html2db.xsl

Oliver Steele

Revision History
Revision 1 2004-07-30

Table of Contents

Abstract

`html2db.xsl` is an XSLT stylesheet that transforms an XHTML source document into a Docbook document.

Overview

`html2db.xsl` converts an XHTML source document into a Docbook output document. It provides features for customizing the generation of the output, so that (in principle) the output can be tuned by annotating the source, rather than hand-editing the output. This makes it useful in a processing pipeline where the source documents are maintained in HTML, although it can also be used as a one-time con-

version tool too.

This document is an example of `html2db.xsl` used in conjunction with the Docbook XSL stylesheets. The source file [index.src.html] is an XHTML file with some embedded Docbook elements and processing instructions. `html2db.xsl` compiles it into a Docbook document [index.xml], which can be used to generate this output file (which includes a Table of Contents), a chunked HTML file [docs/index.html], a PDF [html2db.pdf], or other formats.

Status

This is a work in progress. It serves my needs, but doesn't attempt to be much more general than that. If you run into anything it can't handle, please send a note, or better yet, a patch, to `<steele@osteele.com>`. I can't promise to address problems (I have a day job too), but knowing what people have run into will help my prioritize my work when I do have time to work on this.

Features

XSLT implementation	This tool is designed to be embedded within an XSLT processing pipeline. `html2html.xslt` can be used in a custom stylesheet or integrated into a larger system. See Overriding.
Customizable	The output can be customized by the means of additonal markup (Docbook elements and XML processing instructions) in the XHMTL source. See the section on customization.
Creates outline structure	`h1`, `h2`, etc. are turned into nested `section` and `title` elements (as opposed to bridge heads).
Accepts a wide variety of XHTML	In particular, automatically wraps *naked text* (text that is not enclosed in a `<p>`) that is inside a table cell or list item. XHTML allows this; Docbook is more persnickety.[1]

Requirements

- Java: JRE or JDK 1.3 or greater.
- Xalan 2.5.0 or greater. *Don't* download this if you're using Java 1.4, or you may run into a nasty bug! (This isn't specific to `html2db.xsl`; it affects every Java+Xalan installation.)
- Familiarity with installing and running JAR files.

`html2db.xsl` might work with earlier versions of Java and Xalan, and it might work with other XSLT processors such as Saxon and xsltproc.

Installation

- Install JRE 1.3 or higher.
- Install Xalan, if necessary.
- Download `html2db-1.zip` from http://osteele.com/sources/html2db-1.zip [http://osteele.com/sources/html2db.zip].

[1] This feature is limited. See Text Wrapping.)

- Unzip `html2db-1.zip`.

Usage

Use Xalan to process an XHTML source file into a Docbook file:

```
java org.apache.xalan.xslt.Process -XSL html2dbk.xsl -IN doc.html > doc.xml
```

See `index.src.html` for an example of an input file.

If your source files are in HTML, not XHTML, you may find the Tidy [http://tidy.sourceforge.net/] tool useful. This is a tool that converts from HTML to XHTML, and can be added to the front of your processing pipeline.

(If you need to process HTML and you don't know or can't figure out from context what a processing pipeline is, `html2db.xsl` is probably not the right tool for you, and you should look for a local XML or Java guru or for a commercially supported product.)

Specification

XHTML Elements

`code/i` stands for "an `i` element immediately within a `code` element". The notation is from XPath.

XHTML elements must be in the XHTML Transitional namespace, `http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd`.

XHTML	**Docbook**	**Notes**
`b`, `i`, `em`, `strong`	`emphasis`	The `role` attribute is the original tag name
`dfn`	`glossitem`, and also `primaryindexterm`	
`code/i`, `tt/i`, `pre/i`	`replaceable`	In practice, `i` within a monospace content is usually used to mean this
`pre`, `body/code`	`programlisting`	
`img`	`[informal]figure/mediaobject/imageobject/imagedata`	If it has a `title` attribute or `db:title` it's wrapped in a `figure`. Otherwise it's wrapped in an `informalfigure`.[a]
`table`	`[informal]table`	XHTML `table` becomes Docbook `table` if it has a `summary` attribute; `informaltable` otherwise.
`ul`	`itemizedlist`	But see the processing instruction below.

