e-mail it!

XML Matters #6
A roundup of editors

David Mertz, Ph.D.
Transformer, Gnosis Software, Inc.
January 2001

In this column David Mertz gives an up-to-date review of a half-dozen leading XML editors. He compares the strengths, weaknesses and capabilities of each -- especially for handling text-heavy prose documents. The column addresses the very practical question of just how one goes about creating, modifying, and maintaining prose-oriented XML documents.

Contents:

Working with marked-up prose

Perhaps it is obvious enough that the very first requirement of any approach to working with XML documents is assurance that we are producing *valid* documents in the process. When we use a DTD (or an XML schema), we do so because we want documents to conform to its rules. Whatever tools are used must assure this validity as part of the creation and maintenance process.

Most of the tools and techniques I discuss are also a serviceable means of working with more data-oriented XML documents, but the emphasis in this column is working with marked-up prose. A few main differences stand out between prose-oriented XML and data-oriented XML. Some XML dialects, moreover, fall somewhere between these categories, or outside of them altogether (MathML or vector graphic formats are neither prose nor data in the usual ways). Prose-oriented XML formats are generally designed to capture the features one expects on a printed page (and therefore in a word processor). While most such formats aim to capture semantic rather than typographic features (e.g. the *concept* "foreign word" rather than the *font style* "italic"), their connection to traditional written and read materials is close. On the other hand, data-oriented XML formats mirror more closely the contents of (relational) database formats; the contents can often be thought of as records/attributes (rows/columns), and one expects patterns of recurrent data fields.

In prose-oriented XML dialects, one tends to encounter a great deal of *mixed* content: In data-oriented XML dialects, one tends to encounter little or no *mixed* content. That is, most "data" is text that has character-level markup scattered through it. In terms of a DTD, one sees elements that look something like the following example (taken from an IBM *developerWorks* DTD used for tutorials):

Listing 1. Example from *developerWorks*

```
<!--Contains a paragraph of text.-->
<!ELEMENT p  (#PCDATA | code | img | br | i | b | a)* >
```

Of course, block-level markup is also used to provide overall organization, but character-level markup is important in prose-oriented XML formats in a way that it rarely is in data-oriented XML formats. In my experience, the dichotomy between these markup levels poses the biggest challenge for XML editing tools to handle gracefully.

Let's see how a few tools and approaches hold up under these requirements.

Use a text editor and a validator

The first approach to consider is what we might think of as the "zero case" (or "null hypothesis" to borrow from science, or "first principles" from philosophy) of XML document modification. XML documents, after all, really are just text files in the end. So why not use the tools for working with text files that have been highly refined, and about which a programmer already has a lot of knowledge and deeply felt opinions? If you are like me, no matter how much you like some of the other tools discussed, there will always be those occasional times when you just want to see what is *really* going on by using a text editor.

A good text editor for working with XML will have syntax highlighting that is generic for all XML dialects, and also probably the option of configuring something more specific for a given dialect. It will have flexible (maybe regular-expression based) search and replace capabilities. If the text editor can support *folding* (sometimes called "code hiding"), that often proves to be of great benefit, especially in handling large documents. Obviously, being able to perform operations on blocks is useful, whether it is indent/outdent, cut/copy/paste, or templating. And probably your favorite text editor (if you are a programmer) is easy to configure to call external programs that take the current working file as input.

With a powerful text editor in hand, a few guidelines make working with prose-oriented XML easier. Most of this is common sense, and most of it is the *same* common sense that applies when making your code in other programming languages look nice. Use indentation and whitespacing well; parts of an XML document should stand out as being what they are at a quick glance. Although XML itself is whitespace retentive, most applications normalize whitespace before further processing. So indenting and adding vertical space are not functionally important, but just make your XML documents easier to look at (if the few extra bytes are that important, you probably made a bad choice by choosing XML in the first place; it is quite verbose compared to other formats). In general, character-level elements are best kept inline, and block-level elements are best starting new (hierarchically indented) lines. Obviously, use judgement rather than hard-and-fast rules for what looks neatest, but value clarity in appearance.

