A Comparative Look at Embedded Index Entry Encoding in Selected Publishing Systems

by David K. Ream[1]

While the traditional approach to deliver an index to a publisher is to provide the index as a distinct file, usually an RTF file, many publishers now request embedded indexing. Publishers often generate the index using the intrinsic index compiling feature of their composition software (Word, InDesign, etc.). This removes the complete control the indexer has had via stand-alone software for sorting, styling and formatting the index. This also requires time to write the index within the publishing system or to embed the index entries after writing the index in stand-alone software.

I have seen many publishing systems come and go throughout the years. Some previous generation examples are Framemaker, PageMaker, Quark, and Ventura. These have been mostly deprecated for general use. Two likely reasons are that these software systems didn't support XML and/or Unicode. All the clients I've worked with who previously used any of these systems have moved on to more current publishing systems, usually InDesign. But I'm sure some of older systems are still in use at some publishers.

To provide indexers with more in-depth knowledge of embedding, I will here compare aspects of embedded indexing within eight specific publishing systems. While I was aware of some of this information, it wasn't until I was developing IXMLembedder[2] that I dove deep into each of these publishing systems.

Table 1 lists broad aspects of these publishing systems.

This article is not meant to be a tutorial. XML is not a standard set of tags but rather a standard way to define a set of tags for a specific purpose. Often this purpose is to publish content. A book, a journal, and a manual require different tagging and hence a different schema or Document Type Definition (DTD) to provide the necessary structure hierarchy for each type of publication. XML is a *de jure* standard but it has become as well the *de facto* standard in the publishing industry, and not just for publishable content. XML is also used for metadata (the ONIX standard), ebooks (EPUB3 uses XHTML), and index data exchange (the IXML format). All forms of XML, including IXML, use Unicode by default. AsciiDoc is not Unicode-enabled as far as I can tell. Neither is LaTeX but there are extensions (XeTeX for example) that do implement Unicode.

The following discussion pertains to books, not to journals or other publication formats. Similarly, in compiled indexes, generated locators are page numbers for at least print versions of books. If a publishing system also produces ebooks, locators may be page numbers, but for proprietary systems, other locators may be provided. Also, while this discussion is geared to print publications, embedded indexing is a big step towards providing active, linked indexes in ebooks.

Following is a brief description of each publishing system:

- AsciiDoc[3] is a text markup language for documentation. It doesn't use XML tags but rather it uses unique character strings. This article references O'Reilly Media's extensions to AsciiDoc that provide embedded entry encoding. AsciiDoc easily maps its unique character strings to DocBook tagging.
- DocBook[4] has 15-year-old roots in SGML. It was developed in part by O'Reilly Media. Its current version is XML-based tagging.
- HTMLBook[5] uses XHTML-based tagging for authoring and production of books. It was developed by O'Reilly Media.
- InDesign[6] is a publishing system product from Adobe. Its native files are in a proprietary format accessed via InDesign.
- LaTeX[7] is a text markup language for documentation. It is built on TeX which was developed in the 1970s. Its tags are unique to TeX. It is often used in scientific publications because it excels at typesetting mathematical equations.
- TEI[8] (Text Encoding Initiative) is an XML-based tagging language. It is used chiefly in humanities, social sciences, and linguistics.
- Word is Microsoft's document editor. Embedded entries are encoded in a *field* and may include *switches* (similar to XML attributes) that alter or supply more information about the index data or its generation.
- XML is a metalanguage used to define tagging for specific instances of data such as those listed earlier. It is not a defined tag set. Publisher can develop their own XML schema. It is important to discuss the specifics of any proprietary tagging your client wants to use. The XML row in the table examples below illustrate one specific client's XML tagging.

Each of these publishing systems can be used to create print and digital formats, i.e. PDFs, RTFs, HTML web sites, and ebooks. Table 1 identifies these systems' *out-of-the-box* features (except for the XML row which is a client-specific example). Scripts and applications to make it easier to create or embed index entries, particularly for Word and InDesign, are available from third-party vendors.[9]

continued . . .

David K. Ream is Leverage Technologies' chief consultant for publishers. He has an MS in Computer Science from Case Western Reserve University, and has spent over 30 years working with publishers in the areas of typesetting design and production, database creation, editorial systems, and electronic publication design and production. He has served on ASI's DTTF (Digital Trends Task Force) since its inception and is currently its co-chair.

