Contents

Legal, Copyright, Download..............................................................................................................13

Chapter 1: Introduction..................................................................................................................15
  About DITAinformationcenter......................................................................................................15
  Setting up a DITAinformationcenter project in your environment...............................................17
  About the authors.....................................................................................................................17
  About the project.......................................................................................................................18
  Acknowledgements...................................................................................................................19

Chapter 2: Getting started.............................................................................................................21
  Planning your initial DITA project..............................................................................................21
  Setting up your initial DITA project...........................................................................................22

Chapter 3: Trends in global communication..................................................................................23
  About the challenges...................................................................................................................23
  About the solutions.....................................................................................................................24

Chapter 4: Evaluating DITA..........................................................................................................27
  About workflow.........................................................................................................................27
  Legacy workflow.......................................................................................................................27
  Changes required to the legacy workflow..................................................................................29
  Our recommended DITAworkflow™ methodology....................................................................29
    Step 1: Evaluate DITA...........................................................................................................31
    Step 2: Create ROI objectives...............................................................................................31
    Step 3: Analyze your current documentation.........................................................................32
    Step 4: Create a documentation strategy and plan .............................................................32
    Step 5: Create a prototype implementation..........................................................................32
    Step 6: Implement your design (in stages)............................................................................33

Chapter 5: Key concepts................................................................................................................35
  About DITA..............................................................................................................................35
  About DITA Open Toolkit...........................................................................................................37
  About DITA projects..................................................................................................................38
    Source files............................................................................................................................38
    Processing files......................................................................................................................41
    Output files............................................................................................................................42
  About processing and publishing..............................................................................................45
Chapter 6: Installing the DITA Open Toolkit.................................51
  Installation overview.................................................................51
  Installing the Toolkit...................................................................53
  Verifying the Toolkit installation...................................................53
  Directories and files in the ditaot directory......................................54

Chapter 7: Setting up your working environment..........................55
  Configuring your authoring tool to use DTD or schema URLs...........55
  Setting up your source and output file directories...........................55

Chapter 8: Processing and publishing DITA documents...............57
  About Ant......................................................................................58
  About Ant scripts..........................................................................58
  Processing overview......................................................................59
  Ant processing parameters..........................................................60
  Processing to the key Toolkit output targets...............................65
    Processing to XHTML targets..................................................65
    Processing to PDF targets......................................................66
    Processing to HTML Help targets...........................................68
    Processing to Eclipse targets..................................................69

Chapter 9: Troubleshooting the build process...............................73
  Capturing and using the log..........................................................73
  Error messages generated during processing.....................................74
    Error messages overview.......................................................74
    Messages generated from other sources.....................................75
  Debugging, reporting, repair, and file generation tools.....................76
    About the tools.........................................................................76
    Using the tools.........................................................................77
    Repairing bad references with DITArepair....................................78

Chapter 10: Creating DITA topics......................................................81
  About information types.............................................................81
Chapter 20: Distributing your published content.................155

About distributing published content.................................................................155
Creating DITAmashups™....................................................................................155
About DITAmashups..............................................................................................155
Architecting DITAmashups..................................................................................156
Publishing DITAmashups.....................................................................................157
Distributing published content using RSS.........................................................158

Appendix A: Getting more information.....................................................159

General information..............................................................................................159
Sample DITA projects............................................................................................160
DITA Open Toolkit release information...............................................................161
Specialization..........................................................................................................161
DITA language standard.........................................................................................161
Tools and plug-ins.................................................................................................162
User groups............................................................................................................162
Services..................................................................................................................163
Events.....................................................................................................................163

Appendix B: Commercial tools.................................................................165

Authoring (editing) tools......................................................................................165
Processing tools.....................................................................................................166
PDF rendering tools..............................................................................................167
Source control and library systems.......................................................................168
Content management systems.............................................................................168
Localization and translation tools and services..................................................169
Legacy content conversion tools and services.....................................................171

Appendix C: Use cases.............................................................................173

DITA use cases.....................................................................................................173
Use case template..................................................................................................174

Appendix D: DITA 1.2 language standard.............................................177

DITA 1.2 language standard..................................................................................177
Examples of DITA 1.2 usage................................................................................179

Appendix E: DITA Open Toolkit plugins.............................................181
Appendix F: DITAcorevocabulary™ .........................................................183

Administrator or manager audience category.........................................................183
Agile development process.........................................................................................183
Apache Ant..................................................................................................................184
Ant processing parameter.........................................................................................184
Ant script......................................................................................................................184
Audience......................................................................................................................184
Authoring tool (DITA)...................................................................................................185
Back translation..........................................................................................................185
Best practice..............................................................................................................185
Block element............................................................................................................185
Body element (<body>)..............................................................................................186
Build file.....................................................................................................................186
CMS (Content Management System).........................................................................186
CSS (Cascading StyleSheet).......................................................................................186
Choice table (<choicetable>)......................................................................................186
Collection-type attribute (collection-type)................................................................187
Command element (<cmd>).......................................................................................187
Concept......................................................................................................................187
Concept analysis........................................................................................................188
Concept information type.........................................................................................188
Conditional processing (filtering and flagging content)...........................................188
Content......................................................................................................................188
Content inventory......................................................................................................188
Content reference attribute (conref).........................................................................189
Content reuse............................................................................................................189
Content specialist audience category......................................................................189
Context element (<context>).....................................................................................189
Controlled vocabulary..............................................................................................190
Controlled natural language (CNL)..........................................................................190
Cross-reference element (<xref>)..............................................................................190
Crowdsourcing.........................................................................................................190
Definition list (<dl>).................................................................................................191
Distributing content....................................................................................................191
DITA (Darwin Information Typing Architecture)......................................................191
DITAinformationcenter™.........................................................................................192
DITAmashup™............................................................................................................192
DITAworkflow™.........................................................................................................193
DITA Open Toolkit.....................................................................................................193
DITA Open Toolkit distributions...............................................................................194
DocBook..................................................................................................................195
Appendix G: Category keywords.................................................................227
DITAinformationcenter is a DITAmashup: a structured/unstructured information solution published as a collection of content topics that share the same category/tag keywords to facilitate information access. DITAinformationcenter solutions are posted on WordPress (XMLdocs.info) and Drupal (DITAinfo.info).

The DITAinformationcenter source files are available to you free of charge to use as a model and educational tool: simply download the files and copy them to your hard drive.

The authors of DITAinformationcenter are Anna van Raaphorst and Richard H. (Dick) Johnson, principals, VR Communications, Inc., a micro-consultancy specializing in content-rich information solutions (DITA/Drupal/WordPress/print). We provide DITAinformationcenter on an as-is basis, without warranties or conditions of any kind, either express or implied.

Portions of the structured documentation were previously published and licensed through a Common Public License (CPL) granted by OASIS (Organization for the Advancement of Structured Information Standards) under the name DITA Open Toolkit User Guide.

© 2006-2011 VR Communications, Inc.. Topics in this document can be used in accordance with the Creative Commons License Attribution-ShareAlike 3.0, which allows anyone to copy, modify, and redistribute modifications of all or part of the DITAinformationcenter documentation as long as the license is included with all copies or redistributions, and the DITAinformationcenter documentation is attributed as the originating document.

Eighth Edition: September 2011

This edition of the DITAinformationcenter was tested and published with release 1.5.2 or 1.5.3 of DITA Open Toolkit, which supports the DITA 1.2 language standard.

The key changes in this edition were to:

- Create a Drupal bulkpub (bulk publishing) web service to publish the structured documentation
- Take additional advantage of the native Drupal services to link and navigate the structured topics
- Create aliases for all structured topics on the Drupal site
- Merge the DITAmashup topics into the distribution section
- Consolidate all topics on customization

DITAmashup websites and download locations

Browse the DITAinformationcenter DITAmashups and download any of our free XML/DITA resources from one of the following locations:

- XMLdocs.info (a WordPress site)
- DITAinfo.info (a Drupal site)
Chapter 1

Introduction

The following topics contain introductory information about the DITAinformationcenter project and its authors.

Sections in this topic:

- About the authors on page 17
- About the project on page 18
- Acknowledgements on page 19

About DITAinformationcenter

DITAinformationcenter consists of:

- A structured/unstructured information solution, or DITAmashup
- Sample DITA projects useful as educational or modeling tools
- Debugging, reporting, repair, file generation, and content automation tools, written in Python or PHP, and suitable for direct use with the DITA Open Toolkit or as models for processing and publishing solutions

DITAinformationcenter, originally published in 2006 as an information-only package under the title DITA Open Toolkit User Guide, was produced using the principles and procedures described in the documentation and exhibited in the models.

You can browse the DITAinformationcenter host sites for ideas and information (XMLdocs.info is the WordPress site and DITAinfo.info is the Drupal site). You can also download from the sites' Downloads pages any of the free resources that would be of use to you.

Structured information collection (the “formal documentation”)

The structured information collection is a formal DITA project of moderate complexity containing approximately 350 source files organized into elementary, advanced, and reference topics. The formal topics are professionally architected and tightly controlled by the content owners.

The structured topics are written to the latest DITA language standard and are custom-published to WordPress and Drupal with a Python script using XML-RPC.

The source topics for the structured topics are the same DITA files used to transform and publish to the more traditional DITA output targets: PDF, XHTML, HTML Help, and Eclipse help. We store the DITA source files in folders on our hard drive.

Unstructured information collection

The unstructured collection consists of informal blogs, posts, articles, and static pages; for example, announcements about changes to the site, author blogs, informal user contributions, and contact information.
Many of the unstructured topics are added or updated frequently. They require editor permission to publish but are handled more informally than the structured topics. They share an approved list of keyword tags with the formal topics.

The unstructured topics are created one of the following ways:

- As WordPress or Drupal files using the native WordPress or Drupal editor
- As WordPress or Drupal files using another blog-capable editor (for example, Microsoft Word or Firefox ScribeFire) and published to WordPress or Drupal with a Python script using an RPC interface

Sample DITA projects

The DITA source files for the structured information collection are available as a model of a moderately complex DITA project. In addition, the following simple DITA projects are available to users new to DITA or structured writing:

- Garage sample, a project of approximately 20 topics
- Grocery shopping sample, a project of 7 topics (one topic, two concepts, two tasks, and two reference topics)

Debugging, reporting, repair, file generation, and content automation tools

These tools are useful in the DITA Open Toolkit publishing environment, written in the PHP or Python programming languages, and meant to be invoked either from a command line prompt or from an Ant build script.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ditaauthors.py</td>
<td>Returns a set of unique author and copyright strings, with counts for each.</td>
</tr>
<tr>
<td>ditadebug.py</td>
<td>Provides information useful in debugging processing errors.</td>
</tr>
<tr>
<td>ditaedit.php</td>
<td>Can be used to search for and perform bulk updates in all files in one or more DITA maps.</td>
</tr>
<tr>
<td>ditaids.py</td>
<td>Produces an alphabetical list of IDs for all files in one or more DITA maps, with duplicates marked with an asterisk.</td>
</tr>
<tr>
<td>ditakeywords.py</td>
<td>Produces a list of all metadata keywords defined.</td>
</tr>
<tr>
<td>ditalinks.py</td>
<td>Tests all URLs referenced by files in one or more DITA maps for validity.</td>
</tr>
<tr>
<td>ditarepair.py</td>
<td>Lists and optionally repairs incorrect references to other DITA topics.</td>
</tr>
<tr>
<td>ditastat.py</td>
<td>Lists the DITA elements used and the number of times each was used.</td>
</tr>
<tr>
<td>ditaunused.py</td>
<td>Produces a list of unused files.</td>
</tr>
</tbody>
</table>
Setting up a DITAinformationcenter project in your environment

Keyword tags: DITAinformationcenter, introduction

If you are new to DITA, we recommend installing the DITA Open Toolkit (by reading the topics in Installing the DITA Open Toolkit on page 51 and per instructions accompanying the download distribution package) to experiment with DITA processing in your environment. As you gain knowledge and experience with DITA, you will probably want to purchase one of the commercial products available on the market today to edit and process your files.

The DITAinformationcenter components are available from one of the following download sites:

• XMLdocs.info (a WordPress site)
• DITAinfo.info (a Drupal site)

To set up the DITAinformationcenter model projects in your environment, follow the instructions below.

1. If you plan to process and edit the garage and grocery shopping samples according to the instructions in the Topics and Maps sections of this document, unzip the files and copy them to your root (C:) directory. Because the original garage and grocery shopping sample files might be overwritten when you edit or process them, we strongly recommend that you back up those projects to a safe location outside your working directory structure.

2. If the DITAinformationcenter source files would be useful to you as a model project, download and unzip those files to your C: root directory, as well.

3. If the DITAinformationcenter published documents (PDF, XHTML, HTML Help, or Eclipse help) would be useful to you as sources of information, download one or more of those files or zip packages from one of the download pages.

About the authors

Keyword tags: Anna van Raaphorst, Dick Johnson, introduction, Richard Johnson

The document authors (also the authors of DITA Open Toolkit User Guide) are Anna van Raaphorst and Richard H. (Dick) Johnson, principals, VR Communications, Inc. (vrcommunications.com), an organization
whose business focus is structured documentation solutions and structured/unstructured mashups. DITA logo artwork is by Elizabeth Tichenor.

Anna van Raaphorst has worked for over 20 years helping organizations with their information architecture, technical and marketing communications, and training needs. She has worked as an architect, team leader, manager, consultant, and practitioner in a wide variety of organizations both domestic and international. Anna is principal and content specialist at VR Communications, Inc.

Richard (Dick) Johnson, with over 20 years of experience doing software design and development, is principal and technology specialist for VR Communications. He has served as principal software engineer, technical team leader, user consultant, user group speaker, and curriculum developer in a number of software, hardware, and scientific organizations. He frequently contributes information and inventions to the open-source, technical community.

About the project

Keyword tags: DITAinformationcenter, introduction

The DITAinformationcenter project was first published under the title DITA Open Toolkit User Guide in 2006 and has been maintained under that title and as DITAinformationcenter ever since. The authors continue to provide it as a free resource to the DITA community for the following reasons:

- We have significant interest and involvement with structured writing, content management, scripting, and programming.
- It gives us a way to build our knowledge and experience in DITA, DITA Open Toolkit, and web content management.
- We want to help the DITA community by documenting and promoting DITA and the Toolkit.
- It is an opportunity for us to use our individual skill sets in a collaborative effort.
Acknowledgements

Keyword tags: DITAinformationcenter, introduction

The authors are grateful for the openness and sense of collaboration that characterize the DITA community. We would like to thank those members of the community who have provided valuable feedback and suggestions for future improvements to this project.
Chapter 2

Getting started

Keyword tags: assessing, DITA workflow, evaluating, getting started, implementing, piloting, planning, prototyping

We assume you've heard of DITA, are somewhat interested in its possibilities for your environment, and want to know what it would take to find out for sure. Based on our own experience as DITA strategists and practitioners, we believe DITA is easier than you probably think it is, but you shouldn't simply jump in with both feet without doing some thoughtful planning, prototyping, and serious discussions with your key stakeholders.

This document has been written to help information teams and their managers get started with DITA, and to give them additional information to help guide them as they gain DITA experience. This section provides a summary roadmap of the steps your organization is likely take as you move into a DITA planning and implementation effort, and how the information in this document can help.

Sections in this topic:

Planning your initial DITA project on page 21
Setting up your initial DITA project on page 22

Planning your initial DITA project

Keyword tags: assessing, DITA workflow, evaluating, getting started, implementing, piloting, planning, prototyping

The following steps are typical in the planning stage of a DITA project. See the referenced sections of this document for more information.

1. In Trends and challenges in global communications, read our take on the current trends in global communications, and how DITA offers a promising solution.
2. Read the topics in Evaluating DITA for suggestions on how to evaluate DITA for use in your environment, how to choose your initial pilot project, and make appropriate changes in your legacy workflow process.
3. If you are new to XML and DITA, read the topics in Key concepts to help you learn more about the DITA operational environment.
4. Install the DITA Open Toolkit.
    Be sure your system environment meets the requirements in Installation overview and then download and install the "full easy install" of the DITA Open Toolkit from http://sourceforge.net/projects/dita-ot/.
5. Install and become familiar with the authoring tool you plan to use to create DITA content.
    For information about choosing an authoring tool, see Authoring tools. Many of the popular authoring tools are available on a 30-day free-trial basis.
Setting up your initial DITA project

Keyword tags: assessing, DITA workflow, evaluating, getting started, implementing, piloting, planning, prototyping, setting up

Follow these steps to set up a working environment for your first DITA pilot project.

1. Set up your DITA source file and processing (build) environments.
   Information and instructions are in Setting up your working environment.

2. Process (build) the garage sample, provided as part of the DITA information center or DITA Open Toolkit for your use and reference.
   For information about processing, see Processing (building) and publishing DITA documents.

3. Create some demo-level DITA topics and a map to aggregate them; process them in the same way you did the garage sample files.
   For instructions, see the DITA tutorial (part of the structured collection), which explains how to create and process a small set of DITA files called "grocery shopping." The Grocery Shopping sample contains template files and a complete Ant script.

4. Using the skills gained by working through the tutorial, design and create a small prototype project of your own (no more than 30-50 topics). Demo to your extended team and other key stakeholders, and get their input on the suitability of this type of project for your environment.

5. Brainstorm with your extended team on what to do next.
   Examples: Another prototype? A small DITA implementation included in the next product release? A collaboration with other departments (for example, Training or Support)? A DITA mashup (structured plus unstructured content)?
   If any of these look promising: What are the key objectives? Requirements? Dependencies? Timeframe? Pros and cons of various approaches?
Chapter 3

Trends in global communication

Keyword tags: trend

The following topics outline some of the current trends in global communication, the challenges they bring to information organizations, and how DITA can participate effectively in global information solutions.

Sections in this topic:

  About the challenges on page 23
  About the solutions on page 24

About the challenges

Keyword tags: challenge, trend

The following global trends are creating serious challenges for the organizations and individuals that create and publish technical information:

- Increased availability of information, primarily because of the wealth of information available through the Internet
- Advances in technology, especially tools and technologies related to the production and distribution of information
- Business and financial pressures resulting from an increasingly globalized and economically challenged marketplace
- Organizational stress (often related to the other challenges) for information creators and publishers

Increased availability of information

Much has been written about the Internet information explosion, where users can instantaneously get thousands of answers to a question typed into a search engine. There may be some quality concerns with this informal information (also known as user-generated content, or UGC), but many users prefer the “experienced-derived” nature of the information to much of the “official” (but less “authentic”) documentation typically available from product owners.

Advances in technology

Opportunities include integration of a product with the information describing it, and the automation of information (where documentation is taken directly from the product).

On the other hand, incompatible, overlapping, and expensive tools, frameworks, and platforms provide a constant challenge for both product and information producers.
Business and financial pressures
As the global marketplace expands, product and service providers can expand their business by moving into new markets. However, these new opportunities also bring additional requirements, for example localizing and translation costs.

Whatever the reason, organizations are constantly looking for ways to reduce costs, and information departments are particularly vulnerable.

Organizational stress for information producers
Many additional pressures on information organizations and individuals are caused by the situations described above, and include:

- Need for increased cooperation among information organizations (for example, marketing, technical publications, training, and product support)
- Need to restructure workflows governing information creation, production, and translation
- Significant role shift for many individuals from "information creator" to "information manager"
- Fear of job loss due to outsourcing, crowdsourcing, or offshoring
- Pressure to learn new tools and technologies
- Pressure to produce information products "faster, better, cheaper"
- Loss of control to the global user community, which may produce more popular and more "authentic" information than the individuals hired to do the job

About the solutions
Keyword tags: solution, trend
We recommend the following responses to meet the current challenges in global communication.

Increase the scope of your solution space.
Expand your focus and workflow procedures to include all aspects of information creation, localization (translation), and production.

Reduce your information creation activities and expand your information management role.
Assume that you are in charge of providing your users with the product “big picture” (including the semantic tags that categorize the information solution), the details of your product's features, and telling your users about other, related, informal-but-valuable information about your product they might want to consult.
Adopt DITA as the standard for the formal product information you produce in-house, and apply your structured information framework to the informal, community-produced information that is an important part of your total information solution.

The following are usually cited as the key benefits to adopting DITA. DITA is:

• XML based and now widely used in the technical writing community
• Supported by an industry standard sponsored by the OASIS (Organization for the Advancement of Structured Information Standards) organization
• A topic-based, information-typed solution that promotes content reuse
• A scalable solution
• Ideally suited for information customization (for example, by audience, environment, or product line)

DITA allows you to:

• Create a single information source and publish to many output targets (for example, XHTML, Eclipse help, PDF, and mobile devices)
• Separate content from rendered style
• Achieve quicker turnaround, higher quality, and reduced costs over time
• Avoid being locked into proprietary tools and platforms

DITA promotes:

• Collaboration among information organizations (for example, marketing, technical publications, training, and support) to achieve greater consistency and additional content reuse
• Collaboration with other, related organizations (for example, software and hardware engineering, testing, and user experience) to achieve higher information quality and increased production efficiencies
• Integration of product and information for a more positive user experience
• Automation of information creation and publication
• Application of a governing semantic framework to related formal and informal information collections to both increase information access and maintain quality control
Chapter 4

Evaluating DITA

Would DITA help you better meet the global communication challenges described in *Trends in global communications*? The following topics describe the kind of changes many organizations need to make to implement DITA successfully.

It pays to do a thorough evaluation of the suitability of DITA for your environment before you commit to a multi-month or possibly even a multi-year project. The topics in this section can help you begin that process.

Sections in this topic:

- About workflow on page 27
- Legacy workflow on page 27
- Changes required to the legacy workflow on page 29
- Our recommended DITA workflow™ methodology on page 29

About workflow

A workflow is a model that is illustrated as a sequence of operations and that represents the real work accomplishments of a person, a group of people, or a concrete or abstract mechanism (for example, the flow of a high-level computer program).

The traditional (legacy) workflow processes typically used in technical writing organizations over the past several decades simply will not meet the challenges described in *Trends in global communication* on page 23. The following topics compare those legacy processes with our recommended DITA workflow™ methodology.

Legacy workflow

The workflows formerly used by information organizations to create and publish information reflect the relative isolation in which the individual organizations worked.

In the past, information organizations were responsible for "silos" of information, for example technical documentation, marketing fact sheets, and training material. The organizations themselves often reported to different parts of a company, for example with the technical documentation team reporting to product engineering and the training department reporting to marketing.

Examples of information silos are shown in the following figure.
The key characteristics of the legacy (traditional) technical documentation department workflow are illustrated in the following figure.

The process is similar to the "waterfall" software development cycle, which has long been out of favor. Planning is done on a release-by-release basis, with focus primarily on product changes rather than information quality or usability. Writers were typically assigned ownership of one or more books, and the writer and department manager worked out the details of the plan book-by-book, with little collaboration among writers or discussion with other information departments.

This workflow process is of little use in planning information on the scale typical of today's globally linked community of information providers and users.
Changes required to the legacy workflow

For organizations to achieve DITA success they need to make the kinds of changes to their legacy workflow shown in the following diagram.

This process covers a broader scope of responsibility, and features a more collaborative and strategic focus.

Our recommended DITAworkflow™ methodology

The workflow we recommend for DITA projects is a staged, iterative planning methodology based on the agile software development process.
The steps are described in the following topics.

**Origin of the methodology**

The methodology originated in our own experience in creating DITA and DITA Open Toolkit documentation through several years and over 10 editions. We refined the methodology as we worked with our DITA clients. We looked for an information development process that:

- Promotes the key themes of **collaboration, integration, iteration, and reuse**
- Is simple to understand and implement
- Works for organizations of any size and type
- Can begin "wherever you are today"

**Features and benefits of the methodology**

The key benefits are:

- Simple, six-step, iterative process
- Sequenced, but you can "jump in" anywhere
  
  This is an important point. Even if you haven't yet determined your ROI objectives or done a complete analysis of your current documentation, you can still begin with a preliminary strategy, plan, and prototype. The information you gain doing that and through demos and discussions will allow you to "circle back" to step 1 and do more meaningful ROI planning on the second iteration of the workflow process.
- Can mesh well with most corporate strategies (business, marketing, technical)
- Can produce "demo-able" content with the first iteration (and the demo can be a very effective vehicle for communicating with department members, management, and related organizations)
- Working through the sequence of steps can provide constant progress toward your ultimate strategic (ROI) objectives
Step 1: Evaluate DITA

Keyword tags: assessing, DITAmashup, DITAworkflow, evaluating, evaluating DITA, evaluating DITA Open Toolkit, implementing, piloting, planning, prototyping, workflow

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?"

One of the key goals of this document is to help you get answers to your questions and explore the possibilities.

Step 2: Create ROI objectives

Keyword tags: assessing, DITAmashup, DITAworkflow, evaluating, implementing, piloting, planning, prototyping, ROI objectives, ROI objectives, workflow

In step 2 of the methodology, you create your ROI objectives:

- In the "DITA Maturity Model" (pictured below), where do you want to get ultimately?
- Where do you believe you are today?

What is the DITA Maturity Model?

You can read about the DITA Maturity Model in an excellent white paper by Michael Priestley and Amber Swope that is available on the JustSystems website: justsystems.com. The Maturity Model graphic is shown below.

The idea of the Maturity Model is that you begin DITA implementation at stage 1 by creating topics (to replace the narrative-style documentation that probably dominates your legacy documentation). As you move up to the next stage (stage 2, scalable reuse), you need to make an additional investment, but you collect a higher return (ROI).

When you do your up-front strategic planning you need to estimate where you expect to be in a few years (for example, at stage 4, automation and integration), and then figure out how, how much, and when to invest to achieve your desired return.
Step 3: Analyze your current documentation

Step 4: Create a documentation strategy and plan

Step 5: Create a prototype implementation
A DITAmashup is a structured DITA project that includes unstructured content from one or more information sources external to the structured topics. The DITA structured framework provides organizational and semantic integrity to the entire body of information, both structured and unstructured.

The key advantage of such an information collection is that a large body of high-value, related content can be assembled and presented to users without the cost and effort of a total DITA conversion.

As an example, suppose the documentation group working for a software company pulls together ("mashes up") the "official" DITA-based documentation for their software product along with unstructured content produced by their sister training and support organizations and a knowledge base of user-produced tips contributed to the support website. In assembling this eclectic but useful collection of information they apply the semantics associated with their structured DITA project and then publish the entire mashup to a website accessible to internal and external users alike.

Various platforms could be used as host sites for mashups. As our first two we chose WordPress and Drupal. We were looking for platforms with the following characteristics:

- Platforms that support XHTML, which is one of the standard output types of the DITA Open Toolkit
- Platforms that support the web services protocol XML-RPC, which allows us to programmatically post-process and publish the DITA-based XHTML output as a structured, interconnected collection of topics
- Well-established, popular platforms that provide editing, publishing, and workflow capabilities for the unstructured content, and also provide additional support (for example, search) to the structured content participating in the solution

Our DITAmashups are hosted on our two DITAinformationcenter sites:

- WordPress site: XMLdocs.info
- Drupal site: DITAinfo.info

Step 6: Implement your design (in stages)

Keyword tags: assessing, DITAmashup, DITAworkflow, evaluating, implementing, implementing your design, piloting, planning, prototyping, staged implementation, workflow

In step 6, you implement the prototype design using a staged approach.

It would be rare for an organization to be able to do a full DITA implementation in a single product release. For your first implementation stage, choose a project that will be a good learning exercise and highly visible, but not critical to the success of the product.

Finally begin the next iteration of the cycle. Repeat until you achieve your objectives, and then look at other ways you can improve your documentation or broaden your solution.
Chapter 5

Key concepts

Keyword tags: key concept

The following topics contain information about some of the key concepts required to understand and use DITA and DITA Open Toolkit.

Sections in this topic:

- About DITA on page 35
- About DITA Open Toolkit on page 37
- About DITA projects on page 38
- About processing and publishing on page 45

About DITA

Keyword tags: key concept

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

What is DITA's origin and history?

DITA originated and is extensively used in the IBM Corporation; in 2005 it was adopted as a standard by OASIS (Organization for the Advancement of Structured Information Standards). DITA is currently used in many organizations worldwide, and it 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.

How is DITA used?

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

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

What are the benefits of using DITA?

Some of the commonly referenced benefits of using DITA include:

- Standards-based solution
• Single source, publish to many output targets
• Topic-based, information-typed solution that promotes content reuse across documents and document sets
• Ideally suited to increased customization (for example, by audience)
• Can be integrated with a product being documented and other technologies (for example, localization tools)
• Promotes the kind of collaboration among participating groups that improves quality and increases content reuse across organizations
• Quicker turnaround
• Scalable solution
• Reduced costs (over time)

Why is "Darwin" in the name of the DITA architecture and DITA Open Toolkit?
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 (Document Type Definitions) 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?
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.

What is a DITA project?
A typical DITA project is a collection ("repository") of source, output, and processing files, as illustrated in the following figure.
About DITA Open Toolkit

Keyword tags: DITA Open Toolkit, dita4publishers, DocBook, Eclipse content, Eclipse help, EPUB, HTML Help, JavaHelp, key concept, PDF, PDF, Portable Document Format, troff, Word RTF, XHTML

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

DITA Open Toolkit publishes to the following environments:

- Key targets
- XHTML
Plugins allow you to publish to additional environments, for example to ePub format using DITA for Publishers (dita4publishers), which was contributed by Eliot Kimber.

