An OASIS White Paper

Best Practices Guide for Delivering DITA-Authored Help

OASIS DITA Technical Committee Online Help Subcommittee

October 28, 2008

DRAFT

OASIS (Organization for the Advancement of Structured Information Standards) is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. Members themselves set the OASIS technical agenda, using a lightweight, open process expressly designed to promote industry consensus and unite disparate efforts. The consortium produces open standards for Web services, security, e-business, and standardization efforts in the public sector and for application-specific markets. OASIS was founded in 1993. More information can be found on the OASIS website at http://www.oasis-open.org.

The purpose of the OASIS DITA Technical Committee (TC) is to define and maintain the Darwin Information Typing Architecture (DITA) and to promote the use of the architecture for creating standard information types and domain-specific markup vocabularies. The Translation Subcommittee defines best practices and guidelines for DITA authoring, translation and localization, and recommends solutions for industry requirements for consideration by the OASIS DITA TC. The group recommends widespread adoption of these concepts through liaisons with industry, other standards, and providers of commercial and open source tools.

DRAFT

Contents

Introduction....2
DITA and User Assistance....3
Definition of DITA Help....3
Developing DITA-based Help for Existing Help Environments....4
Arbortext Digital Media Publisher....4
Eclipse Help....9
CSHelp Plug-in....9
Eclipse_CSH Plug-in for Dynamic Context-Sensitive Help....11
Eclipse Help....16
Leximation AIR Help Plug-in....19
Microsoft HTMLHelp....22
Context-Sensitive Help using the Enhanced HTML (htmlhelp2) Plug-In....22
The DITA Open Toolkit HTMLHelp Transform....23
Developing Custom DITA-based Help Systems....26
DHTML Effects in HTML Generated from DITA....26
DITA-OT Plug-ins....28
HTMLSearch Plug-in....28
TOCJS and TOCJSBIS Plug-ins....36
Dynamic Rendering of DITA into XHTML....41
JavaScript-Based Context Sensitive Help....42
WinANT Options Supporting HTML-Based Output....45
WinANT Options Supporting Microsoft® HTML Help....47
Developing DITA-based Help for Existing Help Authoring Tools....50
Converting DITA Content to WebHelp using RoboHelp®....50

DRAFT

Introduction

This topic introduces the *DITA Help Best Practices Guide*.

Need a general introduction here ....

DRAFT

DITA and User Assistance

This topic describes the relationship between DITA and forms of user assistance, Help systems in particular.

DITA can be used in the process of creating user assistance, and especially of Help systems, but is not currently (and may never be) used as a delivery format for user assistance itself.

DITA is a storage and authoring format, not a delivery format; it is a presentation-neutral format. The separation of content and form is fundamental to DITA's design; content is written in DITA and must be transformed to a reading format before it can be delivered to the reader.

In principle, content written in DITA can be *transformed* to any reading format. In practice, it's not that simple. Before DITA can be transformed, a transformation process has to be devised. Many DITA authoring and publishing tools come with standard transformers for most common delivery formats, such as PDF, RTF and HTML. The *DITA Open Toolkit*, an open source collection of utilities and documentation to help writers work with DITA, includes basic transformers for PDF, RTF, HTML, DocBook, Eclipse Help, and Microsoft® HTML Help.

User assistance content is not defined by its format. A document in Microsoft HTML Help format isn't necessarily a Help system; user assistance is defined by the nature of the content. Conversely, user assistance content doesn't have to be delivered in a *traditional* Help format.

DITA promotes a single-source approach to documentation, so user assistance may commonly be one of a number of deliverables produced from a *repository* of information topics. The process of producing simple Help systems from DITA content using the standard DITA Open Toolkit transformers is straight-forward. It is a little more complicated to deliver such DITA-generated content for context-sensitive Help, but still readily achievable. Likewise, in principle, it is a trivial matter to incorporate DITA content into embedded user assistance and user interface elements using standard XML tools and techniques. There is not yet a standard approach to user assistance, so there is also no standard way of using DITA in this way. Different organisations tend to develop their own individual, custom approaches, using in-house technical expertise to do so.

Moving beyond simple Help systems, however, is currently difficult, but not impossible. The DITA Technical Committee is developing some enhancements to the DITA standard to allow these processes to be simplified. However, the apparent simplicity or complexity of using DITA for Help authoring will be in future determined by the capabilities of DITA editing and publishing tools. When it comes down to it, DITA is just a standard, and good tools are needed to work with good standards.

DRAFT

Definition of DITA Help

This topic details how the OASIS DITA Help Subcommittee defines the term *DITA Help*.

DITA Help, as defined by the OASIS DITA Help SC (DHSC), is:

A set of recommendations for the design and implementation of commonly recognized user assistance components using the DITA architecture. These components include, but are not limited to, navigation components, context-sensitive linking, embedded Help, browse sequences, associative links and window definitions. These recommendations can be used as a foundation for the development of authoring models and tools to support solutions in a variety of formats and for a variety of platforms.

Developing DITA-based Help for Existing Help Environments

This topic introduces solutions for moving your DITA source content into one of several existing help run-time environments.

If your company delivers information to a commonly used Help viewer or run-time Help environment, you have options for authoring content in DITA and for transforming that content into ready-to-run Help deliverables.

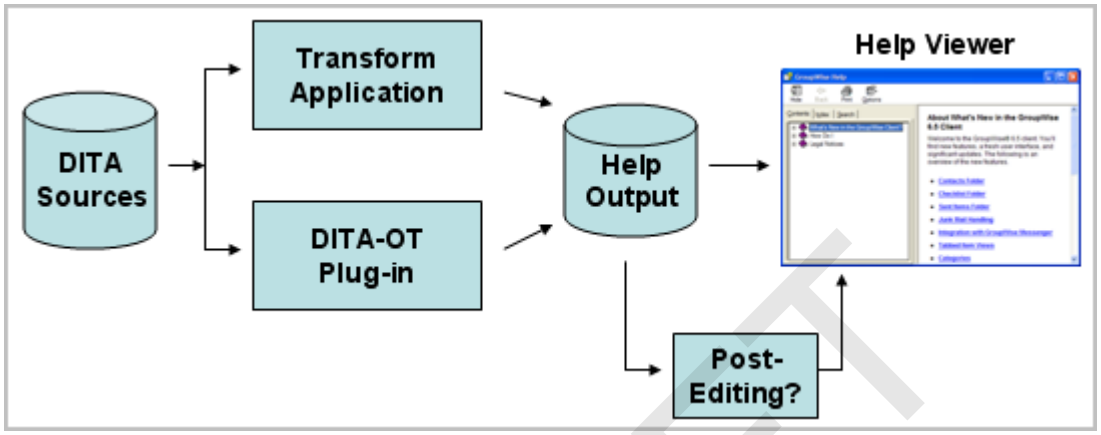


Figure 1: DITA Source to Existing Help Environments

The following topics in this section of the *Best Practices Guide* explain how to use one or more of these plug-ins or transforms to generate Help from DITA sources.

- *Arbortext Digital Media Publisher* on page 4
- *CSHelp Plug-in* on page 9
- *Eclipse_CSH Plug-in for Dynamic Context-Sensitive Help* on page 11
- *Eclipse Help* on page 16
- *Leximation AIR Help Plug-in* on page 19
- *The DITA Open Toolkit HTMLHelp Transform* on page 23

DRAFT

Arbortext Digital Media Publisher

This topic provides an overview of how the Arbortext Digital Media Publisher uses DITA to create Help systems and other types of online information systems.

Arbortext Digital Media Publisher (DMP) has been available as a standalone product for some time, but in the upcoming release DMP provides a DITA-based integration with Arbortext Editor that enables you to create a Help system or other online information system directly from a DITA map.

Overview

DMP enables you to take a set of documents and compose them to various types of output. The documents can be of several different types including DITA and other types of XML documents, HTML, PDF, Microsoft Word, Microsoft PowerPoint, Micosoft Excel, Javadoc, or just text documents. Once you have compiled your set of content, you can use DMP to compose that content to the following types of output:

- Help systems
- Standalone knowledge bases or information systems
- Web applications

DMP is a browser-based application and uses either an embedded browser (based on Internet Explorer) or the native browser installed on a system to serve up the content. DMP is a cross-platform application and runs on Windows, Linux, and some types of UNIX. DMP also supports multiple languages in the same image.

More information on DMP can be found on the *PTC web site*.

Setup and configuration

DMP can be purchased from PTC, either as a standalone product or bundled with Arbortext Publishing Engine. DMP comes with its own installation program. To create a DMP image directly from a DITA map also requires Arbortext Editor.

You can configure a DMP image in the following ways:

- DMP as a standalone product

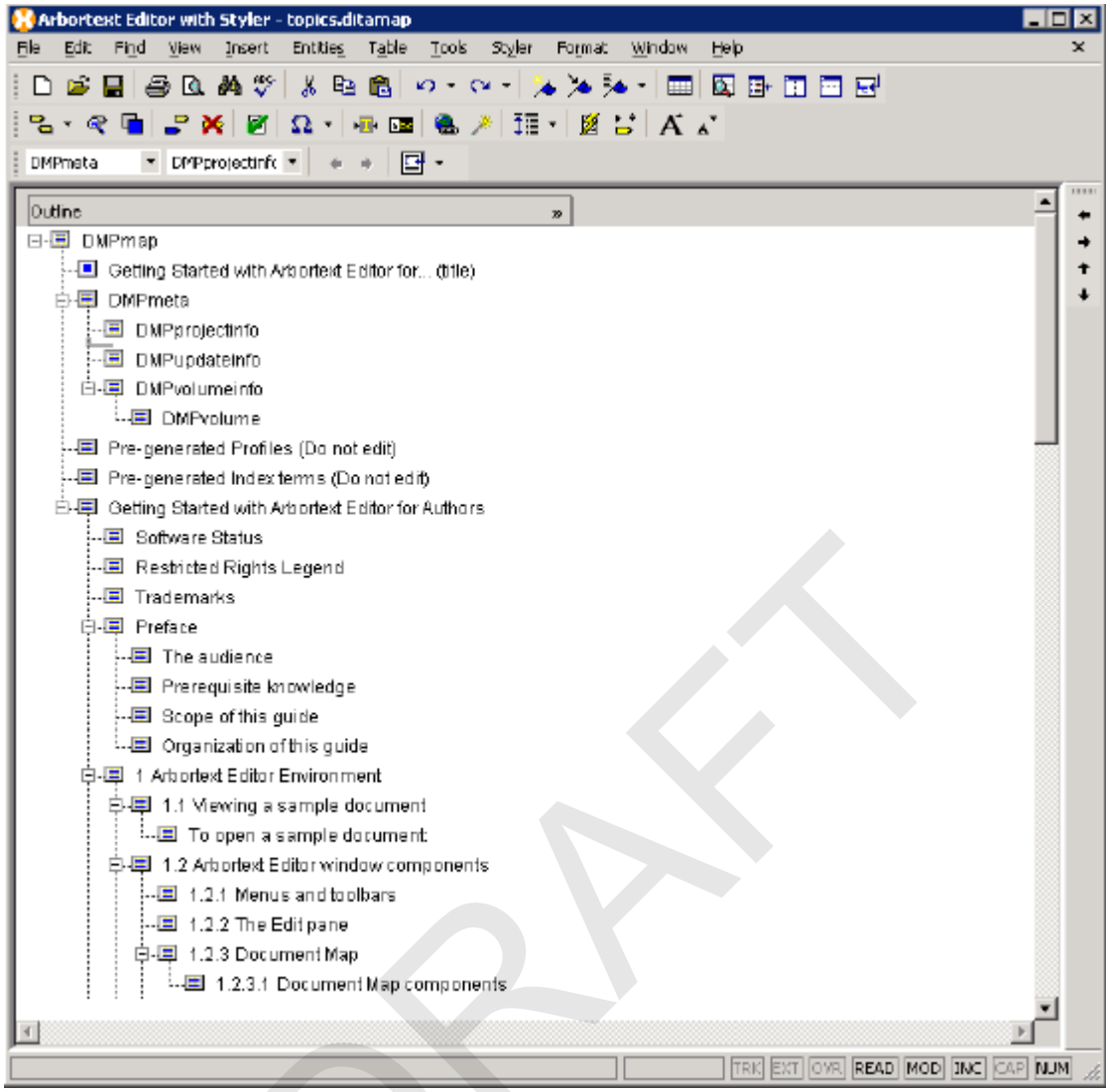
  DMP has some Java property files that you use to set up the application properties and a content configuration XML file that you use to identify both the content sets to go in the application and the structure of the table of contents. You can also apply metadata to the table of contents to filter the content displayed in the image based on any criteria you define. Once you have completed setting up your configuration files and content sets, you can run a DMP build to create the DMP image.

- DMP in conjunction with Arbortext Editor

  You can use Arbortext Editor to edit a specialized DITA map called a "DMP Map" to construct the table of contents and reference the associated content. In this case, you can create a DMP image directly from the map. A DMP Map supports all of the configuration options available in the standalone DMP configuration.

Following is an example of a DMP Map:

DRAFT



DRAFT

Authoring

Authors will need to learn about the specialized elements and attributes in a DMP Map to use the map. However, for authoring regular DITA topics and maps, nothing special is required.

Integration

DMP provides both a Java API and an XML RPC API. The same features are available in both of these APIs. The API provides several capabilities, including displaying an individual topic based on an ID and searching through HTML metadata for strings or IDs to find the appropriate topic to display. Application developers can use the DMP API to integrate a DMP image with their software for use as online Help.

All PTC documentation groups will be using DMP images for their products' online Help.

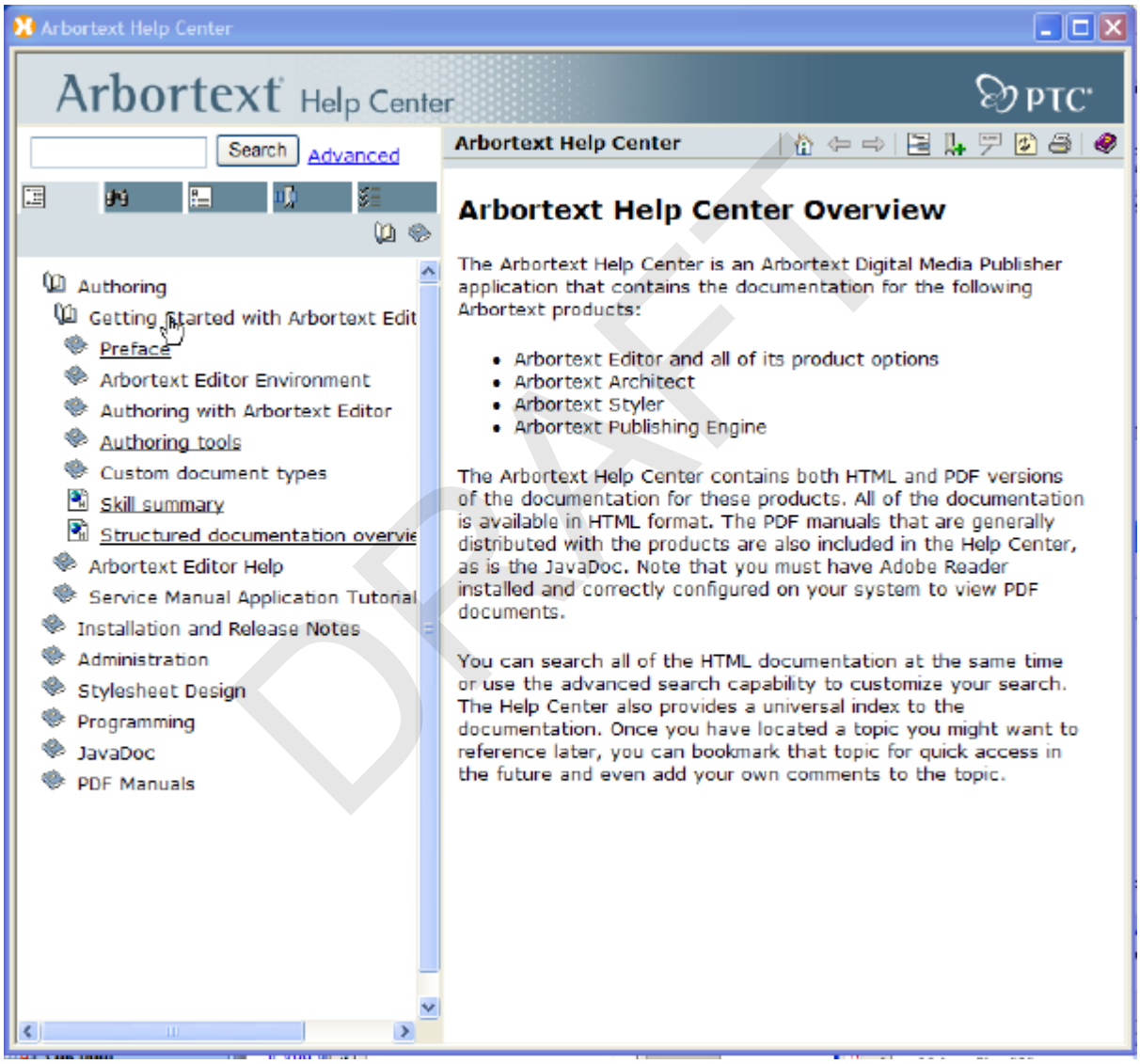
Output

A DMP image is highly customizable. All of the images/icons, colors, text, and features of a DMP image can be modified or removed during configuration. The default image contains the following features:

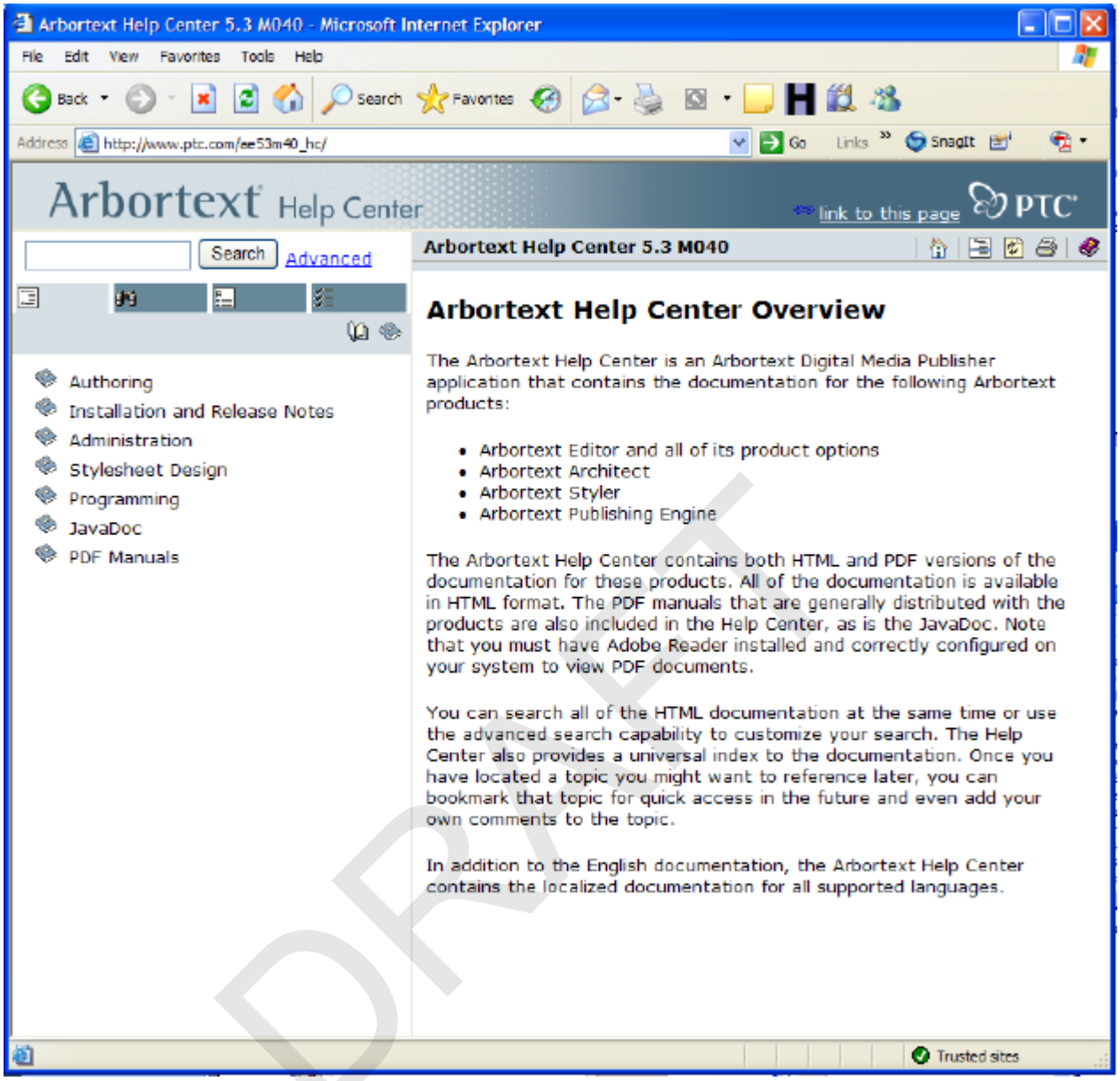
- Table of Contents tab — Provides a table of contents for the content in the image.
- Search tab — Contains search results, arranged either by relevance or position in the table of contents.

