Ant Quick Start Guide for DITA Open Toolkit

Contents

Introduction to the DITA Open Toolkit and Ant

Understanding the role of DITA Open Toolkit and Ant for technical writers.

This section describes the role of DITA Open Toolkit, Ant, and when DITA-OT is a logical choice for your team.

Overview

Introducing the most important authoring tool since FrameMaker.

DITA-based writers gain numerous advantages using this powerful, single-source, document solution. Indeed, I think of DITA as the Holy Grail of technical documentation, the object for which many a manager has sent me on long, fruitless searches in the past, only to return with half-hearted recommendations for a combination of tools that required hours of manual tweaking to reproduce a document in just one alternative format.

Today, I can tell my manager that the Grail has been found, and I can produce a handful of different document output types simultaneously. This is a breakthrough technology for technical writers. In industry jargon, "single source" has previously meant writing in FrameMaker, then importing your source into another expensive application to produce a second output format, typically online help. To gain just a second format from the source often required tedious hours, sometimes days, massaging the text after it had been "translated" into an online help system. DITA-OT allows technical writers to produce seven output formats at the same time.

However, several technologies make this magic happen. Motivated, or just curious, writers will want a more advanced understanding of what makes DITA tick. To do so, you will need to learn how to write Ant build scripts for the DITA-OT and invoke them from the command line.

- *Ant*
- *DITA Open Toolkit*

Hopefully, this guide will motivate you to study DITA-OT further and encourage your publications team to implement a single-source, DITA-based documentation solution.

About DITA-OT Toolkit Roles

Descriptions of the following user roles for the toolkit: CSS Customizers, Build Scripters, and potentially End Users of third-party authoring tools.

If you don't know what a build script is, or whether your authoring tool should or does use DITA OpenToolkit, you may be reading the right document in the in the DITA-OT documentation suite. User Roles are a useful way to present and approach the documentation, enabling the reader to bypass information that is not relevant to the specific task she is performing when this document is read. The followige roles are addressed in this guide:

- CSS Stylesheet Customizers
- Ant Build Scripters
- Online Help developers

Many readers may be unaware of the toolkit's presence, but it is the "engine" of whatever writing tool you use to write and maintain XML-based, DITA-compliant content. CSS Stylesheet Customizers read this guide when they are unable to specify custom stylesheets, or `.css` files from their third-party authoring application. Build Scripters consult this guide when the toolkit is unable to process an Ant build script. The Quick Start Guide features an up-to-date tables of all supported parameters for both Ant and the Java command line interface, useful information if your documentation is generated automatically as part of an automated daily build. As my first manager warned me long, long ago, 'The howling hordes descend when the build breaks'.

If you need to fix a broken build, time is of the essence. If you know how to edit CSS stylesheessor run Ant build scripts, then read on. If you don't recognize CSS and Ant, you're likely an End User and you should refer to your

third-party documentation. If like the author, you also enjoy hacking with your tools this is definitely the guide for you. Although the Quick StartGguide currently provides more information for build scripters than css customizers, revisions will describe the use of XSLT and the new plugin architecture for this audience. When your third-party authoring tool breaks down, this guide is the mechanic's manual you didn't know you had. If you're unafraid of a command line and willing to "get your hands dirty" under the hood, an up-to-date, and maintained table of Ant build properties can be the difference between a broken build and a bad day or a gently purring build in a background `chron` chron job that never demands attention. If the build has already broken by the time you read this, your may be the "build hero" who magically "fixes" the build which no one cares about, except that it is usually only then that the fire-breathing IT dragon returns to it's cave and everyone can relax again until the next "emergency".

What is Ant?

Learning about the blurred line between code and documentation and why technical writers need to learn Ant.