[a]Note that this is wrong for inline images.

Links

Table 1. Link Translation

XHTML	Docbook	Notes
`<a name="name">`	`<anchor id="{$anchor-id-prefix}name">`	An anchor within a `hn` element is attached to the enclosing `section` as an `id` attribute instead.
`<a href="#name">`	`<link linkend="{$anchor-id-prefix}name">`	
`<a href="url">`	`<ulink url="name">`	
`<a name="mailto:address">`	`<email>address</email>`	

Tables

XHTML `table` support is minimal. `html2db.xsl` changes the element names and counts the columns (this is necessary to get table footnotes to span all the columns), but it does not attempt to deal with tables in their full generality.

An XHTML `table` with a `summary` attribute generates a `table`, whose `title` is the value of that summary. An XHTML `table` without a `summary` generates an `informaltable`.

Any `tr`s that contain `th`s are pulled to the top of the table, and placed inside a `thead`. Other `tr`s are placed inside a `tbody`. This matches the commanon XHTML `table` pattern, where the first row is a header row.

Text Wrapping

XHTML allows `li`, `dd`, and `td` elements to contain either inline text (for instance, `<li>a list item</li>`) or block structure (`<li><p>a block</p></li>`). The corresponding Docbook elements require block structure, such as `para`.

`html2db.xsl` provides limited support for wrapping inline text in these positions in `para` elements. If a list item or table cell item directly contains text, all text up to the position of the first element (or all text, if there is no element) is wrapped in `para`. This handles the simple case of an item that directly contains text, and also the case of an item that contains text followed by blocks such as paragraphs.

Note that this algorithm is easily confused. It doesn't distinguish between block and inline XHTML elements, so it will only wrap the first word in `<li>some <b>bold</b> text</li>`, leading to badly formatted output. This is easy to fix now that I understand the difference between block and inline elements (I didn't when I was implementing this), but I probably won't do so until I run into the problem again. In the meantime, the workaround is to wrap troublesome content in explicit `<p>` tags.

Docbook Elements

Elements from the Docbook namespace are passed through as is. There are two ways to do this:

Global prefix

A Docbook namespace declaration may be added to the document root element. Anywhere in the document, the prefix from this namespace declaration may be used to include a Docbook element. This is useful if a document contains many Docbook elements, such as `footnote` or `glossterm`, interspersed with XHTML. (In this case it may be more convenient to allow these elements in the XHMTL namespace and add a customization layer that translates them to docbook elements, however. See Customization.)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:db="urn:docbook">
  ...
  <p>Some text<db:footnote>and a footnote</db:footnote>.</p>
```

Local namespace

A Docbook element may be introduced along with a prefix-less namespace declaration. This is useful for sticking a *hierarchy* of elements in Docbook namespace into the midst of a XHTML document.