- Index Terms tab — Contains a universal index for the content in the image.
- Bookmark tab — Contains personal bookmarks and comments for individual topics.
- Configuration tab — Enables the Help Center image to be customized in various ways (for example, switching to a different language or filtering the content based on various metadata criteria).
- Search field — Enables you to enter search terms with full Boolean, wildcard, and other standard search support.
- Advanced search — Enables you to filter search results based on various criteria.
- Toolbar — Provides access to various DMP features.
- View frame — Displays the current topic.

Following is a DMP image delivered in the embedded browser for use as online Help or a standalone knowledge base:



Following is the same image delivered as a web application:



The main difference between the two images is that the bookmark and commenting feature in the standalone image is replaced by the web browser's favorites or bookmarks capability. For either case, displaying documents other than HTML leverages browser plug-ins. For example, PDF documents are displayed in the View frame in the Adobe Reader plug-in.

Summary

Arbortext Digital Media Publisher enables you to go directly from a DITA map to a composed image. It is a highly flexible product that can be customized as needed. The DMP API enables integration with other products for use as online Help or a knowledge base.

Chris Goolsby

OASIS DITA Help Subcommittee

Eclipse Help

CSHelp Plug-in

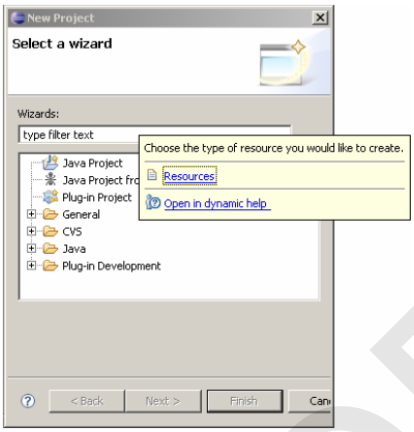
This topic provides an overview of the DITA Open Toolkit (DITA-OT) plug-in named cshelp (for Context-Sensitive Help).

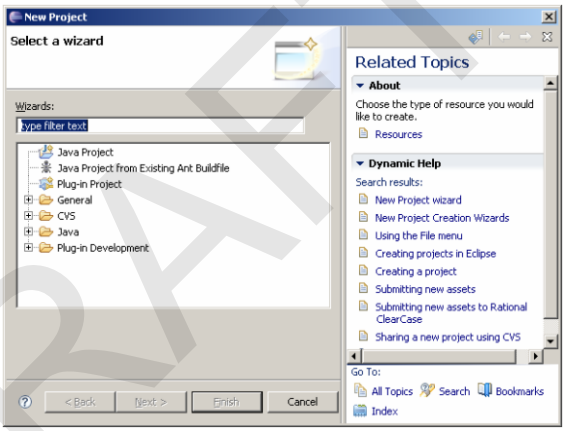
The cshelp plug-in generates contexts files in the format that Eclipse-based applications use for Context-Sensitive Help. The standard DITA-OT transforms can be used to produce XHTML output.This chapter focusses on producing Eclipse contexts files.

Overview

If you develop code plug-ins that extend the Eclipse user interface or develop documentation for development teams that do, you either already are or should be incorporating Context-Sensitive Help into the user interface.

The Eclipse user interface allows you to display Context-Sensitive Help as an infopop or in the Dynamic Help view. The latter option includes information and functionality in addition to the contents of the Context-Sensitive Help topic.





For more information on contexts files and developing Context-Sensitive Help for Eclipse, refer to the documentation for the current Eclipse release, available on *http://www.eclipse.org*.

The cshelp plug-in was developed by a team of writers representing the various software brands in IBM, led by the author. It is currently in use by products in several brands that develop applications for the Eclipse environment.

Setup and configuration

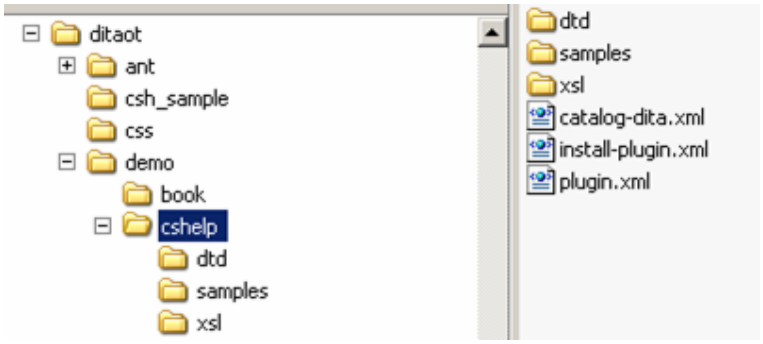
The cshelp plug-in is available from the DITA-OT site on sourceforge.net:

http://sourceforge.net/project/showfiles.php?group_id=132728

Package	Release	Date	Notes / Monitor
dita-ot	DITA Open Toolkit 1.4	August 12, 2007	
dita-ot under Apache License	DITA OT 1.2.2 ASL	May 22, 2006	
Plug-in: apiref	apiref-0.8	February 23, 2006	
Plug-in: cshelp	cshelp1.1	April 2, 2007	
Plug-in: dockbook2dita	docbook2dita 1.0	October 26, 2006	

Download and unzip the plug-in into your DITA-OT installation (typically, the `<DITA-OT>/demo` directory).

DRAFT



To complete the installation, open a command prompt and run the Ant integrator build file from your DITA-OT directory (`ant -f integrator.xml`).

Authoring

The structure of a cshelp DITA file mirrors that of an Eclipse contexts file. A contexts file is an XML file that contains one or more context elements inside a containing <contexts> element. Each context element contains a unique ID, the text of the Context-Sensitive Help topic, and optionally, links to Help topics in the Help system.

```
<?xml version="1.0" encoding="UTF-8"?>
<?NLS TYPE="org.eclipse.help.contexts"?>
<contexts>
    <context id="new_wizard_selection_wizard_page_context">
        <description>
         Choose the type of resource you would like to create.
        </description>
        <topic label="Resources" href="concepts/concepts-12.htm"/>
    </context>
    …
</contexts>
```

Similarly, a cshelp DITA file contains one or more cshelp elements inside a containing <cshelp> element (which is otherwise unused). Each nested cshelp element contains a unique ID, text, and related links.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE cshelp PUBLIC "-//OASIS//DTD DITA CSHelp//EN"
 "..\demo\cshelp\dtd\cshelp.dtd">
<cshelp id="csh_outer_container" xml:lang="en-us">
  <title>sample1_context</title>
  <shortdesc></shortdesc>
  <csbody></csbody>
  <cshelp id="new_wizard_selection_wizard_page_context">
    <title></title>
    <shortdesc>Choose the type of resource you would like to create.</shortdesc>

    <csbody>
    </csbody>
    <related-links>
      <link format="htm" href="concepts/concepts-12.htm" scope="peer">
        <linktext>Resources</linktext>
      </link>
    </related-links>
  </cshelp>
  ...
</cshelp>
```

It is important to note that the only highlighting markup that can be used inside Eclipse Context-Sensitive Help is the bold (`<b></b>`) tag, and the only formatting options are the carriage return and the spacebar. All of these are permissible only inside the description element.When sourcing Context-Sensitive Help in DITA, all of the highlighting and formatting markup options that are available in the shortdesc and body elements of the topic type may be used. The transform that

produces a contexts file from the DITA source produces output that conforms to the restrictions of the contexts file. In most cases, the output approximates what would be seen in HTML output, but some DITA elements are simply ignored:

- Table and simpletable
- Image (only the text in the <alt> attribute or tag will display)
- Object
- Figure (only the text in the <desc> tag will display)
- Footnote
- Indexterm, indextermref
- Xref

Use as many DITA files per Eclipse plug-in that you want, and create a simple DITA map file inside the plug-in that points to each of them. Include any copyright information in the DITA map; this information will appear in comment tags in each generated contexts file. Note, however, that relationship tables (reltables) cannot be used to create related links when producing Eclipse contexts files.

Integration

Refer to the Eclipse documentation for instructions on incorporating context IDs in code plug-ins. This information is typically in Platform Plug-in Developer Guide > Programmer's Guide > User assistance support > Help > Context-Sensitive Help > Declaring a context ID.

http://www.eclipse.org/documentation/

Output

When processing the DITA map file, there are two parameters that need to be set:

- Use the switch that identifies a separate XSL file, and point to the `dit2context.xsl` file in the `cshelp/xsl` directory. For example, if you are using Ant:

```
<property name="args.xsl"
value="${dita.dir}${file.separator}demo${file.separator}
cshelp${file.separator}xsl${file.separator}dit2context.xsl"/>
```

- Use the switch that indicates the output extension of the generated file will be XML. For example, if you are using Ant:

```
<property name="args.outext" value="xml"/>
```

The contents of the generated XML file should resemble the example contexts file in the Authoring section.

Summary

Context-Sensitive Help is an integral part of the user assistance for software applications. If you already use DITA to source your Eclipse documentation plug-ins, sourcing the Context-Sensitive Help in DITA allows you to maintain a consistent authoring and build environment and consistently formatted output.

Jeff Antley

OASIS DITA Help Subcommittee

Eclipse_CSH Plug-in for Dynamic Context-Sensitive Help

This topic describes the eclipse_csh plug-in for the DITA Open Toolkit (DITA-OT), which generates complete, deployable Eclipse plug-ins that support dynamic context-sensitive help for Eclipse-based applications.

Eclipse plug-ins generated by the DITA-OT, with the eclipse_csh plug-in, provide dynamic context-sensitive help for Eclipse-based applications that define abstract help contexts and use the ContextProviderDelegate (supplied by org.eclipse.datatools.help).

DRAFT

Abstract help contexts allow UI developers to isolate the Eclipse help system's concrete help context ID strings and help search expressions from the UI implementation (Java code). That gives user assistance (UA) content developers more control over help contexts and context-specific content, so they can provide more useful, and more precisely targeted, dynamic context-sensitive help.

Overview

The Eclipse help system uses two key pieces of data to find context-specific UA content for dynamic context-sensitive help:

- A help context ID string, which associates a UI context with a particular chunk of context-specific content
- A context-specific help search expression, which the help system uses tofind additional, related content in its online documentation "library"

For many developers, one of the main problems with Eclipse dynamic context-sensitive help is the necessity to embed help context ID strings and context-specific help search expressions in the UI code. That can reduce code portability, and it forces recompiling whenever a help context ID string or a help search expression needs to be changed. It makes UI implementation more cumbersome, particularly in an agile development (or RAD) environment. Unless all UI contexts are explicitly specified, and frozen in advance, developers are tempted to delay implementing context-sensitive help to avoid otherwise unnecessary recompiling and rebuilding.

By using the org.eclipse.datatools.help.ContextProviderDelegate (the DTP help-helper, provided by the Eclipse Data Tools Platform project), and defining abstract help contexts in their code, UI developers are free to implement dynamic context-sensitive help at any point in the UI development cycle.

By using the DITA-OT, with the eclipse_csh plug-in, UA content developers (writers) and information architects can define the associations of help context IDs with UA content, using special context-sensitive help markup in a standard DITA map that produces an Eclipse online documentation plug-in. Content developers are free to revise the UI-UA context associations and context-specific help search expressions at any time, without impacting UI code.

Setup and configuration

The eclipse_csh plug-in is available from the DITA-OT project on sourceforge.net: [URL pending final release to open source]

Download the eclipse_csh zip file (eclipse_csh_1.0.0.zip), and extract all files and folders into the plugins/ directory, in a previously installed DITA-OT.

Run the DITA-OT integrator.xml Ant buildfile to complete the plug-in installation.

Authoring

Any DITA map document that defines an Eclipse online documentation (doc) plug-in can be modified to also define a corresponding context-sensitive help plug-in, by inserting the appropriate context-related markup.

Note: The DITA map markup described in this section does not require any DITA specializations; it relies entirely on standard DITA map elements and attributes.

Identifying the UI plug-ins associated with help contexts

DITA maps for context-sensitive help plug-ins must contain a `<topicmeta>` element as the first child of the `<map>` element, and for each UI plug-in whose help context IDs are identified in the map, that `<topicmeta>` element must contain one `<othermeta>` element that identifies the UI plug-in.

The <othermeta> element's name and content attribute values will be used to identify UI plug-ins in the org.eclipse.help.contexts extension, which is declared in the plug-in manifest (plugin.xml file) of the context-sensitive help plug-in. For example:

```
<map id="org.eclipse.datatools.ui.doc">
   <topicmeta>
      ...
```

DRAFT

```
      <othermeta name="ui-plugin"
         content="org.eclipse.datatools.connectivity.ui"/>
      <othermeta name="ui-plugin"
         content="org.eclipse.datatools.connectivity.ui.dse"/>
   </topicmeta>
   ...
</map>
```

The name attribute value "ui-plugin" is a fixed, literal string. The content attribute value is the Eclipse plug-in ID of a UI plug-in.

Defining related topics associated with help contexts

DITA maps for context-sensitive help plug-ins contain `<resourceid>` elements, each of which identifies a concrete help context ID associated with a DITA topic.

The `<resourceid>` element is a child of a `<topicmeta>` element. For example:

```
<topicmeta>
    <resourceid id="help_context_ID_string"/>
</topicmeta>
```

For each DITA topic associated with a help context ID, the `<topicref>` element that points to that topic contains a `<topicmeta>` element (with a `<resourceid>` child element). This defines the association of a help context ID with that topic. For example:

```
<topicref navtitle="label attribute in contexts.xml topic element"
    href="path/to/topic.xml">
    <topicmeta>
        <resourceid id="help_context_ID_string"/>
    </topicmeta>
</topicref>
```

Any DITA topic can be mapped to multiple help context IDs by inserting as many `<resourceid>` child elements as necessary in the `<topicmeta>` element. For example:

```
<topicref navtitle="label attribute in contexts.xml topic element"
    href="path/to/topic.xml">
    <topicmeta>
        <resourceid id="help_context_ID_1"/>
        <resourceid id="help_context_ID_2"/>
        <resourceid id="help_context_ID_3"/>
    </topicmeta>
</topicref>
```

Other considerations for DITA maps that define context-sensitive help plug-ins:

- Markup for context-to-topic mapping is needed only for the DITA topics that will be context-sensitive help targets (i.e., Related Topics shown in the Help view).
- When a single DITA topic appears more than once in a map, only the first instance of a `<topicref>` that points to that topic needs the context-sensitive help markup.
- Nested maps that contribute DITA topics related to help contexts must include the same context-sensitive help markup.

Context-specific help content

Each `<topicmeta>` element that wraps a `<resourceid>` element may optionally contain one `<searchtitle>` element or one `<shortdesc>` element, or both (one of each), to provide context-specific help content.

- The `<searchtitle>` element is used to supply the value of the title attribute on the `<context>` element in the Eclipse context XML file.

DRAFT

- The `<shortdesc>` element is used to supply the content of the`<description>` element in the Eclipse context XML file.

For example:

```
    <topicref navtitle="label attribute in contexts.xml topic element"
        href="path/to/topic.xml">
        <topicmeta>
            <searchtitle>Optional text to override the help About
title.</searchtitle>
            <shortdesc>Text for context description.</shortdesc>
            <resourceid id="help_context_ID_string"/>
        </topicmeta>
    </topicref>
```

Note: The `<searchtitle>`, `<shortdesc>`, and `<resourceid>` elements must appear in the`<topicmeta>` element in the order shown above.

A DITA map that contains the example shown above produces the following content in an Eclipse context XML file:

```
    <contexts>
        ...
        <context id="help_context_ID_string"
            title="Optional text to override the help About title.">
            <description>Text for context description.</description>
            <topic label="label attribute in contexts.xml topic element"
                href="PLUGINS_ROOT/doc_plugin_ID/path/to/topic.xml"/>
        </context>
        ...
    </contexts>
```

where *doc_plugin_ID* is the value of the id attribute on the `<map>` element.

Integration

To provide dynamic context-sensitive help, an Eclipse-based application must define the associations between its UI controls and help contexts dynamically, by implementing methods of org.eclipse.help.IContextProvider. One of those methods (getContext) must return a concrete help context ID string, which matches an IContext object contributed by an extension of org.eclipse.help.contexts, and defined in an Eclipse context XML file. Another method (getSearchExpression) must return a context-specific help search expression, if the application requires more targeted search results than the default help search expression provides.

The DTP help-helper plug-in (org.eclipse.datatools.help) provides a "help key" extension point (org.eclipse.datatools.help.helpKeyProperties), and supplies a context provider delegate implementation (org.eclipse.datatools.help.ContextProviderDelegate).

- The helpKeyProperties extension point allows any plug-in to contribute ResourceBundle properties files that define the mapping of abstract help keys to concrete help context IDs and help search expressions.
- The ContextProviderDelegate, along with abstract help keys, enables help context abstraction for any UI control that implements methods of IContextProvider.

The eclipse_csh plug-in for DITA-OT provides the processing to generate context-sensitive help plug-ins, which serve as companions to Eclipse online documentation plug-ins generated by DITA-OT (using its standard dita2eclipsehelp transtype). Indeed, the dita2eclipse_csh transtype target depends on the dita2eclipsehelp target to first generate an Eclipse online documentation plug-in.

Eclipse plug-ins produced by eclipse_csh handle the mapping of help contexts to context-specific help content, and provide support for the DTP help-helper infrastructure by contributing:

- Eclipse context XML files, which supply the context-specific help content for each help context, and point to other help topic contributions directly related to the help context

DRAFT

- Java properties files, which define key-value pairs that map abstract help contexts (helpKey constants) to concrete help context IDs and context-specific help search expressions

Eclipse plug-ins produced by eclipse_csh do not contribute (or contain) any topic-based UA content. Their context XML files refer to topics contributed by their companion documentation plug-ins.

Team collaboration

Successful delivery of Eclipse dynamic context-sensitive help requires close coordination of UI components and UA components. It imposes responsibilities on both Development teams and Documentation teams, and it requires ongoing collaboration.

Development teams are primarily responsible to:

- Implement the Eclipse classes and methods necessary to enable dynamic context-sensitive help for all appropriate UI controls.
- Implement interface classes that declare helpKey constants for each UI plug-in.
- Provide lists of the helpKey constants to Documentation teams in a timely manner.
- Test the context-sensitive help UI implementation, with context-sensitive help UA plug-ins and online documentation plug-ins provided by Documentation teams.

Development teams should provide lists of helpKey constants in the form of Java source files for the helpKey constants interface classes. Java source files provided as helpKey lists must include appropriate help context comments to provide information about each associated UI control.

Documentation teams are primarily responsible to:

- Develop context-specific help content and context-specific help search expressions for appropriate UI help contexts.
- Define the concrete help context ID strings that associate abstract help contexts with context-specific help content.
- Create the Java properties files that define the mapping of abstract help contexts to concrete help context IDs.
- Create the context-sensitive help UA plug-ins that contribute Eclipse context XML files and the Java properties files.
- Test the context-sensitive help UA plug-ins and online documentation plug-ins, with UI components provided by Development teams.

Documentation teams should rely on the Java source files (for the helpKey constants interface classes) as the original and definitive sources of all helpKey constant strings.

👉 **Note:** Creating the Java properties files can be somewhat automated (e.g., by processing the Java source files with a simple perl script).

Workflow

The following summarizes the Documentation team workflow to create Eclipse context-sensitive help plug-ins:

1. Get the helpKey list (provided as a Java source file) from the UI Development team for each UI plug-in.
2. Analyze the helpKey list and associated UI controls to define the help contexts.

   The Documentation team must determine whether:

   - The helpKey constants alone are sufficient to identify actual help contexts, and thus, a helpKey constant could be mapped directly to a concrete help context ID, with the same string value as the helpKey constant.
   - Distinct help context ID strings must be defined to combine groups of helpKey constants into common help contexts.

   👉 **Tip:** It may be preferable to combine help contexts in the helpKey properties file, instead of defining the mapping for multiple help contexts to a single topic in a DITA map. This is a judgment call for the Documentation team responsible for maintaining DITA maps.

3. Analyze the help contexts and existing (or planned) help topics to define context-specific help search expressions.
4. Create helpKey properties files, based on the content of each Java source file.

DRAFT

- Define the mapping of helpKey constants to concrete help context IDs and context-specific help search expressions, based on results of the help context analysis and help topic (content) analysis.
- Save the helpKey properties files in source control, as appropriate.

Tip: The helpKey properties files are flat ASCII text files, so authors (or IAs) responsible for defining the help context IDs and context-specific help search expressions should use a suitable ASCII text editor to create and edit those files.

5. Modify existing DITA maps (if used to produce Eclipse online documentation plug-ins) to add the markup for context-sensitive help.
6. Build Eclipse plug-ins, using the DITA-OT with the eclipse_csh plug-in, and test the Eclipse plug-ins with UI components provided by Development teams.

Summary

Help context abstraction is a technique to simplify the handling of help context IDs and help search expressions in the UI code, by abstracting them to "help keys."

Help context abstraction provides the following benefits:

- Development teams are free to associate new help contexts with UI controls, without necessitating that corresponding help context IDs or help search expressions exist.
- Documentation teams are free to define UA help contexts, modify help context IDs, and control the mapping from abstract help contexts to concrete help context IDs, without necessitating any change in the UI code.
- Documentation teams are free to define and modify context-specific help search expressions, and the mapping from abstract help contexts to help search expressions, without necessitating any change in the UI code.

This separation of responsibilities enables the project team to provide higher quality, and more precisely targeted, dynamic context-sensitive help.

For more information about the DTP help-helper, see:
http://www.eclipse.org/datatools/doc/20080310_DTP_Help-Helper.pdf

Eclipse Help

This topic provides an overview of how to use the DITA Open Toolkit to develop content for the Eclipse Help system.

According to the Eclipse Web site, the Eclipse integrated development environment (IDE) is an "open source platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle." Eclipse can be installed on Windows, Mac OS X, or Linux (32–bit or 64–bit).

The Eclipse Help system is itself extensible, allowing additional documentation to be delivered using Eclipse's plug-in framework. It can be used in three different modes: workbench, standalone, and infocenter.

- Workbench mode is for documentation for tools integrated into the Eclipse IDE.
- Standalone mode serves a similar purpose, but is used for software that is not Eclipse-based.
- Infocenter mode allows the Eclipse Help system to deliver topics through the World Wide Web. Local instances of the Eclipse Help system can also retrieve and integrate content from a remote server running Help in infocenter mode.

The Eclipse Help system contains search and indexing features, allows the triggering of workbench actions or Eclipse code from within Help topics, supports dynamic content and link filtering (when using XHTML), and supports globalization. You can even use Eclipse to manage the development, building, and testing of your DITA-sourced documentation plug-ins.

Overview

The Eclipse Help displays HTML- or XHTML-formatted topics, and organizes the topics according to the XML-formatted Table of Contents (TOC) files that are provided in each plug-in that contains documentation.

DRAFT

For more information on developing Help for Eclipse, refer to the documentation for the current Eclipse release, available on *http://www.eclipse.org*.

Setup and configuration

DITA Open Toolkit

The DITA Open Toolkit contains the transformations necessary to produce all of the files required for Eclipse Help plug-ins. No special setup or configuration is necessary. Refer to installation and configuration instructions within the DITA Open Toolkit to set it up.

Eclipse

Download Eclipse from *http://www.eclipse.org*. Installation involves unpackaging an archive file to any location on your machine. A Java Runtime Environment (JRE) is required. Eclipse manages your development resources in workspaces, that also can be any location on your machine. You are prompted to select or create a workspace location to use each time you start Eclipse.

Eclipse documentation plug-ins

Documentation for the Eclipse Help system can be included in any plug-in, as long as the plug-in extends Eclipse's `org.eclipse.help.toc` extension point. It is up to the individual organization whether to include documentation inside code plug-ins or in their own plug-ins. There are several advantages to the latter, such as unambiguous plug-in ownership, or if the contents of the documentation plug-in will be globalized.

Plug-ins are named according to Sun's Java package naming guideline (for example, `org.eclipse.help`). Plug-ins that contain only documentation typically have `.doc` at (or near) the end of the plug-in name.

Within a plug-in, two XML files are required in the root (a manifest and a table-of-contents (TOC) file). Any number of content files may be included, in any location within the plug-in. The TOC file can be generated from a DITAMAP file, and HTML files will be generated from DITA files.

Plug-ins can be delivered to the Eclipse run-time environment as folders or as Java archive (JAR) files. Documentation within folders may exist in archives (for example, `doc.zip`). Archives cannot be nested (that is, a `doc.zip` cannot be included in a plug-in delivered as a JAR file).

Authoring

Author DITA topics as you would normally. You can include any number of topic files, folders, DITA maps, or other file-based resources that can be delivered within browser environment (for example, images or multimedia).

When creating links to other topics (XREFs or LINKs), note links to topics in other plug-ins should be coded as follows:

```
PLUGINS_ROOT/PLUGIN_ID/path/to/target.html
```

Where `PLUGINS_ROOT/` indicates that the target file is in another plug-in and `PLUGIN_ID` is the ID of the plug-in as declared in the manifest file. Refer to the Eclipse documentation regarding Help server and file locations in Help content: *http://www.eclipse.org/documentation/*.

Integration

To integrate your content into Eclipse, manually create a plugin.xml file that points to one or more TOC files in the plug-in, and a `manifest.mf` file (in a META-INF folder).

Plugin.xml file

This example `plugin.xml` file is similar to the one provided with the Garage sample in the DITA-OT. This example shows the minimum amount of information required to declare a TOC file to the `org.eclipse.help.toc` extension point.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
 <extension point="org.eclipse.help.toc">
  <toc file="hierarchy.xml"/>
```

DRAFT

```
 </extension>
</plugin>
```

Manifest.mf

In current versions of Eclipse, some of the manifest information, such as the plug-in ID, is separated into the `manifest.mf` file. `Manifest.mf` is stored in the plug-in in a folder named `META-INF` For the Garage sample, here is a possible manifest:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Garage Plug-in
Bundle-SymbolicName: org.dita.garage.doc
Bundle-Version: 1.0.0
```

Note that what was the plug-in ID in the `plugin.xml` file is referred to as the Bundle-SymbolicName in the `manifest.mf` file.

TOC file

TOC files are generated from DITA map files. You may include any number of TOC files in a plug-in, as long as they are declared in the `plugin.xml` file.

From the Garage sample, here is `hierarchy.xml`:

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

<?NLS TYPE="org.eclipse.help.toc"?>

<toc label="Garage (hierarchy)" topic="concepts/garagetasks.html">
<topic label="Garage Tasks" href="concepts/garagetasks.html">
<topic label="Organizing the workbench and tools" href="tasks/organizing.html"/>
<topic label="Taking out the garbage" href="tasks/takinggarbage.html"/>
<topic label="Spray painting" href="tasks/spraypainting.html"/>
<topic label="Washing the car" href="tasks/washingthecar.html"/>
</topic>
<topic label="Garage Concepts" href="concepts/garageconcepts.html">
<topic label="Lawnmower" href="concepts/lawnmower.html"/>
<topic label="Paint" href="concepts/paint.html"/>
<topic label="Shelving" href="concepts/shelving.html"/>
<topic label="Tool box" href="concepts/toolbox.html"/>
<topic label="Tools" href="concepts/tools.html"/>
<topic label="Water hose" href="concepts/waterhose.html"/>
<topic label="Wheelbarrow" href="concepts/wheelbarrow.html"/>
<topic label="Workbench" href="concepts/workbench.html"/>
<topic label="Windshield washer fluid" href="concepts/wwfluid.html"/>
</topic>
</toc>
```

Building DITA content in Eclipse

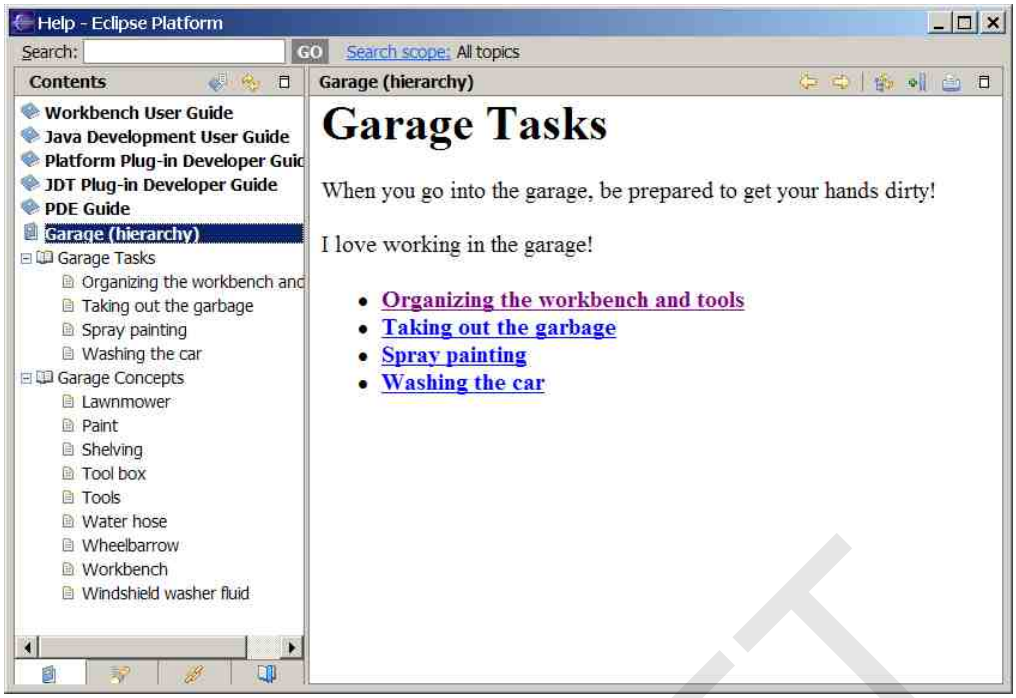
It is possible to use Eclipse as your IDE for developing, building, and testing your Eclipse plug-in projects. Refer to the DITA Open Toolkit User Guide topic entitled "Running DITA builds in Eclipse."

Output

Detailed instructions exist in the DITA-OT Help for producing Eclipse Help. Refer to DITA Open Toolkit User Guide topic entitled "Processing to Eclipse Help targets."

The image shows the content after it has been transformed and the plug-in has been added to a basic Eclipse run-time environment.

DRAFT



If you intend to take advantage of active Help or the dynamic content capabilities within the Eclipse Help system, such as link filtering, be prepared to create your own post-processing transforms to incorporate the required markup.

Summary

Whether creating documentation for an Eclipse-based application or one that simply uses the Eclipse Help system to deliver user assistance, consider DITA to source content. Since Eclipse Help is one of the key output targets for the DITA Open Toolkit, setup, configuration, and processing are all straightforward and well-documented. For additional instructions on such advanced Eclipse Help capabilities as active Help, dynamic content, or remote Help, refer to the current Eclipse documentation, located at *http://www.eclipse.org/documentation/*.

Jeff Antley

OASIS DITA Help Subcommittee

DRAFT

Leximation AIR Help Plug-in

This topic introduces the concept of AIR Help and provides information about generating AIR Help from DITA.

The Leximation AIR Help DITA-OT plug-in is not yet publicly available; information about the availability of this plug-in will be posted to www.leximation.com/airhelp.

Overview

AIR Help is an evolving new option for delivering online user assistance.This Help delivery format does not specifically define appearance or functionality, but refers to any user assistance application that was developed with Adobe's AIR (Adobe Integrated Runtime) technology. Applications developed using AIR can be installed on the Mac, Windows, and Linux operating systems, and are highly customizable, making it an ideal technology from which to develop an online Help deliverable.

Currently there are two commercially available products for generating AIR Help, Adobe's "RoboHelp Packager for AIR" and MadCap Software's "Flare." RoboHelp Packager for AIR (beta) allows you to export an AIR Help file from a RoboHelp WebHelp project. At this time, there is no connection between RoboHelp and DITA other than the option

of authoring DITA files in FrameMaker and importing the FrameMaker files into RoboHelp. Flare offers AIR Help as one of the standard output formats from their latest version.

The other option for creating AIR Help is custom development. You can take any browser-based set of HTML files and "wrap" them up in an AIR application. The most basic AIR application is one that contains an embedded web browser (based on WebKit) which essentially lets you present your HTML-based Help in your custom browser.

The Leximation AIR Help DITA-OT plug-in provides a direct path from DITA to AIR Help. This implementation of AIR Help provides the following features:

- Tabbed navigation panels for Contents, Index, and Search
- Full text search that allows for both local and remote search indexes
- Forward, Back, Home buttons as well as Next/Previous Browse buttons that follow the structure of topics in the Contents tab
- The ability to make context sensitive calls from an external application
- Position and size of window is preserved between sessions
- Sync with TOC functionality

Additionally, you may customize the AIR Help interface to add features or to modify the appearance as needed.

Setup and configuration

Currently, in order to install and use this plug-in you must recompile the `dost.jar` file. For this reason, the plug-in is not publicly available. Updates are being made to the DITA-OT 1.5 that will support the modifications to the indexing pipeline required by this plug-in. Setup and configuration information will be made available once the OT has been updated.

Assuming the plug-in has been installed, the following information will apply.

For the most basic output, no extra effort is needed. Run your DITA map through the plug-in, and it will:

- Generate the "local" full text search index
- Compile the SWF
- Build batch files for testing and final AIR file creation

The following data is used from the map file when building the AIR file:

- /map/topicmeta/copyright
- /map/topicmeta/prodinfo/prodname
- /map/topicmeta/prodinfo/vrmlist/vrm (the last one)
- /map/topicmeta/prodinfo/prognum
- /map/topicmeta/othermeta/@name='remote-search-index-url' (@content specifies the location for the remote search index which is loaded on startup)

The Index is built from available indexterm elements.

After a successful build, just run the `airhelp-test.bat` batch file to view and test the new AIR Help file. When you're ready to build the final AIR file, just run the `airhelp-packager.bat` batch file.

Authoring

No specific authoring requirements are needed.

Integration

Because an AIR Help file is actually a desktop application, you launch the Help system by making a system call to run the executable. If no command-line parameters are provided, the AIR Help file will open with the default topic displayed.

In order to launch the Help system on a specific topic, use the following command-line syntax:

```
<path/app_executable> <target_topic>
```

DRAFT

For example, the following call will display the “Image” topic in the DITA Reference AIR Help file:

```
"c:\program files\ditaref\ditaref.exe" image.html
```

To launch the Help system with the search panel selected and pre-populated with a query, use the following syntax:

```
<path/app_executable> "search:using conrefs"
```

To launch the Help system with an external URL (website) displayed, use the following syntax:

```
<path/app_executable> http://google.com
```

Additional command-line parameters can be added as needed.

Because only one instance of eachAIR Help application can exist at one time, if you make a system call to the application when one is already running, the result of the new call replaces the running instance.

Output

This plug-in uses the HTML generated by the dita2html target, and only controls the interface features of the“container” application.

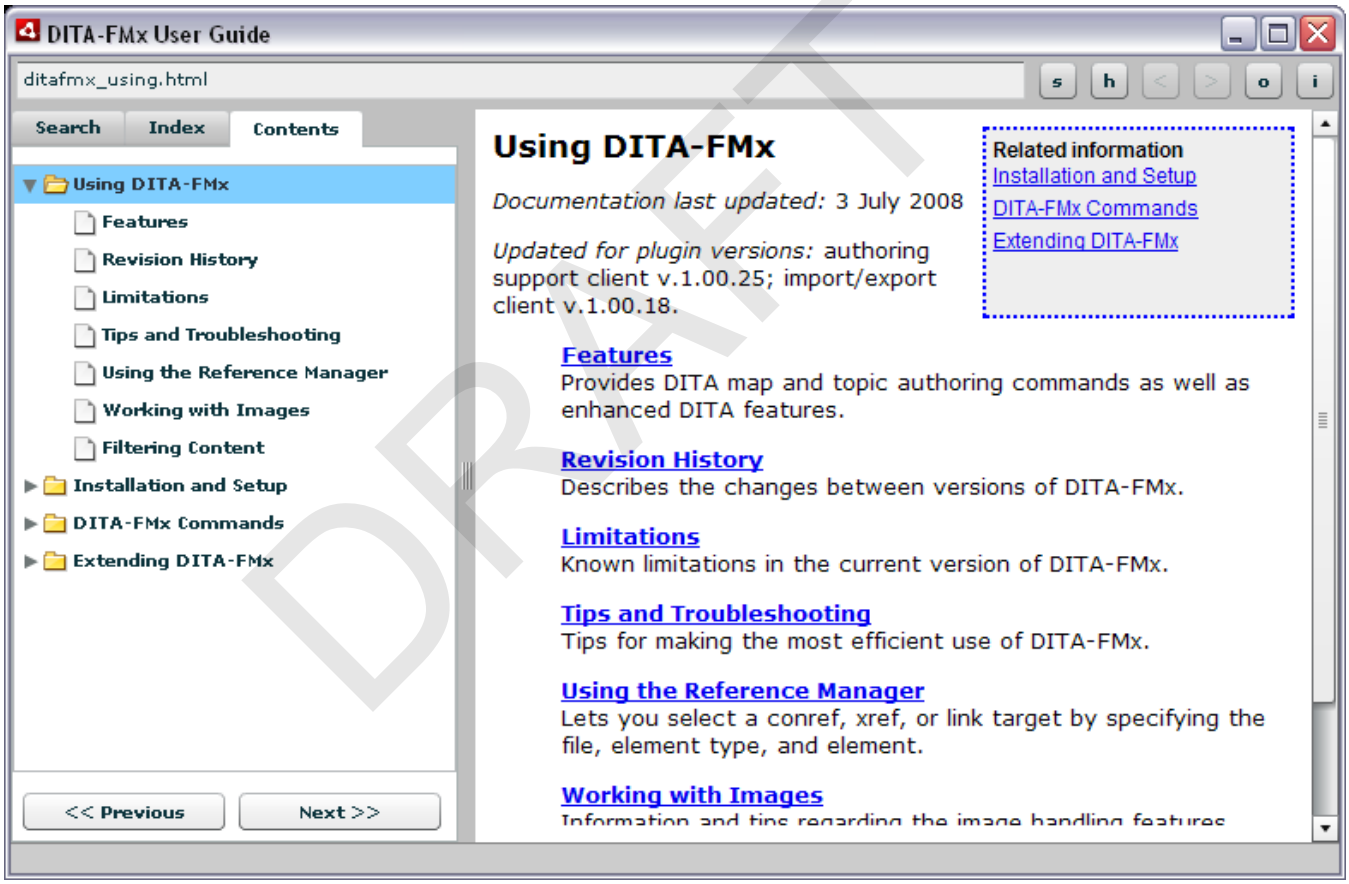


Figure 2: Example AIR Help application window

Summary

Because AIR Help is very new and still under development, it may be best for some to wait for further development to be completed. However, because of its cross-platform support and reduced testing effort, you may find that it is worth closer investigation as a potential Help delivery format.