If your DITA authoring tool uses the Open DITA Toolkit to generate your documents, you're already using Ant. So, what is Ant, anyway? Ant is a *build tool,* a program used to compile other programs. If you work as a writer in the enterprise software industry, you know that software engineers regularly produce several versions of whatever software they are working on before they release it to the public. Each compilation is called a *build*. Dozens, sometimes hundreds, of builds are compiled before the RTM (release to manufacturing) or GA (general acceptance) build is certified as the official release build. You can often determine the release build of whatever software you are using by reading the Help->About dialog box. For example, my version of XMetal is 5.5.0.219. This means that build 219 was the official release build for XMetal, version 5.5.

Writers also draft, write, revise, and rewrite their documents many times before releasing a document to the public. We tend to call these drafts, rather than builds. You probably saved drafts of your documents in a document repository or CMS, a content management sytem, in the past, but I doubt you thought of your draft, even though it was versioned by the repository, as a software build that either compiled or failed to compile. A successful document "build" meant only that a document opened in your authoring tool the next day, not that all the related documents also opened successfully and "compiled" together to produce a version, albeit incomplete, of the documentation that will eventually make its way to your readers. Hence, the "build" metaphor did not extend beyond the programming code in the engineers' cubicles to the documents crafted by the writers.

DITA changes that forever; the build metaphor is as relevant to you as to the engineers. Behind the user interface of your authoring tool, the DITA Open Toolkit uses Ant to compile a build every time you try to generate your single-source documents. If you want to customize the way DITA-OT generates your documents, you will need to open the hood, so to speak, and get your hands dirty with the internals of the Toolkit and Ant.

What is DITA-OT?

What is the DITA Open Toolkit, anyway?

The DITA Open Toolkit is a popular, free, open-source tool used to transform DITA documents and maps into the output document formats you desire. In fact, most of the proprietary authoring tools use the DITA-OT to transform DITA documentation projects, so you aren't wasting your time learning about how it works. Many errors are more quickly fixed if you understand what the toolkit is doing "beneath the hood" of your authoring tool.

DITA-OT uses Ant to generate your documents. The primary ant script is `build.xml`, which imports several other build scripts to initialize, validate, and transform your .dita documents.

See *Introducing Document Generation* on page 10 for more information about how DITA-OT processes documents.

When Should I Use DITA-OT?

Determining when DITA OT makes sense for your team.

There a several scenarios where using DITA-OT is the appropriate choice for your documentation team, and describing them all is beyond the scope of this guide. However, here are three simple criteria where DITA-OT provides the best solution:

- Your documentation suite contains a lot of content that is reusable for different documents and audiences.
- Your documentation suite contains documentation for developers using the Eclipse IDE.
- Your documentation suite includes both Microsoft HTML Help and PDF-based documents.

DITA-OT is the only publication tool capable of producing both PDF and Eclipse-based documentation from the same source. Eclipse is the industry-standard IDE for many Java developers, and DITA-OT generates the the content and plugin file required for this environment. If your developer audience uses Eclipse, you can easily add IDE-specific online help to your documentation suite.

If your primary audience is end users, rather than software developers and system administrators, you likely need to provide Microsoft HTML Help for them, in addition to PDF-based documentation. DITA-OT is the only tool that produces both from the same source files.

Running DITA Open Toolkit

Learn how to run DITA-OT to generate published output documents.

