an asterisk.
ditakeys.php	Produces a list of all metadata keywords defined.
ditalinks.php	Tests all URLs referenced by files in the map for validity.

Software prequisites 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.

```
<!-- 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>
```

Production notes (troubleshooting)

Sections in this topic:

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\nstalling_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:\DITAOT_UGRef_SOURCE\processing\../installing/nstalling_fo.dita does not
exist!
Bad reference: C:\DITAOT_UGRef_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 DITAOT_UGRef_mastermap.ditamap
dir: .\ file: DITAOT_UGRef_mastermap.ditamap

6 unused files in directories used by this map:

C:\DITAOT_UGRef_SOURCE\images\dita_finch_logo.jpg , *No DOCTYPE*
...
C:\DITAOT_UGRef_SOURCE\ugxref.txt , *No DOCTYPE*

27 directories in this map:

C:\DITAOT_UGRef_SOURCE
C:\DITAOT_UGRef_SOURCE\accessing
...
C:\DITAOT_UGRef_SOURCE\troubleshooting

349 files and links in this map:

C:\DITAOT_UGRef_SOURCE\DITAOT_UGRef_bkinfo.dita , bkinfo
C:\DITAOT_UGRef_SOURCE\DITAOT_UGRef_mastermap.ditamap , bookmap
...
https://sourceforge.net/projects/dita-ot , *https*

493 references in this map:

DITAOT_UGRef_mastermap.ditamap , bkinfo ,
C:\DITAOT_UGRef_SOURCE\DITAOT_UGRef_bkinfo.dita
DITAOT_UGRef_mastermap.ditamap , chapter ,
C:\DITAOT_UGRef_SOURCE\release_current\relcurrent_map.ditamap
...
C:\DITAOT_UGRef_SOURCE\introduction\aboutdita.dita , conref ,
C:\DITAOT_UGRef_SOURCE\core_vocabulary\darwininfo_typingarch.dita
#darwininfo_typingarch/darwininfo_typingarch_term
C:\DITAOT_UGRef_SOURCE\core_vocabulary\ditaot.dita#ditaot/ditaot_term
...
C:\DITAOT_UGRef_SOURCE\core_vocabulary\xalan.dita , xref ,
C:\DITAOT_UGRef_SOURCE\release_current\sysreqs.dita
C:\DITAOT_UGRef_SOURCE\core_vocabulary\xalan.dita , xref ,
http://archive.apache.org/dist/xml/xalan-j/
```

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 (*Messages generated by the Toolkit* on page 107). This reference topic has been generated multiple times during the production cycle of the *DITA Open Toolkit User Guide and Reference*, as the Toolkit moved to new point releases.

```
<?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>

<!-- 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>

</refbody>
</reference>

</xsl:template>

<!-- 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>

<xsl:template match="link">
<a>
<xsl:attribute name="href">
<xsl:value-of select="."/>
</xsl:attribute>
Item link</a>
</xsl:template>

</xsl:stylesheet>
```

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).

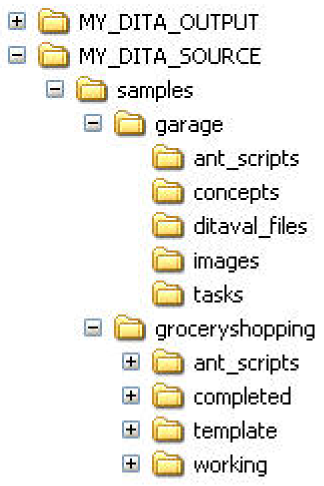
This chapter and the next one are a short tutorial; you should work through the topics chapter (this one) before the maps chapter. The key concepts and tasks in this chapter are meant to be read and performed in the order shown below.

Sections in this topic:

About the grocery shopping sample

The grocery shopping sample, which is located in the `ditaot/doc/ot-userguide/MY_DITA_SOURCE/samples` directory of the Toolkit source distribution, 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 the XHTML, HTML Help, and PDF2 targets are also provided.

Before you begin to use the sample files (which include both the garage sample and the grocery shopping sample), we recommend creating two directories in your root directory called `MY_DITA_SOURCE` and `MY_DITA_OUTPUT` (Windows examples would be `C:/MY_DITA_SOURCE` and `C:/MY_DITA_OUTPUT`) and then copying both the garage and grocery shopping sample files from the `ditaot/samples` directory to `MY_DITA_SOURCE`. Your directory structure should then look like this:

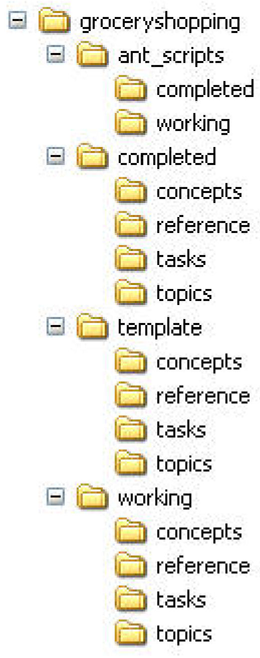


The grocery shopping sample assumes you are already familiar with the garage sample provided as both a project model as well as a tool to verify your DITA Open Toolkit installation, and that you have processed the garage sample as described in *Processing (building) and publishing DITA documents* on page 65.

Creating and processing the grocery shopping sample

You will be working in the `MY_DITA_SOURCE/samples/groceryshopping` directory, which contains a number of subdirectories.

Your directory structure should look like this:



Files in the `template` directory 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 could modify to try and test other DITA features.

About topics

Topic is the base DITA information type.

Creating topics

In this topic you will create a simple DITA topic based on a template already provided. You will be working in the `MY_DITA_SOURCE/samples/groceryshopping` directories. This topic assumes you are familiar with the information in *About the grocery shopping sample* on page 123.

1. Go to the `groceryshopping/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:

   ```
   <topic id="groceryshopping" xml:lang="en-us">
   <title>Shopping for groceries</title>
   <shortdesc>Tips on buying groceries.</shortdesc>
   <prolog>
   <author type="creator">Tom McIntyre</author>
   <copyright>
   <copyryear year="2006"/>
   <copyrholder>Acme Company</copyrholder>
   </copyright>
   <critdates>
   <created date="2006-August-07"/>
   <revised modified="2006-August-08"/>
   </critdates>
   <metadata>
   <keywords>
   <keyword>grocery shopping</keyword>
   </keywords>
   </metadata>
   </prolog>
   <body>
   <!-- This is a container topic—it has no body content. -->
   <!-- Do not type anything here. -->
   <p/>
   </body>
   </topic>
   ```

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 `MY_DITA_SOURCE/samples/groceryshopping` directories. This topic assumes you are familiar with the information in *About the grocery shopping sample* on page 123.

1. Go to the `groceryshopping/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 `groceryshopping/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.</p>
    <p>You can save money by buying produce when it is in season.</p>
    ```

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 `MY_DITA_SOURCE/samples/groceryshopping` directories. This topic assumes you are familiar with the information in *About the grocery shopping sample* on page 123.

1. Go to the `groceryshopping/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>
<step>
<cmd>Put the can in your cart, finish your shopping,
and check out.</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:

```
<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>
<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 `MY_DITA_SOURCE/samples/groceryshopping` directories. This topic assumes you are familiar with the information in *About the grocery shopping sample* on page 123.

1. Go to the `groceryshopping/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:

```
<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 drivers.

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.

Production notes (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*.

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

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 version="1.0" encoding="utf-8"?>
<!-- (c) Copyright 2006 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="css.dita"/>
<topicref href="customizing_production_notes.dita"/>
<topicref href="customizing_formoreinfo.dita"/>
</topicref>
</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 `MY_DITA_SOURCE/samples/groceryshopping` directories. This topic assumes you are familiar with the information in *About the grocery shopping sample* on page 123, and that you have created the topics according to the instructions in *Creating DITA topics* on page 123.

1. Go to the `groceryshopping/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 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:

```
<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:

```
<!-- 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>
```

Because we have concept, task, and reference information for each "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 131. You will be working in the `MY_DITA_SOURCE/samples/groceryshopping/ant_scripts` 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 123, and that you have created the topics according to the instructions in *Creating DITA topics* on page 123. If you need more information about Ant or Ant scripts, see *About Ant* on page 66 or *About Ant scripts* on page 67.

1. Go to the `groceryshopping/ant_scripts/working` directory.
2. Using your DITA authoring tool or a plain text editor, open the version of the Ant script you want to run. You can process to one of three target environments: XHTML, HTML Help, or PDF2.
3. Make sure the Ant script is set up correctly for your environment.
4. In the Command Prompt, navigate to the `ant_scripts/working` directory.
5. Do one of the following:

   To build XHTML output for the grocery shopping sample, enter the command `ant -f grocery_xhtml.xml`.

   To build HTML Help output for the grocery shopping sample, enter the command `ant -f grocery_htmlhelp.xml`.

   To build PDF2 output for the grocery shopping sample, enter the command `ant -f grocery_pdf2.xml`.
6. Check the output directory to be sure the output files are correct.

Production notes (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.

Linking content

This section contains information on how to link DITA topics using cross-references (xrefs), related links, and relationship tables.

Sections in this topic:

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 <xref> and <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.

```
<strow>
<stentry>
<xref href="../release_current/release_current.dita">Release 1.3 information</xref>
</stentry>
<stentry>Information about release 1.3
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.

```
<p>Sections in this topic:
<sl>
<sli>
<xref href="#aboutditaotugref/contents">
Document contents</xref>
</sli>
<sli>
<xref href="#aboutditaotugref/audience">
Audience</xref>
</sli>
<sli>
<xref href="#aboutditaotugref/prerequisites">
Prerequisites</xref>
</sli>
```

```
<sli>
<xref href="#aboutditaotugref/howproduced">
How this document was produced</xref>
</sli>
</sl>
</p>
```

The following example from this document uses an xref to reference an external website.

```
<dl>
<dlentry>
<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>
</dlentry>
</dl>
```

Linking using related links

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

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

Linking using relationship tables

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

```
<map id="reltables_map" title="Relationship tables">
<!-- Relationship table -->
<reltable>
<relrow>
<relcell>
<topicgroup collection-type="family">
<topicref href="../release_current/sysreqs.dita"/>
<topicref href="../installing/aboutinstalling.dita"/>
</topicgroup>
</relcell>
<relcell/>
</relrow>
<relrow>
<relcell>
<topicref href="../introduction/aboutditaot.dita"/>
</relcell>
<relcell>
<topicref href="../core_vocabulary/docbook.dita" linking="targetonly"/>
</relcell>
</relrow>
<relrow>
<relcell>
<topicref href="../introduction/aboutditaot.dita"/>
</relcell>
```

```
<relcell>
<topicref href="../core_vocabulary/eclipse_content.dita" linking="targetonly"/>
</relcell>
</relrow>
```

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>
<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>
```

Production notes (linking)

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 both cross-references (xrefs) and relationship tables: where contextual information would seem to be helpful to the user ("I'll tell you *why* might might you want to click this link") we have used xrefs embedded in the content, 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.

To help ensure that our links continue to be accurate throughout the document production process, we run a a set of debugging and link-checking tools we produced ourselves approximately once a day. 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.

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

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.

Plug-in	Description
apiref0.8	**The API reference specialization** provides a general-purpose basis for documenting callable programming libraries.
FrameMaker Adapter 0.8	**The FrameMaker Adapter** (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. **Note:** This is *not* the Adobe **FrameMaker Application Pack for DITA**.
Idiom FO 1.1	The **Idiom FO plug-in** 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.) For specific installation instructions, see *Installing the Idiom FO plug-in*.
javaapiref0.8	The **Java API reference** specialization provides a basis for documenting Java class libraries.
Thesaurus0.8.1	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.

