XML for publishing

Move content to XML without missing a beat

Kay Whatley

October 28, 2008

Smoothly transition documents designed for print publishing to XML. Discover how logical elements, attributes, and hierarchy make for easier print (and PDF) publishing with the structure in XML.

When you move into an XML-based publishing environment, the right approach can save time and even make new publishing paradigms possible if you properly plan and design the structure. XML is a powerful medium for content, as it turns your documents from a mash of text and objects into a sortable, adjustable, hierarchical collection of pieces. Evaluating existing unstructured content is imperative to reach short-term and long-term publishing goals.

This article describes how you can convert documents designed for print publishing to structured documents. The sections that follow cover the logical musts to ensure that publishing is possible—even easier—following a transition to XML. The focus is on how to design structure for your content.

Planning with long-term XML goals in mind

Frequently used acronyms

- DITA: Darwin Information Typing Architecture
- HTML: Hypertext Markup Language
- PDF: Portable Document Format
- XML: Extensible Markup Language
- XSLT: Extensible Stylesheet Language Transformations

As an author, you might be used to working with your word processor's formatting plus the Enter key. (You type, choose a format, press Enter, type more, choose another format, press Enter, and so on.) With structure, instead of using the Enter key, you insert structural elements. You adjust to not thinking about the formatting of your content. In the structure world, your authoring tools or XML stylesheets handle the formatting.

Word-processed documents include typed text, graphics, and tables. Converted to structure, each of these components is identified, along with any special information that might be needed to drive the publishing process or control formatting. The document parts become XML elements and can be treated like fields in a database—able to be located, sorted, used for retrieval, and otherwise
manipulated. The elements can also be treated differently based on their context—the parent elements in which they are nested or elements hierarchically above them in the document tree (ancestors).

The tool you start with should not affect the first step of your move to structure, which is to confirm your publishing goals and analyze your existing documents with those goals in mind. Think about the following questions:

- What kind of data does your document contain?
- Are there tables of complex information?
- Does your document break down easily into smaller topics or sections?
- Is your content mostly free-form text rather than organized into sections?
- What parts of your documents do you need to identify as special elements or attributes (to ensure that you can target them as desired for re-use, sorting, tracking, or enabling other XML benefits)?
- If your goals include translation, do you want to include attributes for identifying content that has been translated?
- Will you want to comment any document content or mark it for use in different versions of a document?

Determine your document's structure

Evaluate your existing content, and determine the structure that the content implies. For example, if you publish technical manuals, your documents might consist of sections of text, screen shots, illustrations, tables, procedures, reference data, and more. Your text might be broken into body paragraphs, lists, captions, headings, and highlighted phrases. Figure 1 shows a section of an example document. The section includes a heading, several paragraphs, and a set of procedural steps.

Figure 1. Example document section

Checking Your Installation

In this section, you will need to open an XML file to test that your sample setup is working properly. More detailed explanation will be provided including the why and how of this setup in future chapters.

1. From the .\xml\simpقدoc folder, open simpقدoc.xml.
2. When prompted, select the simpقدoc structured application.
3. Click Continue.
4. When the file appears, save it as simpقدoc.fim.

If you do not have the simpقدoc folder, refer to "Preparing Your Desktop" on page 119 for instructions on placing these sample files.

If at Step #2 in the preceding instructions your simp Qaeda structured application is not Listed, go back through the instructions in "Installing Sample XML Applications" on page 120, making sure to perform the step to Read Structured Application (as done in Step 14 on page 122).
Assign paragraph types to your elements

Evaluate these components, then assign the paragraph types to elements, giving your elements logical names. You have a <title> or <head> element to start (the heading, "Checking Your Installation"), followed by a paragraph that you might map to in a <p> element and numbered procedures that you might map to <step> elements within a <procedure> element. With the inclusion of the steps, you might decide that this part of the document should be identified as a <task> type of section, or you might just call it a <section> element.

So, you evaluate your content piece-by-piece and decide how you see the resulting structure. Element names are—unless you choose a ready-made structure—at your discretion. Keep element names logical to you and for your documents.

Using this document as an example, when evaluating it, you would find that the heading uses a "Heading 1" format, as noted in Figure 2.

