Migrating HTML to DITA, Part 1: Simple steps to move from HTML to DITA

Get a quick start with DITA using available HTML topics

Robert Anderson (robander@us.ibm.com)
Developer, Information Development Workbench
IBM

Don Day (dond@us.ibm.com)
Lead DITA Architect
IBM

Erik Hennum (ehennum@us.ibm.com)
Information Architect
IBM

31 January 2005

The Darwin Information Typing Architecture (DITA) has emerged as a standard topic-oriented document architecture. DITA holds many advantages over information authored directly in HTML, including better reuse, easily changed presentation styles, and easy single sourcing. This article, the first of two parts, explains how to get a quick start with DITA using HTML topics that are already available. It shows you how to use the provided XSLT transform to do the migration, and examines what is needed to ensure quality results.

With the advent of the Web, IBM created a large body of documentation in HTML pages. To move this documentation to a more reusable, topic-oriented architecture, many groups within IBM had to migrate much of the information to DITA.

The migration process, however, posed a challenge. The legacy content was produced using many tools, each with different HTML styles, which made it difficult to use a common migration process. Unless you use an extremely strict HTML authoring template, there will often be something that confuses a simple migration utility.

This article is the first of a two-part series on how we met these challenges. Part 1 focuses on the migration itself; when it works best and what you need to do to use it. We also share some of the techniques that we have learned during our own migration efforts.
The code provided here is used within IBM to convert HTML articles to DITA. Although many authors have used this migration tool without changes, we know that this is not always possible. We distribute this to you knowing that it is a generic transform and that it cannot handle every HTML article. In many cases, articles already contain specific markup that can easily match DITA structures. In this case, you are advised to override parts of the migration to get a more effective migration and to save any existing information architecture. Part 2 will give details on how to override the migration tool to get results specific to your information.

If you have simple HTML topics, do not want to get your hands dirty with XSLT, or already expect to do cleanup on the DITA output, this article should be all you need. The tool provided is enough to get you well on your way to the total DITA experience.

If you want to play with the XSLT in order to minimize manual cleanup, you should treat this article as an introduction. You still need to know what is required prior to migration and what to expect before you add your extensions. Then, you can move on to the second article for information on how the XSLT is organized, as well as a sample of how to easily override the default migrations.

If you are unfamiliar with DITA, the following two articles are a good place to start:

- "Introduction to DITA"
- "Why use DITA to produce HTML deliverables?"

When are articles good candidates for migration?

Not every HTML topic can or should be migrated to DITA. Several examples of topics that should not be converted are given in the article "Why use DITA to produce HTML deliverables?" These topics include special-purpose HTML-based UI, articles created for appearance instead of content, and one-of-a-kind articles that do not need to change or be reused. Some other topics should be converted, but cannot be migrated well with an automated tool. These topics are typically written with an old HTML authoring tool, or make heavy use of tables, lists, and indented quotes solely to aid in presentation.

The DITA architecture is focused on topics, relatively small pieces of information that can fit in a browser window with minimal scrolling. Any migration to DITA works better for information that is already stored in this manner.

The best candidates for conversion are topics that use well-structured HTML, or those using a standard authoring template.

Analyze your content

Is your content already divided among concepts, tasks, and reference articles? If not, you might need to rework your articles before migrating. DITA articles are usually authored using these three information types, or using a more generic topic type for information that does not fit any of the main categories. If possible, you should try to migrate directly into one of the three specific types; otherwise, you will likely end up needing another migration down the road.
The migration technique discussed here involves the use of some commonly available tools, such as HTML Tidy and any version of XSLT. We will explore premigration cleanup (such as global edits or HTML Tidy), up-translation using XSLT migration scripts/transforms, and post-migration validation and touch-ups.

The basic migration script discussed in this article targets one information type at a time. If you can divide your information into directories to represent the three types, it's easier to convert each as a group. We use a batch utility that converts every HTML file in a given directory using the same information type. Having each type in a distinct directory makes it very easy to convert all topics using only a few commands. Of course, you can also write a script that selectively chooses the best way to migrate each file in a mixed directory of HTML topics.

**Items to clean up**

This migration utility only works with valid XHTML files, so most HTML files must be cleaned up before processing. The best way to do this is to use the Tidy command (see Resources). When using Tidy, you should have it leave off the document type so that the migration utility does not attempt to validate each of your files.