Microsoft HTMLHelp

Context-Sensitive Help using the Enhanced HTML (htmlhelp2) Plug-In

This topic describes the `htmlhelp2` plug-in for the DITA Open Toolkit.

Overview

The process of generating Microsoft® HTML Help (CHM) output from DITA content using the DITA Open Toolkit does not provide the abilty to store context-sensitive Help identifiers in the topics or in the ditamap. Context identifiers have to be created and managed outside the DITA authoring enviornment. The `htmlhelp2` plug-in created by Deborah Pickett extends the DITA OT functionality to allow context-identifiers to be generated from values stored in the resourceid attribute of the ditamap's `topicref`.

Setup and configuration

To install the plug-in, download the plug-in ZIP file from *Yahoo! DITA Users Group* and extract the files into the `\plugins` directory in the DITA-OT directory, and run the integrator task. The `htmlhelp2` transtype will then be available as an Ant build output.

```
The htmlhelp2 functionality is built into the ditaplus plug-in, also created by
Deborah Pickett.
```

Authoring

The plug-in uses the `resourceid` element within the `topicmeta` element in the `topicref` in the ditamap as the basis for building the map and alias files required for HTML Help context-sensitive Help. The `resourceid` element doesn't contain data, and has two attributes: `id` and `appname`. The `id` attribute can hold only value, whereas the context-hooks in HTML Help require a string and a numeric indentifier. It is expected you will enter a numeric value into the `id` attribute of the `resourceid`, and the plug-in will derive from that number a string value.

For example, an `id` of `3254` will be transformed to a map file entry of:

```
#define identity_3524 3524
```

In other words, the `resourceid`'s `id` attribute is used as the context number, and used to create the context string by prefixing the number with the string identity_.

In addition to entering a context number in the `id` attribute, you must also enter an `appname` attribute of `WindowsHelpId` in the `resourceid` element in the `topicref` in the ditamap. The `WindowsHelpId` value indentifies the particular `resourceid` as being used for context-sensitive Help, thus differentiating it from `resourceid` elements used for other purposes.

You can have add more than one `resourceid` element if you need several different help IDs to be pointed to the one topic.

How It Works

In Microsoft® HTML Help, map (`.h` or `.map`) files correlate context strings to context numbers, and alias (`.ali`) files map context numbers to topic file names.

The htmlhelp2 transformation extracts from all topics in the ditamap any `resourceid` elements where the `appname` attribute is `WindowsHelpId`. The map and aliasfiles generated by htmlhelp2 are named *`[map_file_name]`*`.map` and *`[map_file_name]`*`.ali` respectively.

DRAFT

Example

The following markup:

```
<map>
<title>Cat flossing </title>
<topicref href="abc.xml">
  <topicmeta>
   <resourceid appname='WindowsHelpId' id='3524'/>
  </topicmeta>
</topicref>
...
</map>
```

will produce the following map and alias file lines:

- map:

  ```
  #define identity_3524 3524
  ```

- alias:

  ```
  identity_3524 abc.html
  ```

Summary

The htmlhelp2 plug-in provides a simple method for generating HTML Help context map and alias files from values entered in the resourceid element in the ditamap's topicref.

The DITA Open Toolkit HTMLHelp Transform

This topic provides a brief introduction to the HTMLHelp transform in the DITA Open Toolkit.

We would be remiss not to call attention to the original HTMLHelp transform that has been bundled with the DITA Open Tookit for several years.

Overview

Output from the DITA Open Toolkit HTMLHelp plug-in serves as input to the Microsoft HTML Help Compiler. If you choose to build your DITA sources on teh same Windows system on which you have installed a copy of the Microsoft HTML Help Software Development Kit, you will be able to generate a ready-to-deploy MS HTML Help `.chm` library from your DITA-OT ant script. Otherwise, you an copy output from the DITA Open Toolkit HTMLHelp transform on your DITA-OT build system to some other Windows system on which the HTML Help SDK has been installed.

Setup and configuration

Once you have installed and configured the DITA Open Toolkit, you have the HTMLHelp transform installed by default and ready to use. See the *DITA Open Toolkit User's Guide* for detailed information about working with transforms and about specifying the HTMLHelp transform specifically as a processing target (dita2htmlhelp).

To download the latest Microsoft HTML Help 1.4 SDK, go to the following MSDN web site. It is a free download.

http://msdn.microsoft.com/en-us/library/ms669985.aspx

Note: If you run the test build script that is bundled with the Open Toolkit, `sample_all.xml`, you can verify that the HTMLHelp transform is installed and configured correctly by looking in the `ant/out/samples/htmlhelp` subdirectory.

DRAFT

Authoring

To generate ready-to-deploy `.chm` libraries from most any DITA source collection, you do not really have to customize or rework your DITA source files.

DITA source	MS HTML Help output
DITA map entries	Contents tab entries
index entries	Index tab entries
running text	full-text search index

Integration

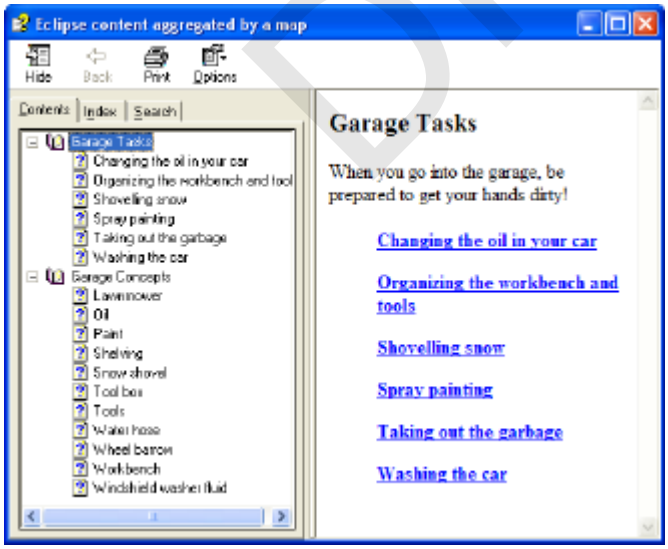
Truth be known, there are also limitations to both this DITA-OT transform and to the MS HTML Help run-time environment. Factor these into your planning.

- *Context-sensitive mapping*: The DITA-OT HTMLHelp transform is not designed to process context-sensitive mappings between DITA source topics and the context IDs associated with those target topics. To feed the MS HTML Help compiler with the appropriate context-sensitive ID maps and/or alias files, consider developing a custom shell for the Open Toolkit or building your HTML Help output with WinANT. See *WinANT Options Supporting Microsoft HTML Help* on page 47 for more information about the latter.
- *HTML support in MS HTML Help*: The Microsoft HTML Help application displays HTML topics in a captive HTML viewer. This viewer supports most, but not all, of the HTML output that you can generate from the DITA-OT XHTML transform. If, for example, you are working with SVG images or some interactive JavaScript libraries, test them with output from the HTMLHelp transform before deploying them.

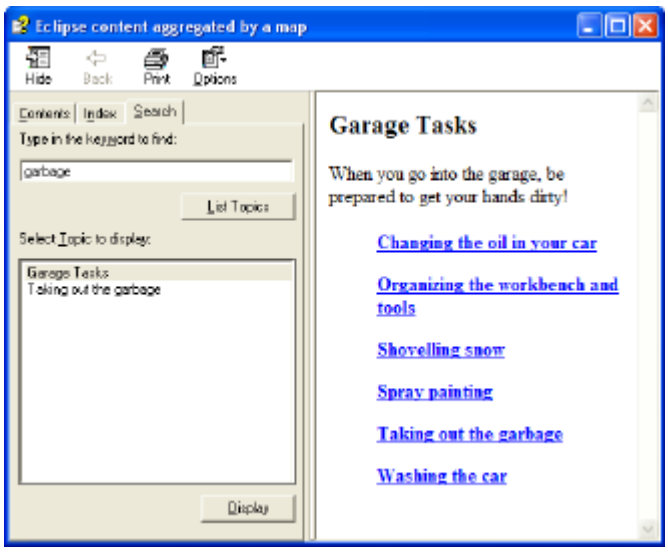
Output

The `.chm` library produced by the combination of the DITA-OT HTMLHelp transform and the MS HTML Help compiler is very functional.

Here's what output from the DITA-OT test collection looks like.



DRAFT



👉 **Note:** The DITA-OT test collection source files contain no index entries, so no index entries are generated in MS HTML Help.

Summary

If you have requirements to deploy a general-purpose Help collection to Windows customers *without* context-sensitive integration, consider using the DITA-OT HTMLHelp transform. It is functional, realiable and fast. Once all your content in is `.chm` format, you can convert it to other distribution formats with second-tier conversion tools such as *chm2web*. If you have requirements to integrate `.chm` output with Windows applications as Context-sensitive Help, you'll need to invest some time managing maps and alias files outside the DITA environment and then integrating them with `.chm` output using WinAnt or other custom processing environments.

DRAFT

Developing Custom DITA-based Help Systems

This topic introduces solutions for developing custom Help implementations using DITA source and/or output.

If your company has developed a custom Help viewer or custom run-time environment, it is unlikely that the default output from an existing DITA plug-in will be what you need. This section of the *Best Practices Guide* explains how to use DITA plug-ins and other tools to generate the base ingredients for a custom help implementation. You will need to invest some amount of post-processing effort, therefore, to integrate output from multiple DITA plug-ins and/or to customize output for exactly what you need.

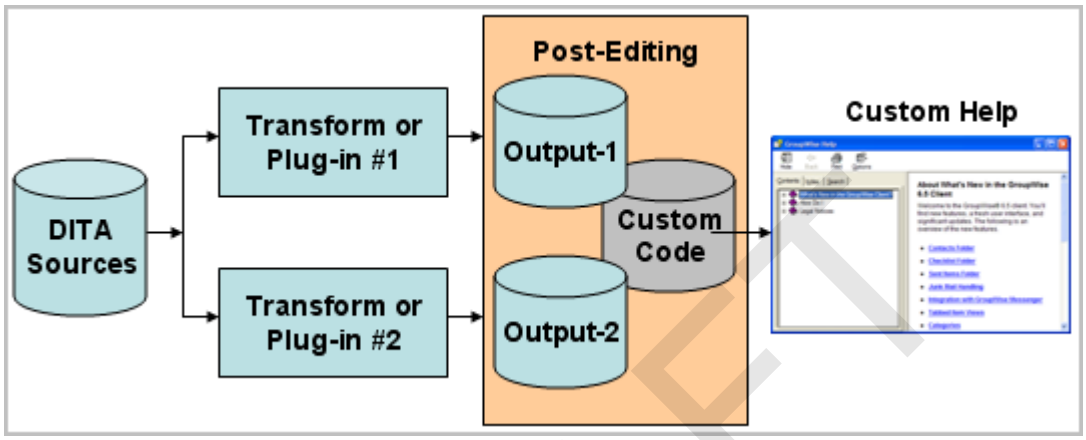


Figure 3: DITA Source to Customized Help Environments

We can't really help you with these post-processing or customization tasks, but we can help you generate the ingredient content and point out some common issues that might affect your customizations.

- *DHTML Effects in HTML Generated from DITA* on page 26
- *Dynamic Rendering of DITA into XHTML* on page 41
- *JavaScript-Based Context Sensitive Help* on page 42
- *HTMLSearch Plug-in* on page 28
- *TOCJS and TOCJSBIS Plug-ins* on page 36
- *WinANT Options Supporting HTML-Based Output* on page 45
- *WinANT Options Supporting Microsoft HTML Help* on page 47

DRAFT

DHTML Effects in HTML Generated from DITA

This topic describes an approach to creating expanding text and other DHTML effects in HTML-based output generated from DITA content.

It is common for help systems to use *layering* techniques to limit the amount of information presented to the reader. The reader chooses to view the information by clicking on a link. Most layering techniques, including expanding text, drop-down text and popup text, are implemented using Dynamic HTML.

Overview

The DITA Open Toolkit HTML transforms do not provide for layering effects. However, some changes to the XSL-T files, and the use of outputclassmetadata in the DITA topic content, along with some judicious use of JavaScript and CSS, can deliver these layering effects.

Authoring Example

In the following example illustrating this technique, a note element is encoded to output as drop-down text, where the note label is used to toggle the display of the note text. The note element is simply marked up with an outputclass distinct attribute value (in this case, `hw_expansion`).

```
< note outputclass="hw_expansion" type="note">Text of the note</note>
```

Without any modification, the DITA OT will transform the note element to a paragraph element with a CSS class of the outputclass value.

XSL-T Changes Example

In the example illustrating this technique, note how the following lines change the way that the XSL-T file processes the note element:

```
<xsl:template match="*[contains(@class, 'topic/note ')]">
<xsl:if test="@outputclass='hw_expansion'" >
  <p>
   <a class="pseudolink_up" onclick="hwToggle(this);">Note: </a>
  </p>
</xsl:if>
<div >
<xsl:if test="@outputclass='hw_expansion'" >
 <xsl:attribute name="class">
   <xsl:value-of select="@outputclass" />
 </xsl:attribute>
</xsl:if>
</div>
</xsl:template>
```

JavaScript and CSS Addition Examples

For the layering DHTML effect to work, a JavaScript routine must be written to control the *toggling* of the display of the dropdown text. Likewise, CSS classes used in the layering (in this example, pseudolink_up and hw_expansion) must be defined in the CSS file for the output.

A simple JavaScript toggling routine is as follows:

```
function hwToggle(me) {
 var nextS=getNextSibling(me.parentNode);
 if(nextS.style.display!='block') {
  nextS.style.display='block';
  me.className='pseudolink_down'
 }
 else {
  nextS.style.display='none';
  me.className='pseudolink_up'
 }
}

function getNextSibling(startBrother){
  endBrother=startBrother.nextSibling;
 try {
  endBrother.nextSibling;
 }
 catch(er) {
  return null;
 }
  while(endBrother.nodeType!=1){
    if(!endBrother.nextSibling){
        //alert('Sorry! No more siblings!');
        return null;
        }
    endBrother = endBrother.nextSibling;
```

DRAFT

```
  }
  return endBrother;
}
```

A simple set of CSS settings are:

```
a.pseudolink_up
{
    color: blue;
    cursor: hand;
    border-bottom: blue 1px dashed;
    background-position: left center;
    background-image: url(images/purple_right_sm.jpg);
    background-repeat: no-repeat;
    padding-left: 15px;
}
a.pseudolink_down
{
    color: green;
    cursor: hand;
    border-bottom: blue 1px dashed;
    background-position: left center;
    background-image: url(images/purple_down_sm.jpg);
    background-repeat: no-repeat;
    padding-left: 15px;
}

div.hw_expansion
{
    display:none;
}
```

When the build file is created, references should be made to include the CSS file (containing the CSS rules) and the JavaScript file (containing the toggle routine).

Working Example

The layering technique described here is used in a production environment on a Web site which dynamically renders DITA content as XHTML. (Refer to *Web page using DHTML layering technique*.)

Summary

Although some technical knowledge is required to implement DHTML efects in output, the techniques are not onerously complex. With a small investment in effort, a big payoff in functionality can be achieved.

DRAFT

DITA-OT Plug-ins

HTMLSearch Plug-in

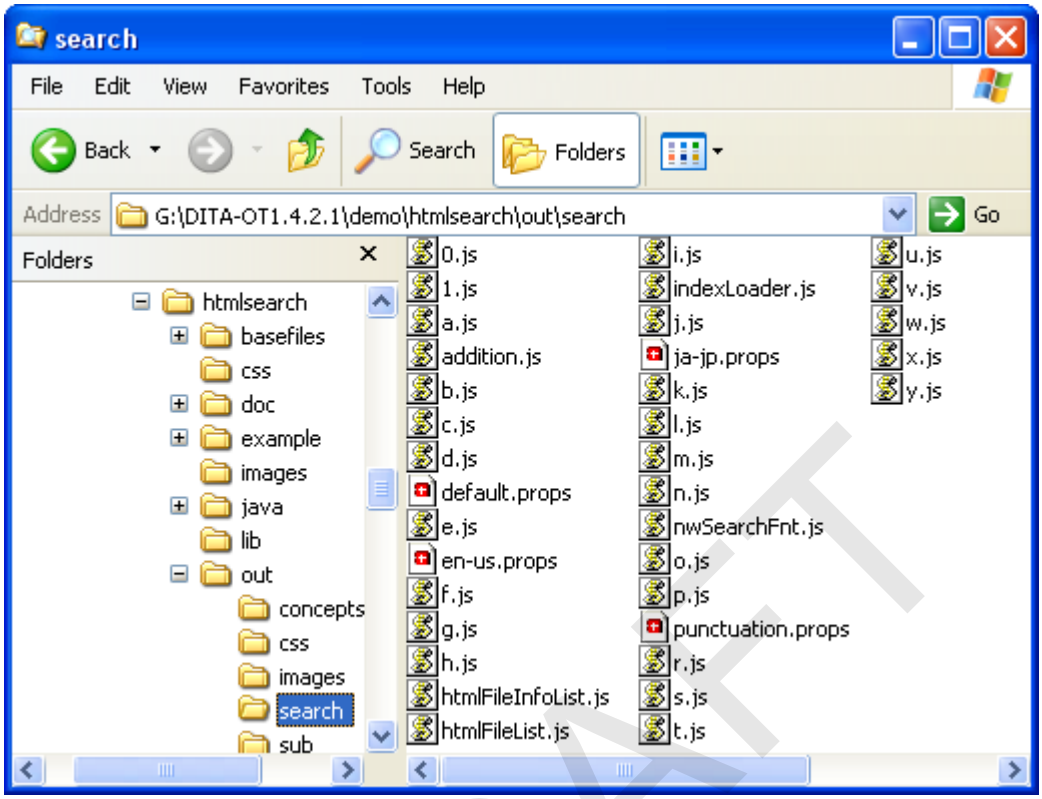
This topic provides an introduction to the DITA plug-in named HTMLSearch.

The HTMLSearch plug-in for the DITA Open Toolkit builds a keyword-based search index and a default frame-based user interface for any collection of DITA topics referenced by a DITA map file.