Table 1 – File Formats

PUBLISHING SYSTEM	NATIVE FILE FORMAT	METHOD OF ENTRY FOR EMBEDDED ENTRIES	IMPORTS/EXPORTS XML FILES
AsciiDoc (with O'Reilly Media extensions)	Text files with non-XML markup	Manually via a text editor	n/a
DocBook	Text files with XML markup	Manually via a text or XML editor	XML native files
HTMLBook	Text files with XHTML markup	Manually via a text or XML editor	XML native files
InDesign	Proprietary format	Built-in index entry GUI dialog box	IDML files – zip files containing folders and XML files for document components – can be saved and opened.
			Simple XML, where each XML tag name corresponds to a paragraph or defined character style, can be imported. This is a useful method for loading data from CINDEX or another database.
			Complex XML can be configured to match a proprietary DTD or schema
LaTeX	Text files with non-XML markup	Manually via a text editor	n/a
TEI	Text files with XML markup	Manually via a text or XML editor	XML native files
Word	Proprietary format	Built-in index entry GUI dialog box	XML native files are a detailed, equivalent representation of what you would see in an RTF file
client-specific XML example	Text files with XML markup	Manually via a text or XML editor or client-supplied method	XML native files

Table 2 compares common tagging methods for embedded entries for the following categories:

- wrapper: a begin and end character string enclosing all the information for an embedded entry. For XML files, this is a pair of begin and end tags.
- nesting: whether or not heading levels are nested in the tagging scheme.
- heading levels: whether headings are encoded with tags, or in the order they are presented usually separated with punctuation.
- sortas: how alternate sorting information is provided.
- cross references: how cross references are encoded.

Table 2 – Tagging Methods

PUBLISHING SYSTEM	WRAPPER	NESTING	HEADING LEVELS	SORTAS	CROSS REFERENCES
AsciiDoc	begin/end character string markup	no	ordinal position	via attribute	via an attribute
DocBook	begin/end tags	no	begin/end tags	via attribute	begin/end tags
HTMLBook	single tag	no	via attribute	via attribute	via an attribute
InDesign	single tag; ranges require begin/end tags	no	ordinal position	via attribute	single tag

Table 2 continued – Tagging Methods

PUBLISHING SYSTEM	WRAPPER	NESTING	HEADING LEVELS	SORTAS	CROSS REFERENCES
LaTeX	begin/end markup	no	ordinal position	positional	positional
TEI	begin/end tags	yes with wrapper	begin/end tags	via attribute	n/a
Word	XE field markup	no	ordinal position	positional	via a switch
client XML	none	yes	begin/end tags	via attribute	via an attribute

InDesign does something no other publishing systems do: in a separate designmap XML file, InDesign maintains a list of topics as they are entered into index entries. This is where alternate sorting values and cross references are stored. In the examples, Topic tag indicates the encoding is in the topic list. *PageReference* tag indicates encoding is embedded in the content.

In all eight systems, sortas value is a distinct element separate from the heading text. In stand-alone indexing, alternate sorting information is integrated inline with heading text. There is no way to specify sortas values for cross references.

As the following tables illustrate, every publishing system uses different syntax for encoding embedded entries. XML variants are similar, but not identical. The tables compare the following situations:

- One heading (Table 3)
- Alternate sort sequencing of headings (Table 4)
- Styled heading (Table 5)
- Multiple levels of headings (Table 6)
- Cross references (Table 7)
- Range (Table 8)
- Styled locator (Table 9)
- Multiple indexes (Table 10)

Examples of each are presented in the tables in their entirety where possible. Most examples also show the wrapper or container markup delimiting all the elements of an embedded entry. When multiple elements are needed to specify a more complex situation, such as an embedded entry that starts a range, specifies an index type, and supplies an alternate sorting value, multiple elements can then be combined as appropriate.

Table 3 – Encoding One heading "Dogs"

PUBLISHING SYSTEM	ENCODING
AsciiDoc	(((“Dogs”)))
DocBook	<indexterm><primary>Dogs</primary></indexterm>
HTMLBook	<a data-type="indexterm" data-primary="Dogs"/>
InDesign	<PageReference Self="ud3f" PageReferenceType="CurrentPage" ReferencedTopic="ud3cTopicnDogs"/>
LaTeX	\index{Dogs}
TEI	<index><term>Dogs</term></index>
Word	{ XE “Dogs” }
client XML	Dogs

Table 4 – Encoding Alternate Sort Sequencing of Headings "4H Club" and "The Ohio State University"

PUBLISHING SYSTEM	ENCODING
AsciiDoc	(((“4H Club”, sortas="fourh club”))) (((“The Ohio State University”, sortas="ohio state university”)))
DocBook	<indexterm><primary sortas="fourh club">4H Club</primary></indexterm> <indexterm><primary sortas="ohio state university">The Ohio State University</primary></indexterm>
HTMLBook	<a data-type="indexterm" data-primary="4H Club" data-primary-sortas="fourh club"/> <a data-type="indexterm" data-primary="The Ohio State University" data-primary-sortas="ohio state university"/>

Table 4 continued – Encoding Alternate Sort Sequencing of Headings "4H Club" and "The Ohio State University"