Text editors will require an external tool to perform actual validation. A lot of good ones are out there for various platforms. I personally usually use the Python module *xmlproc*. Whatever you use, your validator should give you sufficiently specific advice to make problems easy to locate and fix (usually including exact position of the errors and warnings). For example, the below is part of a validation I ran:

A sample validation

```
c:\xml-files\>python xvcmd.py test.xml
xmlproc version 0.62

Parsing 'test.xml'
E:test.xml:6:110: Attribute 'zip_file_comment' not declared
E:test.xml:45:152: '<' not allowed in attribute values
E:test.xml:55:34: '=' expected
E:test.xml:55:49: One of '>' or '/>' expected
E:test.xml:60:14: End tag for 'body' seen, but 'p' expected
E:test.xml:783:12: Premature document end, element 'p' not closed
[...]
Parse complete, 22 error(s) and 0 warning(s)
```

Editing and validating works much like an edit/compile or edit/unit-test cycle in a programming language. If you validate frequently, cleaning up a few errors is not difficult or time consuming.

Basic enhanced editors
Basic XML editing tools effectively add three things to a generic text editor to make them more tailored for XML editing:

1. Integrated validation of documents
2. Hierarchical (tree) views of XML documents
3. Integrated "preview" of transformed XML documents (to HTML, using XSLT or CSS2, generally)

Particular tools might offer a subset of these enhancements. I have looked at three specific tools in this class (but others probably exist).

Microsoft *XML Notepad*:
XML Notepad is more or less what you would expect from its name. It is a free tool -- for Win32 only -- that provides a parsed and structured view of XML documents. In addition, *XML Notepad* will perform validation (only when the document is opened) against a declared DTD. The basic interface to *XML Notepad* consists of a collapsible tree view of an XML document in one pane, with the other pane containing lines indicating the content of elements and attributes. Editing elements or attributes is strictly one line per text block, which means you have to scroll left/right within an entry area for a paragraph. Overall, *XML Notepad* is probably more suitable for editing data-oriented XML documents than prose-oriented ones. But as a quick way to look through or make minimal changes in a prose-oriented document, the tool is not bad. *XML Notepad* does not perform any sort of transformations, preview, or visual formatting.

Wattle Software - *XMLWriter*:
XMLWriter is a step up -- or maybe sideways -- from *XML Notepad*. It is also a Win32-only product, but it is shareware; you have to pay AU$75 (about US$40) to use it after a trial period. There is much more to the *XMLWriter* interface than the *XML Notepad* interface. One noticeable exception in particular is a navigable tree/hierarchy view. *XMLWriter* allows you to do a preview-only tree view, but does not allow you to edit in that view or use it to jump to a navigable pane. It is also based on the somewhat flaky MSXML DLL for this function, so it doesn't always work. However, in most respects, *XMLWriter* is a fairly rich working environment for modifying XML documents.

The basic idea of *XMLWriter* is close to that of a text editor, with some XML-specific extras thrown in. The view of a document is a textual view, with syntax-highlighting, optional line numbering, and the general features one would expect in a decent text editor (but with fewer bells and whistles than in a generic programmers' editor). However, beyond the text editing features, *XMLWriter* has options for validation, for checking how well-formed the text is, for XSLT conversion, for browser preview, and a concept of "projects" (collections of related files: XML, XSL, CSS, schema, etc). *XMLWriter* gives you a nice environment for working with XML that is still solidly in the "enhanced editor" category.

TIBCO Extensibility *XML Instance*:
XML Instance is a Java-based XML editor, and therefore is available on the range of platforms Java runs on. Java Virtual Machines (JVM) differ widely in quality and speed, and your satisfaction with *XML Instance* will depend partially on the JVM you use (IBM's have been the best in my experience, across several platforms). Of those tools I call "enhanced editors," *XML Instance* is the most sophisticated; its price of $100 matches that increase in features.

In some ways, *XML Instance* resembles a much better done *XML Notepad*. The same general interface is used, with a collapsible-tree to the left, and an attribute/element content inline to the right. But that is just the starting point. Each element can be viewed as either a sub-hierarchy or as raw markup. By looking at raw markup for block-level elements that include character-level elements, you achieve the right distinction visually between the types of elements. In addition, viewing an element is initially presented as a single line, but you can expand the view to include multiple lines (as many as needed with a scroll bar) on a per-element basis. This lets you both navigate and edit in a flexible way.

Also *XML Instance* does a good job (but not perfect) with validation checking and XSL transformations (including editing XSLT documents). *XML Instance* provides some rudimentary assistance in inserting subtags and attributes, but does not assist nearly as much, nor assure validity as well, as some of the other tools discussed in this column.

