

DHSC Best Practices Guide

Contents

- DITA and User Assistance.....3**
 - Definition of DITA Help.....3
- Developing DITA-based Help for Existing Help Environments.....4**
 - CSHelp Plug-in.....4
 - LMI AIR Help Plugin.....7
- Developing Custom DITA-based Help Systems.....10**
 - TOCJS and TOCJSBIS Plug-ins.....10
 - DHTML Effects in HTML Generated from DITA.....16
- [INTERNAL] DHSC Best Practices Proposal.....19**

DITA and User Assistance

This topic describes the relationship between DITA and user assistance and 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 user assistance format 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 inhouse technical expertise to do so.

Moving beyond simple Help systems, however, is currently difficult, but not impossible. The DITA Technical Committee is developing some changes 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.

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, window definitions. The 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

CSHelp Plug-in

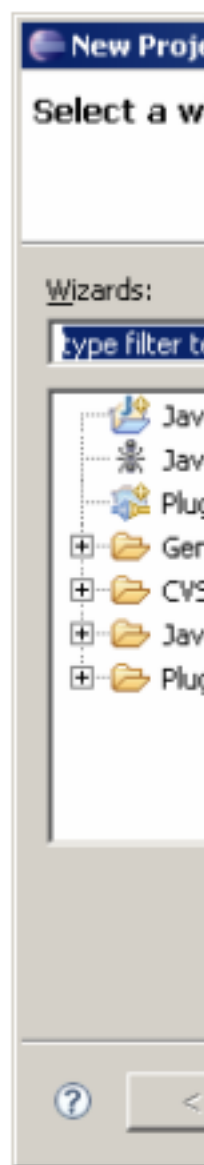
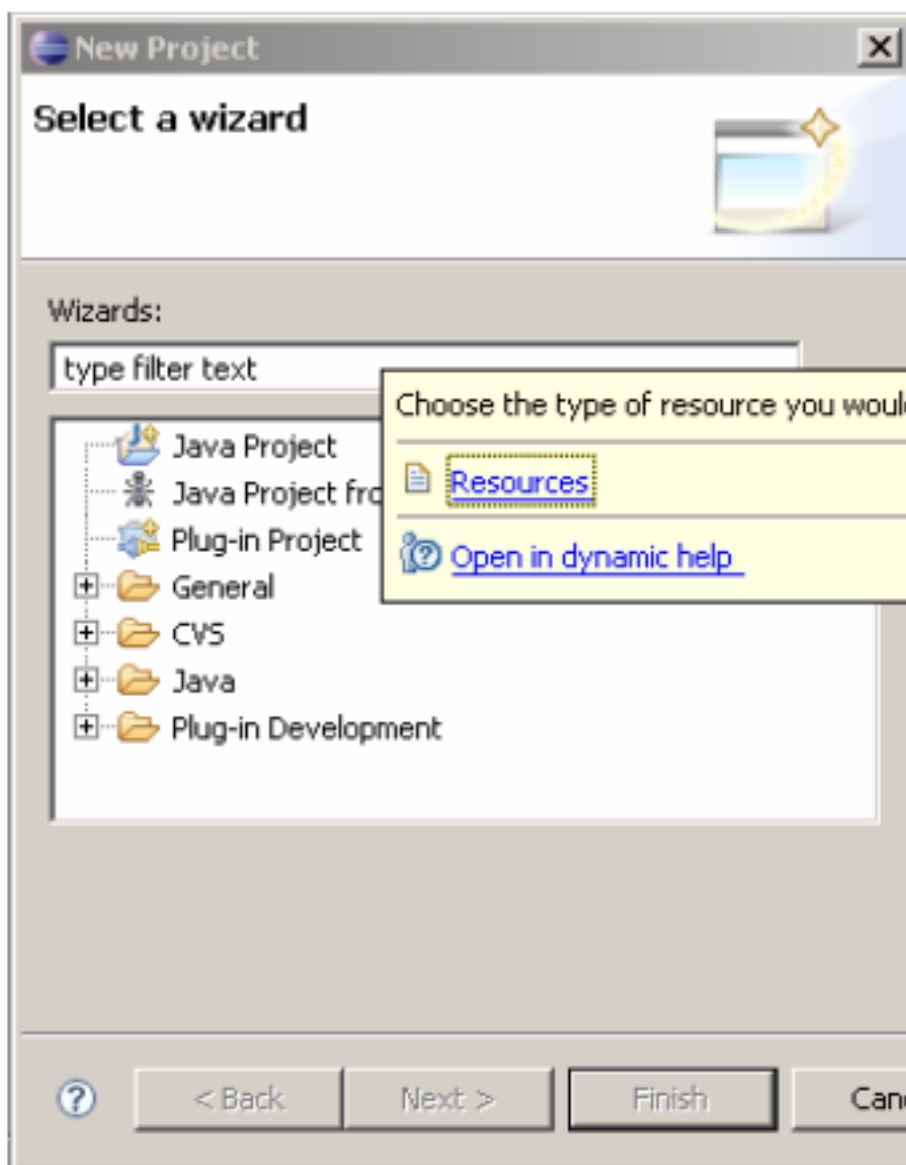
This topic provides an overview of the DITA-OT plug-in named cshelp (for context-sensitive help).