DITA Open Toolkit is a processing tool; it does not have DITA source file authoring capabilities.

DITA Open Toolkit can be used to process DITA source files independent of your DITA authoring tool.

Some DITA authoring tools have the Toolkit bundled with their editing software; others use proprietary processing software.

### About DITA projects

Keyword tags: key concept, project component

Sections in this topic:

- **Source files** on page 38
- **Processing files** on page 41
- **Output files** on page 42

### Source files

Keyword tags: key concept, project component, source file

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

```xml
<taskbody>
  <context>
    <p>Your municipality collects garbage from homes once a week, usually early in the morning.</p>
  </context>
  <steps>
    <step><cmd>Find out from the town what day of the week garbage is collected in your neighborhood.</cmd></step>
    <step><cmd>The night before collection, place the garbage cans on the curb.</cmd></step>
    <step><cmd>After the garbage has been collected, move the cans back into your garage.</cmd></step>
  </steps>
</taskbody>
```
The following figure shows the image that appears in the "washing the car" task. This is a `.png` file. The Toolkit can process `.png`, `.jpg`, and `.gif`.

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<ditaval>
  <prop att="otherprops" val="oil" action="exclude"/>
  <prop att="otherprops" val="snow" action="exclude"/>
</ditaval>
```

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

The following figure shows the hierarchy ditamap.

The following figure shows the ditamap that processes the topics as a sequence.
The following figure shows a portion of the (optional) batch script used to initiate Ant processing of the garage sample.

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

```
<project name="garage" default="xhtml">
  <!-- This message is echoed to the screen at the beginning of the build process -->
  <!-- To process using a different map, you need to change the following statement -->
  <echo>&lt;/echo>
  <echo>Building garage sample as a hierarchy of topics.&lt;/echo>
  <echo>&lt;/echo>
  <property environment="env"/>
  <property name="toolkit_dir" value="${DITA_DIR}"/>
</project>
```

Output files

Keyword tags: key concept, output file, project component

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

<table>
<thead>
<tr>
<th>Garage tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>To change the oil in your car</td>
</tr>
<tr>
<td>To organize your workbench and tools</td>
</tr>
<tr>
<td>To shovel snow</td>
</tr>
<tr>
<td>To take out the garbage</td>
</tr>
<tr>
<td>To spray-paint</td>
</tr>
<tr>
<td>To wash the car</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Garage concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lawnmower</td>
</tr>
<tr>
<td>Oil</td>
</tr>
<tr>
<td>Paint</td>
</tr>
<tr>
<td>Shelving</td>
</tr>
<tr>
<td>Snow-shovel</td>
</tr>
<tr>
<td>Toolbox</td>
</tr>
<tr>
<td>Tools</td>
</tr>
<tr>
<td>Water-hose</td>
</tr>
<tr>
<td>Wheelbarrow</td>
</tr>
<tr>
<td>Workbench</td>
</tr>
<tr>
<td>Windshield-washer fluid</td>
</tr>
</tbody>
</table>

The following figure shows the HTML Help table of contents for the garage sample processed using the hierarchy Ant script.
The following figure shows the HTML Help index for the garage sample.
The following figure shows a portion of the PDF output for the garage sample.

The following figure shows a portion of the PDF output for the garage sample.

**About processing and publishing**

Keyword tags: key concept, processing, publishing

Sections in this topic:

- *Processing and publishing overview* on page 46
- *About XML* on page 47
Processing and publishing overview

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

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

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

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

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

The following figure shows a simple, annotated XML file.

![XML Diagram](image)

About DTDs

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

DTDs are not written in XML syntax.

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

Keyword tags: key concept, schema

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

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

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

The following figure shows the schema for DITA Concept.

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

About XSLT

Keyword tags: eXtensible Stylesheet Language Transformations, key concept, stylesheet, XSLT

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

XSLT stylesheets are written in XML syntax.
In the final stage of processing, the DITA Open Toolkit runs XSLT stylesheet transforms (using either SAXON or Xalan, and SAXON is the default and packaged with the Toolkit) to produce output files. In certain cases, it is possible to override stylesheet processing to customize the output.

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

```
<xsl:template match="*" mode="process.note">
  <div class="note">
    <xsl:call-template name="commonattributes"/>
    <xsl:call-template name="setidname"/>
    <span class="notetitle">
      <xsl:call-template name="getString">
        <xsl:with-param name="stringName" select="'Note'"/>
      </xsl:call-template>
      <xsl:call-template name="getString">
        <xsl:with-param name="stringName" select="'ColonSymbol'"/>
      </xsl:call-template>
    </span>
    <xsl:call-template name="flagit"/>
    <xsl:call-template name="revblock"/>
  </div>
</xsl:template>
```

**About XPath**

Keyword tags: key concept, XPath

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

XPath is a W3C standard and is used in XSLT.

The following figure shows XPath expressions in an XSLT document.

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

**About Java**

Keyword tags: Java, key concept

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

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

The following figure shows a Java code example (Toolkit code).
package org.dita.dost.pipeline;

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

public class PipelineFacade implements AbstractFacade {

    public PipelineFacade() {
    }
}
Chapter 6

Installing the DITA Open Toolkit

Keyword tags: DITA Open Toolkit, installing

The following topics contain information on how to install and upgrade DITA Open Toolkit on Windows, Linux/Unix, and Mac OS.

Sections in this topic:

Installation overview on page 51
Installing the Toolkit on page 53
Verifying the Toolkit installation on page 53
Directories and files in the ditaot directory on page 54

Installation overview

Keyword tags: Ant build processor, DITA Open Toolkit, downloading the Toolkit, FO processor, ICU globalization routines, icu4j globalization routines, installing, installing the Toolkit, Linux, Mac OS, Unix, upgrading the Toolkit, Windows, Xalan XSLT processor, XML catalog resolver

DITA Open Toolkit distributions

DITA Open Toolkit is distributed on its own and as part of other distributions, for example DITA-aware editors like Adobe FrameMaker and SyncRO Soft oXygen.

System requirements

Depending on which distribution of the Toolkit you are using, you may need to install other components as a separate operation.

Tools you need to process DITA files using the DITA Open Toolkit:

• The Java SDK (Software Development Kit, often called JDK, or Java Development Kit). The JRE (Java Runtime Environment) is sufficient.

Note: Prior versions of the Toolkit required the complete JDK. For more information, see the release notes for the Toolkit version you are using.

A version of the JDK compatible with your Toolkit version must already be in your operating environment, or you must install it separately.

• The DITA Open Toolkit itself
• Tools called by the Toolkit during processing (all of which are part of the full package distribution), for example:
  • Ant build processor
  • XML catalog resolver
  • ICU (icu4j) globalization routines
- SAXON XSLT processor
- Shell scripts for setting the necessary runtime environment variables
- Apache FOP (which you will need if you are creating PDF output)

**Download locations**

DITA Open Toolkit: [http://sourceforge.net/projects/dita-ot](http://sourceforge.net/projects/dita-ot)

Java Software Development Kit (SDK) (JDK) (Sun version): [http://java.sun.com](http://java.sun.com)


**Toolkit distributions**

There are three distributions of the Toolkit:

- **full package distribution**
  The full package distribution consists of the Toolkit and all tools called by the Toolkit during processing. We recommend installing the full package distribution of the Toolkit, especially if you are new to DITA and the Toolkit.

- **small package distribution**
  The small package distribution contains only the Toolkit. You must separately install all the other required tools to create a functioning build environment. This distribution is the one typically used to embed the Toolkit in other products. You might also choose to install this distribution if you are upgrading from a prior version of the Toolkit and want to leave part of your environment intact.

- **source distribution**
  The source distribution contains the source and executable code for the Toolkit. You might choose to download this distribution if you need to modify Toolkit Java code or if you want a detailed look at how the Toolkit works.

**Installation considerations**

The tools you use to edit or process DITA files 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. Be sure you understand the Toolkit system requirements and your current operating environment before installing or upgrading the Toolkit.

Tool incompatibilities may require you to move or uninstall one or more tools in your current environment before installing the Toolkit and its prerequisites.

**Upgrade considerations**

Before upgrading to a new version of the 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 update you made to the catalog-dita-template.xml file
- XSLT stylesheets you have added to the xsl directory to override the standard stylesheets
- Plug-ins you have installed
- Catalogs for XML editors

**Optional processing tools**

If you plan to process DITA files to HTML Help (.chm), you will probably want to install the Microsoft HTML Help processor. For information: [http://www.microsoft.com](http://www.microsoft.com) and search the downloads.
If you plan to process DITA files to Eclipse, you will probably want to install the IBM Eclipse content processor. For information: http://www.eclipse.org and search the downloads.

If you plan to process DITA files to JavaHelp, you will probably want to install the Sun JavaHelp processor. For information: http://java.sun.com/javase/technologies/desktop/javahelp/ and search the downloads.

Optional authoring tools

The DITA Open Toolkit is a command-line processing tool and does not have any authoring capabilities.

You can author DITA topics in a plain text editor or any number of "DITA-aware" editors on the market today. In general, it is much easier to author using a tool that enforces the rules of the DTD set you are planning to use.

Some of the authoring tools contain an embedded version of the Toolkit, which may not be the latest Toolkit version available.

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

Installing the Toolkit

Keyword tags: DITA Open Toolkit, installing, installing the Toolkit

We recommend installing the full package distribution of the Toolkit, especially if you are new to DITA or the Toolkit. To install:

2. Unzip or untar the package into the location of your choice.
   The unzip or untar process puts the file into a "DITA-OT" directory. The directory name includes the release number (for example, DITA-OT1.5).
3. (Recommended, on Windows) Create a shortcut for the startcmd.bat file that opens a command shell window where you can run Toolkit builds.

Verifying the Toolkit installation

Keyword tags: DITA Open Toolkit, installing, verifying the Toolkit installation

To verify that the Toolkit has installed properly:

1. Open the processing shell (startcmd.bat or startcmd.sh).
2. At the command prompt, enter ant -f build_demo.xml.
   After a series of processing messages (press Return for each prompt) you should receive a BUILD SUCCESSFUL message, and a new output directory structure (\out\samples\web) should appear in the DITA-OT directory. The out directory contains output from the garage sample. (This is not the same garage sample as the one that accompanies this document.)
3. (If the garage sample processed correctly) Process to an additional set of target output types with the command ant -f build_demo.xml all.
   After a longer series of processing messages, you should receive another BUILD SUCCESSFUL messsage, and the out directory will have additional subfolders containing output for more target types (for example, PDF). By comparing the output with the DITA source samples in the Toolkit distribution, you can get an idea about how Toolkit processing works.
When you have installed DITA Open Toolkit, the following directories and subdirectories should be in your DITA-OT (root) directory.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DITA-OT (root)</td>
<td>System-level Ant scripts and other system files (for example, 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; do not modify them.</td>
</tr>
<tr>
<td>css</td>
<td>Sample CSS (Cascading Style Sheet) files.</td>
</tr>
<tr>
<td>demo</td>
<td>Specializations, plug-ins, and validators that demonstrate extensions to the base DITA language. Many of the subdirectories have README files that provide information about how to use the specializations.</td>
</tr>
<tr>
<td>doc</td>
<td>DITA documentation, including language reference and application notes.</td>
</tr>
<tr>
<td>dtd</td>
<td>Core DITA definitions in XML DTD format.</td>
</tr>
<tr>
<td>lib</td>
<td>Contains dost.jar, the executable .jar file, and other .jar files it depends on.</td>
</tr>
<tr>
<td>out</td>
<td>Sample file output created during the installation verification process.</td>
</tr>
<tr>
<td>plugins</td>
<td>DITA Open Toolkit plug-ins.</td>
</tr>
<tr>
<td>resource</td>
<td>Miscellaneous resource files, including the default (common) CSS files and error messages.</td>
</tr>
<tr>
<td>samples</td>
<td>Sample DITA source files and Ant scripts.</td>
</tr>
<tr>
<td>schema</td>
<td>Core DITA definitions in XML Schema format.</td>
</tr>
<tr>
<td>temp</td>
<td>Temporary files created during the installation verification process.</td>
</tr>
<tr>
<td>tools</td>
<td>Ant processor files.</td>
</tr>
<tr>
<td>xsl</td>
<td>Core and process-specific stylesheets:</td>
</tr>
<tr>
<td></td>
<td>• The common subdirectory contains stylesheets that can be used by any process (for example, internationalization).</td>
</tr>
<tr>
<td></td>
<td>• The docbook subdirectory contains stylesheets used in converting DITA source content into DocBook source.</td>
</tr>
<tr>
<td></td>
<td>• The preprocess subdirectory contains code for conditional, conref, and link resolution.</td>
</tr>
<tr>
<td></td>
<td>• The troff subdirectory contains stylesheets used in converting DITA source content into troff source.</td>
</tr>
<tr>
<td></td>
<td>• The xslfo subdirectory contains code to support the processing of Formatting Objects (FO) output.</td>
</tr>
<tr>
<td></td>
<td>• The xslhtml subdirectory contains code to support XHTML processing.</td>
</tr>
<tr>
<td></td>
<td>• The xslrtf subdirectory contains code to support RTF processing.</td>
</tr>
</tbody>
</table>
Chapter 7

Setting up your working environment

Keyword tags: setting up

The following topics contain 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 DTD or schema URLs on page 55
- Setting up your source and output file directories on page 55

Configuring your authoring tool to use DTD or schema URLs

Keyword tags: authoring tool, editor (authoring tool), online DTDs and schemas, setting up, setting up a working environment

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 DITA concept source file referencing online DTDs:

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

The DTD and schema files for the DITA 1.0 language standard are located at http://docs.oasis-open.org/dita/v1.0.1.

The DTD and schema files for the DITA 1.1 language standard are located at http://docs.oasis-open.org/dita/v1.1/OS.

Setting up your source and output file directories

Keyword tags: setting up, setting up a working environment

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 `DITA-OT` 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.

The following figure shows how the source and output files for the DITAInformationCenter documentation are organized.
The following topics contain information on how to process (build) and publish DITA documents.

Sections in this topic:
- *About Ant* on page 58
- *About Ant scripts* on page 58
- *Processing overview* on page 59
- *Ant processing parameters* on page 60
- *Processing to the key Toolkit output targets* on page 65

**Assumed working environment**

These topics assume the working environment illustrated in the following figure.
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 transformation scripts.

Ant must be installed in your DITA processing environment for DITA Open Toolkit to function. Ant is included in the Toolkit full easy install distribution.

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 DITA-OT 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 also to modify for your own use.

### Main system scripts in DITA Open Toolkit

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

Sample Ant scripts for the garage and grocery shopping samples, as well as for this document, are included in the DITAinformationcenter package. One of the sample scripts processes the garage sample source files with the topics displayed in a hierarchy. The other processes the garage sample with the topics displayed as a sequence. One of the targets in the hierarchy script does conditional processing.

The following example shows invocation of the script that publishes the garage topics in a hierarchy.

ant -f garage_hierarchy_all.xml xhtml

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

Processing overview

Keyword tags: building, building DITA file, processing, processing DITA file, publishing, transforming

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

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

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

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

Keyword tags: Ant parameter, building, building DITA file, processing, processing DITA file, publishing, transforming

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 any of the sample Ant scripts in the DITAinformationcenter sample packages.

<table>
<thead>
<tr>
<th>Parameter (*Required), Target</th>
<th>Definition, Usage</th>
<th>Valid values, Default, Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>args.artlbl</code></td>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml Adds annotation to images showing the filename of the image. Useful for pre-publishing editing.</td>
<td>Valid: yes or no Default: no</td>
</tr>
<tr>
<td><code>args.copycss</code></td>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml Whether to copy user-specified CSS file(s) to the directory specified ( {args.outdir}^{(args.csspath)} ).</td>
<td>Valid: yes or no Default: no</td>
</tr>
<tr>
<td><code>args.css</code></td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
| `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 |
<p>| <code>args.cssroot</code> | 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 <code>args.cssroot</code>. | |</p>
<table>
<thead>
<tr>
<th>Parameter (*Required), Target</th>
<th>Definition, Usage</th>
<th>Valid values, Default, Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>args.dita.locale</strong>&lt;br&gt;These targets only: htmlhelp and javahelp</td>
<td>Locale used for sorting indexterms. If no locale is specified, the first occurrence of &quot;xml-lang&quot; is used as the default locale.</td>
<td>Default (If &quot;xml-lang&quot; is not specified): en-us</td>
</tr>
<tr>
<td><strong>args.draft</strong>&lt;br&gt;All targets</td>
<td>Include draft and required cleanup content (that is, items identified as left to do before publishing).</td>
<td>Valid: yes or no&lt;br&gt;Default: no</td>
</tr>
<tr>
<td><strong>args.eclipsecontent.toc</strong>&lt;br&gt;Target: eclipsecontent only</td>
<td>Root file name of the output Eclipse content toc file.</td>
<td>Default: name of the source ditamap file</td>
</tr>
<tr>
<td><strong>args.eclipsehelp.toc</strong>&lt;br&gt;Target: eclipsehelp only</td>
<td>Root file name of the output Eclipse help toc file.</td>
<td>Default: name of the source ditamap file</td>
</tr>
<tr>
<td><strong>args.eclipse.provider</strong>&lt;br&gt;Target: eclipsehelp only</td>
<td>Provider name of the Eclipse help output.</td>
<td>Default: DITA</td>
</tr>
<tr>
<td><strong>args.eclipse.version</strong>&lt;br&gt;Target: eclipsehelp only</td>
<td>Version number of the Eclipse help output.</td>
<td>Default: 1.0</td>
</tr>
<tr>
<td><strong>args.fo.img.ext</strong>&lt;br&gt;Target: pdf only</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>args.fo.include.rellinks</strong>&lt;br&gt;Target: pdf only</td>
<td>Whether links will appear in the output files.</td>
<td>Valid: yes or no&lt;br&gt;Default: no</td>
</tr>
<tr>
<td><strong>args.fo.output.rellinks</strong>&lt;br&gt;Target: pdf only</td>
<td>Whether links will appear in the output files.</td>
<td>Valid: yes or no&lt;br&gt;Default: no</td>
</tr>
<tr>
<td><strong>args.fo.userconfig</strong>&lt;br&gt;Target: pdf only</td>
<td>Name of the configuration file for FOP processing.</td>
<td>Valid: yes or no&lt;br&gt;Default: no</td>
</tr>
<tr>
<td><strong>args.ftr</strong>&lt;br&gt;These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td>Path to the file containing XHTML to be placed in the body running-footer area of the output file. The file must be well-formed XML.</td>
<td>Example: <code>&lt;property name=&quot;args.ftr&quot; value=&quot;C:/sandbox/myftr.xml&quot;/&gt;</code></td>
</tr>
<tr>
<td><strong>args.gen.task.lbl</strong></td>
<td>An optional parameter that specifies whether to generate locale-based</td>
<td>Valid: yes or no</td>
</tr>
<tr>
<td>Parameter (*Required), Target</td>
<td>Definition, Usage</td>
<td>Valid values, Default, Examples</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>All targets</td>
<td>headings for <code>&lt;section&gt;</code> elements in task topics.</td>
<td></td>
</tr>
<tr>
<td><strong>args.hdf</strong></td>
<td>Path to the file containing XHTML to be placed in the header area of the output file. The file must be well-formed XML.</td>
<td>Example: <code>&lt;property name=&quot;args.hdf&quot; value=&quot;C:/sandbox/myhdf.xml&quot;/&gt;</code>.</td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.hdr</strong></td>
<td>Path to the file containing XHTML to be placed in the body running-header area of the output file. The file must be well-formed XML.</td>
<td>Example: <code>&lt;property name=&quot;args.hdr&quot; value=&quot;C:/sandbox/myhdr.xml&quot;/&gt;</code>.</td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.htmlhelp.includefile</strong></td>
<td>File to be included in the output.</td>
<td></td>
</tr>
<tr>
<td>Target: htmlhelp only</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.indexshow</strong></td>
<td>Whether indexterm entries should display in the output text. Makes it possible to see what has been indexed in a pre-publishing review.</td>
<td>Valid: yes or no Default: no</td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.input</strong></td>
<td>Path and name of the input file. Use the same case as the filename.</td>
<td>Example: <code>&lt;property name=&quot;args.input&quot; value=&quot;mastermap.ditamap&quot;/&gt;</code>.</td>
</tr>
<tr>
<td>All targets</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.javahelp.map</strong></td>
<td>Root file name of the output JavaHelp map file.</td>
<td>Default: name of the input ditamap file</td>
</tr>
<tr>
<td>Target: javahelp only</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.javahelp.toc</strong></td>
<td>Root file name of the output JavaHelp toc file.</td>
<td>Default: name of the input ditamap file</td>
</tr>
<tr>
<td>Target: javahelp only</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.outext</strong></td>
<td>Output file extension name for generated XHTML files. In most browser environments, either html or htm is acceptable.</td>
<td>Valid: html or htm Default: html</td>
</tr>
<tr>
<td>These targets only: eclipsehelp, htmlhelp, javahelp, or xhtml</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>args.logdir</strong></td>
<td>Directory used to store generated Ant log files. If you generate several outputs in a single build, the following rules apply:</td>
<td>Default: output directory specified by output.dir</td>
</tr>
<tr>
<td>All targets</td>
<td>• 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:</td>
<td></td>
</tr>
<tr>
<td>Parameter (*Required), Target</td>
<td>Definition, Usage</td>
<td>Valid values, Default, Examples</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------</td>
<td>---------------------------------</td>
</tr>
</tbody>
</table>
| • 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. |
| An optional parameter that specifies whether to include information about DITA class ancestry within the generated XHTML elements. | Valid: yes or no |
| Root file name of the output XHTML toc file. | Default: index |
| xsl transform file that will replace the default file:  
• For transtype="docbook", dita2docbook.xsl will be replaced.  
• For transtype="eclipsehelp" or transtype="xhtml": dita2xhtml.xsl.  
• For transtype="html" or transtype="javahelp": dita2html.xsl.  
• For transtype="pdf": dita2fo-shell.xsl.  
• For transtype="rtf": dita2rtfImpl.xsl. |
| Example: <property name="args.xsl" value="overrides.xsl"/> |
| args.xhtml.toc | Target: xhtml only |
| 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"/> |
| Whether to clean the temp directory before each build. | Valid: yes or no  
Default: no |
<table>
<thead>
<tr>
<th>Parameter (*Required), Target</th>
<th>Definition, Usage</th>
<th>Valid values, Default, Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dita.dir</code> All targets</td>
<td>Absolute path of the Toolkit's home directory.</td>
<td></td>
</tr>
</tbody>
</table>
| `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 (for example, `.ditamap`) you must specify the format attribute (for example, `format="ditamap"`) in your source file references. If you don't, you will get an error message. | Default: .xml in release 1.2; .dita in release 1.3.1  
Example: `<property name="dita.extname" value=".dita"/>` |
| `dita.input` All targets      | This parameter must be provided if `args.input` is not provided. It must be used together with the `dita.input.dirname` parameter. The result of this combination is equivalent to using only the `args.input` parameter. It is an alternative way to specify the path and name of the input file. |                                  |
| `dita.input.dirname` All targets | This parameter must be provided if `args.input` is not provided. It must be used together with the `dita.input` parameter. The result of this combination is equivalent to using only the `args.input` parameter. It is an alternative way to specify the path and name of the input file. |                                  |
| `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 |
| `generate.copy.outer` All targets | Controls generating/copying if files are referenced outside the top directory of the build. | Valid: 1 (only generate/copy inside parent dir), 2 (generate/copy in any dir), 3 (adjust input.dir based on reference files)  
Default: 1 |
| `onlytopic.in.map` All targets | Should files not referenced in ditamap be resolved. | Valid: true or false  
Default: false |
| `outer.control` All targets   | Controls warning messages if files are referenced outside the top directory of the build. | Valid: fail, warn, quiet  
Default: warn |
<table>
<thead>
<tr>
<th>Parameter (*Required), Target</th>
<th>Definition, Usage</th>
<th>Valid values, Default, Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>*output.dir</td>
<td>Path of the output directory.</td>
<td>Example: <code>&lt;property name=&quot;output.dir&quot; value=&quot;${projdir}/out/xhtml&quot;/&gt;</code></td>
</tr>
<tr>
<td>pdf.formatter</td>
<td>Specify FO processor to be used</td>
<td>Valid: fop, xep, ah Default: fop</td>
</tr>
<tr>
<td>retain.topic.fo</td>
<td>Controls whether the topic.fo file is erased after the output pdf file is generated.</td>
<td>Valid: yes or no Default: no</td>
</tr>
<tr>
<td>*transtype</td>
<td>Type of output to be produced.</td>
<td>Valid: docbook, eclipsecontent, eclipsehelp, htmlhelp, pdf, troff, wordrtf, or xhtml Example: <code>&lt;property name=&quot;transtype&quot; value=&quot;htmlhelp&quot;/&gt;</code></td>
</tr>
<tr>
<td>validate</td>
<td>Should input files be validated.</td>
<td>Valid: yes or no Default: no</td>
</tr>
</tbody>
</table>

### Processing to the key Toolkit output targets

Keyword tags: building, Eclipse, HTML help, PDF, processing, publishing, transforming, XHTML

Sections in this topic:

- [Processing to XHTML targets on page 65](#)
- [Processing to HTML Help targets on page 68](#)
- [Processing to PDF targets on page 66](#)
- [Processing to Eclipse targets on page 69](#)

### Processing to XHTML targets

Keyword tags: building, building DITA file, garage sample, processing, processing DITA file, publishing, transforming, XHTML

This topic assumes you have installed the DITA Open Toolkit and downloaded and unzipped the garage sample files to your C: root drive. We strongly recommend backing up the DITAgarage_SOURCE and DITAgarage_OUTPUT directories and their subdirectories before you begin processing.

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

1. Double-click the DITA Open Toolkit icon to open a command prompt.
2. In the Command Prompt window, cd to the root directory of the garage sample source files (DITAgarage_SOURCE/ant_scripts and invoke the garage_hierarchy_all Ant script (which processes the topics as a hierarchy of topics) or the garage_sequence_all Ant script (which processes the topics as a sequence of topics). For example: `ant -f ant_scripts/garage_hierarchy_all.xml dita2xhtml` or `ant -f ant_scripts/garage_sequence_all.xml dita2xhtml`
Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script `runbuild.bat`: `runbuild dita2xhtml hierarchy` or `runbuild dita2xhtml sequence`. The batch script is in the DITAgarage_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITAgarage_OUTPUT directory during processing.

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

3. After the XHTML file has processed successfully, cd to the DITAgarage_OUTPUT/hierarchy/xhtml or DITAgarage_OUTPUT/sequence/xhtml directory. The xhtml subdirectory should contain these directories and files:

```
- concepts
- images
- tasks
- commonlr
- commonrtl
- index
```

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

```
- Garage tasks
  - To change the oil in your car
  - To organize your workbench and tools
  - To shovel snow
  - To take out the garbage
  - To spray-paint
  - To wash the car
- Garage concepts
  - Lawnmower
  - Oil
  - Paint
  - Shelving
  - Snow-shovel
  - Toolbox
  - Tools
  - Water-hose
  - Wheelbarrow
  - Workbench
  - Windshield-washer fluid
```

**Processing to PDF targets**

Keyword tags: building, building DITA file, garage sample, PDF, processing, processing DITA file, publishing, transforming

This topic assumes you have installed the DITA Open Toolkit and downloaded and unzipped the garage sample files to your C: root drive. We strongly recommend backing up the DITAgarage_SOURCE and DITAgarage_OUTPUT directories and their subdirectories before you begin processing.
In general, the instructions in this topic assume the Windows environment; the procedure is very similar in other operating system environments.

1. Double-click the DITA Open Toolkit icon to open a command prompt.
2. In the Command Prompt window, cd to the root directory of the garage sample source files (DITAgarage_SOURCE/ant_scripts) and invoke the garage_hierarchy_all Ant script (which processes the topics as a hierarchy of topics) or the garage_sequence_all Ant script (which processes the topics as a sequence of topics).
   For example: `ant -f ant_scripts/garage_hierarchy_all.xml dita2pdf` or `ant -f ant_scripts/garage_sequence_all.xml dita2pdf`.
   Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): `runbuild dita2pdf hierarchy` or `runbuild dita2pdf sequence`. The batch script is in the DITAgarage_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITAgarage_OUTPUT directory during processing.

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

3. After the PDF file has processed successfully, cd to the DITAgarage_OUTPUT/hierarchy/pdf or DITAgarage_OUTPUT/sequence/pdf directory.
   The pdf subdirectory should contain these directories and files:
   - Configuration
   - Customization
   - images
   - hierarchy.pdf

4. Open the file hierarchy.pdf in a PDF reader to view the PDF output.
   The window should look something like this:
Processing to HTML Help targets
Keyword tags: building, building DITA file, garage sample, HTML Help, processing, processing DITA file, publishing, transforming

This topic assumes you have installed the DITA Open Toolkit and downloaded and unzipped the garage sample files to your C: root drive. We strongly recommend backing up the DITAgarage_SOURCE and DITAgarage_OUTPUT directories and their subdirectories before you begin processing.

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

1. Double-click the DITA Open Toolkit icon to open a command prompt.
2. In the Command Prompt window, cd to the root directory of the garage sample source files (DITAgarage_SOURCE/ant_scripts and invoke the garage_hierarchy_all Ant script (which processes the topics as a hierarchy of topics) or the garage_sequence_all Ant script (which processes the topics as a sequence of topics). For example: ant -f ant_scripts/garage_hierarchy_all.xml dita2htmlhelp or ant -f ant_scripts/garage_sequence_all.xml dita2htmlhelp

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

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

3. After the HTML Help file has processed successfully, cd to the DITAgarage_OUTPUT/hierarchy/htmlhelp or DITAgarage_OUTPUT/sequence/htmlhelp directory.

The htmlhelp directory should contain these directories and files:

Folder structures:
- images
- tasks
- commonlirr.css
- commonrtl.css
- hierarchy.chm
- hierarchy.hhc
- Index
- hierarchy.hhk
- hierarchy.hhp

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

The window should look something like this:
Processing to Eclipse targets

Keyword tags: building, Eclipse, processing, publishing, transforming

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

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

Sections in this topic:

Processing to Eclipse help targets (overview) on page 69
Processing to Eclipse help targets on page 70

Processing to Eclipse help targets (overview)
Keyword tags: building, Eclipse, processing, publishing, transforming

Eclipse help is one of the DITA Open Toolkit target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plugin. This allows the output to be displayed as Eclipse standalone help (that is, independent of the Eclipse IDE) or in an Eclipse Infocenter, which allows you to access Eclipse help on the web.

Eclipse help output is a set of files in an output directory that can be installed in an Eclipse distribution as a plugin.

To publish a DITA Eclipse help document, simply copy the entire DITA Eclipse output directory into the Eclipse plugin directory.
Processing to Eclipse help targets
Keyword tags: building, building DITA file, Eclipse help, garage sample, processing, processing DITA file, publishing, transforming

This topic assumes you have installed the DITA Open Toolkit and downloaded and unzipped the garage sample files to your C: root drive. We strongly recommend backing up the DITAgarage_SOURCE and DITAgarage_OUTPUT directories and their subdirectories before you begin processing.

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

1. Double-click the DITA Open Toolkit icon to open a command prompt.

2. In the Command Prompt window, cd to the root directory of the garage sample source files (DITAgarage_SOURCE/ant_scripts and invoke the garage_hierarchy_all Ant script (which processes the topics as a hierarchy of topics) or the garage_sequence_all Ant script (which processes the topics as a sequence of topics).
   For example: ant -f ant_scripts/garage_hierarchy_all.xml dita2eclipsehelp or ant -f ant_scripts/garage_sequence_all.xml dita2eclipsehelp

   Alternatively, if you are processing in the Windows environment, you can use the garage sample batch script (runbuild.bat): runbuild dita2eclipsehelp hierarchy or runbuild dita2htmlhelp sequence. The batch script is in the DITAgarage_SOURCE directory. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITAgarage_OUTPUT directory during processing.

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

3. After the Eclipse help file has processed successfully, cd to the DITAgarage_OUTPUT/dita.garagesample.hierarchy_1.0 or DITAgarage_OUTPUT/dita.garagesample.sequence_1.0 directory.
   The eclipselhelp subdirectory should contain these directories and files:

   ![Directory Structure]

4. Install the files in the output directory as an Eclipse plug-in. This involves copying the DITA Eclipse output directory (dita.garagesample.hierarchy_1.0 or dita.garagesample.sequence_1.0 into the Eclipse product's plugins directory. See the Eclipse product documentation for more information.

   The following figure shows the garage sample Eclipse file displayed in an Eclipse Infocenter.
To spray-paint

The garage is a good place to spray-paint.

1. So you will not get paint on your car, drive the car out of the garage.
2. Put newspaper, cardboard, or a drop-cloth on the garage floor.
3. Put the object you want to paint on the newspaper, cardboard, or drop-cloth.
4. To paint the object, follow the directions on the spray-paint can.
5. Before you move the object, let the paint dry thoroughly.

Parent topic: Garage tasks

Related concepts
Paint
Troubleshooting the build process

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

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

The following topics help you deal with the second and third cases, where you need to debug processing problems. They describe tools and mechanisms available to you in the Toolkit itself, other tools available for the Ant/Java environment, and various strategies you can apply to find and fix processing errors quickly.

Sections in this topic:

- Capturing and using the log on page 73
- Error messages generated during processing on page 74
- Debugging, reporting, repair, and file generation tools on page 76

Capturing and using the log

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

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

In order to troubleshoot a build problem, it is useful to capture the Ant build output in a log file and to control the type of output Ant puts in the log. Below is the `runbuild.bat` file we use to "kick off" processing for this document. It eliminates some of the less urgent messages producing during the Toolkit build process and diverts them to a log file.