**Figure 2. Formatting notations on the example section**

```
Checking Your Installation

---

Body Text First

In this section, you will need to open an XML file to test that your sample setup is working properly. More detailed explanation will be provided including the why and how of this setup in future chapters.

1. From the \xml\simplesdoc folder, open simplesdoc.xml.
2. When prompted, select the simplesdoc structured application.
3. Click Continue.
4. When the file appears, save it as simplesdoc.fin.

Numbered

```

If you know you want this to be in a <title> element, then you know that mapping must be made for the conversion. Likewise, for each of the other paragraphs (Body Text First, Numbered, Body Text), you would determine the element to create during the conversion. Don't forget to consider the italics and bold text formatting, cross-references, and other items within each of the paragraphs.

If you know you want this to be in a <title> element, then you know that mapping must be made for the conversion. Likewise, for each of the other paragraphs (Body Text First, Numbered, Body Text), you would determine the element to create during the conversion. Don't forget to consider the italics and bold text formatting, cross-references, and other items within each of the paragraphs.

In Figure 2, you can see the types of paragraphs (headings, numbered steps, and so on). Consider all your paragraphs as well as special items inside them. In Figure 2, you can see highlighted text and quoted phrases, for example. Not only might you break down your document into <title>, <p>, and <steps> but also <xref> or <italic> or <user_action>. Then, you're able to include italicized (or bolded, or otherwise highlighted) text within steps, include cross-references in the following paragraphs, or identify quoted information with a special element.
Though not evident in the figure, this content might have glossary or index terms identified, and you can create elements to enclose those words.

Consider the hierarchy for each element

When the elements are set, you must consider the hierarchy for each element. The structure (XML) includes rules about how you can use elements. Some elements will be wrapped inside a *parent element*—that is, an element that fully contains another element. The element inside the parent is called the *child element*. Child elements can have *siblings*, which are also inside the parent. Multiple levels of this nesting of elements inside each other produces the document hierarchy. As each element is defined, you need to identify which parent elements it can be in and which child elements (if any) it can contain.

Consider the attributes

Also consider any *attributes*—extra data included in your elements—needed to ensure that you can reach your publishing goals. Attributes can be set up to identify parts of a document more specifically, such as giving a `<title>` element an attribute that gives it an identity (such as `<title type="summary">`, where the attribute name is *type* and the attribute value is *summary*). Attributes can also hold extra information, like author names or release dates, or perhaps information important for sorting, retrieval, or document management but that never appears in published documents.

Rewriting when needed

During a detailed evaluation like this, you can not only begin to plan movement of your content into structure but also to verify how well your structure fits your content. To deal with any misfits, you can change the structure (if that is an option) or rewrite your content. Some rewriting might be required to better fit the selected structure, so begin with a looser first analysis, then go back through, tighten up your structure, and identify areas where you require rewriting to make the document do all that you need it to do. While your analysis is proceeding, you will be able to continue working with your content in its current, unstructured form.

Custom elements versus industry standards

You can analyze your documents and prepare a structure that fits them using elements that you name and design. Alternatively, you can use an industry-standard structure (for example, MIL-SPEC, DITA, DocBook). If you need to conform to an existing structure, you might need to make changes to your content to make it fit into the selected structure. To use DITA against the example given previously, you might decide to rewrite the paragraphs that follow the procedural steps. If the example document were to use a DITA task structure, then the `<task>` element offers limited options for content after the procedural steps. You might have to, for example, force your content to fit in as several `<p>` elements inside a `<result>` element, even though logically your paragraphs after the steps may not contain result-oriented text. Or you might have to rewrite those paragraphs to make them fit into the `<task>` element's child element, `<related-links>`. Changes to your content can be subtle or extreme, depending on how your documents are and the structure you need to fit into. Keep potential misfits in mind as you decide on your structure and begin to convert your documents.
Format mapping for conversion

When you determine your structure, map your formatting, and have your tools in place, you should be able to run through your conversion and get your structure into your documents. You might come up with a possible mapping that looks like Figure 3. The first column shows what was in the original, unstructured document. Paragraphs are identified, then text ranges (character-level formatting), then other document components. In the second column, you see the name of the element that will go around that content. In the third column are notes, which you can express in any way logical to you and which show what the hierarchy (ancestry) should end up being in the converted structure.

