



Society for Technical Communication, Wisconsin Chapter

Customizing the XML Authoring Environment

7 February 2006

Peter Meyer

Contents

Abstract.....	1
The use of XML in production of technical documentation	1
The many flavours of XML.....	2
What is unique about using an XML editor?	2
The problem with XML editors.....	3
The role of schema design	4
The solution	4
Consider the writers	4
Schema selection and design	4
Next valid location element insertion	5
Context driven element insertion commands	5
Templates	6
Pre-formed structures	6
Data entry forms	6
Keyboard shortcuts.....	6
Menu items	6
Element selection utilities.....	7
Custom utilities	7
Benefits.....	7

Reduce change management barriers	7
Reduce effort training new writers.....	7
Improve markup consistency and reduce editorial costs	8
Improve writer productivity	8
Conclusions.....	8

Customizing the XML Authoring Environment

Peter Meyer, Managing director, Elкера® Pty Ltd.
www.elkera.com

Abstract

The production of technical and knowledge management documentation increasingly requires the use of specialized tools that support single source publishing requirements. More and more enterprises are finding that XML is the core technology in documentation systems. Content writers need to use XML editing applications to create content in a structured form that is self describing and separated from presentation.

Out-of-the-box, XML editing applications are not particularly easy to use without extensive customization to the DTD or schema and authoring requirements. XML makes it possible to create much more powerful aids for writers than is possible with conventional word processing software but individual customizations are expensive. If the needs of content writers are not properly considered in application planning, writer productivity can be badly affected. Ongoing training, support and data correction costs can be excessive.

In this presentation, Peter Meyer discusses the key issues involved in customizing XML editors to achieve high levels of usability for content writers. Usability is based on DTD design as well as editor configuration. The presentation will compare standard, out-of-the-box applications with carefully tailored applications using the Blast Radius XMetaL Author and the Elкера Utilities for XMetaL Author developed by Elкера to show the levels of usability that can be achieved.

The use of XML in production of technical documentation

There is increasing use of XML in the production of technical documentation. Particular reasons include:

- Increasingly complex systems require complex documentation that is not easily managed by manual processes. Points include content sharing and re-use, linking and the use of multi-media.
- There are increasing demands for multi language documentation and a need to coordinate changes and minimize the costs of translation.
- There is a need to provide documentation in a form that meets user needs in both print and various electronic formats.

- There is a growing range of content management systems and other applications that can help to manage XML content and extract information from it.
- XML is designed for use in automated document workflow, publishing and information retrieval systems.

The many flavours of XML

XML is not a markup language but a tool to define markup languages. There are many XML markup languages for documentation including Microsoft Office Open XML format (WordprocessorML), OASIS OpenDocument, XHTML 1, XHTML 2, DocBook, DITA, TEI, S1000D, Elkera BNML and many others.

Each of these schema or DTDs is designed for a particular purpose. Some are highly presentation oriented and rely on style names to tell us about the nature of the document content. Office Open XML format (WordprocessorML), OASIS OpenDocument, XHTML 1 fall into this category. The others provide a more hierarchical representation of the document and can better describe the nature of different components in the document.

Structural schema such as DITA, DocBook, S1000D, Elkera BNML and similar schema impose rules on content writers that may require the use of elements in particular sequence and mandate the use of certain elements. These rules are enforced through validation of the document against the schema. Some or all these rules must be understood by the writer before he or she can effectively write content using an XML editor.

The points discussed in this presentation relate only to the use of structural schema with an XML editor.

What is unique about using an XML editor?

When using an XML editor, a writer must insert a valid element defined by the schema before typing content. This is quite different to a word processor where the writer can type, press Enter and continue typing to create new content.

After completing the content for an element in an XML editor, the writer has to:

- (1) identify and choose the next element they wish to insert either inside or after the current element;
- (2) if the element is outside the current element, find the valid location at which to insert the new element, usually by moving the insertion point;
- (3) insert the selected element;

- (4) if elements are nested, the writer may have to go back and pick another element to insert before writing new content; and
- (5) continue entering content.

In many XML editors, steps 1 and 2 are bound together. Commonly, the editor provides a pick list of valid elements at each location. The problem is that when the writer finishes a paragraph, the allowable elements that may follow that element cannot be seen until the insertion point is moved to the correct location. This can be quite frustrating, particularly for writers who are new to XML editors.