```
ant -Dbasedir=%DITA_DIR% -f ant_scripts\DITAinformationcenter_all.xml
   -logger org.dita.dost.log.DITAOTBuildLogger %1
```

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 this document (along with subdirectories and other files in the root output directory).
Error messages generated during processing

Keyword tags: debugging, error message, troubleshooting, troubleshooting the build process

Sections in this topic:

- Error messages overview on page 74
- Messages generated from other sources on page 75

Error messages overview

Keyword tags: debugging, error message, troubleshooting, troubleshooting the build process

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\build.xml:59: The following error occurred while executing this line:
C:\sandbox\build.xml:127: [DOTA002F][FATAL] Invalid 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 59 of `build.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\doc\ditaug\concepts\access.dita, Element = draft-comment:1)
[xslt] [DOTX040I][INFO]: Draft comment area found.
If the output is only used as a draft, you do not need to do anything.
If you are producing production-level output, you should not use the /DRAFT option.
```

The Meaning of Message Type (severity)

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

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

Messages generated from other sources

Keyword tags: debugging, error message, troubleshooting, troubleshooting the build process

Other Java messages

<table>
<thead>
<tr>
<th>Message text</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>[pipeline] [Error] :13:39: Element type &quot;...&quot; must be declared.</td>
<td>An error has occurred parsing a DTD.</td>
</tr>
<tr>
<td>[pipeline] [Error] :14:13: The content of element type &quot;...&quot; must match &quot;...&quot;.</td>
<td>An error has occurred parsing a DTD.</td>
</tr>
<tr>
<td>BUILD FAILED C:\sandbox\ant\dotug_xhtml.xml:24: The</td>
<td>Java does not have enough memory allocated to run the build. Change ANT_OPTS to a larger</td>
</tr>
</tbody>
</table>
### Message text and Action

<table>
<thead>
<tr>
<th>Message text</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>following error occurred while executing this line: C:sandbox\build.xml:101: The following error occurred while executing this line: java.lang.OutOfMemoryError</td>
<td>value, for example, ANT_OPTS=-Xmx600M. (The default value is 512M.)</td>
</tr>
<tr>
<td>Unable to instantiate specified logger class org.dita.log.DITAOTBuildLogger ...</td>
<td>Check that your CLASSPATH variable contains dost.jar.</td>
</tr>
<tr>
<td>Can't find resource\messages.xml</td>
<td>Check that your CLASSPATH variable contains dost.jar.</td>
</tr>
</tbody>
</table>

### Debugging, reporting, repair, and file generation tools

**Keyword tags:** debugging, debugging tool, file generation tool, PHP tool, Python tool, repair tool, reporting tool, tool set, troubleshooting

**Sections in this topic:**
- About the tools on page 76
- Using the tools on page 77

#### About the tools

**Keyword tags:** debugging, debugging tool, ditaauthors.py, ditadebug.php, ditaedit.php, ditaids.py, ditakeywords.py, ditalinks.py, ditarepair.py, ditastat.py, ditaunused.py, message topic generation tool, reporting tool, troubleshooting, troubleshooting the build process

These tools are useful in the DITA Open Toolkit publishing environment, written in the PHP or Python programming languages, and meant to be invoked either from a command line prompt or from an Ant build script.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ditaauthors.py</td>
<td>Returns a set of unique author and copyright strings, with counts for each.</td>
</tr>
<tr>
<td>ditadebug.py</td>
<td>Provides information useful in debugging processing errors.</td>
</tr>
<tr>
<td>ditaedit.php</td>
<td>Can be used to search for and perform bulk updates in all files in one or more DITA maps.</td>
</tr>
<tr>
<td>ditaids.py</td>
<td>Produces an alphabetical list of IDs for all files in one or more DITA maps, with duplicates marked with an asterisk.</td>
</tr>
<tr>
<td>ditakeywords.py</td>
<td>Produces a list of all metadata keywords defined.</td>
</tr>
<tr>
<td>ditalinks.py</td>
<td>Tests all URLs referenced by files in one or more DITA maps for validity.</td>
</tr>
<tr>
<td>ditarepair.py</td>
<td>Lists and optionally repairs incorrect references to other DITA topics.</td>
</tr>
<tr>
<td>ditastat.py</td>
<td>Lists the DITA elements used and the number of times each was used.</td>
</tr>
<tr>
<td>ditaunused.py</td>
<td>Produces a list of unused files.</td>
</tr>
</tbody>
</table>
Using the tools

Keyword tags: debugging, debugging tool, ditaauthors.py, ditaedit.php, ditaisds.py, ditakeywords.py, ditalinks.py, ditarepair.py, ditastat.py, ditaunused.py, message topic generation tool, reporting tool, troubleshooting, troubleshooting the build process

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 the DITAinformationcenter package.

DITA debugging tools (ditaedit.php and ditalinks.py)

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

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.

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.py

The ditalinks.py 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.

Cross-referencing problems handled by ditarepair.py

• 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.

For more information, see Repairing bad references with DITArepair

DITA reporting tools (ditaauthors.py, ditaisds.py, ditakeywords.py, ditastat.py, and ditaunused.py)

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?
• Which elements in a collection are used and how many times is each used?
• Which files in a collection are unused?

The ditaauthors.py, ditaisds.py, ditastat.py, and ditakeys.py tools were written to provide answers to these questions.
Software prequisites
To run some of 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.

To run some of these tools you must have the Python interpreter installed on your build machine. Python is a free tool that can be downloaded from http://www.python.org.

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

```xml
<!-- Create the ditamap debug cross-reference -->
<target name="ditadebug">
<echo>Building debugging cross-reference file ditadebug1.txt and ditadebug2.txt</echo>
<mkdir dir="${outdir}/debugreport_results"/>
<exec executable="${PythonExe}" dir="${projdir}" output="${outdir}/debugreport_results/ditadebug1.txt">
<arg value="${debugreporttoolsdir}/ditadebug.py"/>
<arg value="${MAP1_file}"/>
</exec>
<exec executable="${PythonExe}" dir="${projdir}" output="${outdir}/debugreport_results/ditadebug2.txt">
<arg value="${debugreporttoolsdir}/ditadebug.py"/>
<arg value="${MAP2_file}"/>
</exec>
</target>
```

Repairing bad references with DITArepair
Keyword tags: debugging, ditarepair.py, Python, troubleshooting

Problems with DITA references
The DITA standard encourages references between files in a topic set to promote reuse and to allow cross-linking of topics. If you have a set of DITA topics there are several common changes you might make to files that will invalidate references between files. This would include:

- Changing the filetype of a DITA file
- Changing the directory that some DITA files are located in
- Changing the topicid of a DITA file

Once you have made such a change, it can be time-consuming to find all the invalid references and fix them.

The ditarepair Python script
The ditarepair program is a Python script that scans all the DITA topics in a directory looking for invalid references to other files or to ids within other files. It lists those it finds and optionally fixes the problems by changing the references to have the proper value.

The ditarepair program can fix all three of the problems listed above. If the optional fix flag is specified, it will update bad references in source files as long as the change can be made unambiguously.

Here is output from running the script on an example. In this example the images_file directory was changed to images.

```
DITArepair: C:\temp\samples
bad references will be fixed
```
Processing 73 files.
Bad reference: C:\temp\samples\tasks\washingthecar.xml
   -> ..\image_files carwash.jpg
   change from: href ..\image_files/carwash.jpg
       to: href ..\images/carwash.jpg
writing 1 changes to C:\temp\samples\tasks\washingthecar.xml
DITAREpair: exit
Chapter 10

Creating DITA topics

Keyword tags: concept, reference topic, task, topic

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

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

Sections in this topic:

- About information types on page 81
- About topic authoring on page 82
- About the grocery shopping sample on page 82
- About topics on page 82
- Creating topics on page 83
- About concepts on page 83
- Creating concepts on page 84
- About tasks on page 85
- Creating tasks on page 85
- About reference information on page 86
- Creating reference topics on page 86
- Processing (building) a single topic on page 88

About information types

Keyword tags: concept, information type, reference topic, task, topic

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 do the following:

- 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 structures like tables for reference information, and simple sequences of steps for task information)
- Focus on tasks, which is what users are most 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 field to improve information quality. It is based on extensive research and experience, for example Robert Horn's Information Mapping.

The core information types in DITA are concept, task, and reference topic.

### About topic authoring

**Keyword tags:** concept, reference topic, task, topic, topic authoring

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

A document usually contains 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.

### About the grocery shopping sample

**Keyword tags:** concept, grocery shopping sample, reference topic, task, topic

The grocery shopping sample 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 and a runbuild batch file are also provided.

#### Creating and processing the grocery shopping sample

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

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

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

Ant scripts (in the ant_scripts directory) are provided in both the completed and working directories.

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

### About topics

**Keyword tags:** concept, reference topic, task, topic

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

Keyword tags: concept, reference topic, task, topic

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

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

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

```xml
<topic id="grocery_shopping" xml:lang="en-us">
<title>Shopping for groceries</title>
<shortdesc>Information about how to buy groceries.</shortdesc>
<prolog>
<author type="creator">Tom McIntyre</author>
<copyright>
<copyryear year="2010"/>
<copyrholder>Acme Company</copyrholder>
</copyright>
<critdates>
<created date="2010-August-21"/>
<revised modified="2010-August-21"/>
</critdates>
-metadata>
<keywords>
grocery shopping</keywords>
</metadata>
<body>
<!-- Container topics have no body content. -->
<p/>
</body>
</topic>
```

7. Save the changed file.

About concepts

Keyword tags: concept, reference topic, task, topic

A concept information type contains content of a conceptual nature.
Creating concepts

Keyword tags: concept, reference topic, task, topic

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

1. Go to the template/concepts directory.
2. Copy the about_cannedgoods.dita file to the "working" directory (working/concepts).
3. Using your authoring tool, open the "working" version of about_cannedgoods.dita.
4. In the prolog section, change the text of the author element text to your name, the copyrholder element text to your company name, and the revised element text to the current date.
5. In the conbody section, add a paragraph element with some text about canned goods, and an unordered list with some reasons for buying canned goods.
   Here is a suggestion:
   
   ```
   <p>Most people keep canned goods in their kitchen cupboards. Popular canned goods include beans, canned vegetables, and canned fruits.</p>
   <p>To save money on canned goods:
   <ul>
   <li>Buy from discount grocery stores</li>
   <li>Buy large cans instead of small cans</li>
   </ul>
   </p>
   ```
6. Save the changed file.
7. If you have problems creating or validating your working file, compare it with the file by the same name in the completed directory.
8. Go back to the template/concepts directory.
9. Copy the about_fruits_vegetables.dita file to the "working" directory (working/concepts).
10. Using your authoring tool, open the "working" version of about_fruits_vegetables.dita.
11. Edit the same prolog elements you did in the about_cannedgoods.dita file.
12. In the conbody section of about_fruits_vegetables.dita, add a paragraph element with some text about produce.
   Here is a suggestion:
   
   ```
   <p>To stay healthy, eat lots of fruits and vegetables. If possible, buy fresh fruits and vegetables instead of frozen or canned fruits and vegetables.</p>
   <p>You can save money by buying fruits and vegetables when they are 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 DITAgrocery_SOURCE directory. This topic assumes you are familiar with the information in About the grocery shopping sample.

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

   ```
   <context>Canned goods are usually stored on grocery-store shelves by type of food: for example, all canned vegetables will be in one aisle and all canned fruits in another aisle.</context>
   <steps>
   <step>
   <cmd>Read the grocery-store locator-signs or ask a grocery-store clerk to help you find the canned olives that are available.</cmd>
   </step>
   <step>
   <cmd>Find the type of olives you want to buy: green or black.</cmd>
   <info>If you are looking for ingredients for a green salad, buy green olives. If you are making enchiladas, buy black olives.</info>
   </step>
   <step>
   <cmd>Look at the can sizes and prices to get the best buy.</cmd>
   <info>If you plan to make enchiladas tonight and tacos on Friday, buy a large can.</info>
   </step>
   <step>
   <cmd>Select a can and look at it carefully to be sure it has no dents.</cmd>
   <info>A can with a dent might have a broken seal.</info>
   </step>
   <step>
   <cmd>Put the can in your shopping 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 selecting_fruits_vegetables.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>Select your fruits and vegetables after you have selected your canned goods. Otherwise, the heavy cans might bruise the fruits and vegetables in your shopping cart.</prereq>
    <context>Look for fruits and vegetables that are in season. In-season fruits and vegetables are fresher and cheaper. To select peaches:</context>
    <steps>
    <step>
    <cmd>Get a plastic bag or paper bag to hold the peaches.</cmd>
    </step>
    <step>
    <cmd>Select the best peaches and put them gently into your bag.</cmd>
    <info>Do not put more than 6 peaches in each bag.</info>
    </step>
    <step>
    <cmd>Put the bag gently into your shopping cart.</cmd>
    </step>
    </steps>
    <postreq>Be sure the check-out clerk 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

Keyword tags: concept, reference topic, task, topic

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

Keyword tags: concept, reference topic, task, topic

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

1. Go to the template/reference directory.
2. Copy the cannedgoods.dita file to the "working" directory (working/reference).
3. Using your authoring tool, open the "working" version of cannedgoods.dita.
4. In the prolog section, change the text of the author element text to your name, the copyrholder element text to your company name, and the revised element text to the current date.

5. In the refbody section, add a simple table showing product name, can size, and price for several canned goods products.

Here is a suggestion:

```dita
<section>
  <simpletable>
    <sthead>
      <stentry>Food product</stentry>
      <stentry>Size of can</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:

```dita
<section>
  <simpletable>
    <sthead>
      <stentry>Fruit or Vegetable</stentry>
      <stentry>Type</stentry>
      <stentry>Price</stentry>
    </sthead>
    <strow>
      <stentry>Apple</stentry>
      <stentry>Fuji</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

Keyword tags: building DITA file, concept, processing DITA file, reference topic, task, topic

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

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

Keyword tags: map

The topics in this section and the previous one are a short tutorial on how to create DITA maps to define content structure. Work through the topics section before the maps section (this one). The key concepts and tasks in this section are meant to be read and performed in the order presented.

Sections in this topic:

- About maps on page 89
- Creating maps on page 90
- Processing (building) the grocery shopping sample on page 92
- Processing using multilevel maps on page 92

About maps

Keyword tags: ditamap, 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: DITAmappericdfs

The following figure shows the topicrefs in the map for the grocery shopping sample.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "../dtd/map.dtd">
<map title="Grocery shopping" id="grocery_top">
  <topicref href="topics/groceryshopping.dita" type="topic"/>
  <topicref href="concepts/about_fruits_vegetables.dita" type="concept"/>
  <topicref href="concepts/about_cannedgoods.dita" type="concept"/>
  <topicref href="tasks/creating_fruits_vegetables.dita" type="task"/>
  <topicref href="tasks/buying_cannedgoods.dita" type="task"/>
  <topicref href="reference/fruits_vegetables.dita" type="reference"/>
  <topicref href="reference/cannedgoods.dita" type="reference"/>
</map>
```
Creating maps

Keyword tags: ditamap, map

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 DITAgrocery_SOURCE directories. This topic assumes you are familiar with the information in About the grocery shopping sample, and that you have created the topics according to the instructions in Topics.

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

Your working map file initially looks like this:

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

The topicref section of your file should look like this:

```xml
<topicref href="topics/groceryshopping.dita" type="topic">
  <topicref href="concepts/about_fruits_vegetables.dita" type="concept"/>
  <topicref href="concepts/about_cannedgoods.dita" type="concept"/>
  <topicref href="tasks/choosing_fruits_vegetables.dita" type="task"/>
  <topicref href="tasks/buying_cannedgoods.dita" type="task"/>
  <topicref href="reference/fruits_vegetables.dita" type="reference"/>
  <topicref href="reference/cannedgoods.dita" type="reference"/>
</topicref>
```

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

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

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

```xml
<!-- Relationship table -->
<!-- The related concept, task, and reference files point to each other -->
<reltable>
  <relheader>
    <relcolspec type="concept"/>
    <relcolspec type="task"/>
    <relcolspec type="reference"/>
  </relheader>
  <relrow>
    <relcell>
      <topicref href="concepts/about_fruits_vegetables.dita"/>
    </relcell>
    <relcell>
      <topicref href="tasks/choosing_fruits_vegetables.dita"/>
    </relcell>
    <relcell>
      <topicref href="reference/fruits_vegetables.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 conceptual "topic" (the topics would be "canned goods" and "fruits and vegetables") 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 fruits and vegetables. 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**

Keyword tags: ditamap, map

In this topic you will process (build) the map you created in *Creating maps*. You will be working in the DITAgrocery_SOURCE 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*, and that you have created the topics according to the instructions in *Topics*. If you need more information about Ant or Ant scripts, see *About Ant* or *About Ant scripts*.

1. Go to the working/ant_scripts directory.
2. Using your DITA authoring tool or a plain text editor, open the version of the Ant script you want to run. Process to the XHTML target environment.
3. Make sure the Ant script is set up correctly for your environment.
4. In the Command Prompt, navigate to the working directory.
5. Invoke the Ant script for the XHTML target.
   For example: `ant -f ant_scripts/grocery_all.xml dita2xhtml`

   Alternatively, if you are processing in the Windows environment, you can use the grocery shopping sample batch script (`runbuild.bat`); the command is `runbuild dita2xhtml`. The processing is the same, except the batch script filters some of the processing messages into the log file, which is created in the DITAgrocery_OUTPUT directory during processing.

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

6. Check the output directory to be sure the output files are correct.
7. Build the grocery shopping sample files to the HTML Help target (`ant -f ant_scripts/grocery_all.xml dita2htmlhelp` or `runbuild dita2htmlhelp`).
8. Build the grocery shopping sample files to the PDF target (`ant -f ant_scripts/grocery_all.xml dita2pdf` or `runbuild dita2pdf`).

---

**Processing using multilevel maps**

Keyword tags: ditamap, map, multilevel map

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

The following figures show a complete example.

The figure below shows part of the top-level map ("master map") for this document. Note that you need to include the `format="ditamap"` attribute statement in each reference to a lower-level map.
The figure below shows one of the lower-level maps (for the "introduction" chapter of this document). Notice that the topics containing content are nested within a container ("landing-page") topic (which contains no content). Container pages are not required for DITA projects, but they might help to group topics with related content.
Chapter 12

Migrating legacy content to DITA

Keyword tags: migrating, migrating content

The following topics contain information about migrating legacy content to DITA.

Sections in this topic:

- About content migration on page 95
- Migrating reference material programmatically on page 95
- Using Python to migrate tables to DITA on page 96
- For more information: Content migration on page 98

About content migration

Keyword tags: Adobe FrameMaker, FrameMaker editor, Mif2Go, migrating, migrating content, Omni Systems

A number of applications are available to help you migrate legacy content to DITA. The degree of success you will experience in using these tools will depend a great deal on the current state of your legacy content. Content that is not already structured will take a great deal more work than content that has already been published as a set of topics, for example an online help system.

One of the more widely used migration packages is Adobe FrameMaker. Framemaker supports editing DITA topics and map in structured view. It is also possible to convert a DITA map (and all the other files it references) into a FrameMaker book. Output can be produced by using the usual Framemaker output processing or by running the Toolkit build process from within FrameMaker.

OmniSystems makes a product called Mif2Go, which converts legacy FrameMaker files to DITA and other formats.

Migrating reference material programmatically

Keyword tags: legacy content, migrating, migrating content

A constant debate in the DITA community is how to begin a DITA project that will be based on legacy content. Is it better to start with an initial conversion (usually automated) to DITA and follow up with a content rearchitecture effort, or rearchitect first and then convert to DITA format?

As an information architect, I (Anna) have always favored the latter approach, which I realize usually results in a great deal of handwork to get the content back in place. On the other hand, I've seen many "automated" conversions that didn't do the whole job (or anything like the whole job), and which also required days, weeks, or even months of hand conversion of files anyway, to say nothing of the the follow-on architecture effort.

We recently did a project in which we used a modified approach that seemed to offer new possibilities we hadn't thought of before.
We began by separating out all the reference material, most of which was displayed in tables. The source files were in MadCap Flare, which is basically a superset of xhtml and supports mechanisms similar to DITA's links and conrefs, and which lends itself to the kind of customized Python programming described in *Automating DITA processes with Python*. In this case Dick's Python program extracted all relevant information from the MadCap Flare source files and automatically created DITA source files (reference topics) in more-or-less the same format as the original.

Removing the reference material (which tends to have tricky styling) from the legacy documentation makes traditional automated conversion of the more text-rich sections of a document more risk-free, and frees up more time for the architectural planning, which is so critical to the future success of a DITA project.

The same kind of treatment would also be possible using legacy output files, for example online help in standard xhtml format.

### Using Python to migrate tables to DITA

**Keyword tags:** legacy content, migrating, migrating content, migrating content with Python

**Sample Python script**

It is possible to write “one-off” Python scripts to convert reference tables in an XML format to DITA reference topics containing either table or simpletable elements. The following example describes such a script that reads a MadCap Flare source file, extracts table data from it, and writes out a DITA reference topic containing the same table entries.

The script pseudo code goes like this:

1. Read in the file and parse it with ElementTree
2. Loop through all the tables in the file by finding each table element.
3. Locate the tbody element in the table.
4. Loop through all rows in the table by finding each row element.
5. Loop through the td elements in the row and extract the contents of each cell into a list for the row.
6. (Optionally) sort the rows collected by a key column.
7. Create a new DITA reference topic and populate a table with the data just collected.
8. Write out the DITA reference topic containing the table to a new disk file.

It takes about 100 lines of Python code to extract the data from the source file and another 50 or so to write out the new file. Since Python is an interpreted language, it is easy to write and debug this code.

Here is an example of the Python code to create the DITA reference topic:

```python
# build the dita reference topic as output

# create a new XML tree for the DITA reference topic
refattrib = { "xml:lang" : "en-us", "id" : f"_ref" }
# add the reference element
ref = Element("reference", refattrib)
dita = ElementTree(ref)
# add a comment to the output listing the filename
# we are using as input
com = Comment("Based on "+os.path.basename(f))
ref.append(com)
# add a title to the topic
title = SubElement(ref,"title")
# set the title
title.text = f
# add refbody so we can begin inserting content into the topic
refbody = SubElement(ref,"refbody")
```
Sample DITA reference topic produced by the script

Here is part of a DITA topic produced by this script:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<reference id="U_S__State_Codes.htm_ref" xml:lang="en-us">
  <!-- Based on U_S__State_Codes.htm -->
  <title>U_S__State_Codes.htm</title>
  <refbody>
    <section>
      <simpletable>
        <strow>
          <stentry>Alabama</stentry>
          <stentry>AL</stentry>
        </strow>
        <strow>
          <stentry>Alaska</stentry>
          <stentry>AK</stentry>
        </strow>
        <strow>
          <stentry>Arizona</stentry>
          <stentry>AZ</stentry>
        </strow>
        <strow>
          <stentry>Arkansas</stentry>
          <stentry>AR</stentry>
        </strow>
        <strow>
          <stentry>California</stentry>
          <stentry>CA</stentry>
        </strow>
      </simpletable>
    </section>
  </refbody>
</reference>
```
For more information: Content migration

Keyword tags: migrating, migrating content, migrating legacy content to DITA

Presentation: Migrating to DITA
Silicon Valley DITA Interest Group presentation on Tips and Techniques on Migrating to DITA.

Paper: Migrating HTML to DITA
IBM paper.
https://www.ibm.com/developerworks/xml/library/x-dita8a/
Chapter 13

Linking your content

Keyword tags: linking

The following topics contain information on how to link DITA topics using cross-references (xrefs), related links, and relationship tables.

Sections in this topic:

About linking on page 99
Linking using cross-references (xrefs) on page 100
Linking using related links on page 100
Linking using relationship tables on page 101

About linking

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

How to decide which linking paradigm 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 new or changed links. 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.

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

Ensuring continuing link accuracy

To help ensure that our links continue to be accurate throughout the document production process, we run a link-checking tool (ditalink.php) we produced ourselves every time we do a major build. The tool checks both internal and external (URL) links. This tool is available as part of the DITAinformationcenter package.
Linking using cross-references (xrefs)

Keyword tags: cross reference, linking, linking content

The following example links a file in a "Getting started" topic to a file in an "Installation" topic.

<step>
<cmd>Install and become familiar with the authoring tool you plan to use to create DITA content.</cmd>

<info>
For information about choosing and installing an authoring tool, see topics in

<xref href="../installing/installing_overview.dita" scope="local">Installation overview</xref>.
</info>

</step>

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

Keyword tags: linking, linking content, related links

The following example from the garage processing tutorial 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

Keyword tags: linking, linking content, relationship table

The following figure shows the top of the relationship table for this document.

The following example shows the three-column relationship table for the grocery shopping sample.