PUBLISHING SYSTEM	ENCODING
InDesign	<Topic Self="ud3cTopicn4H Club" SortOrder="fourth club" Name="4H Club"> <Topic Self="ud3cTopicnThe Ohio State University" SortOrder="ohio state university" Name="The Ohio State University">
LaTeX	\index{fourh club@4H Club} \index{ohio state university@The Ohio State University}
TEI	<index><term sortKey="fourh club">4H Club</term></index> <index><term sortKey="ohio state university">The Ohio State University</term></index>
Word	{ XE "4H Club;fourh club" } { XE "The Ohio State University;ohio state university" }
client XML	4H Club The Ohio State University

Table 5 – Encoding Styled Heading *"New York Times"*

PUBLISHING SYSTEM	ENCODING
AsciiDoc	(((“__New York Times__”)))
DocBook	<indexterm><primary><emphasis>New York Times</emphasis> </primary></indexterm>
HTMLBook	*specification is silent on this*
InDesign	*any inline style markup needs be changed to character styles after the index is generated since inline style markup is not recognized by InDesign:* <PageReference Self="ud3f" PageReferenceType="CurrentPage" ReferencedTopic="ud3cTopicn<i>New York Times</i>"/>
LaTeX	\index{New York Times@\textit{New York Times}}
TEI	<index><term><emph>New York Times</emph></term></index>
Word	{ XE "New York Times" }
client XML	*different tags are used depending on type of heading rather than by its style:* <ic>New York Times</ic>

Table 6 – Encoding Multiple Levels of Headings "Mammals" > "Canines" > "Dogs"

PUBLISHING SYSTEM	ENCODING
AsciiDoc	(((“Mammals”, “Canines”, “Dogs”)))
DocBook	<indexterm> <primary>Mammals</primary> <secondary>Canines</secondary> <tertiary>Dogs</tertiary> </indexterm>
HTMLBook	<a data-type="indexterm" data-primary="Mammals" data-secondary="Canines" data-tertiary="Dogs"/>
InDesign	<PageReference Self="ud3f" PageReferenceType="CurrentPage" ReferencedTopic="ud3cTopicnMammalsTopicnCaninesTopicnDogs"/>
LaTeX	\index{Mammals!Canines!Dogs}

Table 6 continued – Encoding Multiple Levels of Headings "Mammals" > "Canines" > "Dogs"

PUBLISHING SYSTEM	ENCODING
TEI	<index><term>Mammals</term> <index><term>Canines</term> <index><term>Dogs</term></index> </index> </index>
Word	{ XE "Mammals:Canines:Dogs" }
client XML	Mammals <ii>Canines <iii>Dogs</iii> </ii>

Table 7 – Encoding Cross references "Canines. See also Dogs" and "Felines. See Cats"

PUBLISHING SYSTEM	ENCODING
AsciiDoc	(((“Canines”, seealso=”Dogs”))) (((“Felines”, see=”Cats”)))
DocBook	<indexterm><primary>Canines</primary> <seealso>Dogs</seealso></indexterm> <indexterm><primary>Felines</primary><see>Cats</see></indexterm>
HTMLBook	<a data-type=”indexterm” data-primary=”Canines” data-seealso=”Dogs”/> <a data-type=”indexterm” data-primary=”Felines” data-see=”Cats”/>
InDesign	<Topic Self=”ud3cTopicncanines” SortOrder=”” Name=”Canines”> <CrossReference Self=”udaa” ReferencedTopic=”ud3cTopicnDogs” CrossReferenceType=”See also” CustomTypedString=””/> </Topic> <Topic Self=”ud3cTopicnfelines” SortOrder=”” Name=”Felines”> <CrossReference Self=”udab” ReferencedTopic=”ud3cTopicnCats” CrossReferenceType=”See” CustomTypedString=””/> </Topic>
LaTeX	\index{Canines|seealso{Dogs}} \index{Felines|see{Cats}}
TEI	*specification does not support cross references*
Word	{ XE “Canines” \t “See also Dogs” } { XE “Felines” \t “See Cats” }
client XML	*the attribute name is the same for both types of cross references; the index generation logic creates the appropriate lead words based on the headings in the index:* Canines Felines

Table 8 – Encoding a Range for a Heading "Pets"

PUBLISHING SYSTEM	ENCODING
AsciiDoc	(((“Pets”, id=”range1”, range=”startofrange”))) …. content … (((range=”endofrange”, startref=”range1”)))
DocBook	<indexterm id=”range1” class=”startofrange”> <primary>Pets</primary></indexterm> …. content … < indexterm class=”endofrange” startref=”range1”/>

Table 8 continued – Encoding a Range for a Heading "Pets"