The transform drops the presentation elements `<br/>` and `<hr/>` because DITA has no matching tags. If you use these elements strategically to help convey something to the reader, you might want to look for some other way to present your information. One method might be to use preformatted text instead of `<br/>`, or to use tables instead of separating sections with `<hr/>`. This might also be a sign that the topic generally needs to be revised if you are going to move it to DITA.

Aside from the necessary conversion from HTML to XHTML with Tidy, you also need to determine whether it's easier to do other cleanup before or after the migration. The best way to do this is to try converting a few sample files first. If the HTML contains regular structures that can be easily modified, such as `<div>` elements that serve no real purpose, you might find it easier to update them before the conversion. However, the migration utility emits messages when it encounters problems, which typically makes it easy to find the problems after the migration. With post-migration updates, you can also use a validating editor to ensure that your cleanup is effective.

**Other general migration notes**

Here are some other things to keep in mind as you proceed:

- As written, the provided migration utility operates on a single file. It takes in the desired information type as a parameter (infotype), and creates the topic accordingly. Values for the infotype parameter are either topic (the default), task, concept, or reference; the output is always a single value in a single file.
- If your HTML content already has a topic architecture, you can conserve the existing architecture by migrating HTML pages into specific topic types. Using more specific types has many advantages; for example, links are sorted by type in the output. This also makes it easier to organize your presentation, and makes it easier to find information when you know
what you are looking for. Here at IBM, users who convert straight to the topic often find that they want more specific topics down the road, resulting in another migration step.

- If the migration is done with XSLT 1.0, the doctype cannot be included in the output. You need the doctype to continue using the topics after they are in DITA. In this case, you have several options:
  - The doctype can be added manually with a batch processor or with an editor that supports search-and-replace across multiple files.
  - You can create a wrapper XSLT shell that sets the proper doctype, and imports the migration. This requires that you call a different shell for each information type.
  - Some processors allow you to get around this problem by setting the doctype with a variable. For example, Saxon allows you to specify XSLT 1.1, and then set the doctype dynamically.

This utility assumes that you will set the doctype on your own; however, it contains code that takes advantage of Saxon's XSLT 1.1 processing (this might work with other processors as well). If you want to set the doctype dynamically using Saxon's 1.1 level processing, you can do so very easily, even if you know nothing about XSLT. The provided program opens with the code shown in Listing 1:

**Listing 1. Beginning code**
```xml
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="no" encoding="utf-8"/>
<xsl:param name="infotype">topic</xsl:param>

To switch over to the dynamic version, you must place comment markers around this code (just put <!-- before it and --> after it). Next, you must remove the same comment markers from the code shown in Listing 2.

**Listing 2. XSLT code for creating doctypes dynamically**
```xml
<!--<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:saxon=http://icl.com/saxon
extension-element-prefixes="saxon">
<xsl:param name="infotype">topic</xsl:param>
<xsl:param name="systemid">
<xsl:choose>
<xsl:when test="$infotype='concept'">../dtd/concept.dtd</xsl:when>
<xsl:when test="$infotype='task'">../dtd/task.dtd</xsl:when>
<xsl:when test="$infotype='reference'">../dtd/reference.dtd</xsl:when>
<xsl:otherwise>../dtd/topic.dtd</xsl:otherwise>
</xsl:choose>
</xsl:param>
<xsl:param name="publicid">
<xsl:choose>
<xsl:when test="$infotype='concept'">-//IBM//DTD DITA Concept//EN</xsl:when>
<xsl:when test="$infotype='task'">-//IBM//DTD DITA Task//EN</xsl:when>
<xsl:when test="$infotype='reference'">-//IBM//DTD DITA Reference//EN</xsl:when>
<xsl:otherwise>-//IBM//DTD DITA Topic//EN</xsl:otherwise>
</xsl:choose>
</xsl:param>
<xsl:output method="xml" indent="no" encoding="utf-8"
document-system="{$systemid}" doctype-public="{$publicid}"/>
```
• When you convert links, the extensions for any local HTML topics are changed to ".dita". Links to external or non-HTML resources remain unchanged. If you do not expect to use the ".dita" extension, this can be changed with the dita-extension parameter. Simply pass in your preferred value (such as dita-extension=xml), or change the value in the source code.
• An ID attribute on a DITA topic is required. The migration utility determines the ID by examining the name of the HTML file (which should be passed in as a parameter). If the HTML file name is not passed in, a generated ID is used. If you want to use another method for creating the topic ID, such as the topic title or a phrase near the start of each topic, you should update the genidattribute template or change the default for the filename-id variable.
• In general, the migration tool makes a best guess as to how to migrate any given element. However, in some cases the existing structure of the data does not map reliably to a proper DITA element. Any time the utility cannot determine the correct way to migrate, it places the contents in a <required-cleanup> element, and the tool generates a message that the output must be fixed.

What does this migration not do?

The most important thing to know when using this migration is that it is intended for topics. An HTML page shouldn't contain the equivalent of multiple book sections, but should instead contain a single section without any nested sections. As stated above, the DITA architecture is focused on topics; information that is written for books needs to be redesigned in order to fit into a topic-based architecture. You can also use DITA maps after migration to reconstitute hierarchies by defining nesting for the converted topics.

It is also important to remember that this really is a migration; it is not very useful as part of a processing pipeline. If you do not start with topic-oriented information, you cannot create real topics just by going through DITA. You also need to do at least minimal cleanup after your HTML topics are migrated to DITA. While it would be possible for you to drastically modify the XSLT in order to remove the cleanup, this just gives you problems the next time one of your authors inserts a <br/> tag where you do not expect it. In the long run, it is far easier to migrate the files, clean up the DITA content, and then control any output needs with CSS.

Allowances for special HTML practices

As you are probably aware, you can do all kinds of things with HTML to adjust your presentation, which sometimes has the side effect of misrepresenting your content. For example, a block quote could actually be a block quote, or it could just be a paragraph that somebody wants to indent. Well-structured HTML avoids the <blockquote> element in favor of CSS-based presentation. This is the same approach that DITA takes. So, when migrating to DITA, it is often necessary to fix the markup at the same time.

On several projects, we had users with a large number of topics that were authored in RoboHelp. Many of these files did not convert well because of the method by which the ordered lists were saved. Each item in the list was placed in its own ordered list, and numbering was preserved by using the start attribute on each list. For example, the fifth item appeared in the fifth list as <ol
For lists that needed extra space, an empty paragraph was introduced between each list item.

In DITA it is much better to store this as a single list, and then control the spacing of your output with CSS. This is especially important when working with tasks, which only allow a single `<steps>` element; you cannot separate each item into its own `<steps>` element.

We take care of this by checking the topic to see if it contains any lists with a start attribute. If so, we run an initial pass over the topic that copies the entire tree, while merging any lists that should be together. If you want to see how this is done in the provided XSLT, look for the "shift-lists" mode. The result tree is placed in a variable, and the contents of that variable are then processed as if they were the original file.

This method of processing a variable can cause problems with some XSLT processors (the Saxon processor mentioned above handles it without any problems). If this is the case with your processor, you can send the merged XHTML to a temporary file, and then process that instead. You can also set up an actual two-pass model and run the transforms in sequence, or just ignore the problem and merge your lists manually after the migration. Note that this is only an issue if you actually have lists that use the start attribute; if this attribute is not used, the merging process is skipped.

The process described here was originally written as an override for RoboHelp users. However, anybody who has this markup would want the override, so we folded it back into the original migration.

