A Brief Overview of XML

Timothy W. Cole and Thomas G. Habing
Grainger Engineering Library Information Center
University of Illinois at Urbana-Champaign

INTRODUCTION: What is Text Anyway?[1] This may seem an overly philosophical question with which to begin a practical briefing on XML (Extensible Markup Language), but the quickest way to understand XML is to comprehend the model of text on which it's based. To digitize a text you first need a systematic approach to representing it as bits and bytes. The model of text on which you base this approach will determine the usefulness and define the limits of the digital representation.

For instance, digitizing a text as a set of bit-mapped page images can be a valid approach in certain situations. This approach assumes a model of text as pages in a book, journal, or other print artifact. Assuming sufficient screen resolutions and scanning quality, you can use this approach to make texts previously available only on paper obtainable and readable online. But this approach does not expose the intellectual structure and content of the text in a way that facilitates machine manipulation or search and retrieval across a collection of texts. It does not facilitate updating, modifying, or annotating the text. It does not make possible the decomposition of the text into component elements (other than pages). It does not facilitate linking between documents or parts of documents.

Representing a text as an undifferentiated stream of alphanumeric characters, punctuation characters, and spaces assumes another kind of model of text. This model enables some machine manipulation of the information contained in a text, but makes it harder to display the text properly and really doesn't provide enough granularity of structure to be useful for complex computer processing. It's a valid model as well, but only for very limited uses (e.g., for the body of a simple e-mail message). Additional structure is required to differentiate the elements of a text, preferably in ways both machine readable and human understandable.

HOW XML RELATES TO SGML, PDF, WORD PROCESSOR FORMATS, AND HTML: XML and its immediate antecedent SGML (Standard Generalized Markup Language) are based on the axiom that any text can be represented as an ordered hierarchy of content objects (OHCO). According to this model, a journal article might include content objects such as title, author, author affiliation, sections, paragraphs, sentences, tables, figures, footnotes, etc. XML differs from SGML (which became an international standard in 1986) in that it's optimized for the Web, but both systems are the same in that both are designed to enable an author to accurately define and delineate the content objects of a text.

This makes XML different than most of the other high-level text-encoding schemes in use today. Word processor formats, PDF, and even HTML all focus first on detailing the appearance of a text. These other formats are concerned with font size and weight, placement of text on the page or screen, margins and indents, list numbering and bullet styles, etc. In XML such issues are all second-order. XML is designed for *descriptive* markup rather than *presentational* markup. Document instances expose intellectual structures, while ancillary files (e.g., stylesheets) are used to define presentation. What's most important in XML is the hierarchical structure of a text, not its appearance. (There are tradeoffs here, which we'll get around to later.)

Similarly, most non-XML encoding schemes lack or severely limit extensibility. If your word processor doesn't understand overstriking, then you can't mark text for overstriking using that application. HTML has a fixed set of tag names available. There's a 'paragraph' tag (**<P>**), but no 'sentence' tag. If you want to delineate sentences as well as paragraphs, you're out of luck using HTML. Afterthought mechanisms like **<SPAN>**, **<DIV>**, '**class**' and '**id**' attributes, allow for some extensibility of HTML, but only as a second-order characteristic. XML on the other hand allows you to define tag names and content object attributes as you go. Like SGML, XML is a meta-markup language. It prescribes the syntax to use when marking up a text, but it leaves the author free to specify the semantics used to identify and describe the content objects delineated in the markup. In the long-run this may be its most attractive feature, albeit also its greatest complexity.

Look at it this way. HTML was invented ten years ago for a limited purpose by a group of high-energy physicists working in CERN Switzerland. It is to their credit that the language they came up with has led to the World Wide Web of today, but it'd

---

[1] From ---- DeRose, S.J., D.G. Durand, E. Mylonas, & A.H. Renear (1997). What is Text, Really? *The Journal of ComputerDocumentation (ACM SIGDOC)* 21 (August 1997), 1-25. [Reprinted from *Journal of Computing in Higher Education* 1 (Winter 1990), 3-26.] See also commentaries that follow on pages 26-44.

be naive not to recognize that HTML is being warped almost beyond recognition on a daily basis by all the ways it's now being used. To continue to build and enhance the Web as a tool for everyday business, we need more than HTML by itself.