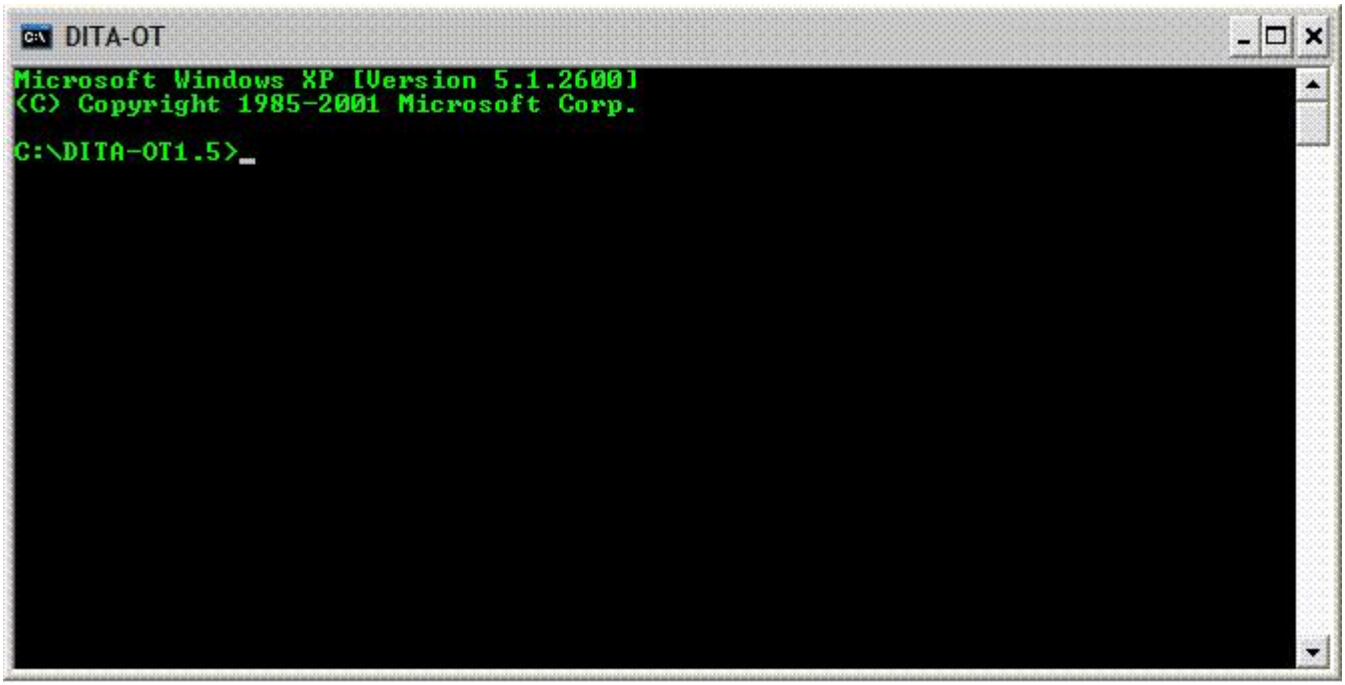
Generating Documents with Ant

How to generate documents from the command line with Ant.

1. Open a command prompt.
2. Change directories to where the DITA-OT is installed on your machine.
3. Set up the processing environment.
   Enter the following command:

   ```
   startcmd.bat
   ```

   Another command prompt appears with DITA-OT in the title bar, as shown in the following figure:

4. Run a convertion to a transformation output type.
   Enter the following command and press the Enter key:

   ```
   ant -Dargs.input=source -Dtranstype=transtype
   ```

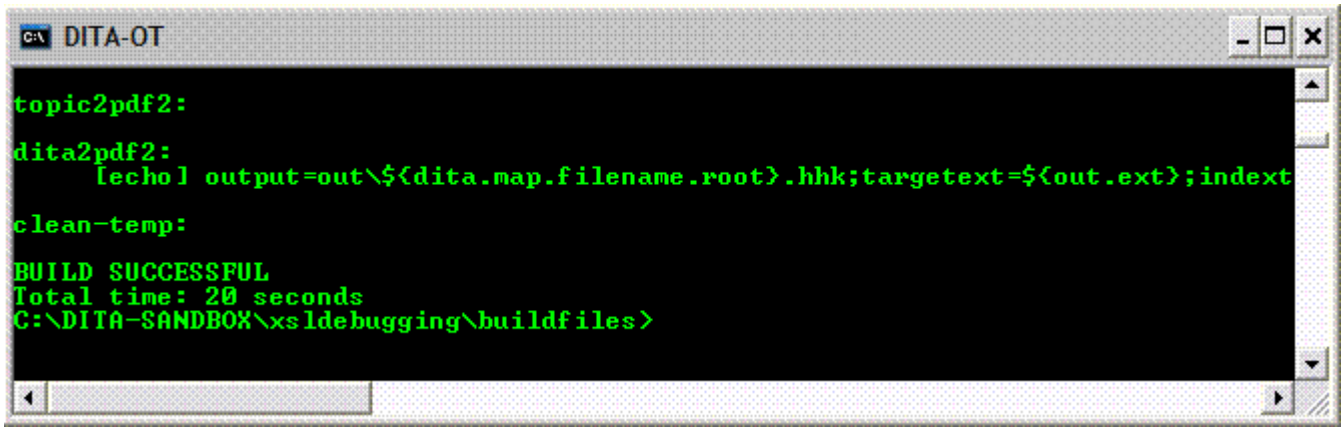
   The following table describes this command.