Installing plug-ins

Generic installation instructions for 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 139.
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.

Production notes (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:

- 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*

Managing your 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

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, if 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.

Using a library or source control system

Storing DITA files in library (source control) systems

DITA source files are stored on disk as ASCII text files, just as are source code files for software projects. Because of this source control or library systems used to store and manage software source code can also be used for DITA source files. A source control system stores and tracks changes to source files. Multiple versions of source files are kept, and most systems have built-in capabilities to display differences between different versions of the same file.

Some library (source control) systems you can use

The following table shows just a few of the many source control systems that can be used for storing DITA files.

System name	Description
CVS (Concurrent Versions System)	Open source project. For more information, see *http://www.cvsnt.org/wiki/*.
Perforce	Commercial software configuration management system. You can download a free version for small projects. For more information, see *http://www.perforce.com/*.
Subversion	Open source project that is meant as a newer replacement for CVS. For more information, see *http://subversion.tigris.org/*.

Using a a content management system

Why a CMS can be useful to manage DITA files

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

In addition most CMSs provide basic library (source control) functionality.

How you can find a CMS that matches your needs, environment, and budget

The DITA focus area web site (dita.xml.org) has a list of DITA-related products, including products that provide content management functionality. See *http://dita.xml.org/products-services* to access the DITA product listings.

How to find general content management information

There are many web sites with information about content management. A good place to start is with CM Professionals at *http://www.cmprofessionals.org*.

Production notes (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)

Name, description	**Location**
CM Professionals, 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.	*http://www.cmprofessionals.org/*
CMS Review provides resources to help you choose a content management solution.	*http://www.cmsreview.com/*
CMS Watch™ provides independent evaluations of content management, records management, enterprise search, and portal solutions.	*http://www.cmswatch.com/*
Managing Enterprise Content: A Unified Content Strategy (book) by Ann Rockley.	*http://www.rockley.com/*
Content Management for Dynamic Web Delivery by JoAnn T. Hackos.	*http://www.comtech-serv.com/*
The international association for **Open Source Content Management**, is an association connecting users and developers of open source content management solutions.	*http://www.oscom.org/*

Reuse concepts and techniques

This section contains information on how to reuse content (using conrefs), information design (using specializations), and processing code.

Sections in this topic:

About reuse

DITA and DITA Open Toolkit support three kinds of reuse:

- **Content reuse**, in which a source topic or part of a topic is written once and used in multiple locations. For example, you might reference the same concept topic (say, processing DITA files) in both the processing and the troubleshooting maps. Another example might be to use the DITA content reference (conref) mechanism to reuse content once (say, using the text of a controlled vocabulary topic in an "about" file) or many times (say, repeating a short warning statement about the proper use of a hardware unit).
- **Information design reuse (specialization)**, in which you extend the definition of an existing DITA element to be used in a special way. Specialization makes use of the fact that DITA is based on the principle of inheritance.
- **Processing reuse**, in which you override stylesheet processing to customize your output.

The following topics provide information about how to implement information design reuse (specialization) and processing reuse, and the production notes topic for this section provides examples of how conrefs have been used in this document (content reuse).

Implementing information design reuse (specialization)

About information design reuse (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.

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

For examples of how to do specializations, see *http://www-128.ibm.com/developerworks/xml/library/x-dita2/*.

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.

Implementing processing reuse

Processing reuse overview

In the final stage of processing, the DITA Open Toolkit runs XSLT stylesheet transforms to produce the output. In certain cases, it is possible to override stylesheet processing to customize the output.

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.

Transtype (*can be overridden)	XSLT stylesheet(s)
*docbook	map2docbook.xsl, dita2docbook.xsl
eclipsecontent	map2eclipse.xsl, map2plugin-cp.xsl

Transtype (*can be overridden)	**XSLT stylesheet(s)**
eclipsehelp	map2eclipse.xsl
htmlhelp	map2hhc.xsl, map2hhp.xsl
*javahelp	dita2html.xsl
*pdf	dita2fo-shell.xsl
troff	dita2troff-step1-shell.xsl, dita2troff-step2-shell
*wordrtf	dita2rtf.xsl
*xhtml	dita2xhtml.xsl

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.

How to override an XSLT stylesheet (specific 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 `xdita2fo-shell.xsl`".
2. Delete the following text in the file near line 134:

```
<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="."></xsl:value-of>] </fo:block>
</xsl:for-each>
```

3. Add the following line to the Ant target in your build script:

```
<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.

Production notes (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:

```
<conbody>
<section id="ant_term">
<p>
<b>Definition</b>
</p>
<p>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.</p>
<p>
<b>Usage</b>
</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.
Ant 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, but it is not part
of the Toolkit installation package.</p>
<p>
<b>For more information</b>
</p>
<p>For information about the Ant version required by the Toolkit,
see <xref href="../release_current/sysreqs.dita"
format="dita">System requirements and supported applications</xref>.</p>
<p>
<b>To obtain</b>
</p>
<p>You can download Ant from <
xref href="http://ant.apache.org/bindownload.cgi"
format="html" scope="external"/>.</p>
</section>
</conbody>
```

Here is the conref in the About Ant topic:

```
<conbody>
<section conref="../core_vocabulary/ant.dita#ant/ant_term">
</section>
```

For more information (reuse)

For more detailed discussions on content reuse and specialization, 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*.

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

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 DITA 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>
</indexterm>
</keywords>
<prodinfo>
<prodname>DITA Open Toolkit</prodname>
<vrmlist>
<vrm version="1.3"/>
</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="2006"></copyryear>
<copyrholder>VR Communications, Inc.</copyrholder>
</copyright>
<critdates>
<created date="2006-June-10"/>
<revised modified="2006-July-23"/>
</critdates>
<metadata>
<keywords>
<keyword>Darwin Information Typing Architecture</keyword>
<keyword>DITA</keyword>
<keyword>DITA Open Toolkit</keyword>
<keyword>managing content</keyword>
<keyword>metadata</keyword>
</keywords>
<prodinfo>
<prodname>DITA Open Toolkit</prodname>
<vrmlist>
<vrm version="1.3"></vrm>
</vrmlist>
</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 that was 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 source for the source file above:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<meta name="security" content="public"/>
<meta name="Robots" content="index,follow"/>
<meta http-equiv="PICS-Label" content='(PICS-1.1 "http://www.icra.org/ratingsv02.html" l
 gen true r (cz 1 lz 1 nz 1 oz 1 vz 1) "http://www.rsac.org/ratingsv01.html" l gen true
r (n 0 s 0 v 0 l 0) "http://www.classify.org/safesurf/" l gen true r (SS~~000 1))' />
<meta name="DC.Type" content="concept"/>
<meta name="DC.Title" content="About metadata"/>
<meta name="abstract" content="Overview information about creating and using metadata."/>
<meta name="description" content="Overview information about creating and using metadata."/>
<meta name="DC.subject" content="Darwin Information Typing Architecture, DITA, DITA Open
 Toolkit, managing content, metadata"/>
<meta name="keywords" content="Darwin Information Typing Architecture, DITA, DITA Open
Toolkit, managing content, metadata"/>
<meta name="DC.Relation" scheme="URI" content="../managing/managing.html"/>
<meta name="DC.Creator" content="Anna van Raaphorst"/>
<meta name="DC.Contributor" content="Richard Johnson"/>
<meta name="DC.Publisher" content="OASIS (Organization for the Advancement of Structured
 Information Standards)"/>
<meta name="copyright" content="VR Communications, Inc. 2006" type="primary"/>
```

```
<meta name="DC.Rights.Owner" content="VR Communications, Inc. 2006" type="primary"/>
<meta name="DC.Date.Created" content="2006-June-10"/>
<meta name="DC.Date.Modified" content="2006-June-10"/>
<meta name="DC.Format" content="XHTML"/>
<meta name="DC.Identifier" content="aboutmetadata"/>
```

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 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:

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

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.

Coordinate the values you use with your team, so that all the information can be consistently processed with the same value definitions.

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 this filter file:

Attribute	Value	Action	Flag
audience	programmer	exclude	
platform	unix	flag	../images/ngunix.gif

You should get this output:

- This is important for lots of reasons
- UNIX This applies on UNIX

The first list item remains because it applies to admin audience 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 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") you want to exclude on.
      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 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 version="1.0" encoding="UTF-8"?>
<val> </val>
```

3. Define the property you want to flag on:
   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 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 filter file you want by using the parameter `/filter:{args.input.valfile}` for the output options.

Content is flagged when any of the flagged values appears in any of the attributes. When flagged values are found in a map, links generated from affected topicrefs will be flagged on output.

Examples

<p audience="admin programmer">	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.
<topicref platform="linux" href="abc.dita">	When linux is set to flag, any links to abc.dita derived from this topic reference will be flagged with the image you specified. In addition, any links derived from child topicrefs will also be flagged: metadata attributes are inherited.

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 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 only works for flagging phrase-level information, and can not be distinguishable to color-blind readers, on monochrome displays, or once an online page is printed. Note: 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 be like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <revprop val="rev1" style="Green" action="noflag" date="2005-10-08" />
  <revprop val="rev2" action="flag" date="2005-10-10" />
</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.

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, amd 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 name of title of a web page is "About metadata".

SKOS

The Thesaurus 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 139 for more information about this plug-in.

Production notes (accessing)

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. Here is the section of the script that references the ditaval file:

```
<!-- Specify the file to be used for filtering. -->
<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:

```
<map title="Garage (hierarchy)">
<topicref href="concepts/garagetasks.dita" format="dita">
<topicref href="tasks/changingtheoil.dita" otherprops="oil" format="dita"/>
<topicref href="tasks/organizing.dita" 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, referenced in the Ant script, that excludes topics tagged as having to do with oil or snow:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Filters out topics about "oil" and "snow" from the garage sample. -->
<val>
<prop att="otherprops" val="oil" action="exclude"></prop>
<prop att="otherprops" val="snow" action="exclude"></prop>
</val>
```

For more information (accessing)

The garage sample has coding to do filtering (conditional processing). See *About the garage sample* on page 74 for more information.

Customizing your published output

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

Sections in this topic:

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:

```
<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 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:

```
<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 you 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:

Parameter	Description
args.copycss	Whether to copy your CSS to the output directory.
args.css	Path to your CSS file.
args.csspath	Location of your CSS file in the output directory.

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:

```
/* 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:

```
<!-- Properties to add a custom CSS -->
<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:

```
<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.

Tailoring XHTML output

XHTML tailoring 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:

Parameter	**Definition, Usage**	**Examples**
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/myftr.xml"/>
args.hdf	URI of the file containing XHTML to be placed in the head area of the output file. The file must be well-formed XML.	Example: <property name="args.hdf" value="file:/C:/sandbox/myhdf.xml"/>
args.hdr	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.	Example: <property name="args.hdr" value="file:/C:/sandbox/myhdr.xml"/>

Instructions for tailoring XHTML ouput (specific example)

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:

```
<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:

```
<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>
```

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:

```
<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:

```
<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:

```
<fo:external-graphic src="/path/to/images/my_logo_file.pdf"/>
```

Production notes (customizing)

Our experience with default CSS behavior

In the Ant scripts used to build the *DITA Open Toolkit User Guide and Reference*, 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:

```
/* Stylesheet overrides for DITA Open Toolkit Users Guide and Reference */

/* DITAOT_UGRef_CSS.css */

.filepath { font-family: courier }
```

For more information (customizing)

For more information about Ant parameters, see *Ant processing parameters* on page 69.