```
  ...
  <articleinfo xmlns="urn:docbook">
    <author>
      <firstname>...</firstname>
  ...
```

The source [index.src.html] to this document illustrates both of these techniques.

Note

For purposes of the *source files*, but not the output files, for this tool, the "Docbook namespace" is `urn:docbook`. Docbook doesn't really have a namespace, and if it did, it wouldn't be this one. See Docbook namespace for a discussion of this issue.

Note

Both these techniques will cause your document to be invalid as XHTML. In order to validate an XHTML document that contains Docbook elements, you will need to create a custom schema. Technically, you then ought to place your document in a different namespace, but this will cause `html2db.xsl` not to recognize it!

Output Processing Instructions

`html2db.xsl` places a handful of processing instructions. The Docbook XSL stylesheets ignore these, but if you write a customization layer for Docbook XSL, you can use the information in these processing instructions to customize the HTML output. This can be used, for example, to preserve the `a` `onclick` and `target` attributes in the source document, where are transformed into `<html2db attribute?>` processing instructions in the Docbook output file.

`<?html2db attribute="`*`name`*`" value="`*`value`*`"?>`

Placed inside a link element to capture the value of the `a` `target` and `onclick` attributes. *`name`* in the name of the attribute (`target` or `onclick`), and *`value`* is its value, with `"` and `\` replaced by `\"` and `\\`, respectively.

`<?html2db element="br"?>`

Represents the location of an XHTML `br` element in the source

	document.
`<?db2html element="br"?>`	Represents the location of an XHTML `br` element in the source document.

You can also include `<?db2html?>` processing instructions in the HTML source document, and they will be copied through to the Docbook output file unchanged (as will all other processing instructions).

Customization

XSLT Parameters

`<xsl:param name="anchor-id-prefix" select="''/>`	Prefixed to every id generated from `<a name=>` and `<a href="#">`. This is useful to avoid collisions between multiple documents that are compiled into the same book.
`<xsl:param name="document-root" select="'article'"/>`	Default document root; can be overridden by `<?html2db class="`*`name`*`">`.

Processing instructions

Use the `<?html2db?>` processing instruction to customize the transoformation of the XHTML source to Docbook:

Processing instruction	Content	Effect
`<?html2db class="`*`xxx`*`"?>`	`body`	Sets the output document root to *`xxx`*. Useful for translating to `prefix`, `appendix`, or `chapter`; the default is *`$document-root`*.
`<?html2db class="simplelist"?>`	`ul`	Creates a vertical `simplelist`. (Note that the current implementation simply checks for the presence of *any*`html2db` processing instruction.)
`<?html2db rowsep="1"?>`	`[informal]table`	Sets the `rowsep` attribute on the generated `table`. (Note that the current implementation simply checks for the presence of *any*`html2db` processing instruction that begins with `rowsep`, and assumes the vlaue is `1`.)

Overriding the built-in templates

For cases where the previous techniques don't allow for enough customization, you can override the

builtin templates. You will need to know XSLT in order to do this, and you will need to write a new stylesheet that uses the `xsl:import` element to import `html2db.xsl`.

The `example.xsl` [examples.xsl] stylesheet is an example customization layer that recognizes the `<div class="abstract">` and `<p class="note">` classes in the source [index.src.html] for this document, and generates the appropriate Docbook elements for them.

FAQ

Why generate Docbook?

The primary reason to use Docbook as an *output* format is to take advantage of the Docbook XSL stylesheets. These are a well-designed, well-documented set of XSL stylesheets that provide a variety of publishing features that would be difficult to recreate from scratch for HTML:

- Automatic Table-of-Contents generation
- Automatic part, chapter, and section numbering.
- Creation of single-page, multi-page, PDF, and WinHelp files from the same source document.
- Navigation headers, footers, and metadata for multi-page HTML documents.
- Link resolution and link target text insertion across multiple pages and numbered targets.
- Figure, example, and table numbering, and tables of these.
- Index and glossary tools.

Why write in XHTML?

Given that Docbook is so great, why not write in it?

Where there are not legacy concerns, Docbook is probably better choice for structured or technical documentation.

Where the only legacy concern is the documents themselves, and not the tools and skill sets of documentation contributors, you should consider using (X)HMTL convertor to perform a one-time conversion of your documentation source into Docbook, and continue to maintain it in that format. You can use this stylesheet to perform this conversion, or evaluate other tools, many of which are probably appropriate for this purpose.

There are two reasons to maintain documentation sources in XHTML instead of Docbook: the availability of cheap (including free) and usable HTML editors and editing modes; and the fact that everyone knows HTML but only documentation specialists generally know Docbook.