Syntax	Description
`ant`	Starts the Ant build tool installed as part of DITA-OT.
`-Dargs.input=source`	The `source` specifies the DITA topic or map to process. If the build file is not in the current directory, you must specify the path to the file.
`-Dtranstype=format`	The `format` specifies the transformation output type to generate.

To build XHTML output for the sample DITA map `samples\hierarchy.ditamap`, run the command:

```
ant -Dargs.input=samples\hierarchy.ditamap -Dtranstype=xhtml
```

DITA-OT displays a lot of output in the console window, including whether the build failed or succeeded at the end of the output.



When your build is unsuccessful, the error message may be difficult to find in the copious output. If you have not configured your console window most of the early output may have already scrolled off the screen. If you add an Ant property, `-l log-file` to the command line invocation, DITA-OT will save the output to a log file that you can study after the build finishes.

Writing more complex Ant build files for the DITA-OT

The sample Ant build scripts provided by the DITA-OT may not be adequate to meet the needs of your project. This topic describes how to customize the default scripts and write your own.

Customizing the Default Ant Script

The DITA Open Toolkit contains sample build files for both the DITA-OT and sample documentation. Writers new to the toolkit may use the `sample_all.xml` Ant build script to create all the sample documents that come with DITA-OT. The toolkit also contains build scripts for individual ouput types, such as `sample_pdf.xml`. You can modify just one or two Ant properties in these scripts for your own documentation.

Here is the Ant project definition from `samples\ant_sample\template_pdf.xml`.

```
<project name="@PROJECT.NAME@_pdf" default="@DELIVERABLE.NAME@2pdf"
basedir=".">

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

  <target name="@DELIVERABLE.NAME@2pdf">
    <ant antfile="${dita.dir}${file.separator}build.xml">
      <property name="args.input" location="@DITA.INPUT@"/>
      <property name="output.dir" location="@OUTPUT.DIR@"/>
      <property name="transtype" value="pdf"/>
    </ant>
  </target>

</project>
```

You simply change the values of the following properties to match the values used in your project:

- Project name: The root element in an Ant build file.
- Target name: Must be one of the supported DITA-OT transtypes.

Note that these scripts assume that your input files are located in same directory structure used by the DITA-OT samples.

Writing Your Own Ant Script

The default build script may not meet the needs of your project for a range of reasons:

- You want to add additional Ant properties not used in the sample template, such XSL and DTD properties to assist your debugging efforts.
- Your content files may not have the same directory structure as the samples.
- You want to place the output files in a different directory.

You need to customize or write your own build file for these use cases. For example, each target for this guide's build script uses a separate value for `dita.temp.dir` to assist debugging for a specific output types; setting `clean.temp` to "no" ensures that the temp directories remain available when processing ends.

Here is an example Ant script that can be used to produce this document.

```
<?xml version="1.0" encoding="utf-8"?>
<project name="userguide" default="dita2pdf" basedir=".">

  <property environment="env"/>
  <property name="DITA_DIR" value="${env.DITA_DIR}"/>
  <property name="args.logdir" value="logs"/>

  <property name="dita.extname" value=".dita"/>

  <property name="outdir" location ="output"/>
  <property name="clean.temp" value="no"/>

  <property name="args.indexshow" value="no"/>

  <target name="dita2pdf">
    <ant antfile="${DITA_DIR}/build.xml">
      <property name="transtype" value="pdf"/>
      <property name="args.input" value="doc/userguide-book.ditamap"/>
      <property name="dita.temp.dir" value="${outdir}/temp_pdf"/>
      <property name="output.dir" value="${outdir}/pdf"/>
      <property name="outer.control" value="quiet"/>
      <property name="clean.temp" value="no"/>
    </ant>
  </target>

</project>
```

This script is designed to run from the DITA-OT main directory. The generated PDF file will be placed in the `DITA-OT/output/pdf/` directory. The temporary processing directory will be left behind in `DITA-OT/output/temp_pdf/`.