Localizing (translating) your DITA content

This section contains information about translating the content in your DITA projects.

Sections in this topic:

About localizing (translating)

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

Supported languages

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

Language	xml:lang value
Arabic	ar-eg
Belarusian	bg-bg
Bulgarian	be-by
Catalan	ca-es
Chinese (Simplified)	zh-cn
Chinese (Traditional)	zh-tw
Croatian	hr-hr
Czech	cs-cz
Danish	da-dk
Dutch	nl-nl
Dutch (Belgian)	nl-be
English (Canadian)	en-ca
English (UK)	en-gb
English (US)	en-us
Estonian	et-ee
Finnish	fi-fi
French	fr-fr
French (Belgian)	fr-be

Language	xml:lang value
French (Canadian)	fr-ca
French (Swiss)	fr-ch
German	de-de
German (Swiss)	de-ch
Greek	el-gr
Hebrew	he-il
Hungarian	hu-hu
Icelandic	is-is
Italian	it-it
Italian (Swiss)	it-ch
Japanese	ja-jp
Korean	ko-lr
Latvian	lv-lv
Lithuanian	lt-lt
Macedonian	mk-mk
Norwegian	no-no
Polish	pl-pl
Portuguese	pt-pt
Portuguese (Brazilian)	pt-br
Romanian	ro-ro
Russian	ru-ru
Serbian	sr-sp
Slovak	sk-sk
Slovenian	sl-si
Spanish	es-es
Swedish	sv-se
Thai	th-th
Turkish	tr-tr
Ukranian	uk-ua

Production notes (localizing)

Using DITA to become "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 that supports 9 output formats and over 40 languages
- 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 needs to be translated only once.

For more information (localizing)

Name, description	Location
The Localization Industry Standards Association (LISA) 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.	*http://www.lisa.org/*
Globalization and Localization Association (GALA) is a non-profit international industry association for the translation, internationalization, localization, and globalization industry. The association gives members a common forum to discuss issues, create innovative solutions, promote the industry, and present a joint voice within the industry and to the outside community.	*http://www.gala-global.org/*
ATA (American Translators Association) is a professional association founded to advance the translation and interpreting professions and foster the professional development of individual translators and interpreters. Its members include translators, interpreters, teachers, project managers, web and software developers, language company owners, hospitals, universities, and government agencies.	*http://www.atanet.org/*
W3C internationalization activity The W3C Internationalization Activity has the goal of proposing and coordinating any techniques, conventions, guidelines and activities within the W3C and together with other organizations that allow and make it easy to use W3C technology worldwide, with different languages, scripts, and cultures.	*http://www.w3c.org/International/*

Distributing your published content

This section contains information about distributing your published DITA content.

Sections in this topic:

About distributing content

Distributing means making your published DITA content available to your customers, potential customers, and other interested users.

Distributing information about published DITA content as RSS

Definition

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.

Usage

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 the output:

```
<?xml version='1.0'?>
<rss version='2.0'>
<channel>

<title>DITA OT User Guide</title>
<copyright>Copyright (c) 2006 Anna van Raaphorst. All rights reserved.</copyright>
<link>http://www.mysite.com/ditaug.html</link>
<description>DITA Open Toolkit User Guide 1.3</description>
<language>en-us</language>
<lastBuildDate> Wed, 15 Aug 2006 15:14:30 PST</lastBuildDate>
<item>
<title>DITA OT User Guide 1.3</title>
<link>http://www.mysite.com/ditaug.pdf</link>
<pubDate>Wed, 15 Aug 2006 15:14:30 PST</pubDate>
<description>
Get the latest version of DITA OT User Guide.
</description>
</item>

</channel>
</rss>
```

Distributing information by publishing DITA 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:

Content migration overview

In September 2006 Adobe released FrameMaker DITA Application Pack, a free plug-in for FrameMaker 7.2.

Note: This is a plug-in for FrameMaker, *not* DITA Open Toolkit.

The Application Pack adds a plug-in to Framemaker that allows editting 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.

Production notes (migrating content)

This document is about 90% new material. The small amount of legacy content was migrated manually.

For more information (migrating content)

Description	Location
Adobe white paper on **migrating from unstructured to structured FrameMaker**	*http://www.adobe.com/products/framemaker/pdfs/migrationguide.pdf*
Adobe Framemaker Application Pack for DITA	*http://labs.adobe.com/technologies/framemaker_ap/*
Bright Path Solutions	*http://www.travelthepath.com/structure4.html*
PTC: What you should know about DITA	*http://www.arbortext.com/resources/dita_q_a.htm*
IBM: Migrating HTML to DITA	*http://www-128.ibm.com/developerworks/xml/library/x-dita8a/*

Sample files

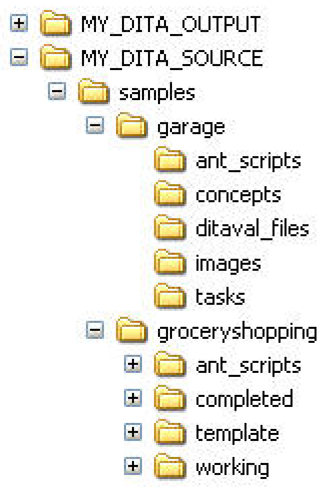
This section contains information about the DITA sample source files that come with DITA Open Toolkit.

The following sample files (garage and grocery shopping) are included in their DITA source form for your reference. Ant scripts are also included. Both samples are in the `samples` directory.

Garage sample

The garage sample, which is located in the `ditaot/doc/ot-usergide/MY_DITA_SOURCE/samples` directory of the Toolkit source distribution, 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 to allow you to publish to all supported target environments.