Other demands imposed on writers in XML editors include:

- The schema may require an element to be inserted or an attribute value to be completed. If it is omitted, the document is invalid.
- When an element has to be moved, it can be moved only to a valid location. If the writer does not know how to find the valid location, moving content can be very frustrating.

Many XML editors provide multiple views of the document, including normal or Tags Off View, Tags On View and a Structure View. To deal with the problems mentioned, writers often have to work in a Tags On or Structured View.

The problem with XML editors

The particular characteristics of XML editors impose several burdens that are not as pronounced when using a word processor. This leads some organizations to try to let technical writers work in Word using style templates. Documents may be converted to XML later in the work flow. This is rarely entirely satisfactory. It is difficult, if not impossible, to reliably convert word processor documents to structural XML without some human intervention to correct errors.

Writers using an XML editor may need a good understanding of the schema rules to understand how to find valid locations for new and moved elements. This imposes a high training burden and may lead some writers to be intimidated by XML editors at first.

The additional steps introduced into the writing process by XML editors can slow down work and break concentration while writing.

If the use of XML editors is to become more widespread, it will be necessary to simplify the process for writers and enable them to work more easily.

The role of schema design

Schema design can accentuate the problems encountered with XML editors:

- Element names and their purpose ought to be distinct and logical to persons who work in the relevant subject domain. When many elements perform similar but slightly distinct functions, they are easily misused. Sometimes subtle distinctions are important but often they are not.
- Some schema define very loose content models, permitting the same content to be marked up in several different ways. This makes it hard for the writer to understand the basic patterns in the content and it produces inconsistent markup that may cause problems in published outputs. As will be seen later, it also makes it more difficult to provide a tailored interface for content writers.

The solution

Consider the writers

Most XML editors are designed to be customized in some way to provide writers with various shortcuts and aids to their work. XML editors must work with a wide range of schema. It is not practicable to treat them as a ready-to-use product. Sometimes, extensive customization is required to achieve a highly usable writing interface. The capacity of an XML editor to facilitate that customization should be carefully considered during evaluation.

The key to customizing an XML editor is to fully understand the content to be created and the conventions followed by writers. A customized interface need not try to deal with every situation. It should aim to make work easier for that common actions that will likely amount to 80% of the writer's work.

A good test of an effective interface is if the writer can perform most work without seeing any tags. While some editors attempt to work wholly without tags, Tags On Views can be very helpful to enable writers to quickly and accurately perform unusual editing actions.

Schema selection and design

As observed earlier, it is important that the schema is carefully designed to minimize confusion and inconsistency in the markup. If an existing schema is used, it may be necessary to customize it to meet these requirements. Some schema attempt to provide elements to meet a wide range of possible circumstances. If these are not absolutely necessary in your application, they should be removed. Otherwise, they will confuse writers, make it harder for them to recognize the common markup patterns, and make it difficult to provide predictable behavior in interface commands.

Ideally, the schema should be the simplest model that will create the desired markup structure. Most documents conform to very standard patterns for section/clause, paragraph and list structures. Ideally, the schema should make it easy for the writer to understand how these work. If the interface is then designed carefully, the writer won't really need to know the finer points of the schema rules until they are ready to learn them.

Next valid location element insertion

Next valid location insertion allows the writer to insert markup structures (elements) using a simple command without having to find the valid location first. Taking DocBook as an example, the writer may have created a `section` with one or more `para` elements. The insertion point is after the period at the end of the last `para`. The writer now wants to create a new `section` after the current `section`. The writer should be able to initiate an "insert section" command from a keyboard shortcut or other command to insert the new `section` without having to find the valid location for the `section` element. The application should take care of this.

Effective use of this strategy requires that the schema use tight content models. If there are many possible valid locations for the section element, constraints must be created in the customization code, thereby making it more expensive to develop. These constraints operate as a pseudo schema change. Ideally, the constraints should be imposed by the schema.

Next valid location element insertion commands can be made more powerful if they recognize shield elements. For example a `para` may be followed by a `blockquote` element. If the insertion point is in the `para` and the writer fires a command to insert a new `para` after, it may be inserted inside the `blockquote`. This may not be desired. Ideally, the application should allow the `blockquote` to be excluded from the valid locations in that circumstance. Instead, the application might only insert a `para` inside `blockquote` if the insertion point is already inside the `blockquote`. These rules should be carefully planned.