The aforementioned products can make a number of things easier, but these tools (and even the more sophisticated tools below) demand that you give up many of the powerful, but generic, capabilities, of a general text editor. Mostly, these tools are good choices for users who do not yet have a programmers' text editor they use daily.

XML development environments
Beyond basic editors, there are several tools available that one might describe as "integrated development environments (IDE)" for XML. *XML Instance* comes close to this category, but lacks several things available with other tools. To make it to this category, there are a few things a tool should have over and above the features provided by the enhanced editors. These tools provide much more assistance with creating valid XML. In most cases, the tools simply disallow the creation of invalid XML, while providing intuitive assistance about what elements and attributes can occur at a given place in a document. The other thing the XML IDEs do is add a wider variety of views of your XML documents, each suited to different specific purpose and type of document.

Icon Information Systems *XML Spy*:
XML Spy is another commercial tool for Win32; it costs about $150. When editing an XML document (or other file type), *XML Spy* provides you with two basic views: the Text View (which looks basically like any text editor) and the Enhanced Grid View (which is a structured schematic representation of the document). There is also a Browser View, which is just an external DDE call to Internet Explorer 5.5 or above; you cannot edit in the Browser View.

In both editing views, you are presented with onscreen prompts and shortcuts for entering valid tags and attributes. If you are using a DTD or schema, *XML Spy* can provide a lot more help. An information window describes the current context (element versus attribute, model, and occurrence rule). Additional "Entry Helper" panes show what subtags, attributes and entities are available in a context, and assist you in entering required or allowed elements. These helpers make your work a whole lot easier in trying to conform to a DTD (or in developing one).

One built-in tool of *XML Spy* that felt like actual magic was its automatic DTD/schema generation. From a typeless XML document, *XML Spy* can do a remarkably good job of inferring the underlying DTD (and creating and attaching it). I found that this generation was not perfect. For example, it believed I was using an enumerated type where there was really just CDATA. But the "errors" it makes are no worse than those a person who had only the XML document, and not the underlying design rules, would make. Cleaning up or customizing an *XML Spy* generated DTD is a lot quicker than writing one from scratch!

Another nice interface device is an option in the Enhanced Grid View. Most of the time, subtags are displayed in a hierarchical tree fashion. However, *XML Spy* will automatically detect when it thinks some data could better be displayed in a Database/Table View of repeating element sequences. You can also manually force one view or another. For working with what is basically tabular data to start with, using this view makes entering and understanding an XML document much easier.

SoftQuad *XMetal*:
XMetal is yet another commercial Win32 product, this one at a rather pricey $500. But it does quite a bit more than other tools. Like *XML Spy*, *XMetal* enforces validity as you work with documents (if a DTD/schema is used, of course). *XMetal*'s default editing interface is called the "normal" or "word processing" view, and will appear quite familiar at first (to Microsoft Word users especially). Like many tools of late, many of the toolbar icons, menu shortcuts, menus, and so on are copied from MS Office.

In addition to the "normal" view, *XMetal* offers several others, with most features borrowed from SoftQuad's popular *HotMetal* HTML editor. You can edit in a "plain text" view, which is just what you would expect in a text editor that knows a little bit about XML -- you get some syntax highlighting, line numbers, Windows/CUA clipboard operations, and search/replace features (including basic regular expressions). Unfortunately, and unnecessarily, some of XMetal's other capabilities are disabled in "plain text" view (i.e. "structure view" and context-sensitive element list).

The view that SoftQuad first invented (as far as I know) is the "tags on" view. This one is interesting, and quite useful. Each tag in the document, opening and closing, is represented by an arrow-like icon that contains the tag name. The words inside the tags are rendered in the style of the "normal" view, i.e. with different fonts, weights, colors, etc. This combination both lets you see without ambiguity what the markup structure of a document is, and also allows you to see a good approximation of a final rendering of an XML document. Obviously, if the particular XML document is being used in a database style rather than in a book/article/documentation style, the rendering has less literal significance (but it can still visually highlight categories of information in the XML). The element rendering is configurable using CSS.