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. If you already use DITA to source your Eclipse documentation plug-ins, sourcing the content-sensitive help in DITA allows you to maintain a consistent authoring and build environment and consistently formatted output.

The Eclipse user interface allows you to display context-sensitive help as an infopop (left) or in a 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 plugin 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.

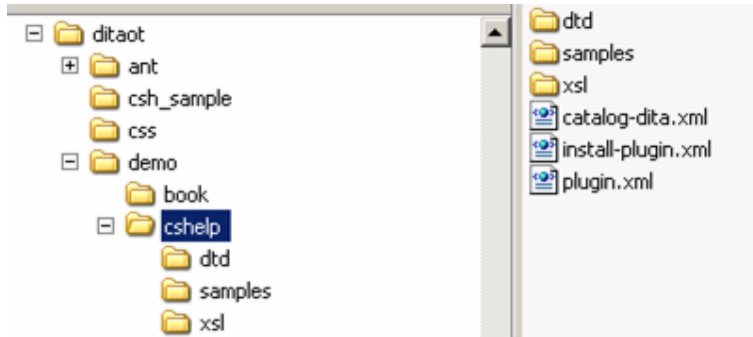
Setup and configuration

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	apiref0.8	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).



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 () tag, and the only formatting options are the carriage return and the space key. 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 simplettable
- 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 DITAMAP file inside the plug-in that points to each of them. Include any copyright information in the DITAMAP; 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 plugins. 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 DITAMAP 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 file:

```
<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

LMI AIR Help Plugin

This topic provides information about generating AIR Help from DITA.

The DITA-OT AIR Help plugin is not yet publicly available.

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 this an ideal technology from which to develop an online Help deliverable.

Currently the only commercially available product for generating AIR Help is Adobe's "RoboHelp Packager for AIR." This (beta) product 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.

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 LMI AIR Help plugin 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 Help interface to add features or modify the appearance as needed.

Setup and configuration

<<WHERE TO GET THE PLUGIN .. SETUP REQUIREMENTS>>

For the most basic output, no extra effort is needed. Run your DITA map through the plugin, 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