```xml
<!-- Relationship table -->
<!-- The related concept, task, and reference files point to each other -->
<reltable>
  <relheader>
    <relcolspec type="concept"/>
    <relcolspec type="task"/>
    <relcolspec type="reference"/>
  </relheader>
  <relrow>
    <relcell>
      <topicref href="concepts/about_produce.dita"/>
    </relcell>
    <relcell>
      <topicref href="tasks/choosing_produce.dita"/>
    </relcell>
    <relcell>
      <topicref href="reference/produce.dita"/>
    </relcell>
  </relrow>
  <relrow>
    <relcell>
      <topicref href="concepts/about_cannedgoods.dita"/>
    </relcell>
    <relcell>
      <topicref href="tasks/buying_cannedgoods.dita"/>
    </relcell>
    <relcell>
      <topicref href="reference/cannedgoods.dita"/>
    </relcell>
  </relrow>
</reltable>
```
Reusing your content

The following topics contain information about various content reuse concepts and techniques.

Sections in this topic:

- About content reuse on page 103
- Reusing glossary content in other topics on page 103
- Getting started with content reuse on page 104

About content reuse

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

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

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

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

Some of the many opportunities for reuse in a DITA content repository include:

- Reusing glossary or core vocabulary topics as introductory content in longer concept topics (for example, "about" topics)
- Reusing legal, copyright, and other "boilerplate" content in multiple documents, document sets, or information centers
- Reusing generic reference information in multiple documents (for example, generic "for more information" topics)
- Including a multipurpose image library in a DITA source repository
- Including a variable library in a DITA source repository
- Creating a library of business- or marketing-sensitive material to be shared by multiple documents or collections of topics (for example, business-sensitive product feature names and descriptions)

Reusing glossary content in other topics

The DITA core vocabulary, which operates as a "super-glossary" in this document, has provided the authors many opportunities for content reuse. For example, the introductory portions of many "about" topics in the
document are "conref'ed" from the core vocabulary topics of the same names. For example, here is the core vocabulary content for the Ant topic:

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

Here is the conref that copies that content to the top of the About Ant topic in the Processing section of the document:

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

---

**Getting started with content reuse**

**Keyword tags:** content reuse, reuse, reusing

My (Anna's) recommendation is to include a "common" section of every new DITA repository that looks like it will ultimately contain more than a single document. I have never worked on a DITA project of any size that didn't benefit from such an approach, for example the DITAinformationcenter project has directories to hold the following kinds of common files:

- core_vocabulary
- gettinginformation
- images
- introduction
- legal_copyright
- processing_files
  - css_files
  - customization files for web and pdf output types
  - ditaval_files
  - relationship_tables
  - splash_pages
I've found that initially labeling such files as "common" helps me to actively look for opportunities to use them in that manner as I work through my early prototypes.
Customizing your content and environment

Keyword tags: customizing

The following topics contain information on these aspects of customizing your DITA and DITA Open Toolkit environment:

- How to expand access to your information through indexing, the use of metadata, and filtering (conditional processing)
- How to customize your published output using conditional processing, CSS, XSLT, and other methods
- How to create custom tools

Sections in this topic:

- **Customizing access to your information** on page 107
- **Customizing your output** on page 111
- **Creating custom tools** on page 123

Customizing access to your information

Keyword tags: accessing, customizing, customizing access

The following topics contain information on how to expand access to your information through indexing, the use of metadata, and filtering (conditional processing).

Sections in this topic:

- **About indexing** on page 107
- **About metadata** on page 109
- **About RDF and DITA Open Toolkit** on page 110

About indexing

Keyword tags: accessing information, customizing, indexing content

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

**Example**

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

The above code produces the following two-level index entry:

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

<keywords>
<keyword>XPath</keyword>
<indexterm>XPath</indexterm>
</keywords>

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

<keywords>
<keyword>evaluating DITA</keyword>
<keyword>evaluating DITA Open Toolkit</keyword>
<keyword>DITA authoring framework</keyword>
<keyword>DITA information development framework</keyword>
<keyword>DITA production framework</keyword>
<keyword>DITA maturity level</keyword>
<indexterm>evaluating DITA and DITA Open Toolkit</indexterm>
<indexterm>authoring framework</indexterm>
<indexterm>production framework</indexterm>
<indexterm>information development framework</indexterm>
<indexterm>components (base) of a DITA/Toolkit system</indexterm>
<indexterm>skills required for a DITA/Toolkit system</indexterm>
<indexterm>maturity levels</indexterm>
<indexterm>demo</indexterm>
<indexterm>pilot project</indexterm>
<indexterm>prototype</indexterm>
<indexterm>broad end-to-end system</indexterm>
<indexterm>enterprise system</indexterm>
</indexterm>
<indexterm>demo maturity level</indexterm>
<indexterm>pilot project maturity level</indexterm>
<indexterm>prototype maturity level</indexterm>
<indexterm>broad end-to-end system maturity level</indexterm>
<indexterm>enterprise system maturity level</indexterm>
</keywords>
About metadata
Keyword tags: accessing information, customizing, metadata

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

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

Example

```xml
<metadata>
  <keywords>
    <keyword>Ant script</keyword>
    <indexterm>Ant script</indexterm>
    <indexterm>definition</indexterm>
    <indexterm>usage</indexterm>
  </keywords>
  <prodinfo>
    <prodname>DITAinformationcenter</prodname>
    <vrmlist>
      <vrm version="4.0"/>
    </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, a prior version of this DITA topic contains the following metadata:

```xml
<prolog>
  <author type="creator">Anna van Raaphorst</author>
  <author type="contributor">Richard Johnson</author>
  <publisher>VR Communications, Inc.</publisher>
  <copyright>
    <copyryear year=""/>
    <copyrholder>VR Communications, Inc.</copyrholder>
  </copyright>
  <critdates>
    <created date="2009-July-18"/>
    <revised modified="2010-June-21"/>
  </critdates>
  <metadata>
    <keywords>
      <keyword>accessing information</keyword>
      <keyword>metadata</keyword>
    </keywords>
    <prodinfo>
      <prodname>DITAinformationcenter</prodname>
      <vrmlist>
        <vrm version="5.0"/>
      </vrmlist>
    </prodinfo>
  </metadata>
</prolog>
```
How the Toolkit processes metadata

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

```
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
<meta content="concept" name="DC.Type" />
<meta name="DC.Title" content="About metadata" />
<meta content="accessing information, metadata" name="DC.subject" />
<meta name="DC.Title" content="accessing information, metadata" name="keywords" />
<meta scheme="URI" name="DC.Relation" content="/accessing/accessing.html" />
<meta name="prodname" content="DITAinformationcenter" />
<meta content="5.0" name="version" />
<meta content="Anna van Raaphorst" name="DC.Creator" />
<meta content="Richard Johnson" name="DC.Contributor" />
<meta content="VR Communications, Inc." name="DC.Publisher" />
<meta name="copyright" content="VR Communications, Inc. 2010" type="primary" />
<meta name="DC.Rights.Owner" content="VR Communications, Inc. 2010" type="primary" />
<meta content="2009-July-18" name="DC.Date.Created" />
<meta content="2010-June-21" name="DC.Date.Modified" />
<meta content="XHTML" name="DC.Format" />
<meta content="aboutmetadata" name="DC.Identifier" />
<link href="/CSS/commonltr.css" type="text/css" rel="stylesheet" />
<link href="/CSS/DITAinformationcenter.css" type="text/css" rel="stylesheet" />
<title> About metadata</title>
</head>
```

About RDF and DITA Open Toolkit

Keyword tags: accessing information, customizing, Dublin Core metatag, OWL, RDF, SKOS

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

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

Dublin Core

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

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

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

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

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

SKOS (Simple Knowledge Organization System) is based on the RDF framework. The Thesaurus (aka Taxonomy) plug-in can be installed with DITA Open Toolkit to provide a DITA specialization that can be used to identify and process content based on what the information is about by generating SKOS output.

Customizing your output

Keyword tags: customizing, customizing output, output

The following topics contain information on how to customize your published output using conditional processing, CSS, XSLT, and other methods.

Sections in this topic:

- Conditional processing on page 111
- Overriding the standard XSLT stylesheets on page 114
- Customizing XHTML output on page 115
- Customizing PDF output on page 119
- Customizing Eclipse output on page 121

Conditional processing

Keyword tags: conditional processing, customizing

Sections in this topic:

- About conditional processing on page 111
- About filtering on page 113

About conditional processing

Keyword tags: accessing information, audience attribute, conditional processing, customizing, ditaval file, filtering content, metadata attributes, otherprops attribute, platform attribute, product attribute, rev attribute

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

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

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

You can exclude content based on its metadata. By default, all content is included.

You can flag content based on metadata. By default, no content is flagged.

You can show the revision information for your content. All revision information is hidden by default.

Attributes used for conditional processing

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

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

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

**Setting conditions**

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

Specify the filtering 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:

```xml
<ul>
    <li audience="admin programmer">
        This is important for lots of reasons.
    </li>
    <li audience="programmer">
        This is important only to programmers.
    </li>
    <li audience="programmer" platform="unix">
        This is important only to Unix programmers.
    </li>
    <li platform="unix">
        This applies to the Unix platform.
    </li>
</ul>
```

And given this filtering file:
You should get this output:

- This is important for lots of reasons.
- This applies on Unix.

The first list item remains because it applies to an administrator and programmer audience. The second list item is removed because it applies to programmers only, and programmer-specific information has been excluded. The third list item is removed because it applies to programmers as well: the fact that it applies to the Unix platform does not save it from exclusion. The fourth list item is flagged based on its platform attribute.

**About filtering**
Keyword tags: customizing, filtering content

**Garage sample files that illustrate filtering**

One of the Ant scripts for the garage processing sample uses filtering to exclude topics having to do with oil and snow. The garage batch file that "kicks off" the processing (`runbuild.bat`) and the garage Ant script (`garage_hierarchy_all.xml`) are set up to do the filtering.

To run the build:

```
runbuild dita2filtered
```

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

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

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"/>
</map>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Action</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>audience</td>
<td>programmer</td>
<td>exclude</td>
<td></td>
</tr>
<tr>
<td>platform</td>
<td>unix</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
Here is the .ditaval file, which is referenced in the Ant script, that excludes topics tagged as having to do with oil or snow:

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

**Overriding the standard XSLT stylesheets**

Keyword tags: customizing, customizing XSLT stylesheets, stylesheet processing, XSLT stylesheets

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

XSLT stylesheets are written in XML syntax.

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

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

```xml
<xsl:template match="*" mode="process.note">
  <div class="note">
    <xsl:call-template name="commonattributes"/>
    <xsl:call-template name="setidname"/>
    <xsl:call-template name="getString">
      <xsl:with-param name="stringName" select="'Note'"/>
    </xsl:call-template>
    <xsl:call-template name="getString">
      <xsl:with-param name="stringName" select="'ColonSymbol'"/>
    </xsl:call-template>
    <xsl:text></xsl:text>
    <xsl:call-template name="flagit"/>
    <xsl:call-template name="revblock"/>
  </div>
</xsl:template>
```

**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 simply 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.

Customizing XHTML output
Keyword tags: customizing, customizing published output, XHTML

Sections in this topic:
- Customizing XHTML output overview on page 115
- Including a header and footer in XHTML output on page 119
- Publishing XHTML output as a frameset on page 117

Customizing XHTML output overview
Keyword tags: customizing, customizing published output, XHTML

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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition, Usage</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>args.ftr</td>
<td>Path to the file containing XHTML to be placed in the body running-footer area of the output file. The file must be well-formed XML.</td>
<td>Example: &lt;property name=&quot;args.ftr&quot; value=&quot;C:/sandbox/myftr.xml&quot;/&gt;</td>
</tr>
<tr>
<td>args.hdf</td>
<td>Path to the file containing XHTML to be placed in the head area of the output file. The file must be well-formed XML, which means, for example, that you can include a JavaScript if it is labeled as CDATA.</td>
<td>Example: &lt;property name=&quot;args.hdf&quot; value=&quot;C:/sandbox/myhdf.xml&quot;/&gt;</td>
</tr>
<tr>
<td>args.hdr</td>
<td>Path to the file containing XHTML to be placed in the running-header area of the output file. The file must be well-formed XML.</td>
<td>Example: &lt;property name=&quot;args.hdr&quot; value=&quot;C:/sandbox/myhdr.xml&quot;/&gt;</td>
</tr>
</tbody>
</table>

Using your own CSS (Cascading Style Sheet)
Keyword tags: cascading stylesheet, CSS, customizing, customizing published output, outputclass

Default CSS behavior for XHTML processing
The Toolkit CSS stylesheet file resource/commonltr.css is copied to the output directory when you process to a target that creates XHTML output. All the generated XHTML output files include a link like the following that references the default CSS file:

```html
<link rel="stylesheet" type="text/css" href="../commonltr.css">
```

The generated XHTML pages reference classes defined in the default CSS file to control the styling of the XHTML page in a web browser.

Overriding the default CSS for a single DITA element
DITA provides an outputclass common attribute that can be used to to explicitly set CSS classes for elements in the XHTML output. For example, if you want an entire section to be rendered as bold, you would code:

```xml
<section outputclass="caution" />
```
How to create your own CSS to override the default behavior

If you want to change the appearance of all the generated web pages, you can create your own CSS file that overrides part or all of the default CSS file. Your CSS will be included after the default CSS in all the generated pages.

For your override CSS to be used, you must set property values for the three Ant parameters in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>args.copycss</td>
<td>Whether to copy your CSS to the output directory.</td>
</tr>
<tr>
<td>args.css</td>
<td>Path to your CSS file.</td>
</tr>
<tr>
<td>args.csspath</td>
<td>Location of your CSS file in the output directory.</td>
</tr>
</tbody>
</table>

CSS override example

The following example makes 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:

```html
/* 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:

```xml
<!-- 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 is copied to the CSS subdirectory in the output directory. In addition, the generated web pages all contain the following lines:

```html
<link rel="stylesheet" type="text/css" href="../CSS/commonltr.css"/>
<link rel="stylesheet" type="text/css" href="../CSS/garage.css"/>
```

This causes all the web pages to have an aqua background color.

Custom CSS used in this document

Here is part of the custom CSS for this document:

```html
/* Stylesheet overrides for DITA Information Center */

body {
    background: #FFFFFF;
    margin: 5%;
    font: 80% Verdana, Arial, Helvetica, sans-serif;
    margin-left: 10px;
}

/* Makes all headings bold */

h1, h2, h3, h4, h5, h6 {
```
Publishing XHTML output as a frameset
Keyword tags: customizing, frameset, XHTML

By default, the Toolkit produces XHTML output in which the index.html page is a hierarchical list of document contents. When you click on a linked topic in the list, your browser opens a new page.

If you have a document of any size, it may be more user-friendly to create an XHTML document using framesets, in which the hierarchical contents are on the left, and the displayed topic is on the right. The following figure shows an example.

Because the TOC for this document is so long, we have customized the XHTML output to use framesets as described above.

With a few customizations to your DITA project, the Toolkit will produce frameset-based XHTML output automatically.
1. Put an XHTML file called `index.html` into a directory accessible by the Toolkit (ours is in a project directory called `build_files`).

The file should look similar to the following.

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></meta>
<title>DITAinformationcenter</title>
<!-- The following two lines cause the default styling for the navigation frame to be the same as the content frame. -->
<link href="CSS/commonltr.css" type="text/css" rel="stylesheet" />
<link href="CSS/DITAinformationcenter.css" type="text/css" rel="stylesheet" />
</head>
<!-- The left-hand navigation frame is 30% of the available space -->
<!-- The right-hand content frame takes up the rest of the available space -->
<frameset cols="30%,*">
  <!-- tocwin is the navigation frame -->
  <frame name="tocwin" src="toc.html"></frame>
  <!-- contentwin is the content frame -->
  <frame name="contentwin" src="DITAinformationcenter_abstract.html"></frame>
</frameset>
</html>
```

The frame called `tocwin` is the left-hand frame for your table of contents. The source (src) file is `toc.html`.

The frame called `contentwin` is the right-hand frame for your content pages. The source (src) file, `DITAOTUG_bkinfo.html` in this case, is the first page displayed in the content frame. For the `DITAinformationcenter` documentation, it is the title page.

The TOC frame will take up 30% of the available space in the browser window, and the content frame will take up the rest.

2. Add code to your Ant script similar to the highlighted examples in the following figure

```xml
<!-- Create xhtml output -->
<target name="dita2xhtml">
  <ant antfile="${toolkit_dir}${file.separator}build.xml" target="init">
    <property name="args.input" value="${projdir}/${MAP_file}"/>
    <property name="output.dir" value="${outdir}/xhtml"/>
    <property name="dita.temp.dir" value="${outdir}/temp"/>
    <property name="transtype" value="xhtml"/>
    <property name="args.draft" value="yes"/>
    <property name="args.indexshow" value="no"/>
    <property name="args.css" value="${projdir}/css_files/DITAinformationcenter_CSS.css"/>
    <property name="args.csspath" value="CSS"/>
    <property name="args.copycss" value="yes"/>
    <!-- Make the "index" file be toc.html -->
    <property name="args.xhtml.toc" value="toc"/>
  </ant>

  <property name="args.xhtml.toc" value="toc"/>
  <property name="dita.extname" value="${EXTNAME}"/>
  <property name="dita.input.valfile" value="${ditaval_nonpdf}"/>
</target>

<!-- Copy index.html to the output directory -->
<copy todir="${outdir}/xhtml">
  <fileset dir="${projdir}/build_files"/>
</copy>
```
In the above code, the args.xhtml.toc parameter directs the Toolkit to substitute toc.html for the default index.html index page.

The above code also directs the Toolkit to copy the index.html file in the build_files directory to the output directory in a postprocessing step.

Including a header and footer in XHTML output
Keyword tags: customizing, footer, header, XHTML

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:

```xml
<target name="tk2xhtml">
    <ant antfile="${basedir}${file.separator}build.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="C:/sandbox/myhdr.xml"/>
        <property name="args.ftr" value="C:/sandbox/myftr.xml"/>
    </ant>
</target>
```

Customizing PDF output
Keyword tags: customizing, customizing published output, PDF

Sections in this topic:

- Adding a product logo to a PDF cover page on page 119
- Setting up your own PDF customization directory on page 120
- Setting the PDF formatter on page 120

Adding a product logo to a PDF cover page
Keyword tags: cover page, customizing, customizing published output, product logo

We wanted to place an image of a penguin on the PDF cover of this document. Here is how we did that. (Note that we are using the bookmap specialization.)

```xml
<!DOCTYPE bookmap PUBLIC "-//OASIS//DTD DITA BookMap//EN" "dtd/bookmap.dtd">
<bookmap id="dita_infocenter_abstract_top">
    <booktitle>
        <mainbooktitle>DITAinformationcenter</mainbooktitle>
        <image href="images/penguin.png" placement="break"/>
    </mainbooktitle>
</bookmap>
```
Setting up your own PDF customization directory
Keyword tags: customization.dir, customizing, customizing published output, product logo

The DITA OT PDF transform allows you to specify a directory in which you place overrides to the default Toolkit PDF processing. By default this is set to demo/fo/Customization in the Toolkit directory. Using a property setting, you can set this to be a directory that is not part of the Toolkit install directory in case you want to set up overrides on a document set basis. That way each PDF target in your ant build script can have its own set of overrides and you don’t have to modify the Toolkit itself.

Here is a fragment from an Ant build script that shows how to set this property:

```
<property name="customization.dir" value="c:/mysource_dir/customize_pdf/">
```

**Note:** The main build script that controls PDF is in demo/fo/build.xml. This is where the default for customization.dir is set.

Setting the PDF formatter
Keyword tags: customizing, customizing published output, pdf.formatter, product logo

The DITA Open Toolkit PDF transform supports the use of several formatters that convert the generated .fo file it produces to an output PDF file.

The formatter to be used by setting the pdf.formatter property in your Ant build script.

The values you can set are:
- fop - use the Apache FOP formatter. This is the default.
- xep - use RenderX xep.
- ah - use Antenna House.

Here is an example of how to set the property:

```
<property name="pdf.formatter" value="xep"/>
```

**Note:** The main build script that controls PDF is in demo/fo/build.xml.

Setting landscape orientation for PDF output
Keyword tags: customizing, customizing published output, pdf landscape

By default, the DITA Open Toolkit PDF transform produces output in portrait orientation. It is possible to customize the processing so that output has landscape orientation instead.

Here are the steps to make this change.

1. Make a copy of the demo/fo/Customization folder in your build environment.
2. Change your build environment to point to this folder. See Setting up your own PDF customization directory on page 120 for how to do this.
3. In your copied folder, rename the file fo/attrs/custom.xsl.orig to fo/attrs/custom.xsl


4. Edit this file and add variables for page-width and page-height. Save the modified file. The file should look like this:

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    version="1.0">
    <!-- override fo/attrs/basic-settings.xsl to set landscape page format -->
    <xsl:variable name="page-width">279.4mm</xsl:variable>
    <xsl:variable name="page-height">215.9mm</xsl:variable>
</xsl:stylesheet>
```

5. In your new copied folder, rename the file `catalog.xml.orig` to `catalog.xml`.

6. Uncomment this line in the file and save it. The line should look like this:

```xml
<uri name="cfg:fo/attrs/custom.xsl" uri="fo/attrs/custom.xsl"/>
```

This will cause the PDF transform to use the new values for the variables set in your `custom.xsl` file instead of the defaults.

Customizing Eclipse output

Keyword tags: customizing, customizing published output, Eclipse

Sections in this topic:

- **Eclipse help overview** on page 121
- **Deleting the Eclipse help Bookmarks tab for a standalone environment** on page 121
- **Customizing the Eclipse help splash page for a standalone environment** on page 122

Eclipse help overview

Keyword tags: customizing, Eclipse help

Eclipse help is one of the DITA Open Toolkit target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plugin. This allows the output to be displayed as Eclipse standalone help (that is, independent of the Eclipse IDE) or in an Eclipse Infocenter, which allows you to access Eclipse help on the web.

The basic procedure consists of the following:

1. Downloading a minimal Eclipse distribution, such as the Platform Runtime Binary from [http://www.eclipse.org](http://www.eclipse.org)
2. Removing unwanted help plug-ins from the distribution `plugins` directory
3. Modifying the look and feel of the Eclipse help processor

Deleting the Eclipse help Bookmarks tab for a standalone environment

Keyword tags: Bookmarks tab of Eclipse help, customizing, Eclipse help Bookmarks tab

The example in this topic is based on the Eclipse platform binary runtime distribution. To do this customization, you need a text editor for editing text files or members of a `jar` file. To modify a `jar` file you can use the `jar` command line tool included in the Java JDK.

Once you have downloaded an Eclipse distribution, you can eliminate the Bookmarks tab that is displayed by default in the Eclipse help navigation panel.
1. Within your Eclipse distribution, navigate to the directory `eclipse/plugins/` subdirectory.
2. Within the directory, locate the `org.eclipse.help.base` plugin. In Eclipse 3.5 this file is `org.eclipse.help.base_3.4.0v20090611150.jar`.
3. Extract the file `preferences.ini` from the jar file with the command `jar -xf org.eclipse.help.base_3.4.0v20090611150.jar preferences.ini`, or edit using a tool like WinZip.
4. In a text editor, locate the line in the file that defines `bookmarksView`. It should look like this:
   
   `bookmarksView=true`

5. Change the line to `bookmarksView=false`.
6. Save the changed file and update the jar file using the command `jar -uf org.eclipse.help.base_3.4.0v20090611150.jar preferences.ini`.

When you restart standalone Eclipse help, you should see no bookmarks tab in the left-hand pane.

---

**Customizing the Eclipse help splash page for a standalone environment**

Keyword tags: customizing, Eclipse help splash page, splash page of Eclipse help

The example in this topic is based on the Eclipse platform binary runtime distribution. To do the customization, you need a text editor for editing text files or members of a jar file. To modify a jar file you can use the jar command line tool included in the Java JDK.

Once you have downloaded an Eclipse distribution, you can customize it to change the splash page displayed on the output screen in both an Infocenter ot standalone help.

1. Within your Eclipse distribution, navigate to the directory `eclipse/plugins/` subdirectory.
2. Within the directory, locate the `org.eclipse.help.base` plugin. In Eclipse 3.4 this file is `org.eclipse.help.base_3.4.0v20090611150.jar`.
3. Within the directory, copy or create your new splash page file. In this example it is called `mysplash.html`.
4. Add this file to the jar file using the command `jar -uf org.eclipse.help.base_3.4.0v20090611150.jar mysplash.html`.
5. Extract the file `preferences.ini` from the jar file with the command `jar -xf org.eclipse.help.base_3.4.0v20090611150.jar preferences.ini`.
6. In a text editor, locate the line in the file that defines `help_home`. It should look like this:
   
   `help_home=/org.eclipse.help.base/doc/help_home.html`

7. Change the line to `help_home=/org.eclipse.help.base/mysplash.html`.
8. Save the changed file and update the jar file using the command `jar -uf org.eclipse.help.base_3.4.0v20090611150.jar preferences.ini`.

Now when you start standalone Eclipse help, you should see your own splash page, like this:
Creating custom tools

Keyword tags: custom tool, customizing, DITAinformationcenter, tool

The following topics contain information on how to create custom tools for the DITA and DITA Open Toolkit environment.

Sections in this topic:

- The why and how of custom tools on page 123
- Custom tool examples on page 124
- Repairing bad references with DITArepair on page 78

The why and how of custom tools

Keyword tags: custom tools, customizing

The why of custom tools

Custom tools are often created for the following reasons:

- To satisfy key requirements because commercial or open-source (that is, tools available off-the-shelf) are not available
- To produce content automatically (that is, pull product information directly out of the product)
- To create a tighter integration between product and documentation or to build the product and documentation using the same process
• To do a customized legacy content conversion from a proprietary or non-DITA XML format

The how of custom tools
DITA is an XML vocabulary, so it presents a hierarchical structure (coincidentally, this is how published information is often structured).

XML source documents, and also many product source code files, can be parsed and modified using scripting languages. The scripting tools can process all files in a DITA map (or set of maps) as a collection.

The following figure shows the XML (DITA) tree structure that makes this kind of scripting possible.

Custom tool examples
Keyword tags: custom tools, customizing

Debugging and reporting tools for processing
We have created a set of debugging and reporting tools, which are available, free, as part of the DITAinformationcenter distribution, and which augment the functionality that is available as part of the DITA Open Toolkit.

These tools are written in Python or PHP, which are also free and run on most operating system platforms. These tools are particularly useful if you keep your DITA source files in system folders rather than in a source control system or content management system.

Custom XSL scripts for styling
If you use the DITA Open Toolkit, and you want to apply additional styling to your output, you might need to do custom XSL scripting. Creating these scripts requires specialized knowledge, and you may need to hire outside consulting assistance to achieve your objectives.

Custom scripts for web services documentation
If you are documenting web services, you may be interested in investigating customized scripts to pull information out of your WSDL or schema (.xsd) product files. Python is especially well-suited to accomplish this.
Chapter 16

Automating DITA processes with Python

Keyword tags: automated documentation, automating

The following topics contain information about automating DITA processes using Python programming.

Sections in this topic:

About Python on page 125
Why Python and DITA? on page 125
Python XML packages available on page 126
Python DITA XML Examples on page 126
Automating DITA documentation describing web services on page 130

About Python

Keyword tags: automated documentation, automating, Python

Python is a programming language that is widely used and runs on most operating system platforms.

Key Python benefits include the following:

- Freely available from www.python.org
- Interpreted language that is reasonably efficient in its execution
- Relatively easy to learn
- Produces programs that are generally compact and easy to read
- Contains a large built-in library of modules

Why Python and DITA?

Keyword tags: automated documentation, automating, Python

I (Dick) have worked on several client DITA projects where using Python helped make me more productive. These projects include the following:

- A legacy conversion of a set of XML files to the DITA standard
  The original files were in an XML dialect that was similar to, but not identical to, DITA. My first task was to determine which XML elements were in the original files.
- The creation of a DITA prototype where I wanted to extract content contained in legacy XML source files and use it to create DITA topics
- Automatic creation of DITA documentation topics from web services WSDL XML definitions

I was able to write Python scripts in all these scenarios to get the job done.
The following topics describe various things I did with the Python programming language on these DITA projects. I have included sample Python code fragments that illustrate how I did it in hopes that it will make a similar task easier for others.

Note: These topics do not describe the Python programming language in detail. The examples I have provided assume you already know something about Python.

Python XML packages available

The XML package in the Python distribution includes several modules that can be used for XML (and DITA) processing.

Some of the modules are:

- **minidom.** Basic parsing of an XML document and storing it in memory.
- **sax.** Parsing XML documents using the SAX API.
- **ElementTree.** General-purpose routines for storing and manipulating hierarchical data, such as from an XML file.
- **lxml.** An add-on library for Python that is a superset of ElementTree. Versions are available for both Python 2.x and 3.x.

The module I use is **lxml,** which is the most convenient interface for doing XML processing with Python. Python lxml can be used to do the following tasks:

- Read a disk file and parse it into an XML document tree.
- Query for the existence of sets of nodes in the tree. The queries can be done in documents with or without namespaces.
- Change/add/delete elements and their attributes in a document tree.
- Create a new XML document tree, add to it, and then write it out to a disk file.

The lxml module can be used for some of the tasks normally done with XSLT, but:

- It is easier to program and debug.
- The programs are compact and easy to read.
- The lxml module is fast.

I originally used ElementTree, but later switched to lxml for the following reasons:

- ElementTree does not preserve XML comments, but lxml does.
- ElementTree does not provide full XPath support, but lxml does.
- The lxml module provides many other extensions of ElementTree that make it easier to process XML.

For more information on lxml, see [http://codespeak.net/lxml/](http://codespeak.net/lxml/).