Context driven element insertion commands

In word processor software, writers are accustomed to pressing the *Enter* key to create new paragraphs and list items etc. In a word processor, this is achieved through chaining styles. The limitation of this is that the sequence can never change according to context.

In a structured schema with some XML editors, it is possible to define different behavior for element insertion commands according to the context. It may even be possible to use the text characters adjacent to the insertion point to further refine context rules. Thus, pressing *Enter* after a list item that is terminated by a semi colon may create

a new list item while pressing *Enter* after one that is terminated by a period may create a new paragraph after the list.

These rules must be based on conventions that are convenient to the writers who will use the application.

Templates

The use of templates is a simple way to create the major structures in a new document, just as with word processor documents. Very often, the major structural divisions in a document are only created once. If they are included in the template, the writer does not have to bother about creating them and it is probably not necessary to create specific interface commands for them.

Pre-formed structures

Pre-formed structures are packaged element structures inserted on a single command. If it is common to insert several nested elements before typing content, a template with those structures can be created and inserted by an interface command. Often these are context dependant so use of context rules may be helpful to make this feature work well. For example, in one context a command may insert a `section` with a `para` element and in another it may insert just the `section` element.

Data entry forms

In most XML editors it is possible to create forms to enter element or attribute data. These can be very helpful for graphics, metadata, cross references and tables.

Keyboard shortcuts

Keyboard shortcuts can be defined to execute common element insertion or other commands. The underlying action maybe to insert pre-formed structures at the next valid location, according to a defined context. In this way, a great deal of markup can be inserted very easily without the writer having to break concentration.

Menu items

Pull down menus and toolbar buttons should be used to organize the more common and the less common commands so that the main interface is not overloaded.

Element selection utilities

Writers frequently need to select elements, cut them and paste them into a new location. Some editors provide keyboard shortcuts to allow the writer to progressively select containing elements. However, for common actions, specific commands may be created to select specific elements and speed up the writer's work. These are more accurate because the writer knows exactly which elements are selected.

Custom utilities

In most applications writers have to perform common editing commands. They may need to insert elements by pasting copied content in particular contexts. Commands can be created to transform markup and to control the destination of pasted content so the writer does not have to find the valid location in Tags On View before pasting.

Utilities can be created to assist writers to create cross references so they don't have to write in ID values at targets.

Benefits

Reduce change management barriers

All too often, XML applications are developed by technical people who are focussed on content management and publishing systems. The needs of writers are given little or no consideration until it is time to train them on the new system. Inadequate attention to writer needs can delay rollout of a new system and incur excessive training and support costs.

Using a thoughtful customization strategy, transition times for XML authors can be reduced from months to a few days or weeks.

Reduce effort training new writers

One of the biggest hurdles for writers new to XML is to learn about XML and the rules of the schema. The aim of an effective customization should be to reduce the interface to a small set of commands that enable writers to create common markup structures without being concerned about the schema rules. Naturally, writers must understand the basic patterns and rules but these only have to be understood at a high level. In this way, writers can begin work quickly without even seeing XML markup. This builds confidence and allows writers to learn the detail as it is needed. Over time they will build up a good understanding of the schema.

Improve markup consistency and reduce editorial costs

The processes described will ensure that the schema and the user interface are coordinated. Redundant elements should be removed and excessively loose content models tightened. When writers work through defined interface commands, the XML markup created will be more consistent than if they must choose elements and contexts through the standard interface components.

Improved markup accuracy at source means fewer problems and lower rectification costs at later stages in the content life cycle.

Improve writer productivity

Standard interfaces in XML editors do not allow writers to work smoothly. They must frequently stop to perform complex or fiddly element selection and insertion actions in Tags On or Structure Views. A more usable interface will enable writers to work more productively.

Conclusions

In most XML editors, this customization work can involve very considerable effort. In some editors, not all these options are available. With the right tools, the benefits are substantial and continue indefinitely. With larger teams of writers, the benefits should clearly justify the development effort.

Greater use of standard schema should facilitate the development of applications with inbuilt configurations for those schema. This will reduce the cost of customization efforts and make improved XML authoring interfaces available to smaller teams.