Figure 3. Example mapping (tabular model)

<table>
<thead>
<tr>
<th>Starting Object</th>
<th>Convert to Element</th>
<th>Hierarchy to Wrap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paragraph: Body Text</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>Paragraph: Heading 1</td>
<td>title</td>
<td>task</td>
</tr>
<tr>
<td>Paragraph: Numbered</td>
<td>cmd</td>
<td>stop &lt; steps &lt; taskbody</td>
</tr>
<tr>
<td>Text range: Bold</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>Text range: Emphasis</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>Cross Reference</td>
<td>xref</td>
<td></td>
</tr>
</tbody>
</table>

Using this data, you can now set up a conversion plan. Depending on the tool that you use or the expertise at your disposal, you can do the conversion through different processes. Some tools, like Adobe® FrameMaker®, provide a conversion process. Tools that don't have their own conversion to structure might require you to export the content first to an intermediate format (for example, text, HTML, or very raw XML). When you have that intermediate format, you might need to tap internal (or external) expertise to use Perl, XSLT, or other scripting options to convert your raw output to structure (XML) that you can use for publishing.

When you finish with conversion, you can review your content in its structure (see Figure 4) and confirm that the structure works for your content—and you.
Figure 4. Potential resulting structure for the example section

A pilot project—whether a full document or a chapter—will help you determine the conversion time you'll need. The pilot allows you to take a small percentage of your documents, run through the full conversion process, and ensure that when you're done, you have structured content that will be appropriately formatted for publication. When you believe the structure and content are coming together, go through your full conversion workflow with a good-sized sample, then review the results. If your design was careful and the structure a good fit for your content, you're likely to be able to proceed without reworking at this point.

Short-term distribution needs

Procedures discussed in this article—document analysis, structure mapping, rewriting—are often done while you continue to publish. You still need to get your documents out the door. As you move to structure, you need to keep your documents in a usable format. Depending on your distribution—paper, PDF, HTML, and so on—you might publish from a mix of unstructured and structured content.

When you're ready to move your content into structure, use the following process:

1. Back up your original unstructured files.
2. Run through your conversion with one document.
3. Review the conversion results.
4. Save the converted document either as a structured document (in your tool of choice) or as an XML file.
5. Publish from the document to ensure that it will publish as needed.
6. Repeat this process for your other documents, one at a time.

By using this process, at any point you can publish your documents—even if some are unstructured and the others have been through the process. After all documents are converted, moving forward, you can publish as needed with all your content, without missing a beat.

**Conclusion**

The move from unstructured publishing to structured publishing takes time and effort. If you plan in advance, you can avoid problems and ensure a smooth transition. Such a transition comes from careful consideration of options, pilot projects to gauge conversion time in advance, and advanced setup for fast conversion.

Ensuring that logical elements, attributes, and hierarchy are in place means publishing can smoothly continue after converting to XML.
Related topics

- **Introduction to XML** (Doug Tidwell, developerWorks, August 2002): Learn what XML is, why it was developed, and how it's shaping the future of electronic commerce. Also, meet a variety of important XML programming interfaces and standards, and look at two case studies that solve business problems with XML.
- **Introduction to XSLT** (Nicholas Chase, developerWorks, January 2007): The need to transform XML is so common that XSLT is considered one of the basic XML specifications. In this tutorial, explore how to transform XML data from one format to another as you create XSLT stylesheets, the basics of XPath which enables you to select specific parts of an XML document, and some more advanced capabilities that XSLT offers.
- **Darwin Information Typing Architecture (DITA XML)** Find additional information on DITA, an architecture to create topic-oriented, information-typed content for reuse and single-sourcing.
- **XML technical library**: See the developerWorks XML zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **IBM certification**: Find out how you can become an IBM-Certified Developer.
- **IBM trial software for product evaluation**: Build your next project with trial software available for download directly from developerWorks, including application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- **developerWorks podcasts**: Listen to interesting interviews and discussions for software developers.

© Copyright IBM Corporation 2008

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)