For example, at Laszlo, most developers contribute directly to the documentation. Everyone knows HTML and XML these days, and XHTML isn't too far a step from these. It would be a significant impediment to either require that every documentation contributor learn the Docbook tag set, or to insert a process whereby every developer contribution had to wait in a bottleneck for the documentation team to assimilate it.

Why not use an existing convertor?

This isn't the first (X)HTML to Docbook convertor. Why not use one of the exisitng ones?

Each HTML to Docbook convertors that I could find had at least some of the following limitations,

some of which stemmed from their intended use as one-time-only convertors for legacy documents:

- Many only operated on a subset of HTML, and relied upon hand editing of the output to clean up mistakes. This made them impossible to use as part of a processing pipeline, where the source is *maintained* in XHTML.
- There was no way to customize the output, except by (1) hand editing, or (2) writing a post-processing stylesheet, which didn't have access to the information in the XHTML source document.
- Many of them were difficult or impossible to customize and extend. They were closed-source, or written in Java or Perl (which I find to be a difficult languages to use for customizing this kind of thing) and embedded in a larger system.
- They didn't take full advantage of the Docbook tag set and content model to represent document structure. For instance, they didn't generate nested `section` elements to represent `h1 h2` sequences, or `table` to represent tables with `summary` attributes.

I got this error. What does it mean?

Q. `Fatal Error! The element type "br" must be terminated by the matching end-tag "</br>".`

A. Your output document is empty except for the `<?xml version="1.0" encoding="UTF-8"?>` line.

Q. `Fatal Error! The prefix "db" for element "db:footnote" is not bound.`

A. You haven't declared the `db` namespace prefix. See the example [index.src.html] for an example.

Implementation Notes

Limitations

- The `id` attribute is only preserved for certain elements (at least `h`*n*, images, paragraphs, and tables). It ought to be preserved for all of them.
- Only very simplest table format is implemented.
- Always uses compact lists
- The string matching for `<?html2b class="`*classname*`"?>` requires an exact match (spaces and all).
- The text wrapping code is easily confused, as documented in that section.

Bugs

- Improperly sequenced `h`*n* (for example `h1` followed by `h3`, instead of `h2`) will result in duplicate text.

Wishlist

- Allow <html2db attribute-name="*name*" value="*value*"?> at any position, to set arbitrary Docbook attributes on the generated element.
- A different technique from the fake namespace prefix to name Docbook elements in the source; for instance, an option to parse the `class` (e.g.) or use a processing attribute (<?html2db classname="footnote"?>).
- Parse DC metadata from XHTML `html/head/meta`
- Option to use `html/head/title` instead of `html/body/h1[1]` for top title
- Allow an `id` on every element.
- Add an option to translate the XHTML `class` into a Docbook `role`.
- Preserve more of the whitespace from the source document especially within lists and tables in order to make it easier to debug the output document.

Design Notes

The Docbook Namespace

`html2db.xsl` accepts elements in the "Docbook namespace" in XHTML source. This namespace is `urn:docbook`.

This isn't technically correct. Docbook doesn't really have a namespace, and if it did, it wouldn't be this one. RFC 3151 [http://www.faqs.org/rfcs/rfc3151.html] suggests `urn:publicid:-:OASIS:DTD+DocBook+XML+V4.1.2:EN` as the Docbook namespace.

There two problems with the RFC 3151 namespace. First, it's long and hard to remember. Second, it's limited to Docbook v4.1.2 but `html2db.xsl` works with other versions of Docbook too, which would presumably have other namespaces. I think it's more useful to *under*specify the Docbook version in the spec for this tool. Docbook itself underspecifies the version completely, by avoiding a namespace at all, but when mixing Docbook and XHTML elements I find it useful to be *more* specific than that.

History

The original version of `html2db.xsl` was written by Oliver Steele [http://osteele.com], as part of the Laszlo Systems, Inc. [http://laszlosystems.com] documentation effort. We had a set of custom stylesheets that formatted and added linking information to programming-language elements such as `classname` and `tagname`, and added Table-of-Contents to chapter documentation and numbers examples.

As the documentation set grew, the doc team (John Sundman) requested features such as inter-chapter navigation, callouts, and index and glossary elements. I was able to beat all of these back except for navigation, which seemed critical. After a few days trying to implement this, I decided it would be simpler to convert the subset of XHTML that we used into a subset of Docbook, and use the latter to add navigation. (Once this was done, the other features came for free.)

During my August 2004 "sabbatical", I factored the general html2db code out from the Laszlo-specific code, refactored and otherwise cleaned it up, and wrote this documentation.

Credits

`html2db.xsl` was written by Oliver Steele [http://osteele.com], as part of the Laszlo Systems, Inc.

[http://laszlosystems.com] documentation effort.