Before you begin to use the sample files (which include both the garage sample and the grocery shopping sample), we recommend creating two directories in your root directory called `MY_DITA_SOURCE` and `MY_DITA_OUTPUT` (Windows examples would be `C:/MY_DITA_SOURCE` and `C:/MY_DITA_OUTPUT`) and then copying both the garage and grocery shopping sample files from the `ditaot/samples` directory to `MY_DITA_SOURCE`. Your directory structure should then look like this:

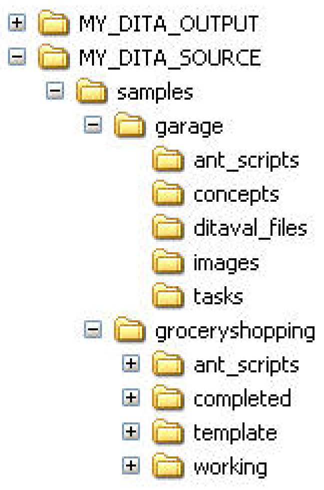


The garage sample includes Ant scripts that process to all supported target environments. A filtering (conditional processing) script is also included: `garage_filtering_xhtml.xml`. This script filters out (excludes) all files having to do with oil or snow, which are tagged with the "otherprops" attribute. Running this script produces a hierarchically organized output file with four of the topics excluded.

Grocery shopping sample

The grocery shopping sample, which is located in the `ditaot/doc/ot-userguide/MY_DITA_SOURCE/samples` directory of the Toolkit source distribution, 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 the XHTML, HTML Help, and PDF2 targets are also provided.

Before you begin to use the sample files (which include both the garage sample and the grocery shopping sample), we recommend creating two directories in your root directory called `MY_DITA_SOURCE` and `MY_DITA_OUTPUT` (Windows examples would be `C:/MY_DITA_SOURCE` and `C:/MY_DITA_OUTPUT`) and then copying both the garage and grocery shopping sample files from the `ditaot/samples` directory to `MY_DITA_SOURCE`. Your directory structure should then look like this:



```
⊞ MY_DITA_OUTPUT
⊟ MY_DITA_SOURCE
    ⊟ samples
        ⊟ garage
                ant_scripts
                concepts
                ditaval_files
                images
                tasks
        ⊟ groceryshopping
            ⊞ ant_scripts
            ⊞ completed
            ⊞ template
            ⊞ working
```

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 65.

Frequently asked questions (FAQs)

This section contains frequently asked questions about DITA and DITA Open Toolkit (OT).

Sections in this topic:

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.

What is topic-based authoring?

Contributing authors: JoAnn Hackos and IBM

Date: 2006

What is a topic?

A topic is a unit of information with a title and content, short enough to be specific to a single subject.

What is the origin of topic-based writing?

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.

What is a DITA topic?

In DITA a topic is the basic unit of authoring and of reuse. 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.

How long should a DITA topic be?

Topics should be short enough to be easily readable, but long enough to make sense on their own.

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.

```
<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 s 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 SouceForge, 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) re-use.

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 som eone from the DITA TC 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

I followed the following general process to architect and edit *DITA Open Toolkit User Guide and Reference* (a document of over 300 topics and approximately 220 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 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.

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 Darwin Information Typing Architecture (DITA) 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 and Reference*
- 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.

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 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 implimenting or planning to impliment 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)

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

Cascading stylesheet (CSS)

A CSS is a file that 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 50.

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>
```

Command element (<cmd>)

In a task step, a command element (<cmd>) describes the action the user needs to take.

Example

```
<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>
```

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 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.

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

```
<taskbody>
<context>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
<filepath>MY_DITA_SOURCE/samples/groceryshopping</filepath> directories.
This topic assumes you are familiar with the information in
<xref href="../topics/aboutgroceryshopping_sample.dita" scope="local">
About the grocery shopping sample</xref>,
and that you have created the topics according to the instructions
in <xref href="../topics/topics.dita" scope="local">Topics</xref>.</context>
<steps>
<step>
<cmd>Go to the <filepath>groceryshopping/template</filepath> directory.</cmd>
</step>
```

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

```
<step>
<cmd>Set the <varname>CLASSPATH </varname>
<xref href="linux_settingenvvariables.dita" scope="local">environment
variable</xref> for <codeph>dost.jar</codeph>
</cmd>
</step>
```

Darwin Information Typing Architecture (DITA)

DITA 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.

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

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 (TC).

Definition list

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

```
<section id="knownproblems_info">
<p>You can get current information about bugs,
patches, and change requests in the following locations:</p>
<dl>
<dlentry>
<dt>Bug tracker</dt>
<dd>
<xref href="http://sourceforge.net/tracker/?group_id=132728&atid=725074" scope="external"/>
</dd>
</dlentry>
<dlentry>
<dt>Patch tracker</dt>
<dd>
<xref href=" http://sourceforge.net/tracker/?group_id=132728&atid=725076" scope="external"/>
</dd>
</dlentry>
<dlentry>
<dt>RFE tracker</dt>
<dd>
<xref href="http://sourceforge.net/tracker/?group_id=132728&atid=725077" scope="external"/>
</dd>
</dlentry>
</dl>
</section>
```

Distributing your published content

Distributing means making your published DITA content available to your customers, potential customers, and other interested users.

DITA

See

Darwin Information Typing Architecture (DITA) on page 187

DITA Open Toolkit (OT)

DITA Open Toolkit is an implementation of the OASIS DITA Technical Committee's specification for DITA DTDs and schemas. The Toolkit, which can be used in the Windows, UNIX/Linux, and Mac OS operating environments, transforms DITA content (maps and topics) into deliverable formats.

DITA Open Toolkit supports the following publishing environments:

- DocBook
- Eclipse content
- Eclipse help
- HTML Help
- JavaHelp
- PDF
- troff
- Word RTF
- XHTML

DITA Open Toolkit User Guide and Reference

DITA Open Toolkit User Guide and Reference (this document) is the definitive source of information about DITA Open Toolkit (OT), 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

```
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN"
"../dtd/concept.dtd">
```

Document type definition (DTD)

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.