<<AUTHORING NEEDS .. ??>>

Integration

<<HOOKING UP FOR CONTEXT HELP>>

Output

<<SAMPLES>>

Summary

<<MORE INFO>>

Developing Custom DITA-based Help Systems

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.

The screenshot displays the user interface of the `tocjsbis` plug-in. On the left, a tree view shows the hierarchy of DITA elements, with 'DITAVAL elements' expanded and 'alt-text' selected. On the right, the detailed view for 'alt-text' is shown, including its definition, containment information, and an example reference.

alt-text

An element allowed inside either `startflag` or `endflag` to provide alternate text for an image, when the `imageref` attribute sets an image to be used for flagging. The default alternate text for `revprop` start of change is a localized translation of "Start of change". The default alternate text for `revprop` end of change is a localized translation of "End of change".

Contains

text data

Contained by

[startflag](#), [endflag](#)

Example

See the example in the [<val>](#) description.

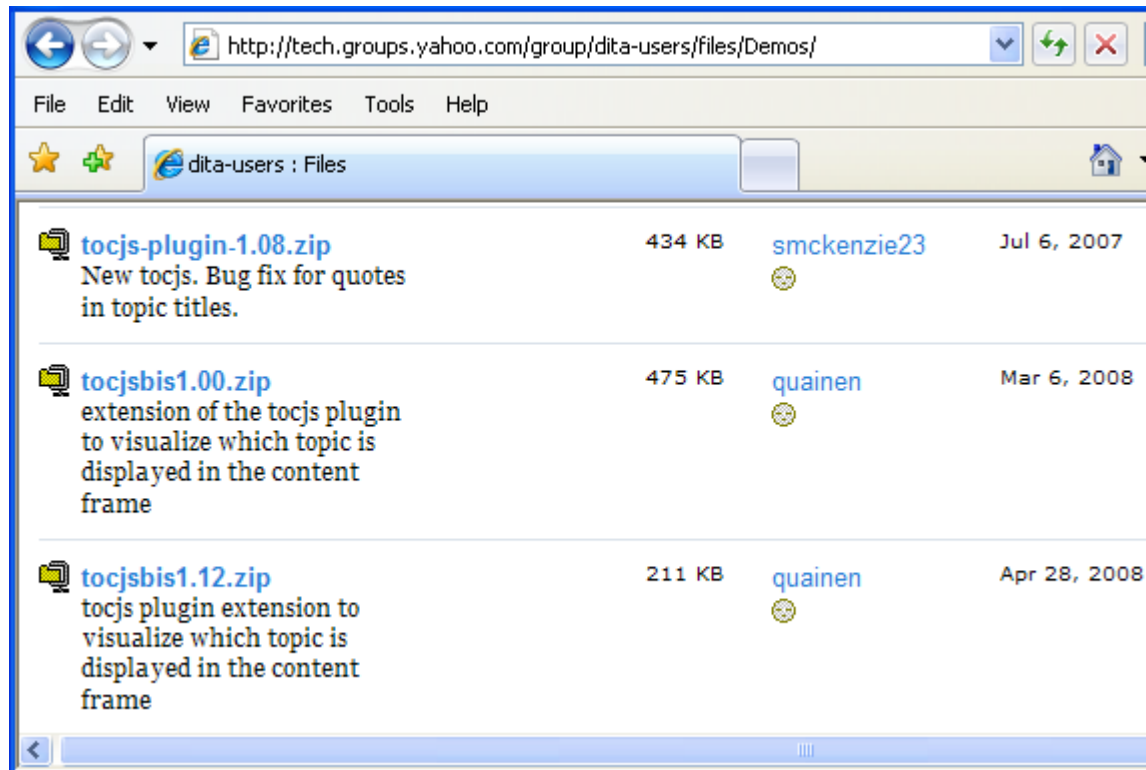
- *tocjs*: The `tocjs` plug-in was developed by Shawn McKenzie, currently working at Sophos in beautiful Vancouver, BC (Canada). The `tocjs` plug-in executes after the standard DITA-OT XHTML transform, so each `tocjs` TOC entry knows the name of its target XHTML topic.
- *tocjsbis*: The `tocjsbis` plug-in 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 synchronizing 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.

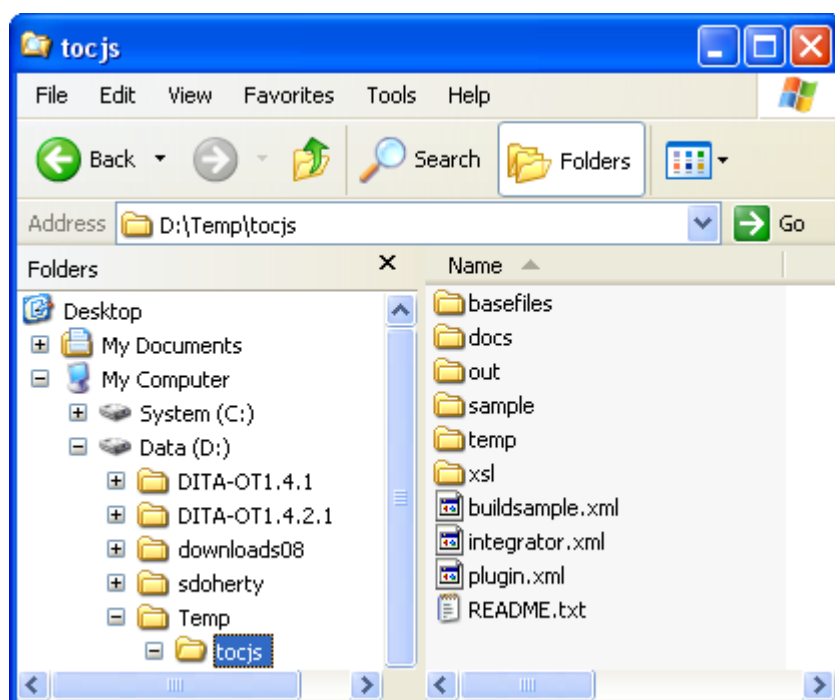
Setup and configuration

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

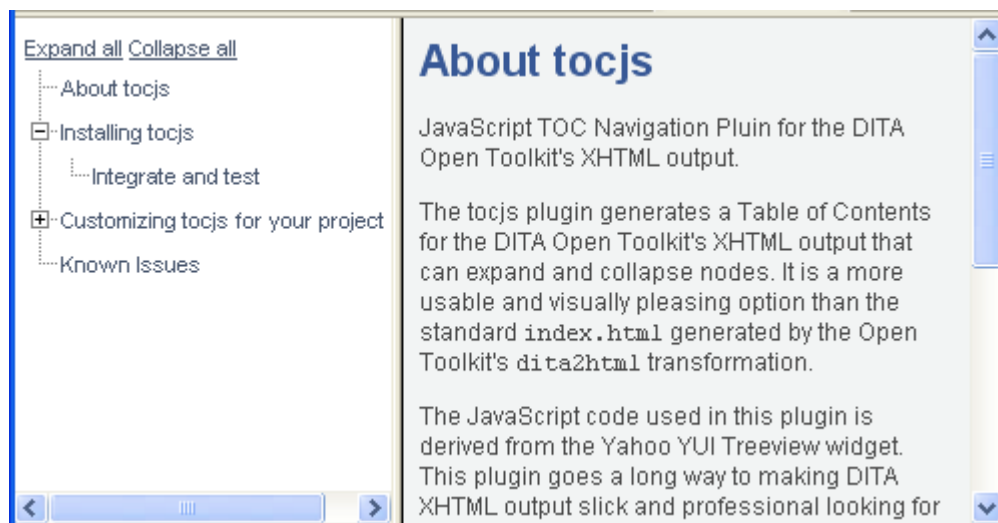
<http://tech.groups.yahoo.com/group/dita-users/files/Demos/>



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.

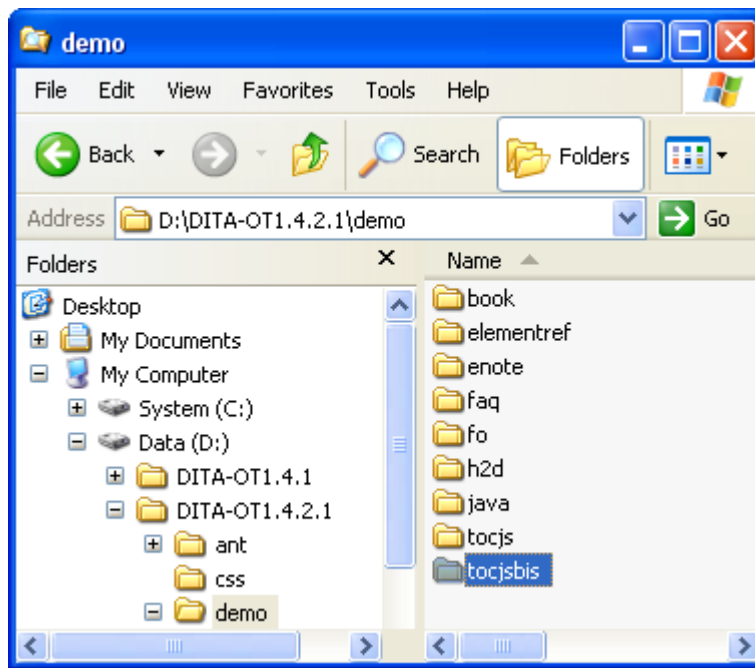


The pre-built documentation for tocjs lives in the /docs subdirectory.



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/>

Yahoo! UI Library: TreeView

The YUI TreeView Control provides a rich, compact visual presentation of hierarchical node data. Support is provided for several common node types, for the association of custom metadata with each node, and for the dynamic loading of node data (via in-page data or via XMLHttpRequest using [Connection Manager](#)) to navigate large datasets with a small initial payload.