Another optional view is available as a separate pane. The "structure view" is a hierarchical tree representation of the document being edited in the main pane. Navigation is synchronized between the two panes. While "plain text" or "tags on" view will show you the tag information, the "structure view"

helps make it more explicit. The structure view itself can take on multiple appearances, some more useful than others, but generally the options involve some helpful nested indentation and/or icons.

The one thing really missing from *XMetal* is built-in XSLT support. You can call external programs, and put that call in the menus, but for $500 one expects every bell and whistle to be included. I would also quibble with SoftQuad's reliance on the MSXML DLL that many Win32 tools seem to use (I've had many problems with that beta DLL).

Worth mentioning

A few other tools are worth mentioning in passing. The column will not have space to provide complete reviews or descriptions, but you can at least investigate further.

Morphon is a commercial Java XML editor/IDE. It is still in free betas as of this writing, but the release version is expected to cost around $150. *Morphon* presents a number of views, and is generally similar to *XMetal* or *XML Spy*. Like *XMetal*, *Morphon* relies on CSS for visual "WYSYWIG" presentation of documents. Betas have been a little unstable, but a lot of that depends on JVM issues. Being Java-based, *Morphon* is likely to have far better cross-platform support than the Win32-only products.

Xeena is a Java XML editor, like *Morphon*. But *Xeena* has the nice extra feature of being free (well, actually a free trial, as it comes with a 90-day license), and part of IBM *alphaWorks*' range of tools. Editing with *Xeena* is somewhat more restricted than with the best IDEs. You get good validation enforcement, and the "hierarchical tree plus line view" as in *XML Notepad* and *XML Instance*. But *Xeena* is not really as friendly for working with prose-oriented XML documents as are *Morphon* and *XMetal*. Like most IBM Java technologies, however, I have found *Xeena* to be stable and fast (which is more than is true of most Java applications).

Conglomerate is a free-software project based on GTK+. It compiles fairly easily on Linux, and in principle should also compile on Windows (since GTK is available). There is good news and bad news here. *Conglomorate* has the *absolute* best interface concept I have encountered for working with prose-oriented XML documents. It does not attempt to render character-level elements with CSS, but instead uses labeled and colored "underscores" for such elements. Block-level elements, in turn, are identified with similarly rendered vertical color blocks. Take a look at the home page for screenshots. The bad news is that *Conglomorate* is still basically alpha-level software. It crashes, and needs additional features. But it is one of the most promising starts I have seen; I hope someone (maybe one of you) advances the development of *Conglomorate* to being a free-software end-all XML editor/IDE.

A last possibility is worth considering. *Corel WordPerfect 2000* builds on the SGML support in earlier *WordPerfect* version to provide nicely structured XML support and validation. It is a powerful word processor in general, and by using it, you can use the same tool for XML document development.

Resources

- Webreference offers a nice general list (with short reviews) of XML editors.
- Also see David Mertz's reviews of a number of text editors.
- You can access Microsoft *XML Notepad* on the Microsoft Web site (MS does not always maintain URLs for very long, so try a search engine if this doesn't work by the time you read this).
- Check out Wattle's Web site for information on Wattle Software's *XMLWriter* and a free 30-day

evaluation.

- TIBCO provides information, as well as press releases and downloads on Extensibility *XML Instance*.
- You can 'test drive' the Icon Information Systems' *XML Spy*.
- Information about SoftQuad and the *XMetal* product is available on the SoftQuad Web site.
- Go to the Morphon Technologies site for screen shots and other information about *Morphon* XML/CSS Editor.
- IBM *Xeena* XML Editor is rated by other users at the *alphaWorks* site.
- Conglomerate: Note that the requirements description on the Web page is misleading; the Conglomerate archive available includes the necessary version of Fluxlib (so you do not need to download it separately as suggested).
- Also see David Mertz's previous columns:
  - XML Matters #1 introduces the Python `xml_pickle` object.
  - XML Matters #2 describes how to use Python's `xml_objectify`.
  - XML Matters #3 introduces DocBook.
  - XML Matters #4 continues how to build a legacy document archive with DocBook.
  - XML Matters #5 explains how to convert an XML document to HTML through XSLT.

About the author

David Mertz must have mislaid his MacGuffin in one of his other articles. It is bound to show up again soon. David may be reached at mertz@gnosis.cx; his life pored over at http://gnosis.cx/publish/. Suggestions and recommendations on this, past, or future, columns are welcomed.