Example (concept DTD)

```
<!-- ============================================================= -->
<!--                    HEADER                                     -->
<!-- ============================================================= -->
<!--  MODULE:    DITA Concept DTD                                  -->
<!--  VERSION:   1.0.1                                             -->
<!--  DATE:      November 2005                                     -->
<!--                                                               -->
<!-- ============================================================= -->

<!-- ============================================================= -->
<!--                    PUBLIC DOCUMENT TYPE DEFINITION            -->
<!--                    TYPICAL INVOCATION                         -->
<!--                                                               -->
<!--  Refer to this file by the following public identifier or an
      appropriate system identifier
PUBLIC "-//OASIS//DTD DITA Concept//EN"
      Delivered as file "concept.dtd"                              -->
```

```
<!-- ============================================================= -->
<!-- SYSTEM:     Darwin Information Typing Architecture (DITA)     -->
<!--                                                               -->
<!-- PURPOSE:    DTD to describe DITA concepts                     -->
<!--                                                               -->
<!-- ORIGINAL CREATION DATE:                                       -->
<!--             March 2001                                        -->
<!--                                                               -->
<!--             (C) Copyright OASIS Open 2005.                    -->
<!--             (C) Copyright IBM Corporation 2001, 2004.         -->
<!--             All Rights Reserved.                              -->
<!-- ============================================================= -->


<!-- ============================================================= -->
<!--                    DOMAIN ENTITY DECLARATIONS                 -->
<!-- ============================================================= -->


<!ENTITY % ui-d-dec     PUBLIC
"-//OASIS//ENTITIES DITA User Interface Domain//EN"
"uiDomain.ent"                                                    >
```

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 content

Eclipse is a open-source platform-independent software framework for delivering "rich-client" applications, as opposed to "thin-client" browser-based applications. Originally developed by IBM, Eclipse is now managed by the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors. Eclipse content is one of the documentation sytem options available in the framework.

Eclipse content is one of the DITA target outputs.

Eclipse help

Eclipse is a open-source platform-independent software framework for delivering "rich-client" applications, as opposed to "thin-client" browser-based applications. Originally developed by IBM, Eclipse is now managed by the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors. Eclipse help is one of the documentation sytem options available in the framework.

Eclipse help is one of the DITA Open Toolkit target outputs.

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 184

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 and flagging content

See

Conditional processing (filtering and flagging content) on page 185.

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.

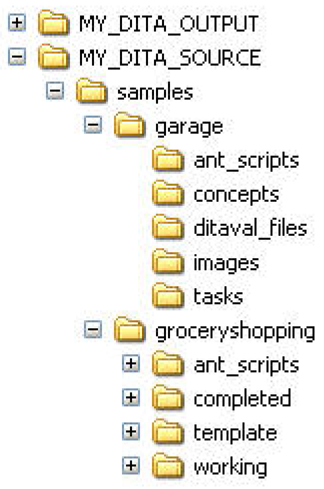
The most common formats for DITA files are "dita,", "xml," and "ditamap." You can also reference other formats like "pdf" or "html."

```
<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 located in the `ditaot/doc/ot-usergide/MY_DITA_SOURCE/samples` directory of the Toolkit source distribution, 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 to allow you to publish to all supported target environments.

Before you begin to use the sample files (which include both the garage sample and the grocery shopping sample), we recommend creating two directories in your root directory called `MY_DITA_SOURCE` and `MY_DITA_OUTPUT` (Windows examples would be `C:/MY_DITA_SOURCE` and `C:/MY_DITA_OUTPUT`) and then copying both the garage and grocery shopping sample files from the `ditaot/samples` directory to `MY_DITA_SOURCE`. Your directory structure should then look like this:



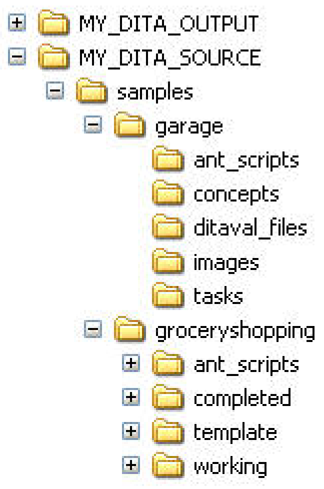
```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
            completed
            template
            working
```

The garage sample includes Ant scripts that process to all supported target environments. A filtering (conditional processing) script is also included: `garage_filtering_xhtml.xml`. This script filters out (excludes) all files having to do with oil or snow, which are tagged with the "otherprops" attribute. Running this script produces a hierarchically organized output file with four of the topics excluded.

Grocery shopping sample

The grocery shopping sample, which is located in the `ditaot/doc/ot-userguide/MY_DITA_SOURCE/samples` directory of the Toolkit source distribution, 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 the XHTML, HTML Help, and PDF2 targets are also provided.

Before you begin to use the sample files (which include both the garage sample and the grocery shopping sample), we recommend creating two directories in your root directory called `MY_DITA_SOURCE` and `MY_DITA_OUTPUT` (Windows examples would be `C:/MY_DITA_SOURCE` and `C:/MY_DITA_OUTPUT`) and then copying both the garage and grocery shopping sample files from the `ditaot/samples` directory to `MY_DITA_SOURCE`. Your directory structure should then look like this:



```
MY_DITA_OUTPUT
MY_DITA_SOURCE
    samples
        garage
            ant_scripts
            concepts
            ditaval_files
            images
            tasks
        groceryshopping
            ant_scripts
            completed
            template
            working
```

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 65.

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

```
<concept id="framework">
<title>The DITA authoring/production framework</title>
```

```
<conbody>
<section id="javahelp_term">
```

Indexing content

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
```