On This Page:

- [Getting Started](#)
- [Using TreeView](#)
- [Known Issues](#)
- [YUI on Mobile Devices](#)
- [Support & Community](#)
- [Filing Bugs and Feature Requests](#)

Quick Links:

- [Examples](#): Explore examples of the TreeView Control in action.
- [API Documentation](#): View the full API documentation for the TreeView Control.
- [Release Notes](#): Detailed change log for the TreeView Control.
- [License](#): The YUI Library is issued under a BSD license.
- [Download](#): Download the TreeView Control as part of the full YUI Library on SourceForge.

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 opportunities, 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.

SOPHOS WEB APPLIANCE HELP

About Your WS1000

Organizations typically expend considerable resources and effort preventing virus, worm and trojan infections from entering their networks via email. These same threats, as well as spyware, adware and phishing scams are increasingly infiltrating organizations' networks via web browsing.

Inappropriate web browsing by employees is also a significant legal liability and productivity concern for many organizations. The Sophos Web Appliance provides extensive URL categorization data that allows you to set acceptable web access policies for

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

Contents Search Support

Expand all Collapse all

- About tocjs
- Installing tocjs
- Customizing tocjs for your project
- Known Issues

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/>

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

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, dropdown text and popup text, are implemented using Dynamic HTML.

Overview

The DITA Open Toolkit HTML transformations 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 the technique, a note element is to output as dropdown 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 the technique, the following changes to the way the XSL-T file processes the note element are made:

```
<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;
  }
  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 effects in output, the techniques are not onerously complex. With a small investment in effort, a big payoff in functionality can be achieved.

[INTERNAL] DHSC Best Practices Proposal

This document describes the general contents for the proposed *Best Practices Guide for Authoring Help in DITA*.

Overview

This document follows up on the proposal for a best practices document approved by the DITA Help Subcommittee (hereafter DHSC) on June 5, 2008 and updated on June 26. The general notion here is that our subcommittee has devoted time to reviewing the current options for developing Help in DITA. Until we can develop Help specializations for DITA 1.3, these options are the only things that DITA-based organizations can implement, short of developing something new from scratch. We would be doing the DITA community in general a service by summarizing our findings and providing links to relevant Help-oriented resources.

Proposed Contents and Contributors

I propose that we keep things pretty simple. If we can develop one DITA topic per content area, I can pull these together into something buildable and reviewable pretty quickly. Following the general shape of the demos that we had over the past few months, here's what I would propose for a starting outline.

Section	Topic	Contributor
Introduction	Introduction	Tony Self (HyperWrite) Stan Doherty (Individual)
Developing DITA-based Help for Existing Help Environments	Introduction	Volunteer??? Stan Doherty (Individual)?
-	EclipseHelp	Jeff Antley (IBM)
-	CSHelp Plug-in	Jeff Antley (IBM)
-	AIRHelp	Scott Prentice (Individual)
-	PTC Help	Chris Goolsby (PTC) Dan Cunningham (PTC)?
-	HTMLHelp	Deborah Pickett (Moldflow)
-	WinANT outputs and optins	Tony Self (HyperWrite)
Developing Custom DITA-based Help Systems	Introduction Need matrix of options/choices.	Stan Doherty (Individual)
-	WinANT context ID management	Tony Self (HyperWrite)
-	TOCJS plugin	Stan Doherty (Individual)
-	HTMLSearch plugin	Stan Doherty (Individual)

Section	Topic	Contributor
-	XHTML (DITA-OT)	Stan Doherty (Individual)
-	CSHelp Plug-in	Jeff Antley (IBM)
-	Dynamically-rendered HTML	Tony Self (HyperWrite)
-	VistaHelp-style expandable content	Tony Self (HyperWrite)
Using DITA Source with Existing HATs	Introduction	TBD???
-	MadCap Flare	TBD???
-	RoboHelp	TBD???
Resources and Demos	Resources	Joe Welinske
-	Demos (links)	Joe Welinske
Futures?	Introduction	Tony Self (HyperWrite) ?

Topic Structure

At the risk of sounding "pushy," I'd recommend that we start with a generic DITA topic that has a common set of headings. If writers need to rearrange them or delete them, fine ... but at least we'd be starting from a common set of items to discuss. Obviously, where there is existing documentation for a plugin or tool, we point off to it ... where there is little or no documentation, we may need to do a little writing.

Here's my take on some common <section> headings to start with.

- *Overview*: discussion of what the plug-in or tool does (features) and why someone might want to use it
- *Setup and configuration*: where to download or purchase it, how to install (high-level) it, and how to configure it (high-level)
- *Authoring*: what to do (if anything) in DITA source files to take advantage of the plugin or tool
- *Integration*: what does someone do to integrate this plugin or tool with an engineering environment (context IDs at a minimum)
- *Output*: screen shots or links to live output
- *Summary*: final comments, evaluations, recommendations

Stan Doherty