PUBLISHING SYSTEM	ENCODING
HTMLBook	<a data-type="indexterm" data-primary="Pets" id="range1"/> .... content ... <a data-type="indexterm" data-startref="range1"/>
InDesign	*ranges are not point-to-point but rather an indicator of how far from the starting point to include;* *some common range types:* <PageReference Self="ud3f" PageReferenceType="ToEndOfDocument" ReferencedTopic="ud3cTopicnPets"/> <PageReference Self="ud3f" PageReferenceType="ToEndOfStory" ReferencedTopic="ud3cTopicnPets"/> <PageReference Self="ud3f" PageReferenceType="ForNextNPages" ReferencedTopic="ud3cTopicnPets"> <Properties> <PageReferenceLimit type="long">3</PageReferenceLimit> </Properties> </PageReference>
LaTeX	\index{Pets|(} .... content ... \index{Pets|)}
TEI	<index spanTo="#range1"><term>Pets</term></index> .... content ... <anchor xml:id="range1"/>
Word	*requires that a bookmark, which highlights the range of text, be created with the name bookmark1; a range appears in the index if the page numbers for the start and end points of the bookmark range are not on the same page:* { XE "Pets" \r "bookmark1" }
client XML	Pets .... content ...

Table 9 – Encoding Styled Locators (italic or boldface)

PUBLISHING SYSTEM	ENCODING
AsciiDoc	*not specifiable in the embedded entry*
DocBook	*not specifiable in the embedded entry*
HTMLBook	*not specifiable in the embedded entry*
InDesign	<PageReference Self="ue13" PageReferenceType="CurrentPage" PageNumberStyleOverride="CharacterStyle/LocatorItalic" ReferencedTopic="ud3cTopicn..." /> <PageReference Self="ue13" PageReferenceType="CurrentPage" PageNumberStyleOverride="CharacterStyle/LocatorBold" ReferencedTopic="ud3cTopicn..." /> *Note: this only works if the book is a single file rather than multiple files say one per chapter.*
LaTeX	\index{...|\textbf} \index{...|\textit}
TEI	*not explicitly specifiable in the embedded entry but enhanced scripting could accommodate specific situations; client discussion would be in order*
Word	*Only bold, italics, or bold italics:* { XE ... \b } { XE ... \i } { XE ... \b \i }
client XML	*not specifiable in the embedded entry*

Table 10 – Encoding Multiple Indexes (examples show the begin tag only)

PUBLISHING SYSTEM	ENCODING
AsciiDoc	*not available*
DocBook	<indexterm type="topics"> <indexterm type="names">
HTMLBook	*not available*
InDesign	*not available*
LaTeX	\index[topics]{...} \index[names]{...}
TEI	<index indexName="topics"> <index indexName="names">
Word	{ XE ... \f "topics" } { XE ... \f "names" }
client XML	*not available*

The appearance and functionality of the index generated from embedded index entries depends on the features or options within the software module, script, or application generating the index. InDesign and Word, for example, have an "insert index" function as well as options for index style. For instance, you can:

- Modify paragraph styles for headings
- Specify a number of columns for displaying the index
- Use run-in or indented format

To proof or review an index, generate a PDF copy of the index either through open source software, client-provided scripts, or the XML editor. For instance, oXygen[10] has a built-in transformation that will generate a PDF version of the book with a linked index from DocBook files. Check with your client to determine what options are available.

There are various reasons why you may need to edit or review the generated index before finally producing a well-formed index. For example:

- to add decorations, differentiators, or styling to locators
- to order multiple cross references
- to remove redundant lead word phrases

This should be the very last task undertaken after the index has been reviewed and deemed good. If a typo or sorting issue is spotted that requires the index to be recompiled, any manual edits will be overwritten.

If you do require other features for the index, such as section number locators, decorations or differentiated locators, etc., the index will generally have to be provided as a separate document—rather than embedded—and the locators will need to be hyperlinked. (This is assuming you want a linked, active index in the electronic version of the publication.) Again, discuss this with your client as they may have enhanced scripts to provide some of these other features.

In conclusion, there are more embedded encoding styles than any one indexer is likely to encounter. It is also likely you will encounter more than one style of encoding. The embedding methods compared in this article provide a list of specific situations to discuss with your client to help get you started. ■

(Endnotes)

[1]Many thanks to Michele Combs, Lucie Haskins, and Jan Wright for proofing of and suggestions for the article.

[2]http://www.levtechinc.com/publishing-indexing-products/utilities/IXMLembedder.asp

[3]http://docs.atlas.oreilly.com/writing_in_asciidoc.html

[4]http://www.docbook.org

[5]https://oreillymedia.github.io/HTMLBook

[6]https://www.adobe.com/content/dam/Adobe/en/devnet/indesign/cs55-docs/IDML/idml-specification.pdf

[7]https://en.wikibooks.org/wiki/LaTeX/Indexing#Sophisticated_indexing

[8]http://www.tei-c.org/index.xml

[9]http://www.asindexing.org/about-indexing/digital-trends-task-force/

[10]https://www.oxygenxml.com/