To run this script, save it in the root toolkit directory with a name like `my_test_pdf.xml`. Run the build with the following command (assuming your command shell is already set up):

```
ant -f my_test_pdf.xml
```

Ant parameters contains a list of Ant properties used by DITA-OT. Use these properties to customize your document's build script for your needs.

Generating Documents with command-line tool

How to generate documents from the command line with the DITA-OT command-line tool.

The DITA Open Toolkit provides a command-line tool to run document conversions. However, the command-line tool is a wrapper for the Ant interface, so you still must install Ant. In addition, only a subset of the Ant properties are supported by the command-line tool

1. Open a command prompt.
2. Change directories to where you installed the DITA Open Toolkit.
3. Set up the processing environment.

   Enter the following command:

   ```
   startcmd.bat
   ```

4. Run a convertion to a transformation output type.

   Enter the following command:

   ```
   java -jar lib/dost.jar [arguments]
   ```

   Three arguments are required:

`/i:source`	defines the location of the .ditamap file for your document
`/outdir:output-dir`	defines the director where the output resides after DITA-OT finishes processing your project
`/transtype:format`	defines the type of document you want to generate for the project.

   For example, the following command instructs DITA-OT to build the `samples/sequence.ditamap` as a PDF in the `out` directory:

   ```
   java -jar lib/dost.jar /i:samples/sequence.ditamap /outdir:out /
   transtype:pdf
   ```

Debugging DITA-OT Transformations

Transforming your DITA-compliant XML into documents.

Understanding the Role of the FO PlugIn. Debugging FO-generated tranformation files.

Introducing Document Generation

Learning the mechanics of document generation with DITA OT.

Your documentation project uses an Ant build script, which calls a target in another Ant build script in the DITA-OT root directory, which imports another Ant build script, which itself imports several more Ant build scripts. Sound confusing? This topic explains this interaction and explains how to identify targets in these scripts related to errors in your document generation.

Each target in the build script for this Quick Start Guide contains the following code snippet.

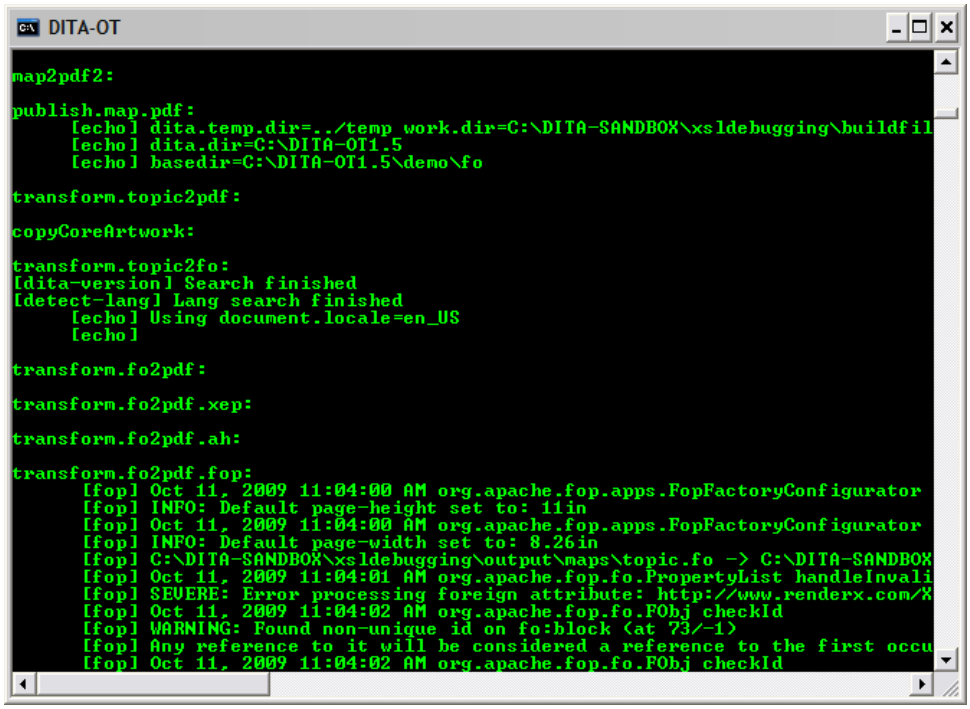
```
<ant antfile="${dita.dir}/build.xml" target="init">
```