Python DITA XML Examples

The following topics are a set of examples for writing Python code with ElementTree to operate on DITA and other XML files. These examples were developed and tested using Python 3.1.
Creating Element Usage Statistics From XML Source Files

Keyword tags: automated documentation, automating, element usage statistics, Python

This example shows a Python code fragment that creates statistics showing how many times each element is used in a set of XML files whose file paths are stored in the "filelist" list.

Example Code

```
# loop through the files counting the tags in
# all of them.
for f in filelist:
    # try to parse the (XML) file
    tree=MakeTree(f)
    if tree!=False:
        # start at the root element
        root=tree.getroot()
        # loop through all the elements
        items=root.getiterator()
        for i in items:
            if iselement(i):
                # count this tag
                thistag = str(i.tag)
                if thistag in dstats:
                    dstats[thistag] = dstats[thistag]+1
                else:
                    dstats[thistag] = 1

    # print out how many tags of each type were found
print(" ")
for tag in sorted(dstats):
    print(tag.ljust(20),str(dstats[tag]).rjust(10))
```

Example Output

```
C:\python ditastat.py
project at: C:\DITAinformationcenter_GARAGESOURCE
21 files in the list

author                                           38
cmd                                              24
conbody                                          13
concept                                          13
context                                           5
copyrholder                                      19
copyright                                        19
copyryear                                        19
created                                          19
critdates                                        19
dd                                                3
dl                                                1
```
Creating a new DITA reference topic

Keyword tags: automated documentation, automating, Python

This example shows a Python code fragment that creates a new DITA reference topic from scratch using ElementTree.

Example Code

```python
# create a new XML tree for a DITA reference topic
f = "somefile.htm"
# set the xml:lang and id attributes in the root element
refattrib = { "xml:lang" : "en-us" , "id" : f+'_ref' }
# add the root reference element
ref = Element("reference", refattrib)
dita = ElementTree(ref)
# add a comment to the topic root
com = Comment("Based on "+os.path.basename(f))
ref.append(com)
# add a title to the topic
title = SubElement(ref,"title")
# set the text of the title
```

```python
128 | DITAinformationcenter All Topics | Automating DITA processes with Python
```
Example DITA reference topic created by this code

```xml
<?xml version="1.0" encoding="UTF-8"?>
<reference id="somefile.htm_ref" xml:lang="en-us">
  <!-- Based on somefile.htm -->
  <title>A simple DITA reference topic</title>
  <refbody>
    <section>
      <simpletable>
      </simpletable>
    </section>
  </refbody>
</reference>
```

Modifying an existing XML file's elements and attributes

Keyword tags: automated documentation, automating, Python

In this example, a DITA topic is read in and parsed, and then various kinds of changes are made to some of the elements found in the file. This example uses ElementTree.

Example Code

```python
# parse the input XML file
tree=MakeTree(f)
if tree==False:
  print("**Error, could not parse",f)
  return False
# get the document root element
root=tree.getroot()
# prepare to iterate over all root child elements
children=root.getiterator("*")

# loop through all the child elements and make some changes
for element in children:
  # add an xml:lang attribute to all topic elements
  if element.tag=="topic":
    element.set("xml:lang", lang)
  # delete any type attributes from either ol or dita elements
  if ((element.tag=="ol") or (element.tag=="dita")):
    # test if a type attribute exists
    tp=element.get("type", default=None)
    if tp!=None:
      # delete the type attribute from the element
      del element.attrib["type"]
```
Automating DITA documentation describing web services

Keyword tags: automated documentation, automating, web services

The following topics describe how to automate the creation of DITA source files describing web services.

Introduction to web services and WSDL

Keyword tags: automated documentation, automating, Python, web service

A web service is an Application Programming Interface (API) that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested service.

In general, a web service API is invoked by an application that creates an XML document called a "request" and sends it to a server on a network. The server then acts on the request and returns its "reply" as another XML document to the application making the request.

The format of the request and the reply is defined in an XML file written in the Web Services Definition Language (WSDL). WSDL is a W3C standard.

What does a WSDL file do?

A WSDL file is an XML document written according to the rules defined in the WSDL schema. The file defines:

- The URL where a request to this service must be sent.
- The requests and replies this service supports. These are called "messages."
- An XML schema definition for each request and reply document sent or received.
- A set of type definitions describing the elements that make up a message. The types may embed references to other types defined in the same WSDL file.
- Optional documentation embedded within the WSDL file.

Here is a fragment from a WSDL file that defines a message containing a simple string field:

```xml
<s:element name="Get_XY">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="USNG_Value" type="s:string"/>
    </s:sequence>
  </s:complexType>
</s:element>
```
Using DITA to document a web service

Using a web service WSDL file as a source document, you can programmatically produce DITA topics to describe the web service. For many web services, there are several issues if you try to do this authoring manually:

- Reading a WSDL file is not easy on the eyes. Some contain several hundred lines of XML schema source.
- The WSDL file may contain hundreds of type definitions that are linked to one another. This makes it difficult to guarantee that the DITA topics derived from the WSDL file are correct.
- Even if you create a set of topics, each time the WSDL changes, you have to go back through the manual authoring process all over again.

DITA and web service documentation

Keyword tags: automated documentation, automating, Python, web service, WSDL

Automating the creation of web service documentation with DITA

To get around the issues with authoring DITA topics from WSDL manually, I (Dick) wrote a Python program that automates the authoring process by creating a set of DITA reference topics to document a web service. The program reads and parses a WSDL source file using the lxml module and creates:

- A DITA container topic that gives an overview of the entire service.
- An “Operations Detail” set of topics listing pairs of requests and their responses with a set of tables showing the elements they use or require. Elements that refer to other element types are linked together so a developer can drill down within element types as needed.
- A “Types Detail” set of topics in alphabetical order containing a table for each type defined in the WSDL.

Any documentation found in the WSDL file is added to the appropriate DITA topic generated by the program. Operations, types, and elements in the WSDL are all indexed by default.

Example of generated DITA topics rendered to PDF

Table of Contents
Overview and Operations

### RefundTransactionResponseType

<table>
<thead>
<tr>
<th>Field</th>
<th>Required Min/Max</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>N 0/1</td>
<td>dateTime</td>
<td>This value represents the date and time (GMT) when the response was generated by a service provider (as a result of processing of a request).</td>
</tr>
<tr>
<td>Ack</td>
<td>y 1/1</td>
<td>AckCodeType</td>
<td>Application level acknowledgment code.</td>
</tr>
<tr>
<td>CorrelationID</td>
<td>N 0/1</td>
<td>string</td>
<td>CorrelationID may be used optionally with an application level acknowledgment.</td>
</tr>
<tr>
<td>Errors</td>
<td>N 0/1</td>
<td>ErrorType</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>y 1/1</td>
<td>string</td>
<td>This refers to the version of the response payload/schema.</td>
</tr>
<tr>
<td>Build</td>
<td>y 1/1</td>
<td>string</td>
<td>This refers to the specific software build that was used in the deployment for processing the request and generating the response.</td>
</tr>
<tr>
<td>RefundTransactionID</td>
<td>y 0/1</td>
<td>TransactionID</td>
<td>Unique transaction ID of the refund.</td>
</tr>
</tbody>
</table>

Generated Index
Chapter 17

Specializing your information design

About 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.

Types of specialization

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 that govern specialization

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 vocabularies. Plug in the modules for your requirements.
• Interoperability at the base type. Guaranteed reversion from special to base.
• Reusable type hierarchies. Share understanding of information across groups, saving time and presenting a consistent picture to customers.
• Output tailored to customers and information. More specific search, filtering, and reuse that is designed for your customers and information, not just the common denominator.
• Consistency, both with base standards and within your information set.
• Learning support for new writers. Instead of learning standard markup plus specific ways to apply the markup, writers get specific markup with guidelines built in.
• Explicit support of different product architectural requirements. Requirements of different products and architectures can be supported and enforced, rather than suggested and monitored by editorial staff.

When to use specialization

Use specialization when you are dealing with new semantics (new, meaningful categories of information, either in the form of new structural types or new domains). The new semantics can be encoded as part of a specialization hierarchy that allows them to be transformed back to more general equivalents, and also ensures that the specialized content can be processed by existing transforms.

Implementing specialization

Keyword tags: information design reuse, specialization, specializing

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 \texttt{xsl} directory that imports the base XSLT script and provides special formatting for your specialized element.

In your Ant build script, add an \texttt{args.xsl} parameter to cause your new XSLT script to be used instead of the default.

\textbf{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:

- \textbf{Specialize from generic elements}. For example, if you want to create a new kind of list but cannot usefully do so specializing from \texttt{<ul>}, \texttt{<ol>}, \texttt{<sl>}, or \texttt{<dl>}, you can create a new set of list elements by specializing nested \texttt{<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.

- \textbf{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.

\textbf{For more information: Specialization}

Keyword tags: information design, reuse, specializing

For examples of how to do specializations, see \url{http://www.ibm.com/developerworks/xml/tutorials/x-ditaspecial/}.

For more detailed discussions on specialization, see these DITA specialization tutorials:

- \url{http://www.xiruss.org/tutorials/dita-specialization/}
- \url{http://www.ditausers.org/tutorials/specialization/}
- \url{http://www.ibm.com/developerworks/edu/x-dw-x-ditaspecial.html}
Localizing and translating your DITA content

The following topics contain information about localizing (translating) the content in your DITA projects.

Sections in this topic:

- About localization on page 139
- Localizing in XHTML output targets on page 140
- Localizing in PDF output targets on page 142
- Preparing your DITA files to be localization-ready on page 144
- Automated machine translation of DITA source files on page 144
- Supported languages on page 148
- For more information: Localization and translation on page 149

About localization

Localization may refer to:

- Language localization, which is the process by which content is translated into another language or adapted for a specific country or region of the world
- Internationalization, which is the adaption of computer software for non-native environments, especially other nations and cultures
- Globalization, which involves planning and preparation for localization

The numeronym "L10n" stands for the first and last letters in "localization" plus the number of letters in between.

The value of DITA in localization

Two of the key reasons organizations cite for using DITA are:

- To respond quickly to customers who need product documentation translated into the primary language spoken in their country or region
- To contain localization and translation costs

Steps required to localize content

Localizing DITA content consists of two steps:

1. Translate the generated labels that appear in DITA output files.
   
   For example, the "Contents" label that appears above the list of contents on the first page of a PDF output file, as shown in the following figure.
Another example would be the "Parent topic" and "Related topic" labels that appear in XHTML output files, as shown in the following figure.

The translation of these generated strings are controlled within the Toolkit build process.

2. Translate the DITA content.

An example of DITA content is the string "You can save money by buying produce when it is in season." string in the figure above.

The translation of the DITA content is done outside the Toolkit build process. (It can be done by human translators and/or by machine, and it is often handled by translation centers.)

Localizing in XHTML output targets

Keyword tags: localizing, translating, xml lang attribute

For XHTML output, the unit of translation for the files included in a DITA map is the individual DITA topic file. In order to translate a topic to another language, it is necessary to translate text in the file and to cause the DITA Open Toolkit to translate any character strings it places in output, such as the string "Related links" in XHTML output.
The language used in an element contained in a DITA topic is controlled by setting two universal attributes. They are:

- **xml:lang** Specifies the language of the element content. If a topic element does not specify the xml:lang attribute, it inherits the value from any ancestor element in the file. If no xml:lang attribute is specified, the default value en-us is assumed.

- **translate** Indicates whether the content of the element should be translated or not.

For more information regarding the XML standards related to the xml:lang attribute, see [http://www.w3.org/International/articles/language-tags/Overview.en.php](http://www.w3.org/International/articles/language-tags/Overview.en.php).

**How DITA Open Toolkit processing handles localization for XHTML output**

When it processes a topic, the Toolkit uses the xml:lang value in each topic element to determine how to do string substitution and to flow text (for example, left to right or right to left). If no xml:lang attribute is specified, it follows the ancestor chain of the current topic to determine xml:lang from any containing element. If none is found, the default language is used.

**Note:** Setting xml:lang in the .ditamap file embedding a topic has no effect on how the topic gets translated by the Toolkit.

**Example**

Here is an example of how this works. In the first case we have a concept that begins:

```
<concept id="about_produce_template">
```

This sets the language to US English for this topic. The XHTML output for the topic looks like the following figure, with the generated strings highlighted:

```
About produce
Overview information about produce.

One of the keys to good health is eating lots of produce. It pays to buy fresh fruits and vegetables and serve them often.

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

**Parent topic:** Shopping for groceries

**Related tasks**

- Choosing produce

**Related reference**

- Available produce

If we change the value of the xml:lang attribute to "de-de", the XHTML output will include generated strings in German, as shown in the following figure.
Finally, if we change the value of the xml:lang attribute to "ar-eg", the XHTML output will include generated strings in Arabic with text flowed right to left, as shown in the following figure.

**Localizing in PDF output targets**

Keyword tags: localizing, translating

**Overview of how translation works for PDF processing**

For PDF output, the locale is set globally for the entire PDF output file. In order to translate the PDF output to another language, it is necessary to translate text in the individual DITA topic files and to cause the DITA Open Toolkit to translate any character strings it places in the PDF output, such as the string "Contents".
Setting the document.locale property

If the locale hasn't been set in the Ant script for a particular PDF document, it is set to US English in the .demo/fo/build.xml file:

```xml
<condition property name="document.locale" value="en_US" />
```

The following locales are supported for Toolkit FO processing:

- Chinese: zh_CN
- English (US): en_US
- German: de_DE
- French: fr_FR
- Italian: it_IT
- Japanese: ja_JP
- Spanish (Spain): es_ES

Note: This is a subset of the languages supported by the DITA Open Toolkit for various XHTML-based target output environments.

PDF localization examples

The following figure shows a page from a PDF output file where the (default) locale of US English was specified:

If we change the locale to German, the same page looks like this:
Preparing your DITA files to be localization-ready

Keyword tags: globalizing content, internationalizing content, localizing, localizing content, translating, translating content

Even if you have no immediate plans to translate your DITA project files into languages other than US English, you can make your projects localization-ready by:

• Including xml:lang statements in your DITA source files.
• Creating well-architected projects that make effective use of content reuse.
• Always including a glossary, core vocabulary, or controlled vocabulary in your DITA projects.
• Investigating translation technologies as you convert your legacy content to DITA to be sure your DITA projects will operate effectively with popular localization applications.
• Ensuring that your content management system integrates well with popular translation technologies.
• Writing your content using "Simplified English" guidelines.

How we made this document localization-ready

Although we have not yet translated this documentation into languages other than US English, it is partially localization-ready for the following reasons:

• The document was written using DITA, an XML vocabulary well supported by popular translation tools.
• We have put xml:lang attributes in all of our root elements.
• The core vocabulary set could provide a translation center with basic information about the DITA, the DITA OpenToolkit and related technologies.
• Approximately 25% of the content is reused throughout the documentation set, which helps to reduce translation costs.

Automated machine translation of DITA source files

Keyword tags: localizing, machine translation, translating

Sections in this topic:

Introduction to machine translation and DITA on page 144
DITA machine translation using Python on page 145
Improving the quality of machine translation on page 146

Introduction to machine translation and DITA

Keyword tags: localizing, machine translation, Python, translating

Machine translation is a subfield of computational linguistics that investigates the use of computer software to translate one natural language to another. The techniques are particularly effective in domains where formal language is used.

Machine translation system are generally rule-based (for example, Yahoo Babel Fish), statistical (for example, Google Translate and Language Weaver), or a hybrid system (for example, Microsoft Translator).

Depending on the tools used to carry it out, machine translation produces results with varying degrees of quality compared to what can be achieved by a skilled human being translating the same text. Today companies that provide translation and localization services might use machine translation as a starting point to translate source text, followed by human post-editing to achieve a final result. Machine translation by itself might be used to provide instant translation of text for static web pages and social networking sites.
Machine translation services on the web

A number of publicly available web services on the Internet can be invoked to machine translate text. Two that are commonly used today are:

- Google Translate - http://translate.google.com/#
- Microsoft Translator - http://www.microsofttranslator.com/

Each of these services offers an API that allows a scripting language, such as Python, to translate text.

DITA machine translation using Python

Keyword tags: localizing, machine translation, Python, translating

To learn about and illustrate machine translation of DITA topics, I (Dick) wrote a Python program that machine translates all the DITA topics in a DITA map file. The program can translate using either the Google or Microsoft translator. It machine translates the topics one by one and places the result in a separate directory from the original source topics. These can then be processed to an output target using the DITA Open Toolkit.

This program performs the following steps:

1. Starting from a DITA map file, creates a list of all source files referred to in the list.
2. Reads in each source file and translates each of its text strings.
4. Writes out the machine-translated results to a separate directory.

Sample output

As the program executes, it generates a log of the machine translations. Here is a sample of a log as the DITA garage sample is machine translated from English to German.

```
MT: DITAgarage_SOURCE using translation provider Google
debugMode False
translating to: de-de

map file: DITAgarage_SOURCE\hierarchy.ditamap
map file: DITAgarage_SOURCE\sequence.ditamap

source: (17) Garage (sequence)
target: (16) Garage (Sequenz)
source: (43) Machine translation to de-de d ...
target: (51) Maschinelle Übersetzung zu de- ...
2 segments translated
writing DITAgarage_SOURCE\GG-de-de\sequence.ditamap
source: (18) Garage (hierarchy)
target: (19) Garage (Hierarchie)
source: (43) Machine translation to de-de d ...
target: (51) Maschinelle Übersetzung zu de- ...
2 segments translated
writing DITAgarage_SOURCE\GG-de-de\hierarchy.ditamap
```

One of the translated topics looks like this:
Improving the quality of machine translation

Keyword tags: localizing, machine translation, Simplified English, Simplified Technical English, translating

I (Anna) was dissatisfied with the quality of the first machine translations we did of the garage and grocery shopping samples. I set out to try to get improved results without having to do a manual translation, using techniques popular in Simplified English, a controlled natural language.

Controlled natural languages (CNLs) are subsets of natural languages, created by restricting the vocabulary and grammar to reduce complexity. CNLs are used to make content more accessible to non-native speakers, as the basis for content to be translated into another language, and to make semantic analysis easier. These languages are often used to increase the quality and understandability of technical material, and to aid in machine translation (MT) into other natural languages.

Most controlled languages are based on English, partly because most translations are done from English to another language. Examples of controlled English projects include ASD (AeroSpace Defence Industries Association of Europe) Simplified Technical English (Simplified English, or ASD-STE100), Attempto Controlled English (ACE), IBM's Easy English, and Special English (used by Voice of America in its broadcasts).

Some of the problems I encountered included:

• My task titles included gerunds in English (for example, "Washing the car." These were generally turned into nouns in the target language.
• Because the language segments ("chunks") were translated one by one in the automated process, context was lost. The result was inconsistently translated titles and steps, which was especially problematic in the set of tasks.
• Some clauses and phrases of any length or complexity caused the translation tool to simply "get lost" and produce silly and/or annoying results.
• The Google and Microsoft translation engines were not consistent in their translations, even with very simple and seemingly straightforward text.

I rewrote the text of the garage and grocery shopping samples using some of the rules of Simplified English:

• Restrict the length of noun clusters to no more than 3 words.
• Restrict sentence length to no more than 20 words.
• Avoid slang and jargon.
• Use articles with nouns ("a/an" and "the") wherever possible.
• Use simple verb tenses (present, past, and future) and avoid gerunds.
• Use active voice.
• Put conditions first in instructions, warnings, and cautions.
• Use only "approved" words (generally those that are in common use, are unambiguous, and can be interpreted only as a single part of speech).
• Hyphenate potentially problematic adjective clusters.
• Hyphenate or write as a single word potentially problematic noun clusters.

It took me about twice as much time to rewrite the few topics in the garage and grocery shopping samples as I had anticipated. The results were improved, but problems remain. As I worked on the rewriting task, results often improved in one target language but got worse in another.

The figures below show two "before" samples, translated by Google.

![Spanish translation](image1)

![German translation](image2)

To see the "after" results in several target languages, see the DITAinformationcenter resource files.

To make machine translation experimentation easier I highly recommend a gadget or other tool such as the one I have on my iGoogle page that shows the same English segment translated by both Google and Microsoft, side by side. (The gadget also does back-translations from each target result to both translation engines.)
## Supported languages

**Supported Languages**

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

<table>
<thead>
<tr>
<th>Language</th>
<th>xml:lang value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>ar-eg</td>
</tr>
<tr>
<td>Belarusian</td>
<td>bg-bg</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>be-by</td>
</tr>
<tr>
<td>Catalan</td>
<td>ca-es</td>
</tr>
<tr>
<td>Chinese (Simplified)</td>
<td>zh-cn</td>
</tr>
<tr>
<td>Chinese (Traditional)</td>
<td>zh-tw</td>
</tr>
<tr>
<td>Croatian</td>
<td>hr-hr</td>
</tr>
<tr>
<td>Czech</td>
<td>cs-cz</td>
</tr>
<tr>
<td>Danish</td>
<td>da-dk</td>
</tr>
<tr>
<td>Dutch</td>
<td>nl-nl</td>
</tr>
<tr>
<td>Dutch (Belgian)</td>
<td>nl-be</td>
</tr>
<tr>
<td>English (Canadian)</td>
<td>en-ca</td>
</tr>
<tr>
<td>English (UK)</td>
<td>en-gb</td>
</tr>
<tr>
<td>English (US)</td>
<td>en-us</td>
</tr>
<tr>
<td>Estonian</td>
<td>et-ee</td>
</tr>
<tr>
<td>Finnish</td>
<td>fi-fi</td>
</tr>
<tr>
<td>French</td>
<td>fr-fr</td>
</tr>
<tr>
<td>French (Belgian)</td>
<td>fr-be</td>
</tr>
<tr>
<td>French (Canadian)</td>
<td>fr-ca</td>
</tr>
<tr>
<td>French (Swiss)</td>
<td>fr-ch</td>
</tr>
<tr>
<td>German</td>
<td>de-de</td>
</tr>
<tr>
<td>German (Swiss)</td>
<td>de-ch</td>
</tr>
<tr>
<td>Greek</td>
<td>el-gr</td>
</tr>
<tr>
<td>Hebrew</td>
<td>he-il</td>
</tr>
<tr>
<td>Hindi</td>
<td>hi-in</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hu-hu</td>
</tr>
<tr>
<td>Icelandic</td>
<td>is-is</td>
</tr>
<tr>
<td>Italian</td>
<td>it-it</td>
</tr>
<tr>
<td>Italian (Swiss)</td>
<td>it-ch</td>
</tr>
<tr>
<td>Language</td>
<td>xml:lang value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja-jp</td>
</tr>
<tr>
<td>Korean</td>
<td>ko-kr</td>
</tr>
<tr>
<td>Latvian</td>
<td>lv-lv</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>lt-lt</td>
</tr>
<tr>
<td>Macedonian</td>
<td>mk-mk</td>
</tr>
<tr>
<td>Norwegian</td>
<td>no-no</td>
</tr>
<tr>
<td>Polish</td>
<td>pl-pl</td>
</tr>
<tr>
<td>Portuguese</td>
<td>pt-pt</td>
</tr>
<tr>
<td>Portuguese (Brazilian)</td>
<td>pt-br</td>
</tr>
<tr>
<td>Romanian</td>
<td>ro-ro</td>
</tr>
<tr>
<td>Russian</td>
<td>ru-ru</td>
</tr>
<tr>
<td>Serbian</td>
<td>sr-sp</td>
</tr>
<tr>
<td>Slovak</td>
<td>sk-sk</td>
</tr>
<tr>
<td>Slovenian</td>
<td>sl-sl</td>
</tr>
<tr>
<td>Spanish</td>
<td>es-es</td>
</tr>
<tr>
<td>Swedish</td>
<td>sv-se</td>
</tr>
<tr>
<td>Thai</td>
<td>th-th</td>
</tr>
<tr>
<td>Turkish</td>
<td>tr-tr</td>
</tr>
<tr>
<td>Ukranian</td>
<td>uk-ua</td>
</tr>
<tr>
<td>Urdu</td>
<td>ur-pk</td>
</tr>
</tbody>
</table>

For more information: Localization and translation

Keyword tags: AMTA (Association for Machine Translation in the Americas), ATA (American Translators Association), Common Sense Advisory, ELIA (European Language Industry Association), GALA (Globalization and Localization Association), globalization (G11N), internationalization (I18N), Localisation Research Centre, localization (L10N), localizing, TAUS (Translation Automation User Society), tcworld, translating, translation, W3C Internationalization (I18N) Activity

GALA (Globalization and Localization Association)

GALA educates general business worldwide on the benefits of localization and offers localization professionals a language and translation community.

http://www.gala-global.org/

ATA (American Translators Association)

ATA 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.
ELIA (European Language Industry Association)
ELIA is a nonprofit organization that provides a forum for information exchange, fosters business relationships, and promotes ethics and quality standards throughout the localization industry.
http://www.elia-association.org

AMTA (Association for Machine Translation in the Americas)
AMTA is an organization dedicated to anyone interested in the translation of languages using computers.
http://www.amtaweb.org/

Localisation Research Centre (LRC)
The Localisation Research Centre is the information, educational, and research centre for the localization community. It is located at University College Dublin.
http://www.localisation.ie/

W3C Internationalization (I18N) 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/

tcworld e-magazine
tcworld is a magazine for international information management. The organization sponsors a yearly conference.
http://www.tcworld.info

TAUS (Translation Automation User Society)
TAUS is a think tank for the translation industry, undertaking research for buyers and providers of translation services and technologies.
http://www.translationautomation.com/
Chapter 19

Managing your DITA content

Keyword tags: managing

The following topics contain information on how to organize and manage your DITA content.

Sections in this topic:

- Organizing your source and output directories on page 151
- Backing up your source files on page 152
- About library and source control systems on page 153
- About content management systems on page 153
- For more information: Content management on page 154

Organizing your source and output directories

Keyword tags: managing, organizing directories, output files, source files

We recommend putting your project source and output files so they appear together in your directory structure. The root source and output directories for the garage sample project are shown in the diagram below.

It is important not to collocate source and output files for multiple DITA projects.

Generic processing files, like DITA Open Toolkit and general-purpose debugging tools should not be located with your individual project files.

Organizing your source file directories

We recommend keeping all source files for a given project together in one place. This would include the ditamap(s), Ant script(s), customization files (for example, a CSS file), image files, and project-specific history and other related informations. The garage sample source files are shown in the diagram below.

DITA source files for the concepts subdirectory of the garage sample are shown in the following diagram.
Organizing your output file directories

We recommend organizing output files by target output. Debugging and reporting tool results for a given project can also be put into the output directory. Below is the directory structure for the garage sample.

Backing up your source files

Keyword tags: backing up files, managing, managing content

All the work you do editing and debugging the files in your DITA project ends up being stored as files on disk. If something happens to one or more of those disk files, you work may need to be re-created. Disk files can be lost for several reasons, including:

- You accidentally erase them. Because DITA projects may have hundreds or thousands of files, it may be relatively easy to do this when you don't have a library system or content management system.
- The hard drive in your computer fails. Hard drives are mechanical devices and will fail after a finite amount of time.
- The file system on your disk drive becomes corrupted.
Strategies for preserving your data

The basic strategy for preserving your data is to make sure it is stored in more than one location. 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 generic processing, debugging, and reporting tools.

About library and source control systems

Keyword tags: library system, managing, managing content, source control system

In library or source control systems, DITA source files are stored on disk as ASCII text files, typically along with the source code files for the software project being documented. A source control system stores and tracks changes to files, and features "check in/out" of files, so users of the system are not making conflicting updates. Multiple versions of source files are kept, and most systems have built-in capabilities to display differences between different versions of the same file.

Although having the code and documentation in the same system may seem useful, source control systems that were written with code in mind may not be the best place to keep content, mostly because these systems are typically not "content-aware."

A better solution would be either a content management system that integrates with the library system, or a separate content management system that has features appropriate for content.

About content management systems

Keyword tags: CMS, content management system, managing, managing content

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 content management system (CMS), which might include:

- Workflow support
- Validation of topic links
- Support for the semantic web
- Localization (translation) support

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

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

Keyword tags: CM Professionals, CMS Review, content management, managing, managing content, Real Story Group

CM Professionals
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
CMS Review provides resources to help you choose a content management solution.

http://www.cmsreview.com/

Real Story Group
Real Story Group, formerly CMS Watch, provides independent evaluations of content management, records management, enterprise search, and portal solutions.

http://www.realstorygroup.com/
Chapter 20

Distributing your published content

Keyword tags: distributing
The following topics contain information on how to distribute your published DITA content.
Sections in this topic:

- About distributing published content on page 155
- Distributing published content using RSS on page 158

About distributing published content

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

Creating DITAmashups™

Keyword tags: distributing, DITAmashup, mashup
The following topics contain information about architecting and publishing structured/unstructured information collections called DITAmashups.
Sections in this topic:

- About DITAmashups on page 155
- Architecting DITAmashups on page 156
- Publishing DITAmashups on page 157

About DITAmashups

Keyword tags: distributing, DITAmashup
A DITAmashup is a structured DITA project that includes unstructured content from one or more information sources external to the structured topics. The DITA structured framework provides organizational and semantic integrity to the entire body of information, both structured and unstructured.

The key advantage of such an information collection is that a large body of high-value, related content can be assembled and presented to users without the cost and effort of a total DITA conversion.

As an example, suppose the documentation group working for a software company pulls together ("mashes up") the "official" DITA-based documentation for their software product along with unstructured content produced by their sister training and support organizations and a knowledge base of user-produced tips contributed to the support website. In assembling this eclectic but useful collection of information they apply
the semantics associated with their structured DITA project and then publish the entire mashup to a website accessible to internal and external users alike.

Various platforms could be used as host sites for mashups. As our first two we chose WordPress and Drupal. We were looking for platforms with the following characteristics:

- Platforms that support XHTML, which is one of the standard output types of the DITA Open Toolkit
- Platforms that support the web services protocol XML-RPC, which allows us to programmatically post-process and publish the DITA-based XHTML output as a structured, interconnected collection of topics
- Well-established, popular platforms that provide editing, publishing, and workflow capabilities for the unstructured content, and also provide additional support (for example, search) to the structured content participating in the solution

Our DITAmashups are hosted on our two DITAinformationcenter sites:

- WordPress site: XMLdocs.info
- Drupal site: DITAinfo.info

**Architecting DITAmashups**

Keyword tags: distributing, DITAmashup

**Structured/unstructured challenges**

Structured information collections have accurate product information, but often lack "real-world" credibility. Unstructured collections typically have that popular "in-the-trenches" look and feel, but the information is often down-level, lacks professional polish, and is difficult for product owners to influence or control.

**Our solution objectives**

Our key DITAmashup objectives are to:

- Combine the structured and unstructured information ("have it all")
- Publish to popular, function-rich web content management systems platforms
- Minimize cost and complexity (for example, take advantage of native platform functionality and meet critical needs with custom scripting rather than expensive commercial products)
- Use the organization and semantics of the structured collection to establish and maintain overall information integrity

**Sample usage scenarios**

We expect our DITAmashup solutions to be popular with organizations:

- That need a relatively inexpensive, content-managed solution published to a popular web content management system
- That need an interactive solution that includes capabilities like comments and multi-person blogs
- That need or want to keep development of structured and unstructured content solutions separate
- Whose content solution is not complete without both structured and unstructured information
- Whose documentation is produced and assembled by multiple in-house organizations that use different tools and content paradigms
- Whose internal website (for example, wiki) has contributors with different tastes in authoring and publishing tools

**Benefits of the solution**

Key DITAmashup benefits include:
• A structured information collection that is tightly controlled, single-sourced, and was created by professionals using the most up-to-date professional tools
• An unstructured information collection that can be created and updated independently from the formally published documents, but are tagged with the same metadata
• A platform that allows all information contributors to choose their favorite authoring and publishing tools
• A powerful delivery platform popular with users of all experience- and skill-levels

Publishing DITA mashups

Keyword tags: distributing, DITA mashup

We have hosted model DITA mashup sites using our DITA information center on WordPress (XMLdocs.info) and Drupal (DITAinfo.info).

We published the DITA-based structured docset that is part of the DITA mashup using Python scripts that have network access to the CMS.

Key solution components

The scripts require these items in order to carry out the publishing process:

• XHTML output for a DITA map processed by the DITA Open Toolkit, release 1.5 or later. No modifications are required to be made to the Toolkit itself.
• The Python 3.2 interpreter installed on a client machine where the publishing operation takes place.
• A web content management system that supports remote blogging using an API that makes use of the XML-RPC protocol. We currently use either the WordPress blog API or the metaWeblog API. These APIs are also used by blogging tools like ScribeFire.
• An XML file containing the URL for the CMS, a CMS userid and password that is allowed to do remote blogging to the CMS, and the location on disk of the DITA Open Toolkit XHTML output files.

Processing procedure

There are two Python scripts involved, one to publish the docset and one to delete the files that have been previously been published, as a single entity. This allows publication and deletion without administrator intervention on the CMS dashboard. The scripts interact with our own custom-created versions of the blog API modules.

The publishing script follows this procedure:

1. Reads the XML file that describes the CMS URL, the userid and password, and the location of the XHTML files.
2. Establishes XML-RPC communication with the CMS.
3. Reads and parses the XHTML index file and creates tables describing the parent-child relationships for all the topics in the docset.
4. Builds up a list of images referenced in the docset and uploads the required images to the CMS, recording where the files were stored in the CMS.
5. Stores the topics one by one in the CMS in their order in the hierarchy and records where the files were stored in the CMS. The file text data stored in the CMS for a topic is a subset of the XHTML in the web pages that were created by the DITA Open Toolkit.
6. Reads back the topic files from the CMS and updates links to other topics or images in the CMS based on where the files were stored by the CMS.
7. Creates an XML file containing an inventory of all the docset files stored in the CMS.

The deletion script can be run at a later time to delete all the files in a docset. This script reads the XML inventory file created by the publishing script and deletes the files and images one by one.
Distributing published content using RSS

Keyword tags: distributing, distributing content, RSS

RSS is a family of web feed formats written in XML and used in content syndication by enabling applications like RSS readers to find out when new information is available on a website.

Since creation of output from DITA source files is automated by using Ant, it is possible to create or update RSS information about the output at the same time the output is produced. The RSS file can then be uploaded along with the output so its availability can be known to those subscribed to the RSS feed.

For example, an Ant build could create a file like the following to announce the availability of a new version of this document:

```xml
<?xml version='1.0'?>
<rss version='2.0'>
  <channel>
    <title>DITAinformationcenter</title>
    <copyright>Copyright (c) 2010 by VR Communications, Inc. All rights reserved.</copyright>
    <link>http://www.mysite.com/ditaug.html</link>
    <description>Supports DITA Open Toolkit 1.5 and DITA 1.2</description>
    <language>en-us</language>
    <lastBuildDate>Wed 30 July 2010, 14:30:25 PST</lastBuildDate>
    <item>
      <title>DITAinformationcenter</title>
      <link>http://www.mysite.com/ditainfocenter.pdf</link>
      <pubDate>Wed 30 July 2010, 14:30:25 PST</pubDate>
      <description>The latest version of DITAinformationcenter is now available.</description>
    </item>
  </channel>
</rss>
```
Appendix

A

Getting more information

Keyword tags: getting information

The following topics provide additional general sources of information about DITA and DITA Open Toolkit.

Sections in this topic:

- General information on page 159
- Sample DITA projects on page 160
- DITA Open Toolkit release information on page 161
- DITA language standard on page 161
- Tools and plug-ins on page 162
- User groups on page 162
- Services on page 163
- Events on page 163

General information

Keyword tags: Cover Pages, DITA information source, DITA Open Toolkit information source, DITA World, dita.xml.org, getting information, 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

DITA World

List of DITA resources.

http://www.ditaworld.com/
Introduction to DITA (book)
Useful tutorial and general reference. Includes sample DITA content. Recommended even if you are not using Arbortext as your editor and processing tool.
http://www.comtech-serv.com

DITA Language Reference (PDF file)
A must-have reference manual for any serious DITA writer. We keep copies handy on our desktops.

Ant: The Definitive Guide (book)
Helpful information about writing Ant scripts.
http://oreilly.com/catalog/9780596006099/

DITA Maturity Model (PDF)
White paper about incremental adoption of DITA.

IBM developerWorks articles on DITA and XML
IBM resource site for developers and IT professionals. Search the site using DITA and XML keywords.
For more information: http://www-128.ibm.com/developerworks/xml/library/x-dita1/

Sample DITA projects
Keyword tags: getting information, sample DITA projects

DITAinformationcenter
Three sets of sample DITA source files are available as part of the DITAinformationcenter distribution:
• Garage sample project
• Grocery shopping sample project
• DITAinformationcenter documentation

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.
**Comstar DITA project**

The Comstar DITA project is part of the two "Introduction to DITA" books published by Comtech Services, Inc.: [http://www.comtech-serv.com](http://www.comtech-serv.com)

---

**DITA Open Toolkit release information**

Keyword tags: DITA Open Toolkit releases, getting information

**DITA Open Toolkit project home**


**Current DITA Open Toolkit release**


**DITA Open Toolkit developers mailing list**


---

**Specialization**

Keyword tags: getting information, information design, reuse, specializing


For more detailed discussions on specialization, see these DITA specialization tutorials:

- [http://www.ditausers.org/tutorials/specialization/](http://www.ditausers.org/tutorials/specialization/)

---

**DITA language standard**

Keyword tags: DITA language standard, getting information

**DITA Technical Committee**


**Roadmap for DITA development**

Tools and plug-ins

Keyword tags: alphaWorks IBM Information Architecture Workbench, DITA plugins, DITA tools, getting information

DITA information center download sites
Download sites for the DITA information center distribution and documentation.
Hosted by VR Communications, Inc.
http://www.xmldocs.info
http://www.ditainfo.info

DITA Open Toolkit download site
Download site for the Toolkit and related/prerequisite software.
Hosted by SourceForge.
https://sourceforge.net/projects/dita-ot

DITA Open Toolkit plug-ins download sites
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

IBM Information Architecture Workbench (formerly known as Task Modeler)
Eclipse-based tool for rapidly creating and analyzing models of human activity for DITA and user experience design.
Hosted by IBM.

User groups

Keyword tags: forum, getting information, user group

DITA Yahoo Group
http://groups.yahoo.com/group/dita-users

DITA/FrameMaker Yahoo Group
http://groups.yahoo.com/group/framemaker-dita

List of DITA user and interest groups (by location)
http://dita.xml.org/user-groups
http://www.ditanews.com/resources/communities/
DITA forums on sourceforge.net
There are three forums:

• Developers
• Help
• Open discussion

http://sourceforge.net/projects/dita-ot/forums

Services
Keyword tags: getting information, service

Services
A list of services is included in the DITA.xml.org website: dita.xml.org/services.

Service categories include:

• Ant customization
• Consulting
• DITA specialization
• Domain specialization
• Getting started
• Implementation
• Information architecture
• Product extensions
• Terminology
• Training
• XSL/XSLT customization

Events
Keyword tags: conference, event, getting information, webinar

News and events on dita.xml.org
http://dita.xml.org/news
http://dita.xml.org/events

Events sponsored by WritersUA (User Assistance Professionals)
http://www.writersua.com

Events sponsored by CIDM (Center for Information-Development Management)
http://www.infomanagementcenter.com

Events sponsored by Congility (formerly X-Pubs)
http://www.congility.com
Events sponsored by IdeAlliance

http://www.idealliance.org

Events sponsored by GALA (Globalization and Localization Association)

http://www.gala-global.org
Commercial tools

Keyword tags: commercial tool

The following topics provide overview information about some of the most popular DITA-related tools. All you need to get started is an authoring tool and basic processing environment. Next, you may want to invest in a source control system. After that, your tool needs depend on your organizational and workflow environments, your ultimate information objectives, and how quickly you can make progress.

Sections in this topic:

- Authoring (editing) tools on page 165
- Processing tools on page 166
- PDF rendering tools on page 167
- Source control and library systems on page 168
- Content management systems on page 168
- Localization and translation tools and services on page 169
- Legacy content conversion tools and services on page 171

Authoring (editing) tools

Keyword tags: Adobe, Altova, Arbortext Editor, authoring tool, commercial tool, DITA-FMx, editing tool, FrameMaker, JustSystems, Leximation, oXygen, PIXware, PTC (Parametric Technology Corporation), SyncRO Soft Ltd., XMetaL Author, XMLmind, XMLSpy

About DITA-aware authoring tools

"DITA-awareness" represents a spectrum of functionality rather than an absolute, but in general an authoring tool that claims this capability needs to meet the following criteria:

- Checks for "well-formedness":
  - Opening and closing tags match
  - Example: <note>Text</note>

- Checks for validity:
  - All DTD rules are obeyed
  - Example: Tags are in the document's DTD and appear in an appropriate place in the document

The authoring tool should "insist" on well-formedness and validity (that is, it won't let you save your file if there are errors).

The following table lists and briefly describes some of the popular DITA-aware authoring tools. For more information about the tools, see the referenced URL.
The following table lists and briefly describes some of the popular DITA processing tools. For more information about the tools, see the referenced URL.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbortext Editor</td>
<td>PTC (Parametric Technology Corporation)</td>
<td>Combines editing and processing for DITA.</td>
<td>Proprietary processing. Generally used by enterprise organizations along with other PTC products for a complete publishing and localization solution.</td>
<td>ptc.com</td>
</tr>
<tr>
<td>FrameMaker</td>
<td>Adobe</td>
<td>Used in structured mode to edit and process DITA source files.</td>
<td>Proprietary processing. Provides a working environment familiar to many writers.</td>
<td>adobe.com</td>
</tr>
<tr>
<td>DITA-FMx</td>
<td>Leximation</td>
<td>FrameMaker add-on. Provides many DITA enhancements and features beyond those available in FrameMaker 8 or 9.</td>
<td>Produces 100% compliant DITA files.</td>
<td>leximation.com</td>
</tr>
<tr>
<td>oXygen</td>
<td>SyncRO Soft Ltd.</td>
<td>Combines editing and processing of DITA and DocBook files.</td>
<td>Popular with both writers and software developers. Uses DITA Open Toolkit for processing. Automatically generates web services doc from schema (.xsd) files. Can be used to edit and process DITA source files in Eclipse. Contains a Subversion (SVN) control client.</td>
<td>oxygenxml.com</td>
</tr>
<tr>
<td>XMetaL Author</td>
<td>JustSystems</td>
<td>Provides editing and processing for DITA files.</td>
<td>Uses DITA Open Toolkit for processing combined with XEP from RenderX.</td>
<td>justsystems.com</td>
</tr>
<tr>
<td>XMLmind</td>
<td>PIXware</td>
<td>Edits DITA and DocBook files.</td>
<td>Along with the XMLmind XSL-FO processor plugin, can produce many output types (see also XSL-FO Converter).</td>
<td>xmlmind.com</td>
</tr>
<tr>
<td>XMLSpy</td>
<td>Altova</td>
<td>DITA-aware editor; can also be used with other XML files.</td>
<td>The Enterprise Edition can be used to automatically generate web services documentation from a WSDL file.</td>
<td>altova.com</td>
</tr>
</tbody>
</table>

**Processing tools**

Keyword tags: commercial tool, DITA Open Toolkit, HyperWrite Pty Ltd, MadCap Flare, MadCap Software, OASIS (Organization for the Advancement of Structured Information Standards), PIXware, processing tool, WinANT, XMLmind XSL-FO Converter

The following table lists and briefly describes some of the popular DITA processing tools. For more information about the tools, see the referenced URL.
The following table lists and briefly describes some of the popular PDF rendering tools. For more information about the tools, see the referenced URL.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>DITA Open Toolkit</td>
<td>OASIS</td>
<td>Reference implementation of the OASIS DITA standard.</td>
<td>Free. Open source project on the SourceForge site. Widely used; embedded in many commercial products.</td>
<td>sourceforge.net</td>
</tr>
<tr>
<td>MapCap Flare</td>
<td>MapCap Software</td>
<td>Can import from DITA source files.</td>
<td>Once the files are imported, you can publish to PDF, XPS, Word, FrameMaker, DITA, XHTML with integrated navigation, HTML Help, and mobile platforms. Source files are stored in a proprietary Flare format.</td>
<td>madcapsoftware.com</td>
</tr>
<tr>
<td>WinANT</td>
<td>HyperWrite Pty Ltd</td>
<td>GUI interface to control DITA Open Toolkit processing.</td>
<td>Free. Has been used to teach DITA at the university level.</td>
<td>hyperwrite.com</td>
</tr>
<tr>
<td>XMLmind XSL-FO Converter</td>
<td>PIXware</td>
<td>XSL formatter.</td>
<td>Similar to Apache FOP, RenderX XEP and Antenna House XSL Formatter. Reads DITA source files and generates output in various formats. Useful if you want to generate output not supported by DITA Open Toolkit (for example, Microsoft Word 2007).</td>
<td>xmlmind.com</td>
</tr>
</tbody>
</table>

**PDF rendering tools**

Keyword tags: Antenna House, Antenna House Formatter, Apache FOP, Apache XML Graphics Project, commercial tool, PDF rendering tool, rendering tool, RenderX, XEP PDF Engine

The following table lists and briefly describes some of the popular PDF rendering tools. For more information about the tools, see the referenced URL.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antenna House Formatter</td>
<td>Antenna House</td>
<td>Renders XSL-FO as PDF.</td>
<td>Supported for PDF by DITA Open Toolkit and several vendor products that embed the Toolkit.</td>
<td>antennahouse.com</td>
</tr>
<tr>
<td>Apache FOP</td>
<td>Apache XML Graphics Project</td>
<td>Renders XSL-FO as PDF.</td>
<td>Free. FOP is included in DITA Open Toolkit and several vendor products that embed the Toolkit.</td>
<td>xmlgraphics.xml.org</td>
</tr>
<tr>
<td>XEP PDF Engine</td>
<td>RenderX</td>
<td>Renders XSL-FO as PDF.</td>
<td>The Personal Edition is free. FOP is included in DITA Open Toolkit and several vendor products that embed the Toolkit.</td>
<td>renderx.com</td>
</tr>
</tbody>
</table>
Source control and library systems

Keyword tags: Apache Software Foundation, commercial tool, library system, Perforce SCM, Perforce Software, source control system, Subversion

The following table lists and briefly describes some of the popular source control and library systems. For more information about the tools, see the referenced URL.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perforce SCM</td>
<td>Perforce Software</td>
<td>Comprehensive source control management system built around a scalable client/server architecture.</td>
<td>Popular in the hardware/software development community. The Perforce client/server runs on most operating system platforms.</td>
<td>perforce.com</td>
</tr>
<tr>
<td>Subversion</td>
<td>Apache Software Foundation</td>
<td>Open-source version control system.</td>
<td>Free. Subversion (SVN, Apache Subversion) clients exist for Eclipse, oXygen, and the command line. Very popular in the software development community. Versions exist for most operating system platforms.</td>
<td>subversion.apache.org</td>
</tr>
</tbody>
</table>

Content management systems

Keyword tags: Astoria On-Demand, Astoria Software, Blustream XML Content Solutions, CMS, commercial tool, content management system, DITA CMS, DocZone, IXIASOFT, Really Strategies Inc., SDL, Trisoft DITA, XDocs

The following table lists and briefly describes some of the popular content management systems. For more information about the tools, see the referenced URL.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astoria On-Demand</td>
<td>Astoria Software</td>
<td>Manages authoring, review, and publishing XML content.</td>
<td>Often used in conjunction with ArborText Editor for DITA content.</td>
<td>astoriasoftware.com</td>
</tr>
<tr>
<td>DITA CMS</td>
<td>IXIASOFT</td>
<td>Manages the DITA documentation process.</td>
<td>Supports the oXygen and XMetaL editors.</td>
<td>ixiasoft.com</td>
</tr>
<tr>
<td>DocZone</td>
<td>Really Strategies Inc.</td>
<td>Supports the content lifecycle.</td>
<td>Part of the Really Strategies RSuite product line. An XMetaL plugin supports DITA authoring.</td>
<td>doczone.com</td>
</tr>
<tr>
<td>Trisoft DITA</td>
<td>SDL</td>
<td>Component content management system.</td>
<td>Manages XML document components. Works with other SDL products to provide translation and localization support.</td>
<td>sdl.com</td>
</tr>
</tbody>
</table>
Localization and translation tools and services

Keyword tags: Alchemy, Asia Online, CEET, commercial tool, EQHO, Lionbridge, localization, Moravia Worldwide, Omni Technologies, SDL Worldserver, TOIN, translation, translations.com, Transperfect, Wordbank, Wordfast

The following table lists and briefly describes some of the popular localization and translation tools. For more information about the tool, see the referenced URL.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDocs</td>
<td>Bluestream XML Content Solutions</td>
<td>Single-source publishing solution for technical documentation.</td>
<td></td>
<td>bluestream.com</td>
</tr>
<tr>
<td>Alchemy set of tools</td>
<td>Alchemy Software</td>
<td>Visual localization solutions.</td>
<td>Products include Alchemy Catalyst, Publisher, and Language Exchange.</td>
<td>alchemysoftware.ie</td>
</tr>
<tr>
<td>Asia Online</td>
<td>Asia Online</td>
<td>Machine translation technologies and services.</td>
<td>Automated language translation platform based on the Statistical Machine Translation (SMT) technology. The platform includes a comprehensive suite of data analysis, data cleaning and data preparation tools and also provides a set of capabilities for post-editing and gathering corrective human feedback on a continuous basis.</td>
<td>asiaonline.net</td>
</tr>
<tr>
<td>CEET</td>
<td>CEET</td>
<td>Translation and localization services focused on languages spoken in Central Europe.</td>
<td>Services include translation, localization, interpreting, and desktop publishing.</td>
<td>ceet.eu</td>
</tr>
<tr>
<td>EQHO</td>
<td>EQHO Communications</td>
<td>Localization services.</td>
<td>Focus on multimedia, voice-over, dubbing, subtitling, document translation, desktop publishing, engineering, and website and software localization.</td>
<td>eqho.com</td>
</tr>
<tr>
<td>Lionbridge</td>
<td>Lionbridge Technologies, Inc.</td>
<td>Translation and localization services.</td>
<td>Provides development, translation, testing, and maintenance services for content and technology applications.</td>
<td>lionbridge.com</td>
</tr>
<tr>
<td>Tool</td>
<td>Vendor</td>
<td>Purpose</td>
<td>Features</td>
<td>For More Information</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Moravia Worldwide</td>
<td>Moravia Worldwide</td>
<td>Translation and localization services.</td>
<td>Translation services and technologies, testing and engineering services.</td>
<td>moraviaworldwide.com</td>
</tr>
<tr>
<td>Omni Technologies</td>
<td>Omni Technologies</td>
<td>Translation and localization services (English to Spanish and Brazilian Portuguese).</td>
<td>Translation, terminology, and localization services. Quality-assurance, transcription, and post-machine translation services.</td>
<td>omnitechnologies.com</td>
</tr>
<tr>
<td>SDL WorldServer</td>
<td>SDL plc.</td>
<td>Linguistic technologies, process automation, content repository integration, and management services.</td>
<td>Formerly Idiom WorldServer. Complete solution for managing authoring, publishing, and localization of DITA content. Also provides automated machine translation.</td>
<td>sdl.com</td>
</tr>
<tr>
<td>TOIN</td>
<td>TOIN Corporation</td>
<td>Localization, translation, authoring, and system development.</td>
<td>Focused primarily on Asian target locales and languages.</td>
<td>to-in.com</td>
</tr>
<tr>
<td>translations.com</td>
<td>translations.com</td>
<td>Translation technologies and services.</td>
<td>Offerings include the GlobaLink product suite.</td>
<td>translations.com</td>
</tr>
<tr>
<td>Transperfect</td>
<td>Transperfect</td>
<td>Family of companies providing business services in over 100 languages.</td>
<td>Services include professional translation, interpretation, website translation, subtitling, voiceovers, multicultural marketing, diversity and inclusion consulting, and litigation support to multinational companies.</td>
<td>transperfect.com</td>
</tr>
<tr>
<td>Wordbank</td>
<td>Wordbank Limited</td>
<td>Translation and localization services.</td>
<td>Specialization is global marketing communication: transcreation of advertising and marketing material worldwide.</td>
<td>wordbank.com</td>
</tr>
<tr>
<td>Wordfast Translation Studio</td>
<td>Wordfast, LLC</td>
<td>Translation memory software.</td>
<td>Wordfast Classic is a Microsoft Word solution; Wordfast Pro is a standalone platform.</td>
<td>wordfast.com</td>
</tr>
</tbody>
</table>
Legacy content conversion tools and services

The following table lists and briefly describes some of the popular legacy content conversion tools and services. For more information about the tools and services, see the referenced URLs.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Vendor</th>
<th>Purpose</th>
<th>Features</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy Data Conversion Center</td>
<td>Single-Sourcing Solutions, Inc.</td>
<td>Legacy content conversion service center.</td>
<td>Converts various proprietary format document source files to SGML and XML (including DITA).</td>
<td>legacydataconversion.com</td>
</tr>
<tr>
<td>Mif2go</td>
<td>Omni Systems</td>
<td>Converts FrameMaker MIF to DITA and other formats.</td>
<td>Installs as a FrameMaker plugin. Can be used to create DITA XML from structured or unstructured FM files. Requires work to assess your source content and define how the conversion is to be done.</td>
<td>omsys.com</td>
</tr>
</tbody>
</table>
Use cases

The following topics contain a list of typical DITA use cases and a sample DITA use case template.

Sections in this topic:

- *DITA use cases* on page 173
- *Use case template* on page 174

DITA use cases

The following sections describe typical DITA use cases.

Problem/solution scenarios

The following scenarios describe a particular problem and its solution using DITA and DITA Open Toolkit (or another processing mechanism).

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

**Industry scenarios**

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

**Publishing environment scenarios**

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

**Access and distribution scenarios**

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

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

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

**Use case template**

Keyword tags: customer success story template, usage scenario template, use case, use case template

This sample template has been included to promote consistency in writing up use cases for DITA and 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)
DITA 1.2 language standard

Keyword tags: DITA language standard, language standard

The following topics contains information about the DITA 1.2 language standard.

Sections in this topic:

- DITA 1.2 language standard on page 177
- Examples of DITA 1.2 usage on page 179

DITA 1.2 features

DITA 1.2 is a major update to the language standard. The key changes in DITA 1.2 are summarized in the following sections.

The keyref attribute

keyref adds indirection to references in DITA topics.

You define a key in a map together with a link target or text.

You then use that key in a keyref attribute to redirect a link or topicref to a new target, redirect a conref to a new target, add text to a term or keyword, or turn terms into links. Undefined keys revert to the already specified href value.

Extensions to conref

In addition to using keys, conref has been extended to:

- Use a conref range (for example, to reference a set of elements like steps 2 through 5 in a task)
- Do a conref push (to force the contents of an element into a location in another topic). This will allow a base set of DITA topics to be modified without making changing to the base source itself.
- Preserve conref without resolving (for systems that accept this)

Learning and training specializations (eLearning)

The DITA 1.2 language standard includes several modules created by the DITA Learning Subcommittee.

These modules were created by experts in the learning field, and include several specializations of topics and tasks used to design information related to learning activities.
Controlled values for metadata

A new subjectScheme map allows the definition of subjects and relationships; it can be used to define a full taxonomy of information.

A classification domain allows an ordinary map to associate those subjects with any given topic.

Rendering engines may use that information to categorize topics and display selected topics to selected users.

Subject schemes also allow a map to define controlled values for metadata attributes (for example, the values a given team wants to allow in *audience*; processors may then provide validation on those attributes. This allows specifying valid values for metadata and its validation when topics are authored.

Glossary expansion

The glossary capabilities in DITA have been expanded:

• From simple term/definition topics to contain more complex information (for example, parts of speech, alternate terms and usages)
• To combine with the abbreviated-form element to provide support for automatic acronym expansion

The new glossary capabilities do not provide full terminology database support.

Constraints

The new constraint mechanism allows DTD and schema developers to constrain existing modules (for example, to provide a topic where sections allow only blocks, or where phrases do not nest). The constraint mechanism allows this to be done without modifying the DTD or schema and without requiring the use of specialization.

Topic and domain integration

The mixing of domain and topic modules is now allowed. For example, the new Task Requirements domain specializes from the task module, and can be used with any task or task specialization.

More general task

The more general task module defined in the 1.2 standard allows users to re-order major sections before and after steps and to include multiple sets of steps. This new task is defined in a new DTD or schema. The existing task model (now known as "strict") is still supported.

New elements

New elements in this release include:

• **bodydiv**, similar to sectiondiv and allows grouping of sections or other body content
• **codelref**, sits inside the codeblock element; acts as a variable for lines of code
• **sectiondiv**, an untitled alternative to nested sections
• **text**, which allows simple text with no associated semantic, and may be placed in keywords, terms, phrases to allow conref of text anywhere
Examples of DITA 1.2 usage

Keyword tags: coderef element, conaction pushreplace, conref extension, DITA language 1.2 standard, DITA language standard, keyref feature, language standard

DITA 1.2 features used in this document

This documentation contains a number of DITA 1.2 feature. The sections below summarize what we used and how.

Extensions to conref, conaction pushreplace

We added a stub shortdesc element to each of our topics as a placeholder for content we could later insert using conaction="pushreplace". This DITA 1.2 feature allows text to be inserted in a topic without changing the topic itself. As an example of how to exploit this, we wrote a Python script that creates a "push" topic for each regular source topic that inserts text listing the keywords contained in the topic and also the keywords from its container topic.

Here is what the shortdesc stub looks like in each topic:

```xml
<shortdesc id="shortdesc"/>
```

The shortdesc is then "pushed" to in an external topic containing this:

```xml
<topic id="topic145">
<title/>
<shortdesc conaction="pushreplace" conref="..\samples.dita#usage/shortdesc">Keyword tags: sample DITA projects</shortdesc>
<body/>
</topic>
```

The keyref attribute

Each of our topics has a prolog that contains values for publisher, copyright and vrm version. In the past these values were hard-coded for each topic. Now we have a single definition for the values in a resource topic and we use keys and keyref to pull them in. It will be much easier to change these values in the future because they are defined only in one place.

Here is the part of the resource topic `variables.dita` that defines the values:

```xml
<prolog>
<publisher id="publisher">VR Communications, Inc.</publisher>
<copyright id="copyright">
<copyrholder>VR Communications, Inc.</copyrholder>
</copyright>
<metadata>
<proinfo>
<prodname id="prodname">DITAinformationcenter</prodname>
<vrmlist id="vrmlist">
<vrm version="4.0" id="version"/>
</vrmlist>
</proinfo>
</metadata>
</prolog>
```
and here is how one of the values is pulled into each of the source topics:

```xml
<publisher conkeyref="vars/publisher"/>
```

They key "vars" is defined in a map as:

```xml
<keydef keys="vars"
href="./DITAinformationcenter_COMMON/variables/variables.dita"/>
```

coderef element

In various places in topics where we include code examples inline, we replaced the inline text with a coderef element instead. Our code inserts are stored in a single code directory.

Features we didn't use

There are other features of DITA 1.2 we looked at, but decided not to use in this version of the documentation. For the most part this was for one of the following reasons:

- The feature was not yet implemented in the Toolkit.
- It didn't do what we needed.
- We couldn't find documentation or examples on how to use it.

The list of such features includes GlossEntry, Learning and Training, the constraint mechanism, and the metadata definition and restriction features.
DITA Open Toolkit plugins

Keyword tags: plug-in

The following topics contain information about DITA Open Toolkit plugins.

Sections in this topic:

- About DITA Open Toolkit plugins on page 181
- Installing the plugins on page 182

About DITA Open Toolkit plugins

Keyword tags: API reference specialization, DITA for Publishers, dita4publishers, Java API reference specialization, Music specialization, plug-in, plugin, specialization

You can extend or enhance the product capabilities provided by DITA Open Toolkit by installing Toolkit plugins. Once installed, a plugin 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/projects/dita-ot/files/

Plug-ins you can install

A few key plug-ins are listed in the following table.

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiref0.9</td>
<td>The API reference specialization provides a general-purpose basis for documenting callable programming libraries.</td>
</tr>
<tr>
<td>DITA for Publishers</td>
<td>DITA for Publishers (dita4publishers) is an open-source, community project to enable the quick and productive use of DITA by publishers.</td>
</tr>
<tr>
<td>javaapiref0.9</td>
<td>The Java API reference specialization provides a basis for documenting Java class libraries.</td>
</tr>
<tr>
<td>Music 1.1</td>
<td>The The Music specialization is a set of sample files that demonstrates how to create a new specialization plug-in. It should be placed in the demo directory inside the DITA Open Toolkit.</td>
</tr>
</tbody>
</table>
Installing the plugins

Keyword tags: installing plugin, plug-in, plugin

Follow these general steps to install a plugin to work with DITA Open Toolkit.

1. Download the plugin .zip file from one of the websites described in About DITA Open Toolkit plugins.
2. Unzip the .zip file and read the installation documentation it contains.
3. Copy the plugin files to a subdirectory in either the demo or plugins subdirectory in the DITA-OT root directory.
4. Run the command ant -f integrator.xml to add the plugin to the Toolkit.
The DITAcorevocabulary is a set of terms related to the DITA language standard and DITA Open Toolkit. It is intended to be used as a controlled or specialized metadata vocabulary for describing and documenting DITA, DITA Open Toolkit, and other resources related to DITA or the Toolkit.

Inspiration for this effort is the Dublin Core Metadata Initiative (DCMI), an organization dedicated to promoting the widespread adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable intelligent information discovery systems. The mission for the DCMI is to provide simple standards to facilitate the finding, sharing and management of information.

In the spirit of the DCMI effort, the key goals (and potential benefits) of the DITAcorevocabulary are to:

- Facilitate the finding, sharing, and management of information about DITA, DITA Open Toolkit, and related technologies (for example, to serve as the basis of a multilingual term base for the DITA domain)
- 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 DITAinformationcenter
- As much as possible, use DCMI metadata in documenting DITAcorevocabulary 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 DITAcorevocabulary.

**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.

**Agile development process**

Agile development refers to one or more workflow methodologies based on the iterative principle, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. Agile development originated in the software code development world, but it is equally applicable to documentation development, both in software and other domains.

Agile methods promote a disciplined project management process that encourages frequent discussion, inspection, and adaptation; a leadership philosophy that encourages collaboration and teamwork; self-organization and accountability; and an agreed-to set of best practices. The key objective is rapid delivery of high-quality product that aligns customer needs with organizational goals.
Our recommended DITA workflow is based on agile development principles and processes.

**Apache Ant**

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

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

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

Ant must be installed in your DITA processing environment for DITA Open Toolkit to function. Ant is included in the Toolkit full easy install distribution.

**Ant processing parameter**

An Ant processing parameter is a property set in the Ant build script.

Examples are "args.draft," "basedir," and "dita.dir."

**Ant script**

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

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

- **System scripts** System-level scripts handle DITA source file processing and transformation into published output. They are an integral part of DITA Open Toolkit and should never be modified by users. The files are located in the DITA-OT 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 also to modify for your own use.

**Audience**

An audience is a target group of users.
This document was written for beginning, intermediate, and advanced users currently implementing or planning to implement DITA or DITA Open Toolkit to produce structured XML documents to be published through any of the supported channels.

Target audience categories and types for this document are:

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

---

**Authoring tool (DITA)**

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

To be DITA-aware, an authoring tool must:

- Check for "well-formedness," that is, that the opening and closing tags match
  An example would be `<note>Text</note>`.
- Check for validity, that is, that all DTD rules are obeyed
  An example would be a check to see whether the tags appearing in the document are nested according to rules in the document's DTDs.

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

---

**Back translation**

Back translation involves translating from one language into another and then translating back into the original to compare the result and thereby avoid translation errors.

---

**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.

**Examples**
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 PDF).

CMS (Content Management System)

If your DITA project involves large numbers of topics, many authors, or geographically distributed authoring and production teams, you may benefit from the features provided by a content management system (CMS), which might include:

- Workflow support
- Validation of topic links
- Support for the semantic web
- Localization (translation) support

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

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

CSS (Cascading StyleSheet)

A CSS file specifies the look and feel of HTML and XHTML documents. DITA Open Toolkit provides default stylesheets; you can override the defaults by including a CSS file of your own in the build.

Choice table (<choicetable>)

In a task step, a choice table (<choicetable>) presents the user with two or more options (choices) to complete the step.
Collection-type attribute (collection-type)

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**

```xml
<reltable>
  <relrow>
    <relcell>
      <topicgroup collection-type="family">
        <topicref href="../release_current/sysreqs.dita"/>
        <topicref href="../installing/installing_overview.dita"/>
      </topicgroup>
    </relcell>
  </relrow>
</reltable>
```

Command element (<cmd>)

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

**Example**

```xml
<taskbody>
  <context>
    <p>Once every 6000 kilometers or three months, change the oil in your car. This will help keep the engine in good condition.</p>
    <p>To change the oil:</p>
  </context>
  <steps>
    <step>
      <cmd>Remove the old oil filter.</cmd>
    </step>
    <step>
      <cmd>Drain the old oil.</cmd>
    </step>
  </steps>
</taskbody>
```

Concept

A concept can mean:

- Conceptual, background, or descriptive information.
- A DITA core information type, which is used to document information of a conceptual nature.
Concept analysis

Concept analysis is analysis of the concept types required in a DITA document or group of documents.

Concept information type

A concept information type contains content of a conceptual nature.

Conditional processing (filtering and flagging content)

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

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

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

You can exclude content based on its metadata. By default, all content is included.

You can flag content based on metadata. By default, no content is flagged.

You can show the revision information for your content. All revision information is hidden by default.

Content

Content is information (for example, the text and graphics that make up a news story appearing on a website) in a DITA file that will be published and delivered to an end user.

Content inventory

A content inventory is an inventory of documents (DITA and non-DITA) in an existing document set.

The inventory is input to a documentation plan involving changes to the existing documents, new use of related documents, or the creation of new documents.
Content reference attribute (conref)

When a topic references a complete topic or smaller piece of content with the conref attribute, the referenced content gets dynamically copied into the referencing topic. If the referenced topic is changed and the document containing the referencing topic is rebuilt, the new version of the referenced topic is automatically replaced.

Content reuse

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

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

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

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

Content specialist audience category

Content specialists are primarily responsible for the content of a DITA project.

This category is a target audience category for this document. It includes information architects; content creators and editors; and graphic, interface, print-document, and website designers.

Context element (<context>)

A context element contains information that helps users understand the background and purpose of a task.

Example

```xml
<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 in 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>
```
**Controlled vocabulary**

A controlled vocabulary is a specified list of topic names or metadata elements and attributes to be included in a DITA document.

**Controlled natural language (CNL)**

Controlled natural languages (CNLs) are subsets of natural languages, created by restricting the vocabulary and grammar to reduce complexity. CNLs are used to make content more accessible to non-native speakers, as the basis for content to be translated into another language, and to make semantic analysis easier. These languages are often used to increase the quality and understandability of technical material, and to aid in machine translation (MT) into other natural languages.

Most controlled languages are based on English, partly because most translations are done from English to another language. Examples of controlled English projects include ASD (AeroSpace Defence Industries Association of Europe) Simplified Technical English (Simplified English, or ASD-STE100), Attempto Controlled English (ACE), IBM's Easy English, and Special English (used by Voice of America in its broadcasts).

**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**

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

**Crowdsourcing**

Crowdsourcing is a neologistic compound of the terms "crowd" and "outsourcing" for the act of outsourcing tasks (often for no remuneration) traditionally performed by employees or contractors to a large group of people in a community or through an open call.

Crowdsourcing is a current trend in the localization and translation industry as a way to get translations done at low or no cost.
**Definition list (<dl>)**

A definition list is a structure for listing terms, products, components, and so forth, along with their definitions. Examples of usage are glossary and product feature lists.

Example

```xml
<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 content**

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

**DITA (Darwin Information Typing Architecture)**

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

DITAinformationcenter consists of:

- A structured/unstructured information solution, or DITAmashup
- Sample DITA projects useful as educational or modeling tools
- Debugging, reporting, repair, file generation, and content automation tools, written in Python or PHP, and suitable for direct use with the DITA Open Toolkit or as models for processing and publishing solutions

DITAinformationcenter, originally published in 2006 as an information-only package under the title *DITA Open Toolkit User Guide*, was produced using the principles and procedures described in the documentation and exhibited in the models.

You can browse the DITAinformationcenter host sites for ideas and information (*XMLdocs.info* is the WordPress site and *DITAinfo.info* is the Drupal site). You can also download from the sites’ Downloads pages any of the free resources that would be of use to you.

**DITAmashup™**

A DITAmashup is a structured DITA project that includes unstructured content from one or more information sources external to the structured topics. The DITA structured framework provides organizational and semantic integrity to the entire body of information, both structured and unstructured.

The key advantage of such an information collection is that a large body of high-value, related content can be assembled and presented to users without the cost and effort of a total DITA conversion.

As an example, suppose the documentation group working for a software company pulls together ("mashes up") the "official" DITA-based documentation for their software product along with unstructured content produced by their sister training and support organizations and a knowledge base of user-produced tips contributed to the support website. In assembling this eclectic but useful collection of information they apply the semantics associated with their structured DITA project and then publish the entire mashup to a website accessible to internal and external users alike.

Various platforms could be used as host sites for mashups. As our first two we chose WordPress and Drupal. We were looking for platforms with the following characteristics:

- Platforms that support XHTML, which is one of the standard output types of the DITA Open Toolkit
- Platforms that support the web services protocol XML-RPC, which allows us to programatically post-process and publish the DITA-based XHTML output as a structured, interconnected collection of topics
- Well-established, popular platforms that provide editing, publishing, and workflow capabilities for the unstructured content, and also provide additional support (for example, search) to the structured content participating in the solution

Our DITAmashups are hosted on our two DITAinformationcenter sites:

- WordPress site: *XMLdocs.info*
- Drupal site: *DITAinfo.info*
**DITA workflow™**

The workflow we recommend for DITA projects is a staged, iterative planning methodology based on the agile software development process.

![DITA workflow diagram](image)

**DITA Open Toolkit**

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

DITA Open Toolkit publishes to the following environments:

- **Key targets**
  - XHTML
  - HTML Help (.chm)
  - PDF
  - Eclipse help

- **Other targets**
  - DocBook
  - Eclipse content
  - JavaHelp
  - troff
  - Word RTF
Plugins allow you to publish to additional environments, for example to ePub format using DITA for Publishers (dita4publishers), which was contributed by Eliot Kimber.

DITA Open Toolkit is a processing tool; it does not have DITA source file authoring capabilities. DITA Open Toolkit can be used to process DITA source files independent of your DITA authoring tool. Some DITA authoring tools have the Toolkit bundled with their editing software; others use proprietary processing software.

**DITA Open Toolkit distributions**

DITA Open Toolkit is available in the following distribution formats:

- Full easy install package distribution
- Small package distribution
- Source distribution

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

**Full package distribution**

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

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

**Note:** This distribution is the one users usually install.

To process DITA documents you must also download and install the Java J2SE SDK.

If one of your target output types is HTML Help, you will probably want to install the Microsoft HTML Help compiler. If you one of your target output types is JavaHelp, you will probably want to install the JavaHelp processor.

**Small package distribution**

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

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

**Source distribution**

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

DocBook is a markup language for technical documentation, available in both SGML and XML forms, and publishable to a variety of formats. DocBook began in 1991 as a joint project between HaL Computer Systems and O'Reilly & Associates. In 1998 it moved to the SGML Open consortium, which subsequently became OASIS.

DocBook is one of the DITA target outputs.

DOCTYPE declaration

A Document Type Declaration, or DOCTYPE, associates a particular SGML or XML document with a Document Type Definition (DTD).

Example

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

DTD (Document Type Definition)

A DTD is the definition of the allowable elements, attributes, and other document pieces of an XML document.

The DITA DTDs are (base) topic, concept, task, reference, map, and bookmap. Every DITA file must reference its DTD in the DOCTYPE declaration.

DTDs are not written in XML syntax.

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

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

A domain element is an element associated with a particular subject area, for example bioengineering, financial services, or software programming.

**Examples of software domain elements:**

<codeblock>, <msgblock>, <varname> and <systemoutput>

**Eclipse**

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

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

**Eclipse content**

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

In contrast to Eclipse help output, which produces XHTML files to be directly displayed by a browser, Eclipse content output produces .dita files that are rendered into .html files on-the-fly by the Eclipse help processor.

**Eclipse help**

Eclipse help is one of the DITA Open Toolkit target outputs. When this output type is specified, the Toolkit produces a directory containing a set of files that can be installed as an Eclipse documentation plugin. This allows the output to be displayed as Eclipse standalone help (that is, independent of the Eclipse IDE) or in an Eclipse Infocenter, which allows you to access Eclipse help on the web.

**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 one 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: ANT_HOME, ANT_OPTS, 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

Figure element (<fig>)

A figure element is a container that allows you to include an image and, optionally, its caption as content in a DITA file.

Filtering content

See

Conditional processing

FO processor

The FO processor (FOP) is an Apache tool that aggregates style and information during the DITA build process for PDF output. FOP is included in the DITA Open Toolkit.

The FO processor (or an equivalent or more sophisticated open-source or commercial processor) must be installed in your DITA processing environment for DITA Open Toolkit to generate PDF output.
Format attribute

The format attribute is the file type of a referenced file or other information source. The most common formats for DITA files are "dita," "xml," and "ditamap." You can also reference other formats like "pdf" and "html."

Example

```
<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 processing sample is a set of DITA source files containing concepts and tasks related to organizing and doing tasks in a garage. The sample map files allow the topics to be published as either a hierarchy or a sequence. The sample also includes Ant scripts that allow you to publish to all target environments.

Note: The garage sample that is part of the DITAinformationcenter is not exactly the same set of files as the garage sample that is part of the DITA Open Toolkit package and used for Toolkit installation verification.

Globalization (g11n)

Globalization is the process of aligning product design, marketing, packaging, and support materials with a global product strategy.

The numeronym "g11n" stands for the first and last letters in "globalization" plus the number of letters in between.

Grocery shopping sample

The grocery shopping sample 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 and a runbuild batch file are also provided.
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.

IBM Information Architecture Workbench

IBM Information Architecture Workbench is an Eclipse-based software tool for modeling human activity as a hierarchy of task and related elements. The Workbench generates graphical representations of DITA maps that can be understood by managers, developers, marketing representatives, and technical communicators.
An information architect can use the Workbench to design DITA maps. A usability practitioner can produce either classic HTA (Hierarchical Task Analysis) diagrams or RAG (Roles and Goals) diagrams.
IBM Information Architecture Workbench was formerly known as the Task Modeler.

**ID attribute**

An ID attribute is an identifier unique within a given topic that can be used to reference the topic. IDs can contain letters, numbers, and underscores.

**Examples**

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

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

**Indexing content**

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

**Example**

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

The above 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*

**Information element (<info>)**

An information element (in a task step), describes additional information required to complete the step beyond the instruction in the command element.

**Example**

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

<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 do the following:

- 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 structures like tables for reference information, and simple sequences of steps for task information)
- Focus on tasks, which is what users are most 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 field to improve information quality. It is based on extensive research and experience, for example Robert Horn's Information Mapping.

The core information types in DITA are concept, task, and reference topic.

Inheritance

In object-oriented programming, inheritance is a way to form new classes using classes that have already been defined. DITA and DITA Open Toolkit are structured around the principle of inheritance.

In DITA, child topics or elements inherit attributes from their parents. For example, metadata applied to a section of a DITA file will automatically be applied to topics contained in the section. Inheritance also plays an important role in DITA specialization, which allows you to extend a base topic to match your specific requirements by defining only the differences between it and its base topic; the bulk of the specialized definition is inherited.

Internationalization (i18n)

Internationalization is the process of creating a flexible system architecture and processes for a product and its documentation so they can be easily customized to meet the needs of a specific locale. Internationalization generally focuses on the technical side of globalization.

The numeronym "i18n" stands for the first and last letters in "internationalization" plus the number of letters in between.

Java

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

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

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

The JDK is an IBM/Sun Microsystems product used by developers to write, compile, debug, and run Java applets and applications.

The JDK must be installed in your DITA processing environment for the DITA Open Toolkit to function, but it is not part of the Toolkit installation package.

» Note: The full JDK may not be required for more recent Toolkit releases.

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.

JavaHelp processor

The JavaHelp processor is a Sun Microsystems product used to generate JavaHelp files, which can be incorporated in applications, components, operating systems, applets, and devices.
JavaHelp is one of the DITA Open Toolkit target outputs. If you plan to produce JavaHelp output, the JavaHelp processor must be installed in your DITA processing environment for DITA Open Toolkit to function, but it is not part of the Toolkit installation package.

**Keyword element (<keyword>)**

A keyword element identifies a term or phrase in an abstract, title, subject heading, or general text that may be used in a special context, such as a glossary or search engine.

**Example**

```xml
<metadata>
  <keywords>
    <keyword>Ant script</keyword>
    <indexterm>Ant scripts</indexterm>
    <indexterm>definition</indexterm>
    <indexterm>usage</indexterm>
  </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. Linking can be implemented through various elements, such as cross-reference (<xref>) and related links (<related-links>), and through relationship tables.

**Locale**

Locale is a language or a language combined with the country where the language is spoken (for example, Afrikaans, Arabic (Egypt), and Arabic (Morocco)).

In a DITA file the locale is specified in the xml:lang attribute (for example, United States English is en-us and Canadian English is en-ca).
Localization (L10n)

Localization may refer to:

• Language localization, which is the process by which content is translated into another language or adapted for a specific country or region of the world
• Internationalization, which is the adaption of computer software for non-native environments, especially other nations and cultures
• Globalization, which involves planning and preparation for localization

The numeronym "L10n" stands for the first and last letters in "localization" plus the number of letters in between.

Localization kit

A localization kit is comprised of content files to be localized (translated) along with additional product and documentation information that will be useful to the translation organization in bidding on the translation job or completing the translation successfully.

The following are often included in a localization kit:

• A glossary or core vocabulary that includes key terms and their definitions (in both source and target languages, if possible)
• The complete set of source files to be localized as target files
• A list of the tools used to produce the source files along with publishing instructions
• Any previously translated material and/or translation memories
• Linguistic style guides for source and target languages
• Localizable graphics in their source format

DITA Toolkit processing log

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

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

Localization Service Provider (LSP)

LSPs typically provide localization services (for example, translate labels that appear in the user interface of software products) and translation services (for example, translate technical documentation from English to Arabic).

Other services might include video voiceovers and subtitles for movies.

Example LSPs: CEET (specializing in Central Europe), iDisc (specializing in Spain and Latin America), and Moravia (worldwide).
Machine translation (MT)

Machine translation is a subfield of computational linguistics that investigates the use of computer software to translate one natural language to another. The techniques are particularly effective in domains where formal language is used.

Machine translation systems are generally rule-based (for example, Yahoo Babel Fish), statistical (for example, Google Translate and Language Weaver), or a hybrid system (for example, Microsoft Translator).

Depending on the tools used to carry it out, machine translation produces results with varying degrees of quality compared to what can be achieved by a skilled human being translating the same text. Today companies that provide translation and localization services might use machine translation as a starting point to translate source text, followed by human post-editing to achieve a final result. Machine translation by itself might be used to provide instant translation of text for static web pages and social networking sites.

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."

Metadata

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

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

Example

```
<metadata>
  <keywords>
    <keyword>Ant script</keyword>
    <indexterm>Ant script</indexterm>
    <indexterm>definition</indexterm>
    <indexterm>usage</indexterm>
  </keywords>
  <prodinfo>
    <prodname>DITAinformationcenter</prodname>
  </prodinfo>
  <vrmlist>
    <vrm version="4.0"/>
  </vrmlist>
</metadata>
```
Navigation title (<navtitle>)

A navigation title is an alternative title for a topic, specified as an attribute on a topicref element.

Navigation titles are usually shorter than the full title, or they could be set, as in the following example, to allow the use of 'scope="peer"' in related links (see Related links).

Example

```xml
<concept id="audience">
  <title>Audience type</title>
  <titlealts><navtitle>Audience</navtitle></titlealts>
  <shortdesc/>
  <prolog>
    OASIS (Organization for the Advancement of Structured Information Standards)
    OASIS is a not-for-profit, global consortium that drives the development, convergence, and adoption of e-business standards.
    DITA is an OASIS standard.
    An ordered list is a list (typically numbered) in which the order of list items is significant (for example, steps in a procedure.)
    Example
    ```xml
    <conbody>
      <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>
    ```
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 a DITA Open Toolkit target output.

Phrase element (<ph>)

A phrase element is used to organize content for reuse or conditional processing (for example, when part of a paragraph applies to a particular audience).

Plugin

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

Postreq element (<postreq>)

In a task, a postreq element specifies something a user needs to do after completing the task.

Example

```
<taskbody>
<prereq>Do your produce shopping <i>after</i> you have bought your canned goods.
Otherwise, the cans might bruise the fruits and vegetables!</prereq>
<context>Remember to look for local produce in season.
The fruits and vegetables you buy will be fresher and cheaper!
Say you're shopping in August for peaches grown locally.
When you get to the produce section of your grocery store:</context>
<steps>
```
<step>
<cmd>Get a plastic or paper bag to hold the peaches.</cmd>
</step>

<step>
<cmd>Pick out the freshest peaches you can find, and put them gently into your bag.</cmd>
<info>To avoid bruising, don't put more than 6 peaches in each bag.</info>
</step>

<step>
<cmd>Put the bag gently into your grocery cart.</cmd>
</step>

</steps>

<postreq>When you check out, be sure the grocery clerk also handles your peaches carefully.</postreq>

</taskbody>

Prereq element (<prereq>)

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

Example

<taskbody>

<prereq>Do your produce shopping <i>after</i> you have bought your canned goods. Otherwise, the cans might bruise the fruits and vegetables!</prereq>

<context>Remember to look for local produce in season. The fruits and vegetables you buy will be fresher and cheaper! Say you're shopping in August for peaches grown locally. When you get to the produce section of your grocery store:</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>

</taskbody>

Print-document designer

A print-document designer is a person responsible for designing DITA print 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 (building)

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

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

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

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

A product manager is a person responsible for designing overall processes and procedures, and setting schedules for DITA projects.

Product manager is one of the target audience types for this document, in the administrators and managers category.

Prolog element (<prolog>)

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

Example

```
<prolog>
  <author type="creator">Anna van Raaphorst</author>
  <copyright>
    <copyryear year=""/>
    <copyrholder>VR Communications, Inc.</copyrholder>
  </copyright>
  <critdates>
    <created date="2010-January-18"/>
    <revised modified="2010-June-20"/>
  </critdates>
  <metadata>
    <keywords>
      <keyword>grocery shopping</keyword>
      <keyword>canned goods</keyword>
    </keywords>
  </metadata>
</prolog>
```

RDF (Resource Description Framework)

RDF (Resource Description Framework) 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.

Reference analysis

A reference analysis is an analysis of the reference information required in a DITA document or group of DITA documents.
Reference information type

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

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

Related links element (<related-links>)

A related links element is a container for linking related topics to a given topic.

Example

```xml
<related-links>
<link href="audience.dita" scope="peer"/>
<link href="../evaluating/framework.dita" scope="local"/>
</related-links>
```

Relationship table

In a DITA map, a relationship table is a systematic structure for creating and maintaining relationships among DITA topics.

The relationships are displayed in a tabular format. Each row of the table represents a relationship, which is usually rendered as a link. Each cell lists one or more participants in the relationship. No relationships exist between the rows of the table.

Example

```xml
<!-- Relationship table -->
<!-- The related concept, task, and reference files point to each other -->
<reltable>
<relheader>
<relcolspec type="concept"/>
<relcolspec type="task"/>
<relcolspec type="reference"/>
</relheader>
<relrow>
<relcell>
<topicref href="concepts/about_produce.dita"/>
</relcell>
<relcell>
<topicref href="tasks/choosing_produce.dita"/>
</relcell>
<relcell>
<topicref href="reference/produce.dita"/>
</relcell>
</relrow>
<relrow>
<relcell>
<topicref href="concepts/about_produce.dita"/>
</relcell>
<relcell>
<topicref href="tasks/choosing_produce.dita"/>
</relcell>
<relcell>
<topicref href="reference/produce.dita"/>
</relcell>
</relrow>
<relcell>
```
Result element (<result>)

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

Example

```xml
<taskbody>
  <context>
    <p>Keep your car looking great by washing it regularly.</p>
  </context>
  <steps>
    <step>
      <cmd>Move the car onto the driveway.</cmd>
    </step>
    <step>
      <cmd>Attach the water hose to a spout and pull the free end over to the car.</cmd>
    </step>
    <step>
      <cmd>Fill a bucket with soapy water.</cmd>
    </step>
    <step>
      <cmd>Use a sponge to apply the soapy water to the car and scrub off the dirt.</cmd>
    </step>
    <step>
      <cmd>Rinse the car by spraying clean water from the hose.</cmd>
    </step>
    <step>
      <cmd>Dry the car using a dampened chamois.</cmd>
    </step>
  </steps>
  <result>
    <p><image href="../images/carwash.jpg" alt="washing the car" height="171" width="249"/></p>
  </result>
</taskbody>
```
RSS (Really Simple Syndication)

RSS is a family of web feed formats written in XML and used in content syndication by enabling applications like RSS readers to find out when new information is available on a website.

Since creation of output from DITA source files is automated by using Ant, it is possible to create or update RSS information about the output at the same time the output is produced. The RSS file can then be uploaded along with the output so its availability can be known to those subscribed to the RSS feed.

SAXON XSLT processor

SAXON transforms DITA source files into rendered output using the DITA XSLT stylesheets.

The SAXON XSLT processor is included in the DITA Open Toolkit.

Schema

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

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

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

The following figure shows the schema for DITA Concept.

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <xs:element name="concept-class">
    <xs:complexType>
      <xs:sequence>
        <xs:group ref="title"/>
        <xs:group ref="title-alt" minOccurs="0"/>
        <xs:choice minOccurs="0">
          <xs:group ref="shortdesc"/>
          <xs:group ref="abstract"/>
        </xs:choice>
        <xs:group ref="prolog" minOccurs="0"/>
        <xs:group ref="conbody" minOccurs="0"/>
        <xs:group ref="related-links" minOccurs="0"/>
        <xs:group ref="concept-info-types" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
      <xs:attribute name="conref" type="xs:string"/>
      <xs:attributeGroup ref="select-atts"/>
      <xs:attribute ref="ditaarch:DITAArchVersion"/>
      <xs:attribute name="outputclass" type="xs:string"/>
      <xs:attributeGroup ref="localization-atts"/>
      <xs:attributeGroup ref="global-atts"/>
      <xs:attribute ref="class" default="- topic/topic concept/concept "/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
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

Before you use DITA Open Toolkit, be sure your operating environment meets the system requirements described in System requirements and supported applications.</section>

Search title element (<searchtitle>)

A search title is an alternative title used in search result summaries.

Segment

A segment is a group of words that is translated as a conceptual unit. Segments are typically clauses or sentences.

Short description

A short description is text that briefly introduces and describes a topic. In DITA, short descriptions are tagged with the short description element (<shortdesc>).

Example

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.

The DITA short description has several functions:

- To act as a sort of "thesis paragraph" for the topic, letting readers quickly assess whether it's worth continuing
- To act as a link hover help in related topics, letting uncertain readers get a sense of the target's content when the title isn't enough
- To act as a summary of the content in generated overviews, letting readers who are drilling down from higher-level content navigate successfully to the right answer through progressive disclosure instead of massive scanning

Simple list (<sl>)

A simple list is a list in which the order of list items is not significant. A simple list is similar to unordered list, but in a simple list the items are not marked with a bullet or other symbol.

The following simple list functions as a topic table of contents.

Example

```
<section>
<p>Sections in this topic:
<sl>
<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 (<simpletable>)

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

Example

```
<simpletable>
<thead>
<stentry>Name, description</stentry>
<stentry>Location</stentry>
</thead>
```
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.