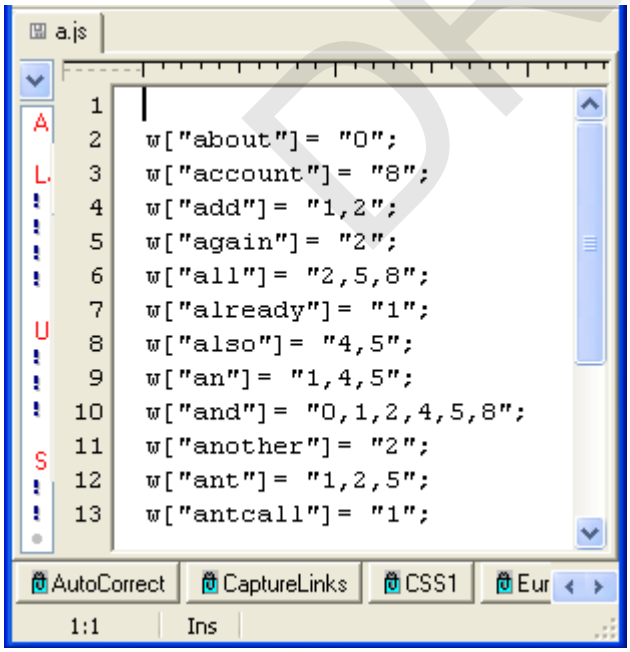
Overview

Nadege Quaine (nquaine@hotmail.com), the contributor of the tocjsbis plug-in, has also contributed this plug-in. As of this writing, the most current version of the plug-in is datedApril 8, 2008 and is distributed as `htmlsearch1.04.zip`.

The HTMLSearch plug-in first runs the XHTML transform that comes with the DITA-OT and then runs an indexing application against those XHTML output files. HTMLSearch deposits the keyword index and the supporting JavaScript worker libraries in a subdirectory named `search`.

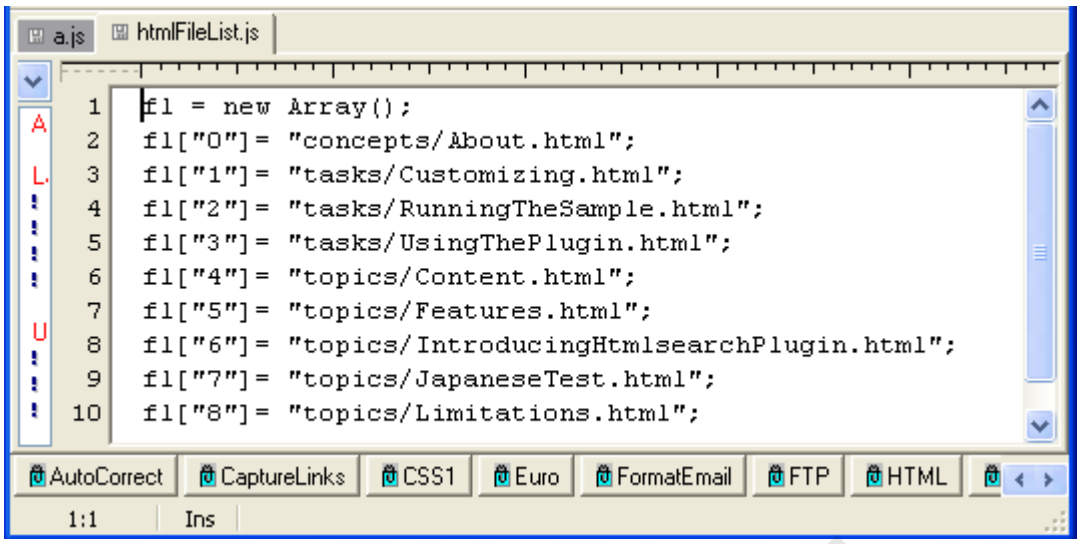


Each of these JavaScript index files correlates discovered keywords to one or more numerals representing particular XHTML output topics.

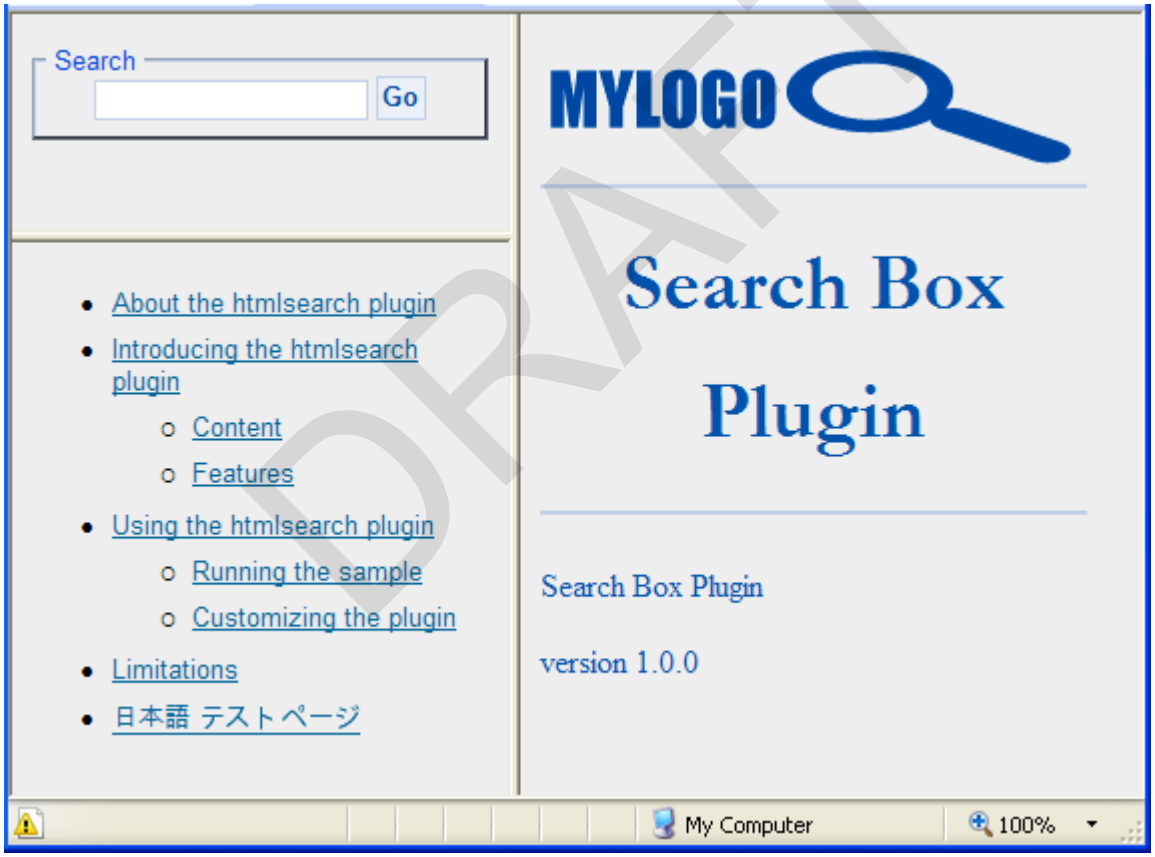


The keyword "ant," for example, appears in three XHTML topics. These target topics are referenced numerically (1, 2, 5) and indexed separately in the file `search\htmlFileList.js`.

DRAFT

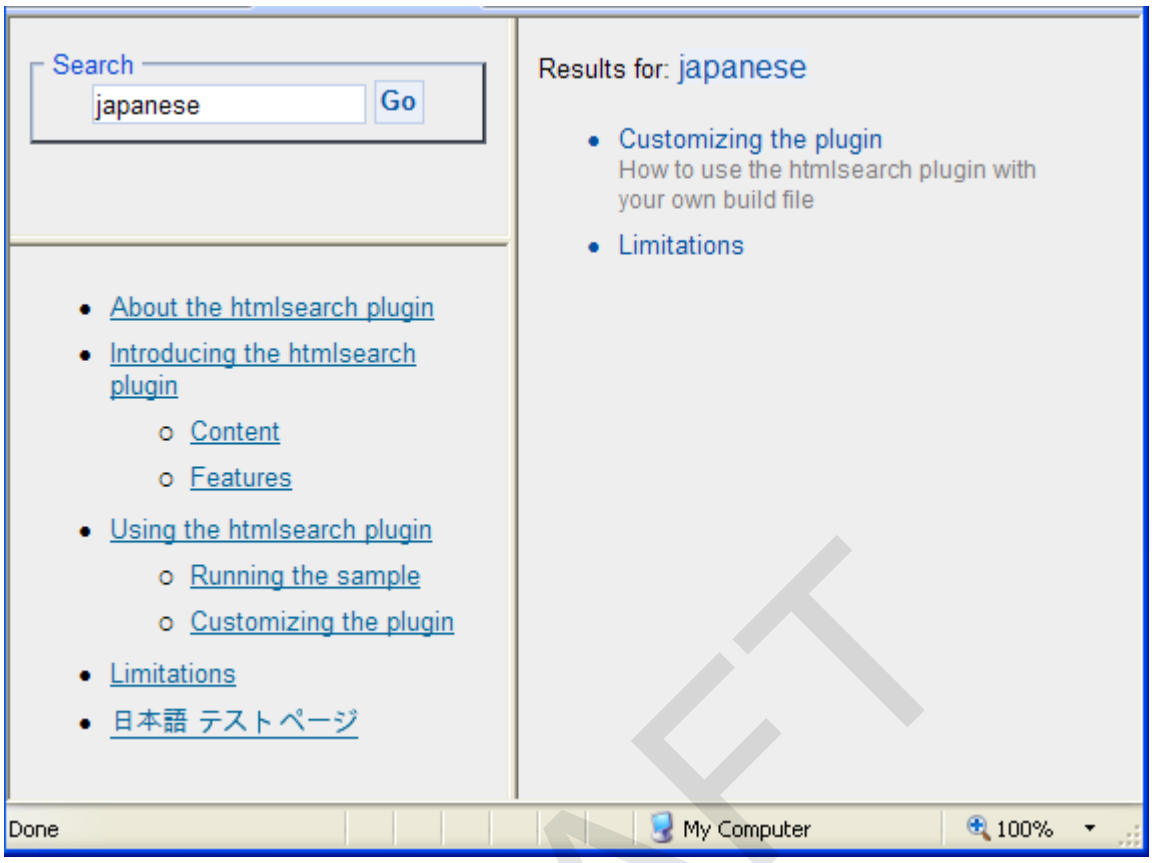


When you open the default HTMLSearch web page (`frameset.html`) in the output directory, it opens three frames containing a keyword input box, some default information about the plug-in, and a splash screen.

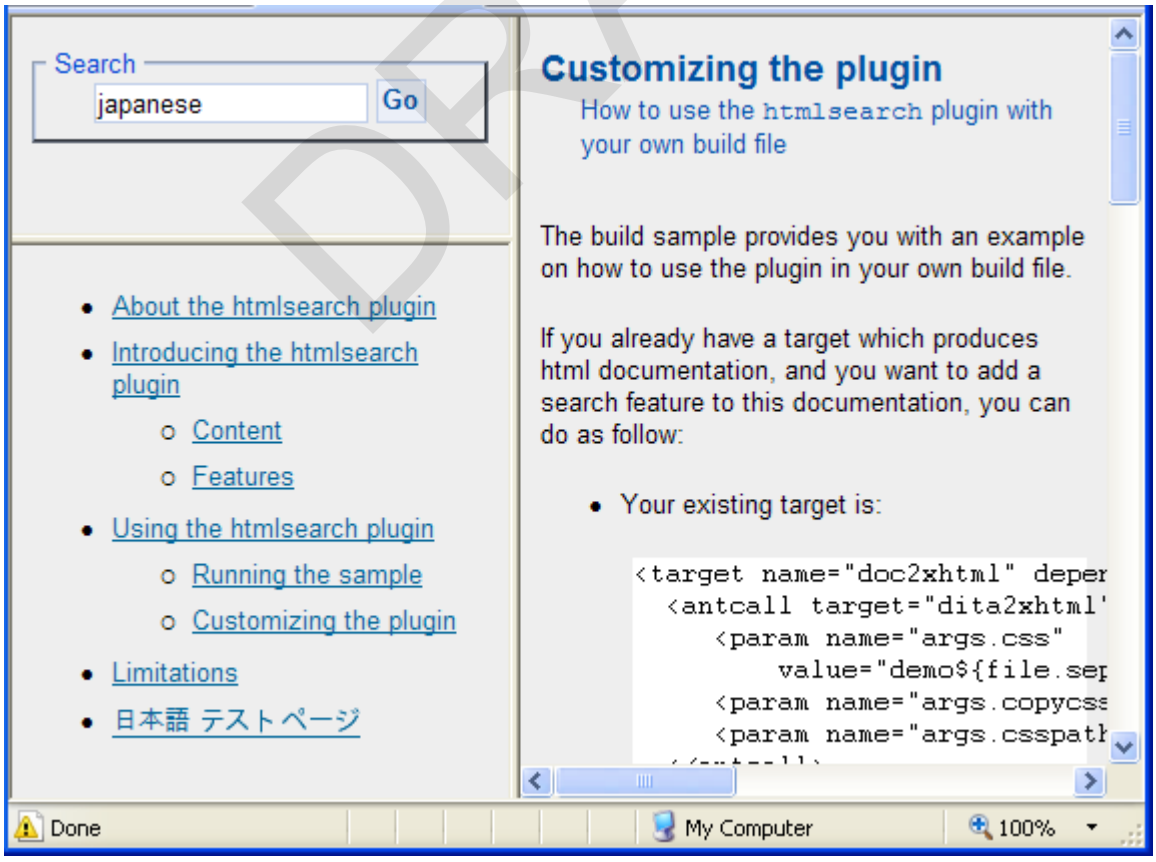


Results from a keyword search are displayed in the right-hand pane.

DRAFT



Clicking a hyperlink in the results list displays the target topic containing the keyword in the same right-hand pane.



DRAFT

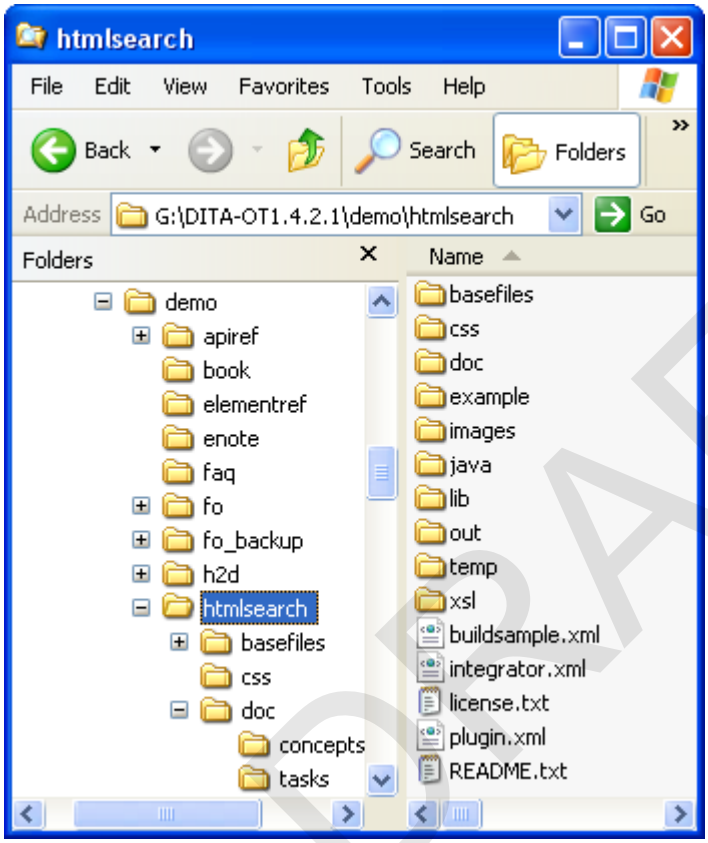
The plug-in works fine in all versions of the DITA-OT because it is really interacting with XHTML output from the DITA-OT, not Open Toolkit resources *per se*. Beyond minor variations in the way the HTML displays in different browsers, HTMLSearch output works reliably in all modern browsers.

Setup and configuration

The HTMLSearch plug-in is a free download from the Yahoo DITA-OT site.

http://tech.groups.yahoo.com/group/dita-users/files/Demos/

After you have unzipped this archive to a local directory, you can browse the `README.txt` file to get a feel for what HTMLSearch offers and how you can install it in your DITA-OT `demo` directory.



Installing the plug-in presents no surprises.

1. Copy the un-archived `htmlsearch` subdirectory into the `demo` subdirectory of your DITA Open Toolkit directory.
2. Open a shell command window from your DITA-OT directory.
3. Enter the following command to integrate the new plugin or plugins with your current DITA-OT environment.

   ```
   ant -f integrator.xml
   ```

4. From the DITA-OT root directory, enter the following command to build the sample DITA topics.

   ```
   ant -f demo/htmlsearch/buildsample.xml xhtml2search
   ```

5. Load the newly generated `demo\htmlsearch\out\frameset.html` file in your browser.

That is about all that is involved with installing and configuring the plug-in.

Authoring

There are no DITA source-level authoring considerations for this plug-in; it works on XHTML output topics generated by the DITA-OT XHTML transform.

DRAFT

Integration

Integrating output from the HTMLSearch plug-in with tocjsbis output or with your local favorite HTML implementation is possible, but challenging. If you have some experience modifying (or hacking) HTML frames and JavaScript code, have at it. Otherwise, you might consider continuing to use HTMLSearch output as a stand-alone complement to your other output transformations.

Consider the following tips when planning your customization effort.