### Running the migration

If you don't want to change the XSLT, you're ready to go; all you need to do is download the XSLT and place it somewhere on your system. You can use any XSLT processor to try converting a topic.

Before you run the XSLT itself, you need to ensure that the files are valid XHTML. One easy way to do this is with the `TIDY` command (see Resources). We use several options with Tidy to make sure that the XHTML is as clean as possible:

<table>
<thead>
<tr>
<th>Tidy option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c</code></td>
<td>Replaces <code>FONT</code>, <code>NOBR</code>, and <code>Center</code> tags with references to CSS.</td>
</tr>
<tr>
<td><code>-n</code></td>
<td>When possible, entities use numeric values rather than named values.</td>
</tr>
<tr>
<td><code>-m</code></td>
<td>Causes the original file to be replaced with the updated file. If you use this option, be sure you have a backup of your original HTML file.</td>
</tr>
<tr>
<td><code>--output-xml yes</code></td>
<td>Ensures that the output is valid XML.</td>
</tr>
<tr>
<td><code>--doctype omit</code></td>
<td>Prevents any doctype from appearing in the output. When the output is XHTML, this prevents your XSL parser from attempting to validate each XHTML topic before converting it.</td>
</tr>
</tbody>
</table>

So a sample call to Tidy might look like this:
tidy -c -n -m --output-xml yes --doctype omit input.htm > output.htm

Again, you should remember to back up your HTML files before using this call. A batch utility might be helpful at this point; at IBM, we have a simple batch script that creates a backup of every HTML file in the directory, and then updates the files.

Now that you have valid XHTML topics, you can move on to the XSLT calls that migrate your topic.

- The call in Listing 3 creates a task, using a generated ID for the task's ID and ".dita" as the default extension:

  Listing 3. Sample calls to the migration, using Saxon and Xalan

  java com.icl.saxon.StyleSheet mytask.htm h2d.xsl infotype=task > mytask.dita
  java org.apache.xalan.xslt.Process -in mytask.htm -xsl h2d.xsl
      -out mytask.dita -param infotype task

- The call in Listing 4 creates a concept, using the file's name as the ID and ".xml" as the default extension:

  Listing 4. More complex calls to the migration, with additional parameters

  java com.icl.saxon.StyleSheet myconcept.html h2d.xsl infotype=concept
      FILENAME=myconcept.html dita-extension=xml > myconcept.xml
  java org.apache.xalan.xslt.Process -in myconcept.html -xsl h2d.xsl -out myconcept.xml
      -param infotype task -param filename myconcept.html
      -param dita-extension xml

If you have a large group of topics to convert, it is a good idea to develop a batch file to do this. Our batch file converts all of the topics in a directory using the same information type, which means that we have to separate each type prior to migration. You might find that this is the easiest path; alternatively, you could create a batch file that queries you for the information type prior to migrating each topic.

**Summary**

You have a lot to gain by moving your topics from HTML to DITA. To start with, you can more easily reuse topics or parts of topics, you can produce multiple output formats, and you can rapidly change the presentation for an entire deliverable just by switching your CSS file. If you are not convinced, take a look at the article "Why use DITA to produce HTML deliverables?"

If you already have well-structured topics, take a few of them and try using the provided conversion tool. You might find that, with very little manual intervention, you can move completely to a DITA-sourced environment. Even with complicated HTML, you might be able to clean up the output without too much work. Of course, if you like playing around with XSLT, be sure to read Part 2 of this article for hints and tips on how to do it.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT migration script and samples</td>
<td>x-dita8asource.zip</td>
<td>21 KB</td>
</tr>
</tbody>
</table>
Resources

- Read Migrating HTML to DITA, Part 2, which provides more detailed information about the migration (and how to override it) (developerWorks, February 2005).
- Learn more about DITA in these developerWorks articles:
  - "Introduction to DITA" (October 2003)
  - "Specializing topic types in DITA" (October 2003)
  - The DITA FAQ (November 2004)
  - "Specializing domains in DITA" (October 2003)
  - "Why use DITA to produce HTML deliverables?" (October 2003)
  - "Design patterns for information architecture with DITA map domains" (September 2004)
- Find additional information on DITA at the OASIS Cover Pages.
- Here are some tools that can be useful to you when doing the migration:
  - HTML Tidy, available from SourceForge.net. A Java version is also available on SourceForge at the JTidy page.
  - The Saxon XSLT engine, also from SourceForge.net.
  - The Xalan XSLT engine, from Apache.
  - As well as these additional XSLT links.
About the authors

Robert Anderson

Robert D. Anderson has worked on IBM's Internal Information Development Workbench since 1999, writing and maintaining transformation tools for both SGML (IBMIDDoc) and XML (DITA). Since 2001 he has worked primarily on conversion tools to and from DITA, as well as general DITA design (prior to its shift to OASIS). You can reach him at robander@us.ibm.com.

Don Day

Don Day designs and supports publishing tools for IBM's Information Development community and has represented IBM on the W3C XSL and CSS Working Groups. For the past three years, Don has led the workgroups that developed and now maintain the DITA DTDs and specification. He has B.A.s in English and Journalism and an M.A. in Technical and Professional Communication from New Mexico State University. You can reach him at dond@us.ibm.com.

Erik Hennum

Erik Hennum works on the design and implementation of User Assistance for the IBM Storage Systems Group. For DITA, he has helped shape the principles of domain specialization. He can be reached at ehennum@us.ibm.com.

© Copyright IBM Corporation 2005
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)