SourceForge website

SourceForge is a download location for many DITA-related tools, documents, and samples.

Source language

A source language is the language from which content is translated.

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.

Step element (<step>)

A step element is a micro-task that a user needs to perform to complete a more high-level task. For example, three of the steps in making a peanut butter sandwich might be finding the peanut butter jar, opening it, and spreading the peanut butter on a piece of bread with a knife.

Example

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 Evaluating DITA and DITA Open Toolkit 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 System requirements and supported applications.</cmd>
</step>
...

## Structure element
A structure element is a base element that you can use with every DITA information type. Examples are topic, title, short description, and body.

## Structured writing
See *Topic-based authoring*

## Stylesheet
A stylesheet is a mechanism for adding style (for example, fonts, colors, and spacing) to web documents.

## Staff manager
A staff manager is a person responsible for managing DITA projects and the personnel involved in project planning and implementation.

Manager is one of the target audience types for this document, in the administrators and managers category.

## Table element (<table>)
A table element is a container element to define display properties and layout of tabular items.

Example

```html
<table colsep="1" rowsep="1">
```
<table>
<thead>
<tr>
<th>Description</th>
<th>Information source</th>
</tr>
</thead>
<tbody>
<tr>
<td>dita.xml.org: Official DITA community gathering place and information resource for the DITA OASIS Standard Includes a knowledge base, a wiki, news items about DITA, events, products and services, case studies, user groups, forums, blogs, and a resource directory. Hosted by OASIS.</td>
<td>[xref href=&quot;<a href="http://dita.xml.org">http://dita.xml.org</a>&quot; format=&quot;html&quot; scope=&quot;external&quot;]</td>
</tr>
<tr>
<td>Cover Pages: Articles, presentations, and facts about DITA</td>
<td>[xref href=&quot;<a href="http://xml.coverpages.org/dita.html">http://xml.coverpages.org/dita.html</a>&quot; format=&quot;html&quot; scope=&quot;external&quot;]</td>
</tr>
<tr>
<td>IBM site containing a collection of information about DITA</td>
<td>[xref href=&quot;<a href="http://www-128.ibm.com/developer-works/xml/library/x-dita1/">http://www-128.ibm.com/developer-works/xml/library/x-dita1/</a>&quot; format=&quot;html&quot; scope=&quot;external&quot;]</td>
</tr>
</tbody>
</table>

**Target language**

A target language is the language into which content is translated.

**Task analysis**

Task analysis is an analysis of 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.

**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.

**Termbase**

A termbase is a database containing terminology (usually in multiple languages) and related information.

When you open a termbase you specify which languages you want to use as source and target. The termbase tool displays the term in the source language and whatever target languages you have chosen.

Example termbases SDL MultiTerm and IATE (InterActive Terminology for Europe).

**Topic information type**

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

**Topic-based authoring (structured writing)**

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

A document usually contains 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.
Translation

Translation is the process of taking information in one language and transferring it to another language. A good translator considers not only the text itself, but also the conventions, idioms, and expectations of the audience for the target language.

Translation memory (TM)

Translation memory is a database tool that allows a human translator to connect the translated segment with its corresponding source segment.

This capability facilitates consistency by pre-matching segments that have been translated previously and flagging them for approval.

Translation memory managers are most suitable for translating technical documentation and documents containing specialized vocabularies.

It is possible to retrieve from the translation memory an exact match (where the match between the current source segment and the one stored in the TM is a character-by-character match), a fuzzy match (where the match is not 100%, and the system may assign a percentage to the match to indicate how close it is), or no match.

There are a number of standards associated with translation memory, including TMX (Translation Memory eXchange), TBX (TermBase eXchange), UTX (Universal Terminology eXchange), OLIF (Open Lexicon Interchange Format), and XLIFF (XML Localization Interchange File Format).

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 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

<ol>
<li>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:</li>
<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>
</ol>

Vendor audience category

Vendors are persons or organizations creating software (for example, plug-ins) complementary to DITA or DITA Open Toolkit.

This category is a target audience category for this document.

Web service

A web service is an Application Programming Interface (API) that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested service.

In general, a web service API is invoked by an application that creates an XML document called a "request" and sends it to a server on a network. The server then acts on the request and returns its "reply" as another XML document to the application making the request.

The format of the request and the reply is defined in an XML file written in the Web Services Definition Language (WSDL). WSDL is a W3C standard.

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.

**Workflow**

A workflow is a model that is illustrated as a sequence of operations and that represents the real work accomplishments of a person, a group of people, or a concrete or abstract mechanism (for example, the flow of a high-level computer program).

**Writer**

A writer is a person responsible for creating DITA topics and implementing effective topic reuse. Writer is one of the target audience types for this document, in the content specialist category.

**Xalan XSLT processor**

Xalan is a tool developed as part of the Apache XML project that transforms DITA source files into rendered output using the DITA XSLT stylesheets.

You need to install either the Xalan XSLT compiler or the SAXON XSLT processor in your DITA processing environment for DITA Open Toolkit to function, but neither processor is part of the Toolkit installation package.

**XEP processor**

XEP (from RenderX) is an XSL FO and SVG processor that aggregates style and information during the DITA build process for PDF output.

XEP is an alternative to Apache FOP, which is included in the DITA Open Toolkit.

**XHTML**

XHTML (Extensible HyperText Markup Language), is a markup language that has the same expressive possibilities as HTML, but a stricter syntax. XHTML is an application of XML. XHTML is a W3C standard. XHTML is one of the DITA Open Toolkit target outputs.
XML (eXtensible Markup Language)

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

The following figure shows a simple, annotated XML file.

```
<?xml version="1.0"?>
<!-- This is an example -->
<people>
  <person>
    <name>Sue Writer</name>
    <employment status="part time"/>
    <motto xmlns:="http://www.w3.org/1999/xhtml">
      I h:b:love</h:b:love> DITA!
    </motto>
  </person>
</people>
```

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

**XML declaration**

An XML declaration is a processing instruction that identifies a document as XML.

DITA documents must begin with an XML declaration.
Example

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

**xNAL (extensible Name and Address Language)**

xNAL, an OASIS standard, is a structured XML language for representing names and addresses. xNAL is governed by the OASIS Customer Information Quality (CIQ) Technical Committee.

xNAL consists of basic constructs that can form complex names and addresses. These constructs can be used for name and address data exchange or storage.

The DITA Open Toolkit distribution contains xNAL domain elements.

**XPath**

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

XPath is a W3C standard and is used in XSLT.

The following figure shows XPath expressions in an XSLT document.

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

**XSLT (eXtensible Stylesheet Language Transformations)**

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

XSLT stylesheets are written in XML syntax.

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

The following figure shows the XSLT stylesheet for the note element.
The following keywords are used in this document to categorize the content:

- assessing
- authoring tool
- automated documentation
- automating
- benefit
- challenge
- content management tool
- building
- commercial tool
- concept
- controlled vocabulary
- core vocabulary
- customizing
- debugging
- distributing
- DITA
- DITAcorevocabulary™
- DITAinformationcenter™
- DITAmashup™
- DITAworkflow™
- DITA language standard
- DITA Open Toolkit
- editing tool
- evaluating
- getting information
- getting started
- glossary
- implementing
- installing
- introduction
- key concept
- language standard
- legacy conversion tool
- linking
- localization tool
- localizing
- managing
- mashup
- map
- migrating
- output
- PHP
- piloting
- planning
- plugin
- processing
- processing tool
- prototyping
- publishing
- Python
- reference topic
- rendering tool
- reusing
- sample project
- setting up
- source control tool
- specializing
- task
- tool
- topic
- transforming
- translating
- translation tool
- trend
- troubleshooting
- tutorial
- use case
- web service
- workflow
Index

A

access and distribution use cases 173
accessing 107
accessing your information 111
Adobe 165
Adobe FrameMaker 95
alphaWorks IBM Information Architecture Workbench 162
Altova 165
American Translators Association (ATA) 149
AMTA (Association for Machine Translation in the Americas) 149
Ant 58, 60
  processing parameters for 60
Ant build processor 51
Ant parameters 60, 115
  args.artlbl 60
  args.copycss 60, 115
  args.css 60, 115
  args.csspath 60, 115
  args.cssroot 60
  args.dita.locale 60
  args.draft 60
  args.eclipse.provider 60
  args.eclipse.version 60
  args.eclipsecontent.toc 60
  args.eclipsehelp.toc 60
  args.fo.img.ext 60
  args.fo.include.rellinks 60
  args.fo.output.rel.links 60
  args.fo.userconfig 60
  args.ftr 60
  args.gen.task.lbl 60
  args.hdf 60
  args.hd 60
  args.html.classattr 60
  args.html.includefile 60
  args.indexshow 60
  args.input 60
  args.javahelp.map 60
  args.javahelp.toc 60
  args.logdir.toc 60
  args.outext.map 60
  args.xhtml.classattr 60
  args.xhtml.toc 60
  args.xsl 60
  args.xsl.pdf 60
  basedir 60
  clean.temp 60
  dita.dir 60
  dita.extname 60
  dita.input 60
  dita.input.dimname 60
  dita.input.valfile 60
  dita.temp.dir 60
  generate.copy.outer 60

  Ant parameters (continued)
    onlytopic.in.map 60
    outer.control 60
    output.dir 60
    pdf.formatter 60
    retain.topic.fo 60
    transtype 60
    validate 60

  Ant scripts 58
    system-level 58
    user-level 58

  Antenna House 120, 167
  Antenna House Formatter 167
  Apache Derby documentation 160
  Apache FOP 120, 167
  Apache Software Foundation 168
  Apache XML Graphics Project 167
  API reference specialization 181
  Arbortext Editor 165
  architecting 156
  DITAmashups 156
  args.artlbl Ant parameter 60
  args.copycss Ant parameter 60, 115
  args.css Ant parameter 60, 115
  args.csspath Ant parameter 60, 115
  args.cssroot Ant parameter 60
  args.dita.locale Ant parameter 60
  args.draft Ant parameter 60
  args.eclipse.provider Ant parameter 60
  args.eclipse.version Ant parameter 60
  args.eclipsecontent.toc Ant parameter 60
  args.eclipsehelp.toc Ant parameter 60
  args.fo.img.ext Ant parameter 60
  args.fo.include.rellinks Ant parameter 60
  args.fo.output.rel.links Ant parameter 60
  args.fo.userconfig Ant parameter 60
  args.ftr Ant parameter 60
  args.gen.task.lbl Ant parameter 60
  args.hdf Ant parameter 60
  args.hd Ant parameter 60
  args.html.classattr Ant parameter 60
  args.html.includefile Ant parameter 60
  args.indexshow Ant parameter 60
  args.input Ant parameter 60
  args.javahelp.map Ant parameter 60
  args.javahelp.toc Ant parameter 60
  args.logdir.toc Ant parameter 60
  args.outext.map Ant parameter 60
  args.xhtml.classattr Ant parameter 60
  args.xhtml.toc Ant parameter 60
  args.xsl Ant parameter 60
  articles 159
  assessing 21, 27
  Association for Machine Translation in the Americas (AMTA) 149
  Astoria On-Demand 168
Astoria Software 168
ATA (American Translators Association) 149
audience attribute 111
authoring in an IDE use case 173
authoring tools 51, 55, 165
   Arbortext Editor 165
   configuring to use URLs 55
   DITA-FMx 165
   FrameMaker 165
   oXygen 165
   XMetaL Author 165
   XMLmind 165
   XMLSpy 165
automated documentation 125
automating 125

B
backing up DITA source files 152
basedir Ant parameter 60
Bluestream XML Content Solutions 168
bodydiv element in DITA 1.2 177
books and other publications 159
build.xml 58, 142
building (processing) 57, 59, 60, 65, 66, 68, 69, 70
   Ant parameters 60
   overview diagram 59
   to Eclipse content targets 65
   to Eclipse help targets 65, 69, 70
   to HTML Help targets 65, 68
   to PDF targets 65, 66
   to XHTML targets 65
business partner use cases 173

C
cascading stylesheet (CSS) 115
   overriding the default with your own 115
case studies 159
catalog-dita_template.xml 58
catalog-dita.xml 58
challenges in global communications 23
changes required to the legacy workflow for DITA success 29
clean.temp Ant parameter 60
CM Professionals 154
CMS Review 154
CMSs (content management systems) 153, 168
   Astoria On-Demand 168
   DITA CMS 168
   DocZone 168
   Trisoft DITA 168
   XDocs 168
coderef element in DITA 1.2 177
commercial tools 165
common directory 54
Common Sense Advisory 149
commonltr.css Ant parameter 115
communication trends and challenges 23
Comstar DITA project 160
concept topics 84
   creating 84
   concepts 81
conditional processing 111
conferences 163
conref 144
constraints in DITA 1.2 177
content management 154
content management systems (CMSs) 153, 168
   Astoria On-Demand 168
   DITA CMS 168
   DocZone 168
   Trisoft DITA 168
   XDocs 168
content management use cases 173
content reuse 103, 104
controlled language 146
controlled vocabulary use cases 173
core vocabulary 183
Cover Pages 159
creating 83, 84, 85, 86, 90, 123, 124
   concept topics 84
   custom tools 123, 124
ditamaps 90
   maps 90
   reference topics 86
tasks 85
topics 83
cross-reference linking 100
CSS (cascading stylesheet) 115
   overriding the default with your own 115
CSS directory 54
custom tools, creating 123, 124
customization.dir 120
customizing 107, 111, 114, 121, 122
   access 107
   Eclipse help Bookmarks tab 121
   Eclipse help splash page 122
   Eclipse output 121
   output 111
   XSLT stylesheets 114
customizing PDF output 119
customizing XHTML output 115

d debugging 73
directories 54, 55
   common 54
   CSS 54
   DITA-OT 54
doc 54
docbook 54
dtd 54
   lib 54
   out 54
   output file 55
   plugins 54
   preprocess 54
   resource 54
   samples 54
   schema 54
   source file 55
temp 54
directories (continued)
tools 54
troff 54
xsl 54
xslt 54
xslhtml 54
xslrf 54
distributing 155
distributing content 155
distributions of the Toolkit 51
full package 51
small package 51
source 51
DITA 21, 22
planning your initial project 21
setting up your initial project 22
DITA CMS 168
DITA for Publishers plugin 181
DITA language 1.2 standard 177, 179
examples 179
DITA language standard 161, 177
DITA Open Toolkit 37, 51, 161, 166
installing 51
releases 161
DITA plugins 162
DITA projects 38
DITA tools 162
DITA World 159
DITA-FMx 165
DITA-OT directory 54
dita.dir Ant parameter 60
dita.extname Ant parameter 60
dita.input.dirname Ant parameter 60
dita.input.valfile Ant parameter 60
dita.temp.dir Ant parameter 60
dita.xml.org 159
dita4publishers plugin 37, 181
ditaauthors.py 76, 77
ditadebug.php 76
ditaedit.php 76, 77
ditaids.py 76, 77
ditakeywords.py 76, 77
ditamaps 89, 90
creating 90
DITAmashups 27, 155, 156, 157
about 155
architecting 156
publishing 157
ditarepair.py 76, 77, 78
ditastat.py 76, 77
ditatargets.xml 58
ditaunused.py 76, 77
ditaval file 111, 113
DITAworkflow 21, 27, 29
doc directory 54
DocBook 37
docbook directory 54
Document Type Definition (DTD) 47
DocZone 168
downloading 51
Java SDK 51
JDK 51
Toolkit 51
Drupal 157
and DITAmashups 157
DTD (Document Type Definition) 47
dtd directory 54
Dublin Core metatags 110

E
Eclipse content 37
Eclipse content processor 51
Eclipse content targets, processing (building) to 65
Eclipse help 37, 121, 122
customizing the splash page 122
deleting the bookmarks tab 121
overview 121
Eclipse help targets, processing (building) to 65, 69, 70
Eclipse output 121
customizing 121
Eclipse software development environment 173
Eclipse targets, processing (overview) 69
ing tools 165
Arbortext Editor 165
DITA-FMx 165
FrameMaker 165
oXygen 165
XMetaL Author 165
XMLmind 165
XMLSpy 165
editors (authoring tools) 55
configuring to use DTD or schema URLs 55
ElementTree Python module 126
ELIA (European Language Industry Association) 149
EPUB format using dita4publishers plugin 37
e rror messages 74, 75
produced by the Java infrastructure 75
produced by the Toolkit 74
produced during processing 74
European Language Industry Association (ELIA) 149
evaluating 21, 27
evaluating DITA and DITA Open Toolkit 31
events 163
eXtensible Markup Language (XML) 47
eXtensible Stylesheet Language Transformations (XSLT) 48

F
file generation tools 76, 77
filtering content 111, 113
FO processor 51
footer, including in XHTML output 119
forums 162
FrameMaker 165
FrameMaker editor 95
frameset, publishing XHTML output as 117
full package distribution of the Toolkit 51
G
GALA (Globalization and Localization Association) 149
generate.copy.outer Ant parameter 60
getting information 159
getting started 21, 27
getting started with DITA 21, 22
    planning your initial project 21
    setting up your initial project 22
global communication trends and challenges 23
globalization (G11N) 149
Globalization and Localization Association (GALA) 149
glossary expansion in DITA 1.2 177
Google Translate 144, 145
grocery shopping sample 82, 92
    processing (building) 92

H
header, including in XHTML output 119
HTML Help 37
HTML Help compiler 51
HTML Help targets, processing (building) to 65, 68
HyperWrite Pty Ltd 166

I
IBM Eclipse content processor 51
ICU globalization routines 51
icu4j globalization routines 51
implementing 21, 27
implementing your design 33
indexing content 107
industry use cases 173
information design 137, 161
information design reuse (specialization) 135, 136
information types 81
installing 51, 53, 182
    DITA Open Toolkit 51
    Java SDK 51
    JDK 51
    plugins 182
    Toolkit 51, 53
integrator.xml 58
internal use cases 173
internationalization (I18N) 149
IXIASOFT 168

J
Java 49
Java API reference specialization 181
Java SDK 51
    downloading 51
    installing 51
JavaHelp 37
JavaHelp processor 51
JDK (Java Development Kit) 51
    downloading 51
    installing 51
JustSystems 165

K
key concepts 35, 38, 45
DITA projects 38
    processing 45
    publishing 45
keyref feature in DITA 1.2 177
keywords 227

L
language standard 177
languages, supported 148
legacy content conversion tools 171
    Legacy Data Conversion Center 171
    Mif2Go 171
legacy content, migrating to DITA 95, 96
legacy workflow 27
Leximation 165
lib directory 54
library system use cases 173
library systems 153, 168
    Perforce SCM 168
    Subversion 168
linking 99, 100, 101
    using related links 100
    using relationship tables 101
    using xrefs 100
Linux, installing the Toolkit on 51
Localisation Research Centre 149
localization (L10N) 149
Localisation Industry Standards Association (LISA) 149
localization tools and services 169
    Alchemy 169
    Asia Online 169
    CEET 169
    EQHO 169
    Lionbridge 169
    Moravia Worldwide 169
    Omni Technologies 169
    SDL WorldServer 169
    TOIN 169
    translations.com 169
    Transperfect 169
    Wordbank 169
    Wordfast 169
localization use cases 173
localizing (translating) 139, 140, 142
    in PDF output targets 142
    in XHTML output targets 140
log file (Ant processing), capturing and using 73
lone writer scenario use cases 173
lxml Python module 126

M
Mac OS, installing the Toolkit on 51
machine translation 144, 145, 146
MadCap Flare 166
MadCap Software 166
managing content 151, 154
maps 89, 90
creating 90
mashups 155
mashups, DITA 155, 156, 157
about 155
architecting 156
publishing 157
messages 74, 75
produced by the Java infrastructure 75
produced by the Toolkit 74
produced during processing 74
metadata 109
metadata attributes 111
metadata controlled values in DITA 1.2 177
metaWeblog API 157
Microsoft HTML Help compiler 51
Microsoft Translator 144, 145
Mif2Go 95, 171
migrating content 95
migrating legacy content to DITA 95, 96, 98
migrating to a new tool use cases 173
minidom Python module 126
modeling content use cases 173
multilevel maps, processing using 92
Music specialization 181

N
NetBeans software development environment 173

O
OASIS (Organization for the Advancement of Structured Information Standards) 166
object model (XML) 47
Omni Systems 95, 171
online DITA DTDs and schemas 55
onlytopic.in.map Ant parameter 60
ontology use cases 173
Organization for the Advancement of Structured Information Standards (OASIS) 166
organizing your source and output directories 151
otherprops attribute 111
out directory 54
outer.control Ant parameter 60
output 111
output file directories 55, 151
output.dir Ant parameter 60
outputclass common attribute 115
outputs supported by DITA Open Toolkit 37
DocBook 37
Eclipse content 37
Eclipse help 37
JavaHelp 37
PDF (Portable Document Format) 37
troff 37
Word RTF (Rich Text Format) 37
XHTML 37
OWL (Web Ontology Language) 110
oXygen 165

P
PDF 37, 120
customization.dir 120
landscape orientation 120
pdf.formatter 120
PDF cover page, adding a product logo to 119
PDF output 119
customizing 119
PDF rendering tools 167
PDF targets, processing (building) to 65, 66
dicta4publishers plugin 181
downloading 181
installing 182
Java API reference specialization 181
Music specialization 181
overview 181
plugins 162, 181, 182
API reference specialization 181
DITA for Publishers plugin 181
dicta4publishers plugin 181
downloading 181
installing 182
Java API reference specialization 181
Music specialization 181
overview 181
plugins directory 54
preprocess directory 54
presentations 159
pretargets.xml 58
problem/solution use cases 173
processing (building) 45, 46, 57, 59, 60, 65, 66, 68, 69, 70, 88, 92
a single topic 88
Ant parameters 60
overview 46
overview diagram 59
the grocery shopping sample 92
to Eclipse content targets 65
to Eclipse help targets 65, 69, 70
to HTML Help targets 65, 68
to PDF targets 65, 66
to XHTML targets 65
using multilevel maps 92
processing tools 166
DITA Open Toolkit 166
MadCap Flare 166
WinANT 166
XMLmind XSL-FO Converter 166
product attribute 111
project components 38, 41, 42
projects 38
prototyping 21, 27
prototyping use cases 173
PTC (Parametric Technology Corporation) 165
publishing 57, 157
DITAmashups 157
publishing content 45, 46
overview 46
publishing environment use cases 173
Python 76, 78, 125, 126, 127, 128, 129, 130, 131, 144, 145, 157
and DITA 125
creating a DITA topic with Python 128
ditarerepair program 78
element usage statistics 127
ElementTree 126
modifying XML with Python 129
scripts to publish DITA mashups 157
tools 76
XML packages 126

R
RDF (Resource Description Framework) 110
Real Story Group 154
Really Simple Syndication (RSS) 158
Really Strategies Inc. 168
reference topics 81, 86
related links 100
relationship tables 101
releases of DITA Open Toolkit 161
rendering tools 167
Antenna House Formatter 167
Apache FOP 167
XEP PDF Engine 167
RenderX 167
repair tools 76
repairing bad references with ditarerepair.py 78
reporting tools 76, 77
Resource Description Framework (RDF) 110
resource directory 54
retain.topic.fo Ant parameter 60
reuse 103, 104, 137, 161
reusing content 103
rev attribute 111
RSS (Really Simple Syndication) 158
runtime environment variables 51

S
sample DITA projects 82, 160
Apache Derby documentation 160
Comstar 160
DITA information center 160
garage sample 160
grocery shopping 82
grocery shopping sample 160
sample files and projects 159
sample project components 38, 41, 42
output files 42
processing files 41
source files 38
samples directory 54
sax Python module 126
schema directory 54
schemas 48
SDL 168
sectiondiv element in DITA 1.2 177

services 163
Ant customization 163
consulting 163
DITA specialization 163
domain specialization 163
getting started 163
implementation 163
information architecture 163
product extensions 163
terminology 163
training 163
XML/XSLT customization 163

setting up 55
setting up a DITA information center project in your environment 17
setting up a working environment 55
source and output file directories 55
setting up your initial DITA project 22
Simple Knowledge Organization System (SKOS) 110
Simplified English 146
Simplified Technical English 146
Single-Sourcing Solutions, Inc. 171
SKOS (Simple Knowledge Organization System) 110
small package distribution of the Toolkit 51
solutions to global communication challenges 24
source control systems 153, 168
Perforce SCM 168
Subversion 168
source distribution of the Toolkit 51
source file directories 55, 151
specialization (information design reuse) 135, 136
specializations 181
API reference 181
Java API reference 181
Music 181
specializing 135
specializing (information design reuse) 137, 161
staged implementation 33
stylesheet 48
stylesheet processing 114
Subversion 168
Sun JavaHelp processor 51
supported languages 148
SyncRO Soft Ltd. 165
system requirements for the Toolkit 51
system-level Ant scripts 58
build.xml 58
catalog-dita-template.xml 58
catalog-dita.xml 58
ditatargets.xml 58
integrator.xml 58
pretargets.xml 58

T
targets supported by DITA Open Toolkit 37
DocBook 37
Eclipse content 37
Eclipse help 37
JavaHelp 37
PDF (Portable Document Format) 37
troff 37
targets supported by DITA Open Toolkit (continued)
  Word RTF (Rich Text Format) 37
  XHTML 37
task changes in DITA 1.2 177
task topic 85
tasks 81, 85
  creating 85
TAUS (Translation Automation User Society) 149
taxonomy use cases 173
tcworld 149
temp directory 54
template for use cases 174
text element in DITA 1.2 177
tool set for debugging, reporting, repair, and file generation 76
Toolkit 51, 53
  downloading 51
  installing 51, 53
  upgrading 51
  verifying the installation 53
tools 123, 165, 166, 167, 168, 169, 171
  authoring (editing) 165
  CMSs (content management systems) 168
  commercial 165
  creating custom 123
  legacy content conversion 171
  localization and translation 169
  PDF rendering 167
  processing 166
  source control and library systems 168
tools and techniques for troubleshooting the build process 77
tools directory 54
topic and domain mixing in DITA 1.2 177
topic authoring 82
topics 81, 82, 83, 88
  creating 83
  processing (building) a single 88
transforming 57
translating (localizing) 139, 140, 142
  in PDF output targets 142
  in XHTML output targets 140
translation 149
Translation Automation Society (TAUS) 149
translation tools 169
  Alchemy 169
  Asia Online 169
  CEET 169
  EQHO 169
  Lionbridge 169
  Moravia Worldwide 169
  Omni Technologies 169
  SDL WorldServer 169
  TOIN 169
  translations.com 169
  Transperfect 169
  Wordbank 169
  Wordfast 169
translation use cases 173
transtype Ant parameter 60
trends in global communication 23
Trisoft DITA 168
troff 37
troff directory 54
troubleshooting 73
troubleshooting the build process 76, 77
  tools and techniques for 76, 77
tutorials 159

U
use cases 173, 174
  access and distribution 173
  business partner 173
  content management system 173
  controlled vocabulary 173
  industry 173
  internal 173
  library system 173
  localization 173
  lone writer scenario 173
  migrating to a new tool 173
  modeling content 173
  ontology 173
  prototyping 173
  publishing environment 173
  taxonomy 173
  template for 174
  translation 173
user groups 162
user-level Ant scripts 58

V
validate Ant parameter 60
vendors or creators of commercial tools 165, 166, 167, 168, 171
Adobe 165, 167
Altova 165
Antenna House 167
Apache Software Foundation 168
Apache XML Graphics Project 167
Astoria Software 168
Bluestream XML Content Solutions 168
HyperWrite Pty Ltd 166
IXIASOFT 168
JustSystems 165
Leximation 165
MadCap Software 166
OASIS (Organization for the Advancement of Structured Information Standards) 166
Omni Systems 171
Organization for the Advancement of Structured Information Standards (OASIS) 166
Perforce Software 168
PIXware 165, 166
PTC (Parametric Technology Corporation) 165
Really Strategies Inc. 168
RenderX 167
SDL 168
Single-Sourcing Solutions, Inc. 171
SyncRO Soft Ltd. 165
verifying the Toolkit installation 53
W
W3C Internationalization (I18N) Activity 149
Web Ontology Language (OWL) 110
web services 130, 131
webinars 163
websites 159
WinANT 166
Windows, installing the Toolkit on 51
Word RTF (Rich Text Format) 37
WordPress 157
    and DITAmashups 157
WordPress blog API 157
workflow 27, 29
    changes required for DITA success 29
    legacy 27
workflow_dita 29
WSDL 131

X
Xalan XSLT processor 51
XDocs 168
XEP PDF Engine 120, 167
XHTML 37
XHTML output 115, 117, 119
    customizing 115
    including a header and footer in 119
    publishing as a frameset 117
XHTML targets, processing (building) to 65
XMetaL Author 165
XML (eXtensible Markup Language) 47
XML catalog resolver 51
xml lang attribute 139, 140, 144
XML object model 47
XML-RPC 157
XMLmind 165
XMLmind XSL-FO Converter 166
XMLSpy 165
XPath 49
xref linking 100
xsl directory 54
xsifo directory 54
xsihtml directory 54
xslrtf directory 54
XSLT (eXtensible Stylesheet Language Transformations) 48
XSLT stylesheets 114