- *Quotation marks in DITA map navtitles*: If the navtitles in your DITA map file(s) contain quotation marks, remove them before running the HTMLSearch plug-in. HTMLSearch passes these quotation marks through to its indexed output, effectively breaking JavaScript syntax in the `search\htmlFileInfoList.js` file.
- *Directory levels for XHTML output*: The HTMLSearch plug-in writes the hyperlinked results of a keyword search to a frame named `contentwin`. The initial static HTML loaded into that frame lives in the output subdirectory named `sub`. The filepaths in the hyperlinks written to that frame, therefore, are relative to an HTML file in `sub`. If you are using default XHTML output from the tocjs or tocjsbis, you may need to edit the JavaScript code that builds that relative path. Test removing the three characters "`../`" in line 131 of `search\nwSearchFnt.js` if you need to adjust HTMLSearch hyperlink paths to match target directory paths from tocjsbis or other DITA-OT XHTML output transformations.

  ```
  linkString = "<li><a href=\"../"+tempPath+"\">"+tempTitle+"</a>";
  ```

- *Target frame name for keyword search input box*: The HTMLSearch plug-in loads its input box into a frame named "searchwin" in its default interface. To integrate that input box into a different set of named HTML frames, you will need to change the name of the input box frame from "searchwin" to the new target frame name in lines 32, 39, 47, 242, and 248 in `search\nwSearchFnt.js`. For example, change "searchwin" to "*your_frame_name*" in the following line in `search\nwSearchFnt.js`:

  ```
  expressionInput=parent.frames['searchwin'].document.ditaSearch_Form.textToSearch.value
  ```

- *Target frame name for keyword search results*: The HTMLSearch plug-in writes its list of keyword search results (hits) to a frame named "contentwin" in its default interface. To have those results displayed in a different frame in your implementation, you will need to change the name of the input box frame from "contentwin" to the new target frame name in lines 146 and 414 in `search\nwSearchFnt.js`. For example, change "contentwin" to "*your_frame_name*" in the following line:

  ```
  with (parent.frames['contentwin'].document) {
  ```

- *Target frame name for target topics*: The HTMLSearch plug-in displays target topics in the same "contentwin" frame as the search results. If you would like to have *both* the list of search results *and* displayed target topics displayed simultaneously in separate frames, you will need to change that way that HTMLSearch builds hyperlinks. Specifically, you'll need to add an HTML "target=*framename*" attribute to line 131 in `search\nwSearchFnt.js`. For example, insert the string `target='topicwin'` (or *your_frame_name*) into line 131. Here's the original ...

  ```
  linkString = "<li><a href=\"../"+tempPath+"\">"+tempTitle+"</a>";
  ```

  and the update ...

  ```
  linkString = "<li><a href=\"../"+tempPath+"\"
  target='topicwin'>"+tempTitle+"</a>";
  ```

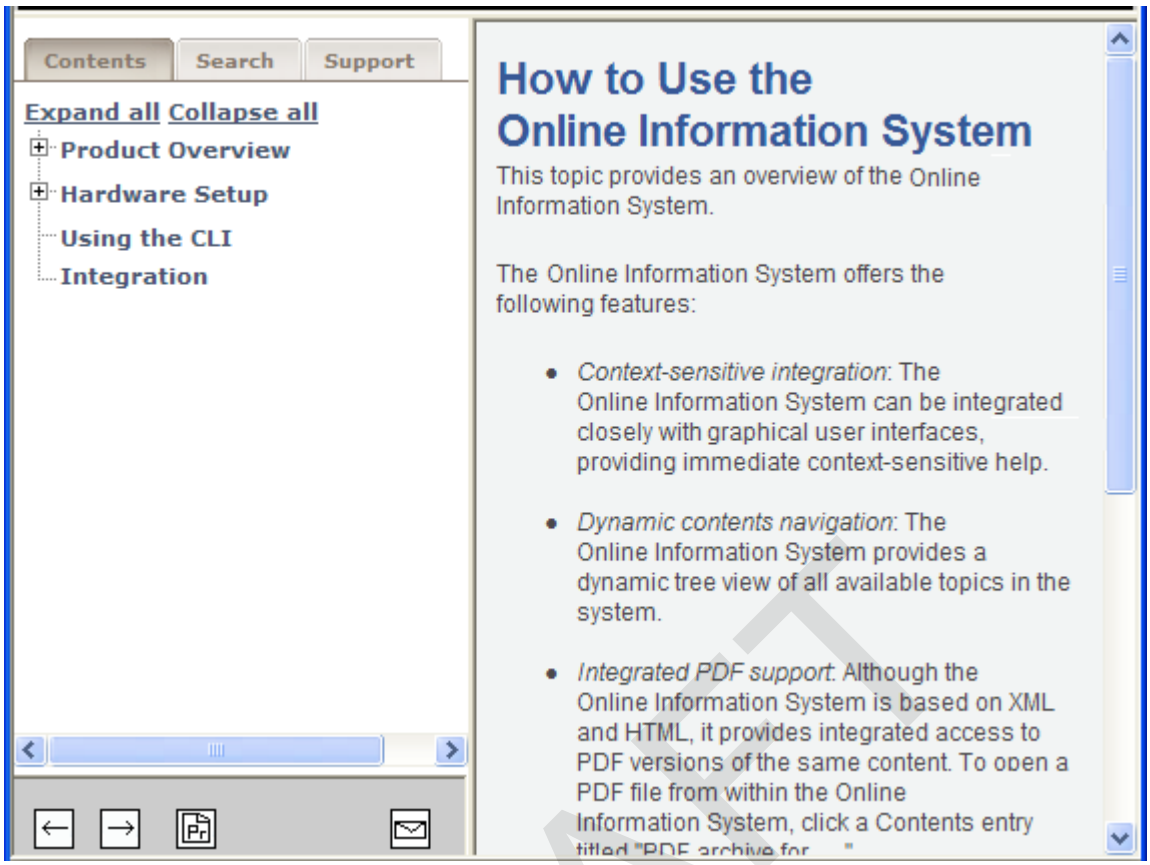
  After this change, HTMLSearch builds hyperlinks in the keyword search results frame that specify a different target window for displaying target XHTML topics.

These are the most visible variables that you'll need to consider when customizing HTMLSearch output for your own help implementation.

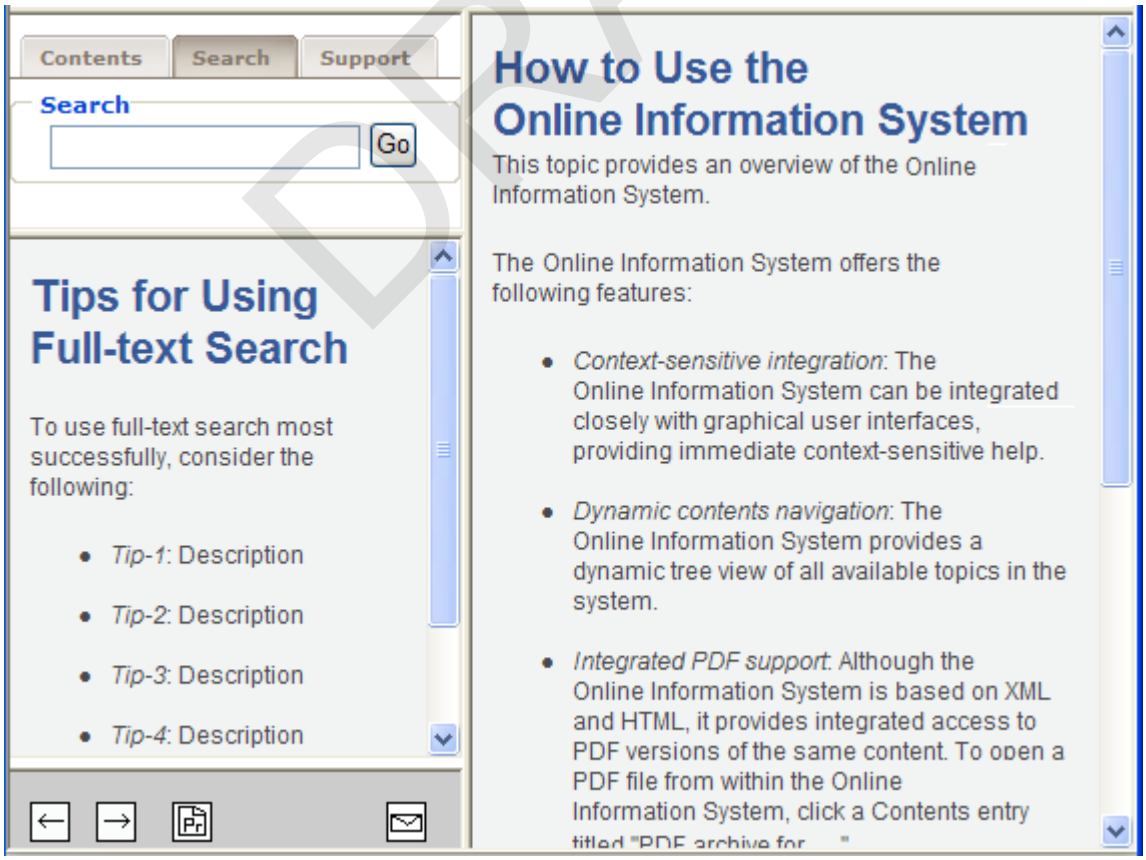
Output

Although the customization process can sound scary, it does produce some very nice results. Here is a prototype Help implementation for my current company. The goal here is to integrate the output from HTMLSearch with output from tocjsbis in a garden-variety three-tab, multi-frame HTML shell. Initially, the search UI is linked to a Search tab.

DRAFT

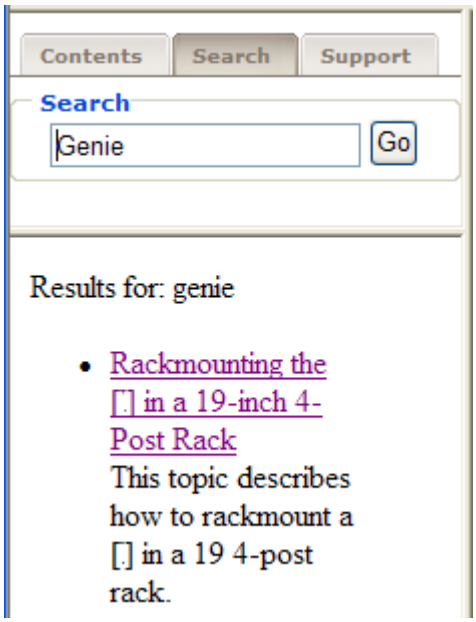


When the customer clicks the Search tab, a re-implementation of the HTMLSearch UI is displayed.

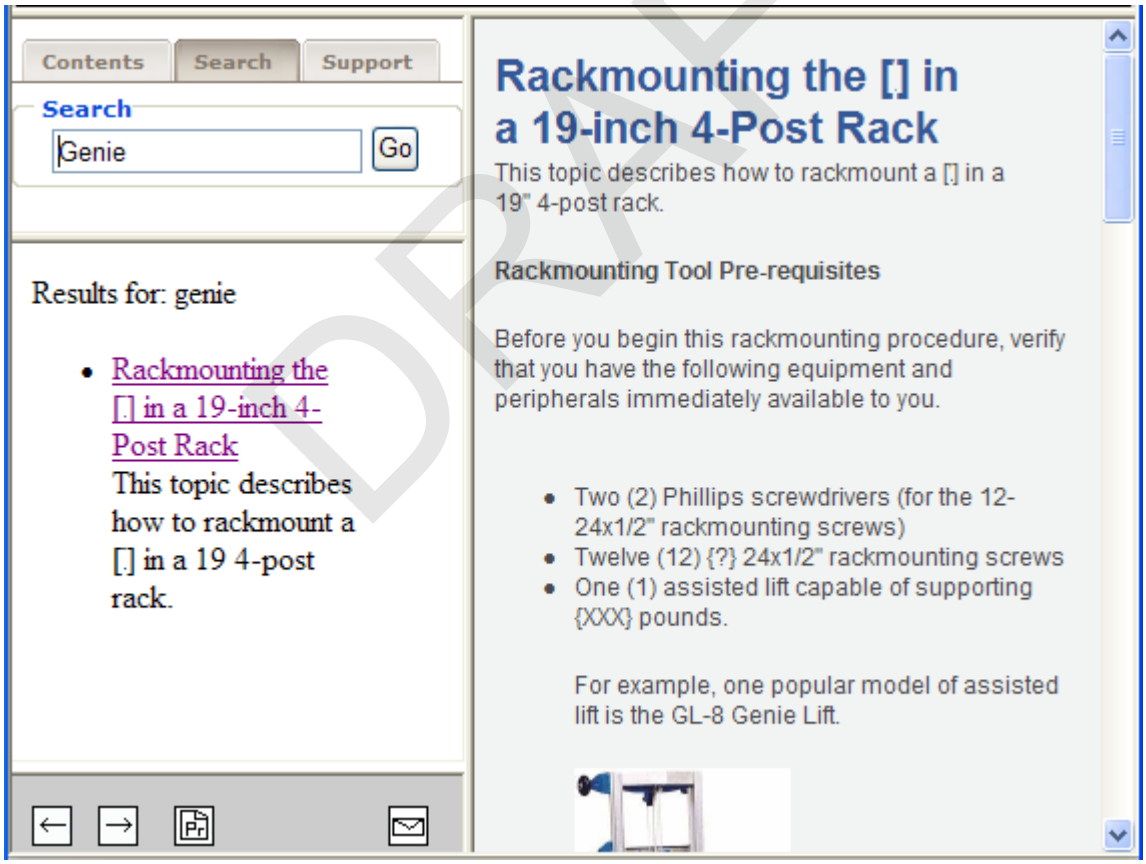


DRAFT

Search results appear in a lower frame in the left-hand navigation frame.



When the customer clicks one of the hyperlinks in the results list, that topic is displayed in the right-hand frame.



DRAFT

Summary

Customizing HTMLSearch to integrate it with tocjsbis or other XHTML output transformations from the Open Toolkit goes a long way toward making DITA-generated Help more usable and familiar to contemporary audiences.

Stan Doherty

OASIS DITA Help Subcommittee

OASIS DITA Technical Committee

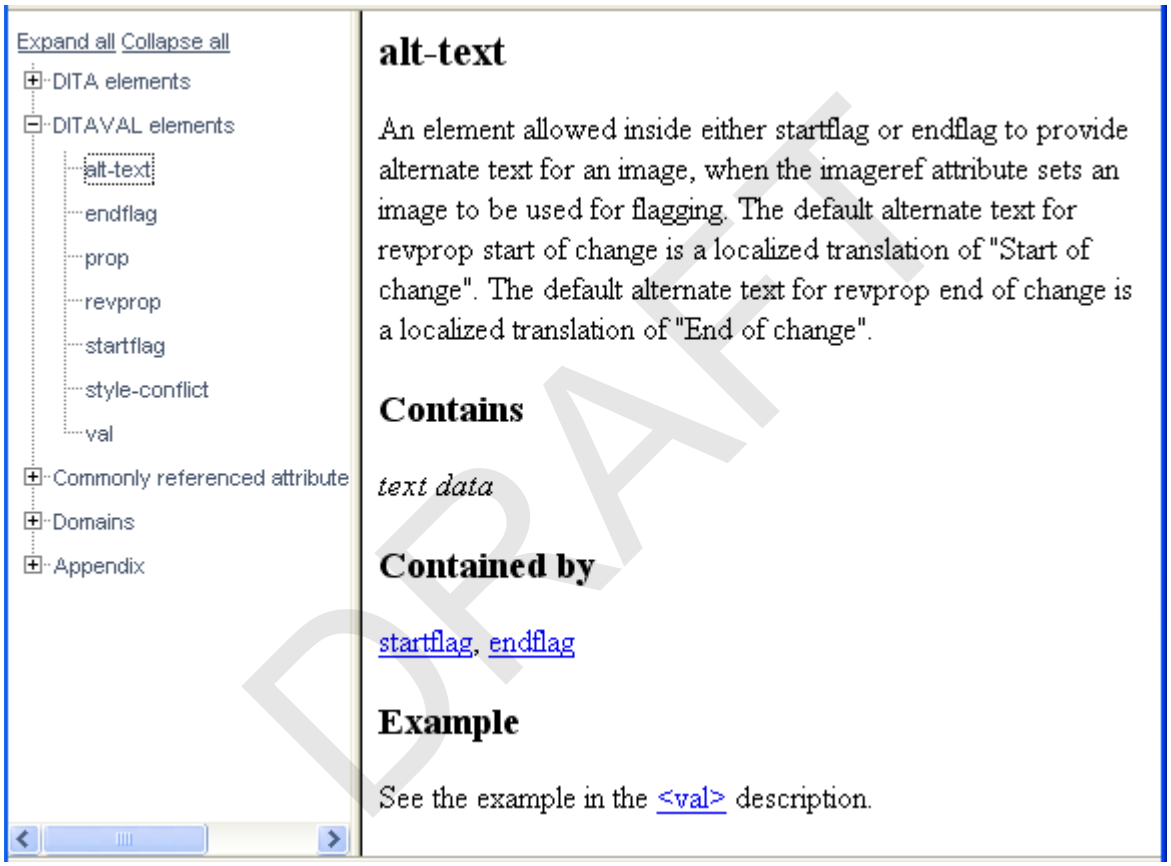
TOCJS and TOCJSBIS Plug-ins

This topic provides an overview of the DITA-OT plug-ins named tocjs and tocjsbis.

The tocjs DITA-OT plug-in and its more recent enhancement named tocjsbis generate a JavaScript-based table of contents page for any DITA topics that you reference in your `.ditamap` file.

Overview

These plug-ins are very popular in the DITA community; we even use them on the DITA Technical Committee for our DITA 1.2 specifications.



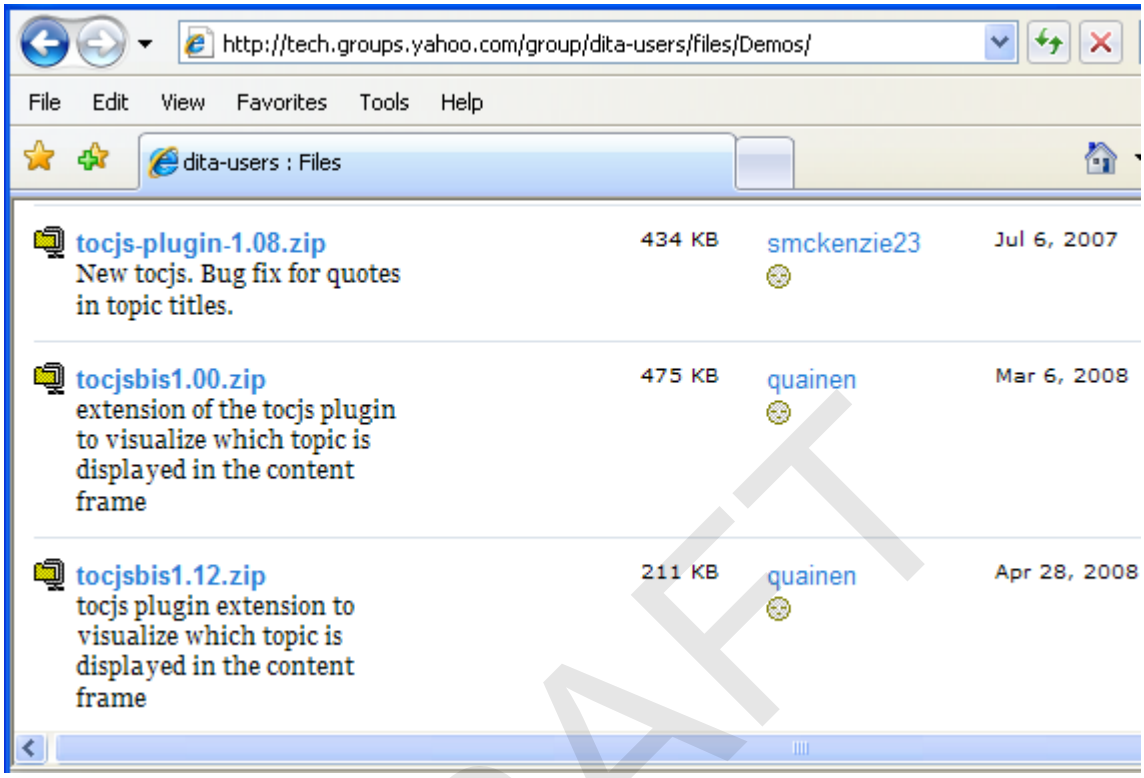
- *tocjs*: The tocjs plug-in was developed by Shawn McKenzie, currently working at Sophos in beautiful Vancouver, BC (Canada). The tocjs plug-in executes after the standard DITA-OT XHTML transform, so each tocjs TOC entry knows the name of its target XHTML topic.
- *tocjsbis*: The tocjsbis plug-in was written by Nadege Quaine and adds the important feature of topic synchronization, i.e. the highlighted topic entry in the TOC updates in sync with the topic being displayed in the contents frame. The plug-in achieves synchronization by adding a unique ID to each XHTML output topic (<meta content="id-tocjsbis_about" name="DC.Identifier" />) and by synchromizing the TOC entry against that topic ID. If you have generated HTML output from RoboHelp or WebWorks Publisher, this technique should be familiar.

If you are generating HTML output of any sort from your DITA sources, you should test one or both of these plug-ins. I use tocjsbis in my context-sensitive Help builds where I work. To the extent that the tree control in tocjs and tocjsbis are based on the Yahoo tree control library, you can customize the way that the final TOC tree displays and behaves in your Help system. They can be tricky, but customizations work.