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 210

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

```
<step>
<cmd>Save and extract the package file
into a Linux home directory. </cmd>
<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.

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 Development Kit (JDK)

The JDK is a 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 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

```
<metadata>
<keywords>
<keyword>Ant script</keyword>
<indexterm>Ant scripts
<indexterm>definition</indexterm>
<indexterm>usage</indexterm>
</indexterm>
</keywords>
```

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 <xref> and <related-links>, and through relationship tables.

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 version="1.0" encoding="utf-8"?>
<!-- (c) Copyright 2006 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="css.dita"/>
<topicref href="customizing_production_notes.dita"/>
<topicref href="customizing_formoreinfo.dita"/>
</topicref>
</map>
```

Metadata

Metadata is semantic information about the information in a DITA 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>
</indexterm>
</keywords>
<prodinfo>
<prodname>DITA Open Toolkit</prodname>
<vrmlist>
<vrm version="1.3"/>
```

```
</vrmlist>
</prodinfo>
</metadata>
```

Migrating legacy content to DITA

In September 2006 Adobe released FrameMaker DITA Application Pack, a free plug-in for FrameMaker 7.2.

Note: This is a plug-in for FrameMaker, *not* DITA Open Toolkit.

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 203).

Example

```
<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

An ordered list is a list (typically numbered) in which the order of list items is significant (for example, steps in a procedure.)

Example

```
<conbody>
<p>A good wheelbarrow will save your back
from extensive trauma when you garden.
wheelbarrows are most often used to haul (in order of importance):</p>
<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 is one of the 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

```
<step>
<cmd>Follow the steps in the HTML Help install guide
wizard to complete the installation.</cmd>
</step>
</steps>
<postreq>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 <filepath>.xml</filepath> scripts
in the Toolkit root directory.
The Toolkit assumes the compiler
is installed on your C drive.
</postreq>
</taskbody>
```

Prerequisite element (<prereq>)

In a task, a prerequisite element specifies something a user needs to know or do before beginning the task.

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.

Processing attribute

A processing attribute controls inclusion of topics in the table of contents (toc attribute) and the print version of a published document (print attribute), if one is specified.

Processing reuse

In the final stage of processing, the DITA Open Toolkit runs XSLT stylesheet transforms to produce the output. In certain cases, it is possible to override stylesheet processing to customize the output.

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 elment is an element containing metadata for a topic (for example, author, creation date, and modification date).

```
<prolog>
<author type="creator">Anna van Raaphorst</author>
<copyright>
<copyryear year="2006"/>
<copyrholder>VR Communications, Inc.</copyrholder>
</copyright>
<critdates>
<created date="2006-August-07"/>
<revised modified="2006-August-07"/>
</critdates>
<metadata>
<keywords>
<keyword>grocery shopping</keyword>
<keyword>canned goods</keyword>
</keywords>
</metadata>
</prolog>
```

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, amd 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 name of title of a web page is "About metadata".

SKOS

The Thesaurus 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 139 for more information about this plug-in.

Reference analysis

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

```
<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.

result element (<result>)

In a task, a result element provides the user with information about what completing a task should have accomplished.

Reuse concepts and techniques

DITA and DITA Open Toolkit support three kinds of reuse:

- **Content reuse**, in which a source topic or part of a topic is written once and used in multiple locations. For example, you might reference the same concept topic (say, processing DITA files) in both the processing and the troubleshooting maps. Another example might be to use the DITA content reference (conref) mechanism to reuse content once (say, using the text of a controlled vocabulary topic in an "about" file) or many times (say, repeating a short warning statement about the proper use of a hardware unit).
- **Information design reuse (specialization)**, in which you extend the definition of an existing DITA element to be used in a special way. Specialization makes use of the fact that DITA is based on the principle of inheritance.
- **Processing reuse**, in which you override stylesheet processing to customize your output.

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 is a shared vocabulary that allows machines to carry out rules made by people.

Schemas provide a means for defining the structure, content, and semantics of XML documents.

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

```
<p>Before you use DITA Open Toolkit,
be sure your operating environment meets the system requirements
described in <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
by the two principals of VR Communications, Inc.
(<xref href="http://www.vrcommunications.com" format="html" scope="external">
www.vrcommunications.com</xref>)
```

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 desciptions are tagged with the short description element (<shortdesc>).

```
<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>
```

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

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.

```
<section>
<p>Sections in this topic:
<sl>
<sli>
<xref href="#aboutditaotugref/contents">Document contents</xref>
</sli>
<sli>
<xref href="#aboutditaotugref/target_audience">Audience</xref>
</sli>
<sli>
<xref href="#aboutditaotugref/prerequisites">Prerequisites</xref>
</sli>
<sli>
<xref href="#aboutditaotugref/howproduced">How this document was produced</xref>
</sli>
</sl>
</p>
</section>
```

Simple table element

A simple table is used to describe tabular information that does not require control over the display properties and layout.

Example

```
<simpletable>
<sthead>
<stentry>Name, description</stentry>
<stentry>Location</stentry>
</sthead>
<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 and other DITA-related processing tools: *http://sourceforge.net/projects/dita-ot/*

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

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

```
<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
<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.</cmd>
</step>
<step>
<cmd>Be sure your system environment meets the requirements in
<xref href="../release_current/sysreqs.dita" scope="local">
System requirements and supported applications</xref>.</cmd>
</step>
...
```

Structure element

A structure element is a base element that you can use with every DITA information type.

Example(s)

Topic, title, short description, and body.

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.

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.

For more information

For more information about Task Modeler, see *http://www.alphaworks.ibm.com/tech/taskmodeler*.

The Task Modeler download contains Eclipse version 3.0 with Task Modeling and XML file editing (XMLBuddy) already configured.

Topic information type

Topic is the base DITA information type.

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>
<b>Definition</b>
</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

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

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 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.

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 one of the DITA Open Toolkit target outputs.

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 version="1.0" encoding="utf-8"?>
```

Index