The `toolkit_dir` directory is the root directory where you installed DITA-OT.

The build file you should understand is `build.xml`, located in folder bound to `dita.dir` property. The Ant targets defined and imported into this script are the same targets that you see on the console as your build script runs.

DITA-OT Build Script	Description
`build_init.xml`	Starts the document transformation, initializes the DITA-OT logger, verifies that the toolkit can locate the files and directories that you specified in your build file, and prints these values to the console and the log file, if you have specified one.
`build_preprocess.xml`	Validates your content files, generates lists of input files, including internal elements distributed across all content, such as index and conref entries. Moves copies of these files and elements into the the directory specified by `output.dir` property in your build script.
`build_general.xml`, `build_dita2wordrtf.xml`, `build_dita2xhtml.xml`, `build_dita2eclipsehelp.xml`, `build_dita2javahelp.xml`, `build_dita2htmlhelp.xml`, `build_dita2pdf`	Output specific Ant files which are not intended to be run directly.

Your console displays the name of each Ant target called inside the build scripts, including the output-specific script. For example, the following screen shot displays the names of Ant targets contained in the `build.xml` script when PDF transtformation type is used.



When you see an error in the output, you should read the Ant target that generated it for clues to solve the problem. To learn more about what caused the `INFO`, `SEVERE`, and `WARNING` errors in the image above, you should read the `transform.fo2pdf.fop` Ant target to learn what the Toolkit was doing when the error occurred and which xsl file generated the error.

Note: The DITA-OT build scripts sometimes continue to run even if they are unable to generate a temporary file for one of your content files. The build later displays an error message stating that a DITA-OT build script cannot find a generated file. This error is often misleading; the problem may be that your content file contains an error other than XML validation, which would stop the DITA-OT build from proceeding.

DITA-OT uses a separate set of Ant targets to process your PDF if you specify a value for the `args.fo.userconfig` property in your document's build script.

Best Practices

Tips and tricks for working directly with the DITA OT.

Create targets only for document types that you need.

DITA-OT's most attractive feature is its ability to produce so many different types of documents from the same source files. However, you may find that you need to tweak the targets in your Ant build file to get a document to meet your customization and style guide requirements. Although the sample documents for DITA-OT ship with every available target, there is no point in ironing out the details of a `dita2rtf` target in your build file if your documentation set doesn't require Word-based documents. If you're not providing JavaHelp, troff, or .rtf, then don't create targets for them.

Place all content inside or within the map directory if HTML Help is one of your output types.

The HTML Help Compiler cannot compile the files generated by DITA-OT for source files that reside outside the folder where your `.ditamap` file resides. If your documentation suite contains HTML Help, you should place all your source files in or below this directory.

For advanced debugging, use a different temp folder for each document type within the same build.

The Ant build script for the DITA-OT samples uses a unique folder for each build. However, many builds will include multiple targets, and some of these targets generate overlapping intermediate files. Specify a unique temp directory for each target within the same build to be sure that the intermediate files that you are reading were generated for the target you're debugging. See the build file for this document for an example.

Index