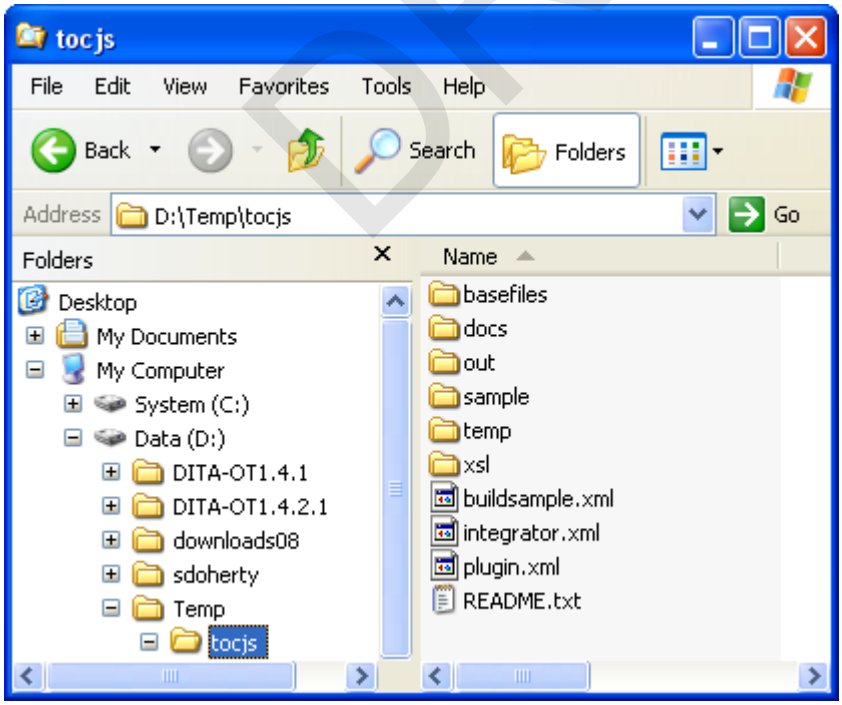
Setup and configuration

The tocjs and tocjsbis plug-ins are free downloads from the Yahoo DITA-OT site.

http://tech.groups.yahoo.com/group/dita-users/files/Demos/

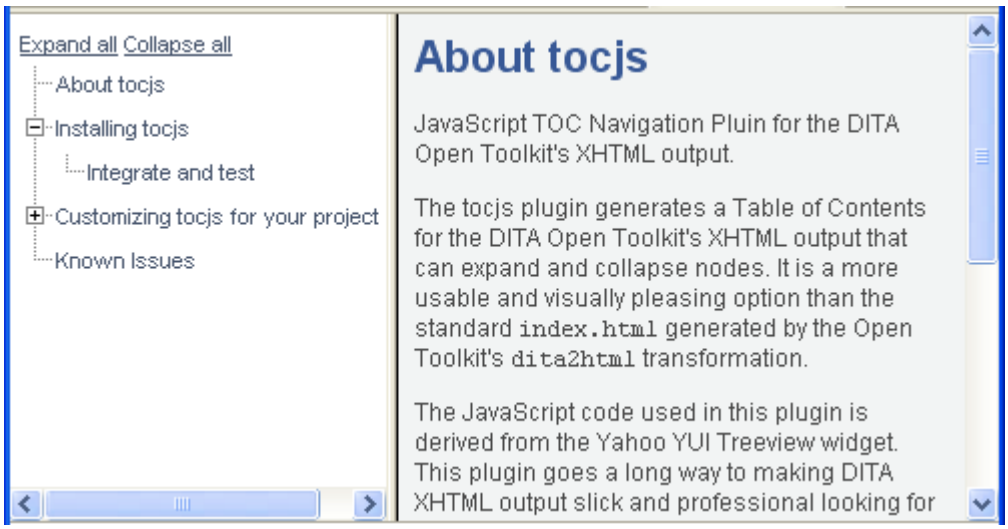


After you have unzipped these archives to a local directory, you can browse the documentation to get a feel for what tocjs offers and how you can install it in your DITA-OT directory.



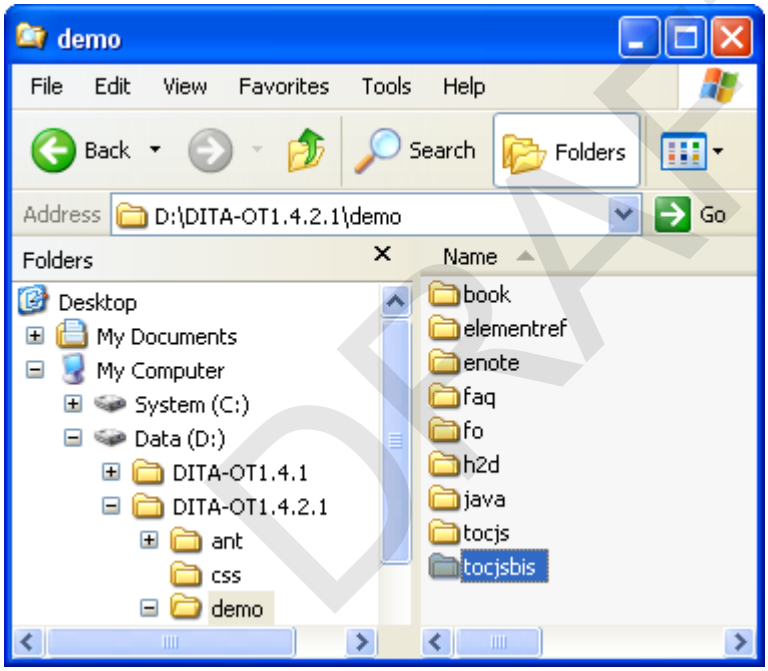
The pre-built documentation for tocjs lives in the `/docs` subdirectory.

DRAFT



Installing tocjs or tocjsbis is very straightforward.

1. Copy the un-achived tocjs or tocjsbis subdirectory into the demo subdirectory of your DITA Open Toolkit directory.



2. Open a shell command window from your DITA-OT directory.
3. Enter the following command to integrate the new plugin or plugins with your current DITA-OT environment.

```
ant -f integrator.xml
```

4. Change directory into the `demo/tocjs` or `demo/tocjsbis` subdirectory and enter the following command to build the sample DITA topics.

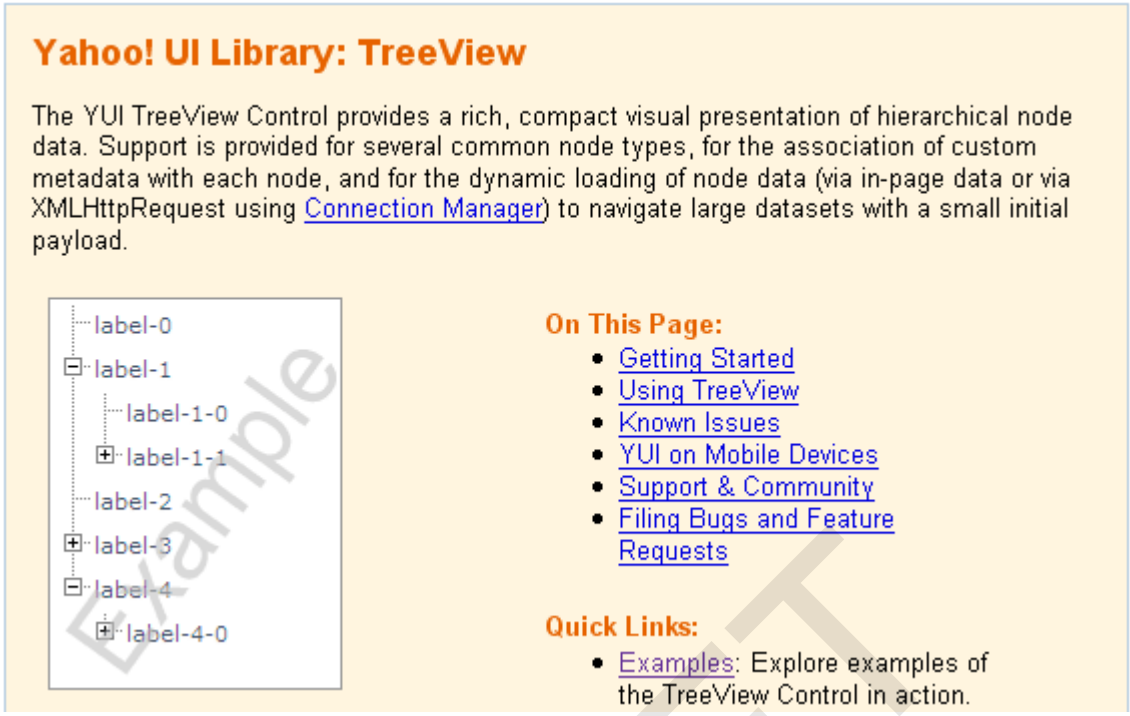
```
ant -f demo/tocjs/buildsample.xml sample2tocjs
```

5. Load the newly generated `demo\tocjs\out\sample\frameset.html` file in your browser.

If you are considering customizations to tocjs or tocjsbis, consult the documentation for the Yahoo UI tree control at the following URL.

http://developer.yahoo.com/yui/treeview/

DRAFT



That is about all that is involved with installing and configuring tocjs and tocjsbis.

Authoring

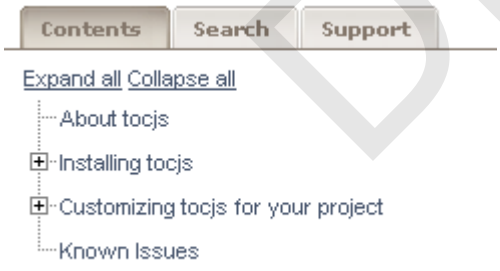
These plug-ins piggy-back whatever investment you have already made in authoring your DITA topics and map files. Beyond setting up a new ant script for tocjs or tocjsbis, there is nothing additional required.

Integration

The tocjs and tocjs plug-ins present few integration problems or opportunites, especially as regards managing context sensitivity between Help output and the calling software application. Many DITA Help writers customize the `frameset.html` file that ships with the tocjs plug-in to personalize or brand the final product. Here's what Shawn McKenzie does with tocjs on his Sophos corporate website.

DRAFT

Wrapping other navigational devices around the tocjs output is not difficult. I add the conventional tabs for tri-pane Help systems.



In terms of translation, note that all the text strings associated with entries in the TOC tree are stored in one file named `toctree.js`. These files can be localized, for sure, but they are not very friendly. To make life easier on our sisters and brothers in L10N, you can post-edit this `toctree.js` file to swap JavaScript resource strings for literal strings.

Output

Here again is a link to a live demo of tocjs running at Sophos.

http://ca-repo1.sophos.com/docs/ws1000/

DRAFT

Summary

If you are comfortable customizing XHTML output to build an HTML-based help system, you should consider tocjs and tocjsbis. They are sufficiently lightweight to work in Help systems or web-based portals.

Stan Doherty

OASIS DITA Help Subcommittee

OASIS DITA Technical Committee

Dynamic Rendering of DITA into XHTML

This topic describes the concept of dynamic transformation of DITA content on a web server into XHTML for delivery to any web browser.

In many cases, Help content is best delivered as HTML. For content authored in DITA, the process is typically to transform the content into HTML (or, more likely, XHTML), and then transfer the output files to a web browser. In some cases, it may be desirable to host the DITA source files on the web server, and use server-side processing to dynamically transform the content to HTML when a browser requests the information.

Overview

When DITA content is transformed to a delivery format through the DITA Open Toolkit or another publishing process, the result is a collection of deliverable files that also need to be managed.The deliverable files (typically XHTML) must be copied to a web server for delivery. When the source content changes, the output has to be regenerated, and then copied once more to the web server.

Most web servers support some sort of server-side processing technology; examples are ASP.Net, PHP and JSP. Many of these technologies are XML-aware, and are capable of handling dynamic XSL-T transformations.

It is possible to set up a system whereby Help content, written and stored in DITA format, is stored on a server, and then dynamically transformed into HTML for delivery.

Setup and configuration

To set up a dynamic transformation system, the services of a developer or someone familiar with a server-side processing and XSL-T is required. The developer will need to devise a menu system (perhaps derived from a ditamap file), and a topic rendering system.

Authoring

Once the system is in place, DITA topics can be either authored directly onto the web server (perhaps through FTP, WebDAV, or a network share).

Limitations

While it is relatively easy to set up a simple menu system and a simple topic rendering system, it is not easy to handle more advanced DITA features such as conrefs, reltables and composite topics. For example, conref elements need to be resolved prior to the XSL-T transformation. In the DITA Open Toolkit, the generation of output comprises a number of stages to first resolve references to build an interim document, and then process the interim document into the output.

However, if the DITA content is simple enough, this limitation is not an impediment.

Example

The *HyperWrite web site* uses these techniques through ASP.Net to generate the breadcrumb trails, navigation system, menu pages, and articles topics from a ditamap file and numerous DITA concept topics. Some content topics are also hosted in DocBook format.

DRAFT

Summary

The primary advantage of the dynamic transformation approach is to minimise file management. The main disadvantage is that the technique is practically limited to simple content structures. Some other dedicated web delivery methods, such as Eclipse Help, Mark Logic Content Server, or other CMS solutions, may provide greater support for complex content structures.

JavaScript-Based Context Sensitive Help

This topic explains how to configure your browser-based Help system to respond to context-sensitive Help requests from a software application.

Overview

If you are deploying a browser-based Help system in conjunction with a software application, you can configure your main HTML Help file to display a specific target Help file when so requested by a calling software application.

👉 **Note:** The following JavaScript functions do not depend technically on anything inside your DITA source files or anything generated by the DITA Open Toolkit. They are generic functions that support generic HTML.

Single-pane Browser Help Systems

In some situations, you may not need to deploy a fancy, multiframe HTML Help solution to support a software application. In these situations, you can add a few lines of JavaScript to the beginning of your main Help file to process incoming hyperlink calls that target a particular HTML Help file.

For example, if my software application needed to display a particular Help screen (`targetHelpTopic.html`), I could build a hyperlink that would call the main Help file (`1file.html`) and ask it to display my target Help topic.

Here's the HTML hyperlink call.

```
<a href="1file.html?targetHelpTopic.html">link</a>
```

Here's the JavaScript to put at in your main Help file.

```
<HTML>
<!-- One-pane Browser Help System 1file.html -->
<body>
<script language="javascript">
targetTopic = window.location.search.substring(1);
window.location.href=targetTopic
</script>
</body>
</html>
```

When the main Help file (`1file.html`) receives the HTML hyperlink call `<a href="1file.html?targetHelpTopic.html">link</a>`, it identifies any text that follows the question mark (?) as one or more aguments being passed to it. In this example, the script identifies `targetHelpTopic.html` as that argument and makes that HTML file name the current topic to be displayed in the browser.

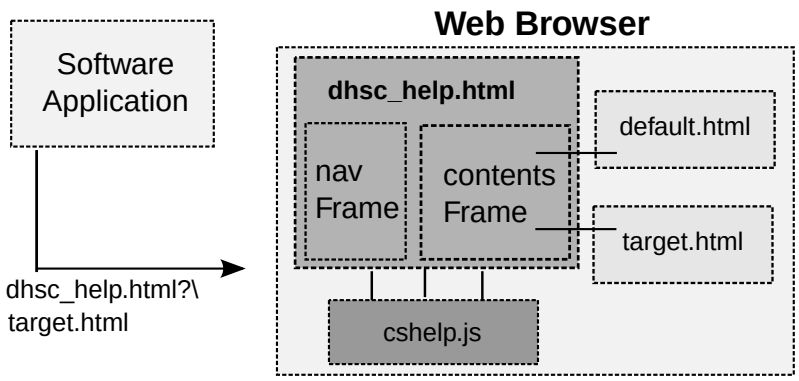
👉 **Note:** This setup works well for captive or embedded Help systems with restrictive or highly predictable calling procedures. If a user or an application calls the main Help file without specifying a target topic, the main Help file will generate an Error 404 "File not found".

Multiframe Browser Help Systems

Many browser-based Help systems use mutlitple frames to provide the customer with topic navigation, search, indexing, and other basic amenities. To get a multiframe Help system to respond to the same context-sensitive hyperlink call is a bit more complicated becuase of some restrictions in how HTML frames and JavaScript interact..

DRAFT

Here are the elements of our little example.



Elements	Description

Software application	Any software application that can generate HTTP calls to a browser will work. For the sake of simplicity, I use an HTML document that contains one hypertext link.
	
	`<html>`
	`<a href="dhsc_help.html?target.html" target=_blank>`
	`Call the help system file and pass it an argument`
	`(the name of the target help topic).</a>`
	`</html>`

Hyperlink	The software application passes two arguments to the Help browser: • Main HTML Help file name (`dhsc_help.html`) • Target Help file name to be displayed (`target.html`) in the main Help window. Preserving the question mark (?) between the name of the main Help file name and the target Help topic file name is critical. Your browser uses this question mark to delimit the name of the target Help topic (`target.html`) from the name of the main Help file (`dhsc_help.html`).

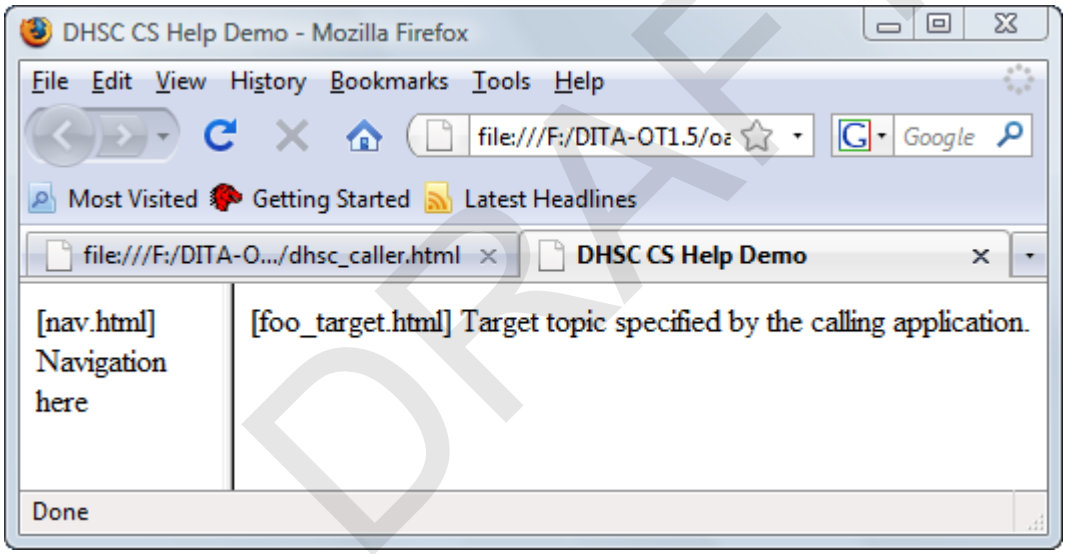
Main Help file	The main Help file here contains little more than a call to a JavaScript library file that perfroms all the real work.
	
	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"`
	`"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">`
	`<html xmlns="http://www.w3.org/1999/xhtml">`
	`<head>`
	`  <title>DHSC CS Help Demo</title>`
	`  <!--Insert cshelp.js library -->`
	`  <script language="Javascript" src="cshelp.js">/script>`
	`  <meta http-equiv="Content-Type" content="text/html;`
	`charset=iso-8859-1">`
	`</head>`
	`<script language="javascript">`
	`insertFrameset()`
	`</script>`
	`</html>`

`cshelp.js`	The `cshelp.js` library contains only one JavaScript function named `insertFrameset`.
	
	`function insertFrameset ()`
	`{`
	`defaultTopic = "default.html";`

DRAFT

Elements	Description

	targetTopic = window.location.search.substring(1);
	if (targetTopic=="")
	{
	target_contentsframe=defaultTopic
	}
	else
	{
	target_contentsframe=targetTopic
	}
	
	document.write('<FRAMESET cols="20%, 80%">');
	document.write('<FRAME src="nav.html" name="navframe">');
	document.write('<FRAME src="'+target_contentsframe+'"
	name="contentframe">');
	document.write('</FRAMESET>');
	}
	
	The variable *targetTopic* points to the name of the target topic (`target.html`) embedded in the hyperlink call. If the JavaScript function receives no argument from the hyperlink that calls it, it uses the name of the HTML topic declared as *defaultTopic*. The final lines of the script build the frameset calls that specify frame dimensions and contents (HTML files).



DRAFT

Summary

These JavaScript functions support small-to-medium Help systems in any language because they place the entire burden of constructing and managing context-sensitive hypertext links on the calling application. With larger Help systems or with distributed Help systems, this overloading becomes risky or impossible.

Stan Doherty

OASIS DITA Help Subcommittee

OASIS DITA Technical Committee

WinANT Options Supporting HTML-Based Output

This topic describes how some features within the WinANT software tool can make generation of HTML-based output from DITA content easier.

The DITA Open Toolkit provides a method for specifying a CSS style sheet, blocks of HTML code to add to the top and bottom of each generated HTML file, and a block of code to add to the <head> section of each generated HTML file. However, the method is cumbersome to use from a command line or terminal window invocation of the Apache Ant build processor. The HyperWrite WinANT software tool, which acts as a Windows interface to Ant, makes this method extremely simple.

Overview

WinANT is a Windows program, build with Microsoft Visual Studio .NET 2003 using VB.NET. It serves as an interface to the Ant build utility, for the sole purpose of processing DITA documents.

WinANT allows a user to select build characteristics using normal Windows interface devices such as drop-down lists, radio buttons, tabs and browse buttons. When all the required settings are in place, the program creates the Ant build file, creates a ditaval file (if required), creates a batch file, and then executes the batch file to trigger the Ant build. When Ant has finished the processing, WinANT displays the generated output file. The settings can be saved (as a build file) and later recalled.

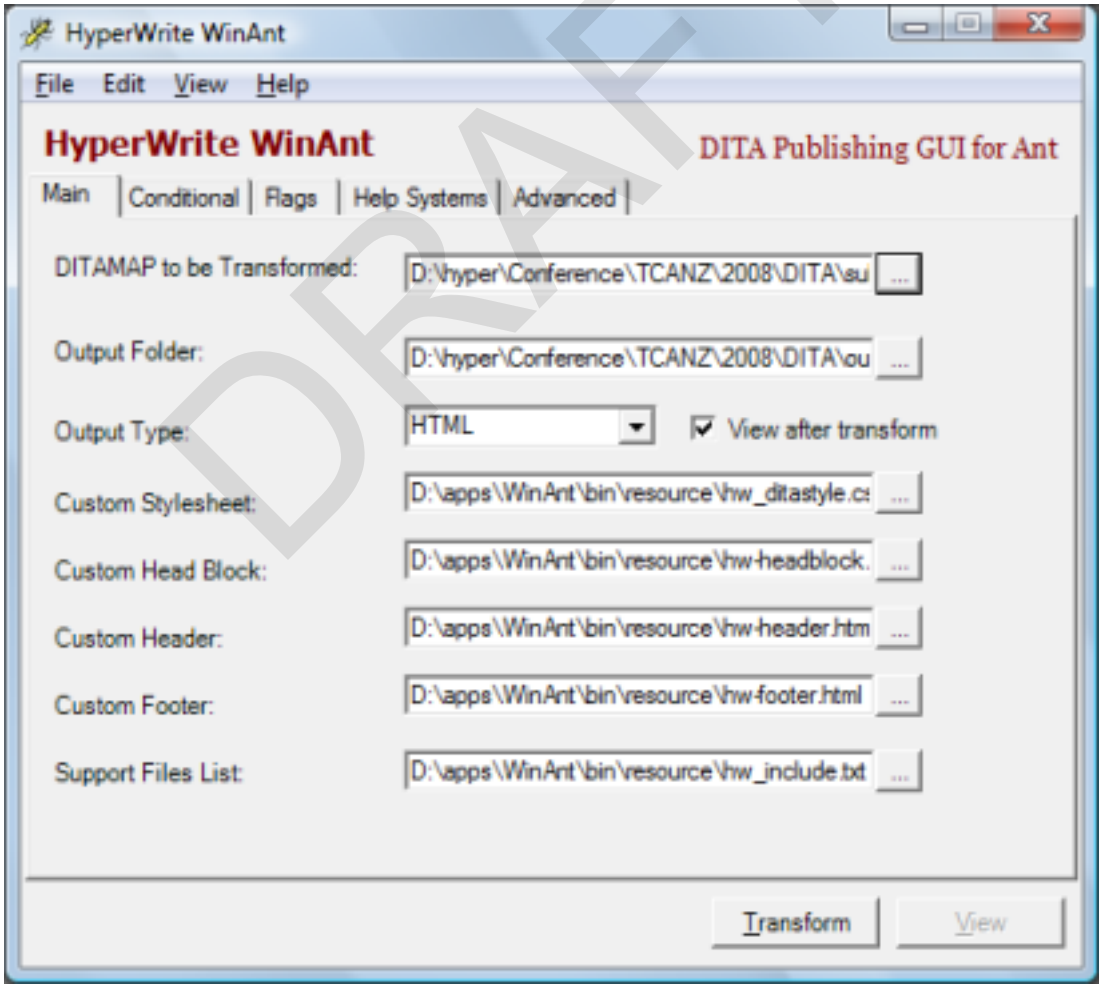


Figure 4: WinANT 1.6 - Main tab

DRAFT

Setup and configuration

WinANT can be downloaded without charge on a "take it as it comes" basis from *http://www.helpml.com/winant_setup.exe*. It installs using a standard Windows installer, and requires *.Net Framework 2.0*.

The Ant *arguments* that can be set within WinANT include:

- The folder where the output will be created.
- The type of output.
- A CSS stylesheet to be applied to each output HTML page.
- A file containing HTML code to be added to the <head> section of each output HTML page.
- A file containing HTML code to be added to the top of the <body> section of each output HTML page.
- A file containing HTML code to be added to the bottom of the <body> section of each output HTML page.
- A list of files to be copied to the output folder (or compiled into HTML Help output).
- Conditional processing rules.
- Images to be used for flagging conditional text.
- The DITA topic file extension used.
- The output HTML file extension to be used.
- Whether content marked as draft will be included in the output.

Authoring

The use of WinANT in the publishing stage does not alter the authoring method.

Publishing

When you are ready to produce HTML-based output from your DITA source, you can process your ditamapfile through WinANT.

WinANT supports the following base DITA Open Toolkit and additional plug-ins:

- HTML
- (Microsoft®) HTML Help
- HTML Help 2 (plug-in)
- PDF
- PDF2
- Eclipse Help
- DocBook
- Word
- HTML - tocjs (plug-in)
- XHTML with Search (plug-in)
- HTML - tocjsbis (plug-in)
- HTML with tocjs
- HTML with search

You will need to prepare the CSS stylesheet to use for presentation of the output, as well as any code blocks for the top (**Custom Header**), bottom (**Custom Footer**), and <head> section (**Custom Head Block**).

👉 **Note:** Make sure your HTML code blocks are well-formed XML. If not, the block will not be included in the output HTML.

If you are producing HTML Help output, you can also nominate an *include file*, which is a simple list of additional files to be compiled into the resultant CHM, in plain text format. If your CSS file references graphics, these graphics files should be listed in the *include file* .

DRAFT

Selecting the CSS and code files

The CSS and HTML code block files are selected on the **Main** tab of WinANT. These fields are:

- Custom Stylesheet
- Custom Head Block
- Custom Header
- Custom Footer
- HTML Help Include File

only active if the **Output Type** field on the **Main** tab is set to *HTML Help* or *HTML Help 2*.

Summary

WinANT provides a simpler way of controlling the HTML-based output from DITA content than the standard DITA Open Toolkit command line. Its ability to store settings for future use also help make it a practical tool for DITA publishing.

WinANT Options Supporting Microsoft® HTML Help

This topic describes how some features within the WinANT software tool can make Microsoft® HTML Help generation from DITA content easier.

The DITA Open Toolkit provides a method for nominating context-sensitive Help *header* (or *map*) and *alias* files to be compiled into the CHM file when a Microsoft® HTML Help output is being generated. The method is difficult to use from a command line or terminal window invocation of the Apache Ant build processor. The HyperWrite WinANT software tool, which acts as a Windows interface to Ant, makes this otherwise difficult method extremely simple.

Overview

WinANT is a Windows program, build with Microsoft Visual Studio .NET 2003 using VB.NET. It serves as an interface to the Ant build utility, for the sole purpose of processing DITA documents.

WinANT allows a user to select build characteristics using normal Windows interface devices such as drop-down lists, radio buttons, tabs and browse buttons. When all the required settings are in place, the program creates the Ant build file, creates a ditaval file (if required), creates a batch file, and then executes the batch file to trigger the Ant build. When Ant has finished the processing, WinANT displays the generated output file. The settings can be saved (as a build file) and later recalled.

DRAFT

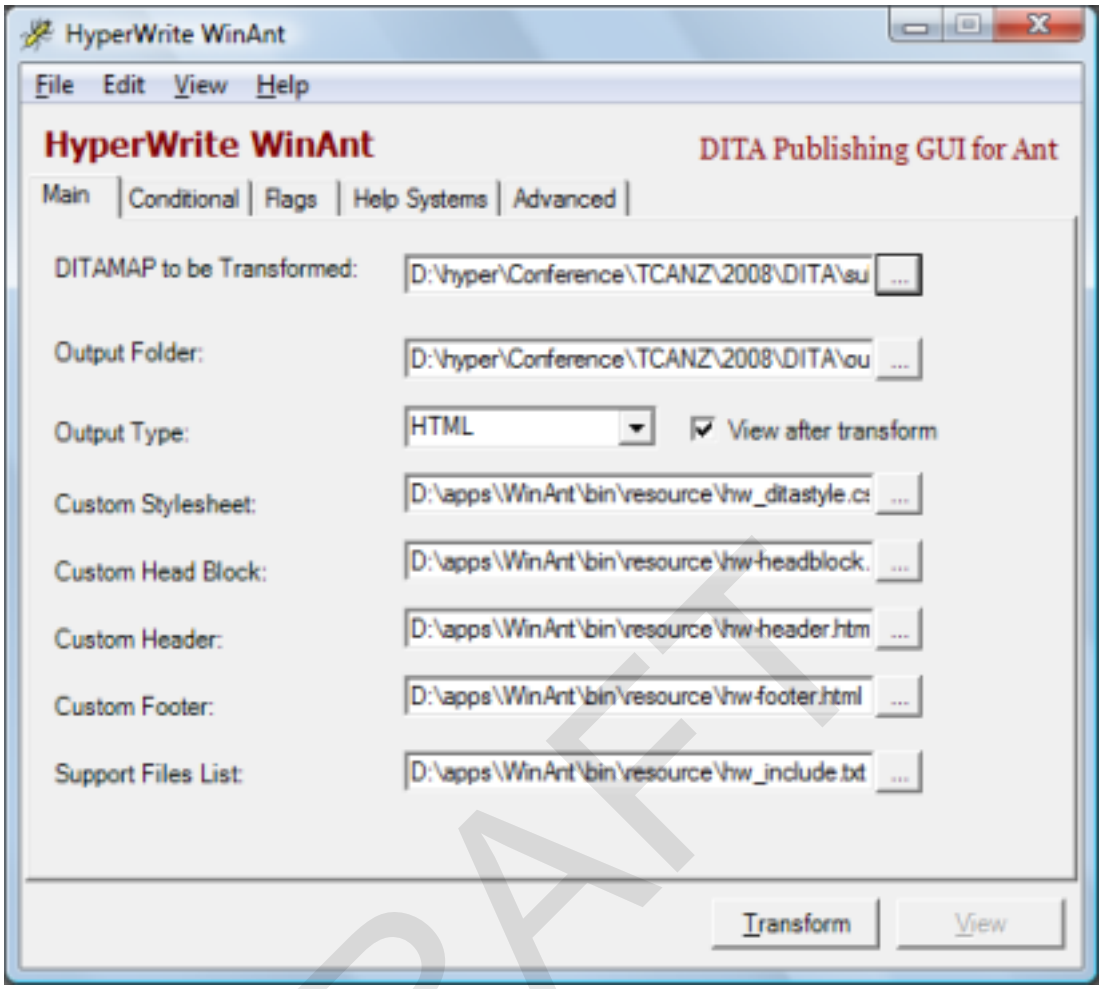


Figure 5: WinANT 1.6 - Main tab

Setup and configuration

WinANT can be downloaded without charge on a "take it as it comes" basis from *http://www.helpml.com/winant_setup.exe*. It installs using a standard Windows installer, and requires *.Net Framework 2.0*.

To enable the incorporation of map and alias files in HTML Help output, you have to manually change the following lines in the standard build_dita2htmlhelp.xml DITA OT file from:

```
<param name="HELPMAP" />
<param name="HELPALIAS" />
```

to

```
<param name="HELPMAP" expression="${dita.map.filename.root}.h" />
<param name="HELPALIAS" expression="${dita.map.filename.root}.ali" />
```

Authoring

Nominating the map and alias files in the HTML Help project file effectively *externalises* the context hooks; the context ids and strings are not incorporated into the DITA source files at all. The map and alias files therefore need to be separately authored outside DITA.

Selecting the *map* and *alias* files

The map and alias files are selected in the **Context-Sensitive Help** panel of the **Help Systems** tab. These fields are only active if the **Output Type** field on the **Main** tab is set to *HTML Help* or *HTML Help 2*.

DRAFT

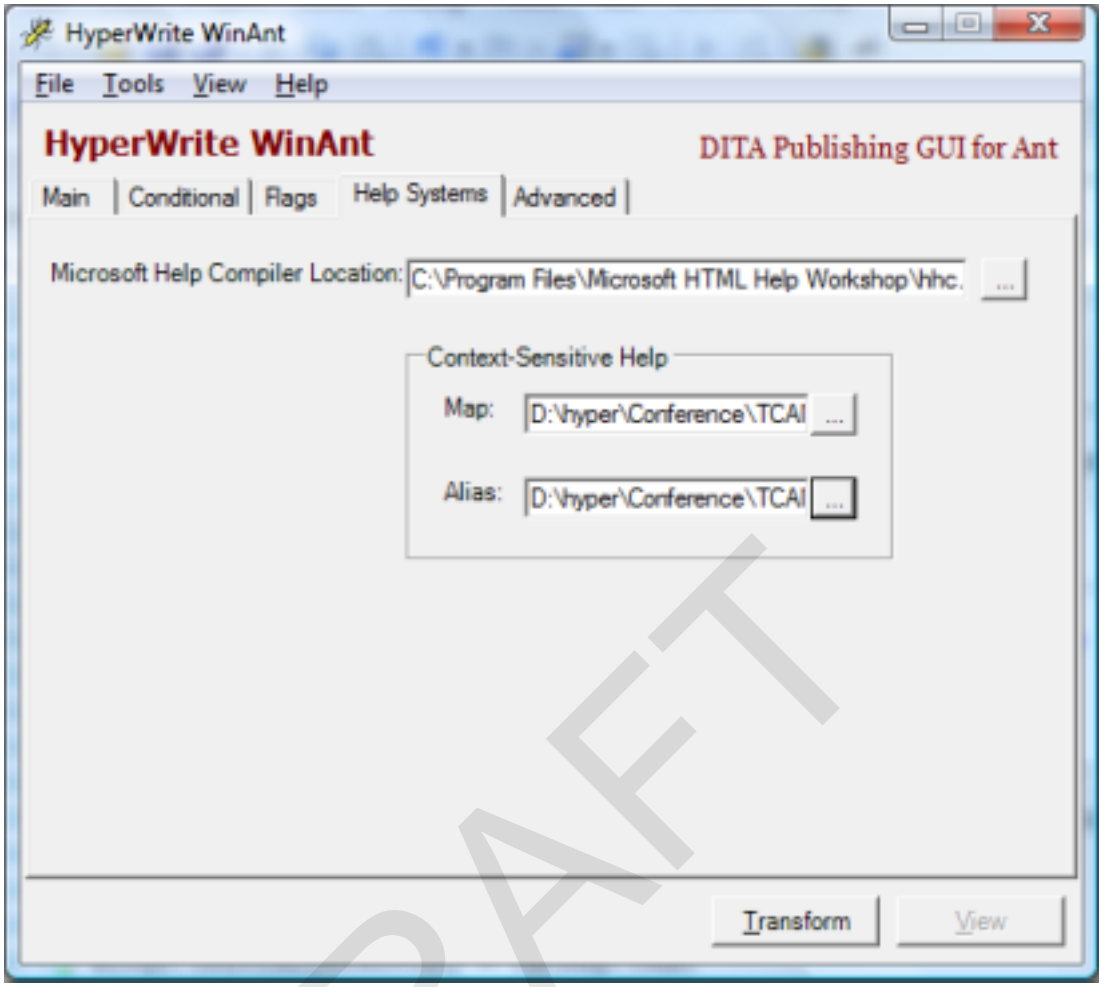


Figure 6: WinANT 1.6 - Help Systems tab

Summary

Provided you have the capacity to create the HTML Help map and alias files outside your DITA authoring environment, this technique is a simple way to produce context-sensitive HTML Help. It only requires very minor (and simple) changes to the standard DITA Open Toolkit files.

It is possible that this technique can be combined and integrated with other methods of generating map and alias files from DITA source.

DRAFT

Developing DITA-based Help for Existing Help Authoring Tools

This topic introduces solutions for integrating DITA source (or in some cases DITA output) with existing HATs (Help Authoring Tools).

If your company develops content in multiple authoring tools and uses a HAT (Help Authoring Tool) to integrate those sources into one or more Help delievrables, you can still author in DITA and work with existing HATs.

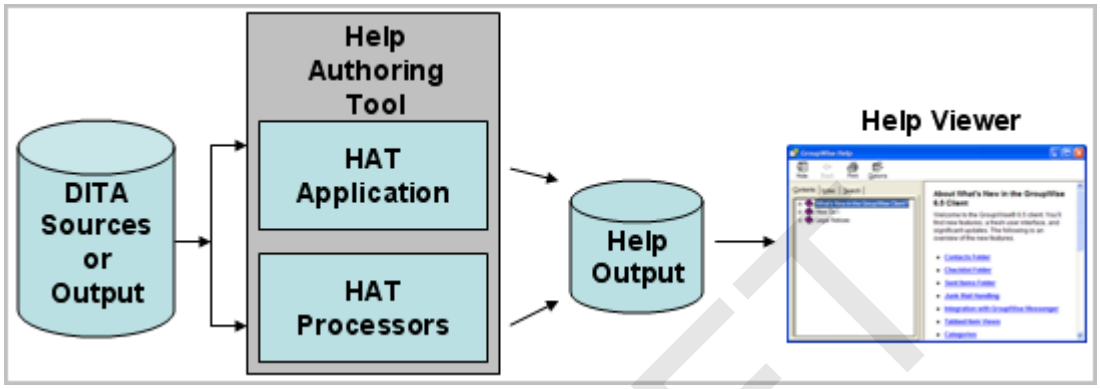


Figure 7: DITA Source to Existing HATs

The following topics in this section of the *Best Practices Guide* explain how to use integrate DITA source files or DITA output files with existing HATs.

- *Converting DITA Content to WebHelp using RoboHelp* on page 50

Converting DITA Content to *WebHelp* using RoboHelp®

This topic describes how to convert DITA content described by a ditamap into RoboHelp's *WebHelp* format.

The DITA Open Toolkit provides only rudimentary support for tri-pane, HTML-based documents. Help Authoring Tools such as RoboHelp have long set the benchmark for feature-rich, HTML-based documents. Adobe® describes the RoboHelp tri-pane HTML output as *WebHelp*.

1. **Generate HTML Help output from your DITA content using the DITA Open Toolkit.**
2. **Start RoboHelp.**
3. **From the File menu, choose New Project.**
   The **New Project** window displays.
4. **Select the Import tab, select Microsoft HTML Help Project (*.hhp), and click OK.**
   The **Select HTML Help Project** dialog box displays.
5. **Browse to the directory into which the DITA Open Toolkit generated your DITA document's output, select the *project (.hhp)* file, and click Open.**
   The HTML Help project files will be imported into a new RoboHelp project.
6. **Generate the WebHelp output.**

RoboHelp will generate the WebHelp output into the `/!SSL!/WebHelp` sub-directory of the project path. The default file name to launch the WebHelp document takes its name from the project file name.

DRAFT