Another perspective — consider what HTML might look today like if it had been invented instead by law students interested in exchanging case notes and class assignments online among study group members. Possibly better, probably worse, but either way different. Maybe better for law students, not as good for hi-energy physicists. XML is designed to allow customization. XML provides intellectual communities an orderly way to transition from HTML to a set of markup semantics customized to their specific needs rather than to the needs of some other community (or every other community). Its underlying OHCO model of text limits XML somewhat, but in most regards it's wide open, much more so than just about anything else out there. With XML, the legal community has an opportunity to replace (or at least supplement) HTML with a schema (or a set of schemas) more like what HTML might have been if built for lawyers rather than physicists.

XML BASICS — CONTENT AND MARKUP: The fundamental syntax of XML is simple. It's a lot like the syntax of HTML. There's content and there's markup. Content consists of bytes representing spaces, punctuation, and other ASCII (unaccented Latin) characters and numerals. You represent non-ASCII characters using entities. Entities are multi-byte strings that start with an ampersand and end with a semicolon. Thus **&copyright;** can be used to represent the © symbol (assuming proper entity declaration is made). A special class of entities is used to represent Unicode characters. By definition all conforming XML applications must understand and be able to deal with Unicode entities without requiring entity declarations. Entities may also be used in XML content to represent other objects bigger than a single character (e.g., proper names or non-text objects like embedded graphics). Entities can even be used to represent ASCII characters and in fact the ASCII characters <, >, and **&** must be represented using entities whenever these characters are used within XML content.

XML markup (tagging) identifies and delineates the content objects of an XML document. Markup can be differentiated from content because markup is contained between angle brackets (< and >). The start of a content object in XML is indicated by a markup structure called an 'open tag.' A corresponding 'close tag' indicates the end of the content object. An open tag is markup consisting of at least a tag (content object) name. The matching close tag is that exact same tag name (in the same case) proceeded by a '/' character. There *must* be a close tag for each and every open tag that contains content (this is stricter than HTML). XML does allow 'empty' tags (that is markup that contains no content -- like an HTML img tag), but such empty tags must include a '/' character as the last character before the closing angle bracket. In addition to the tag name, open tags and empty tags may include attribute name-value pairs that further describe the particular content object associated with the tag. Attribute name-value pairs consist of an attribute name, followed by the '=' character, followed by the attribute value enclosed in quotes. (The requirement for quotes is strictly enforced.) There are rules for construction of tag and attribute names, but those are too esoteric to go into here. Adequate flexibility is provided for most any purpose.

The following are snippets of acceptable XML:

```
<Heading type='article title' id='Article001'>My Article Title</Heading>

<FirmName type="legal" id="Firm027">Cole, Kohl, &amp;amp; Colman</FirmName>

<img src="MyPicture.gif" />
```

But the following is *NOT* acceptable XML (because the case of the tag name in the open tag differs from that used in the close tag and because the attribute values are not enclosed in quotes):

```
<name type=person id=Name025>Cole, Timothy</NAME>
```

You can get away with being sloppy in HTML, but not in XML.

The other key to understanding XML syntax is to remember that it assumes a simple, hierarchical organization of content objects. This assumption allows for nesting of content objects, but does *not* allow for overlap of objects. A single, true hierarchy must be preserved throughout an XML document instance.

Thus the following, which has a non-overlapping object model, is acceptable XML. (Any implied analogy between the way the blind men approached the elephant and the way we collectively approach XML is intentional.)

```
<PoemFragment author='John Godfrey Saxe'>
        <Stanza refNumber='Stanza2'>
                <Sentence>
                        <Line>It was six men of Indostan</Line>
                        <Line>To learning much inclined,</Line>
                        <Line>Who went to see the Elephant</Line>
                        <Line>(Though all of them were blind),</Line>
                        <Line>That each by observation</Line>
                        <Line>Might satisfy his mind.</Line>
                </Sentence>
        </Stanza>
</PoemFragment>
```

But this, which contains overlapping content objects, is *NOT* acceptable XML:

```
<PoemFragment author='John Godfrey Saxe'>
        <Stanza refNumber='Stanza2'>
                <Line><Sentence>It was six men of Indostan</Line>
                <Line>To learning much inclined,</Line>
                <Line>Who went to see the Elephant</Line>
                <Line>(Though all of them were blind),</Line>
                <Line>That each by observation</Line>
                <Line>Might satisfy his mind.</Sentence></Line>
        </Stanza>
</PoemFragment>
```

OVERHEAD COSTS — STYLESHEETS, DTDs, AND XML SCHEMAs: There's one more thing you need to know about XML, particularly if you're going to be authoring XML. All this flexibility comes at a price.

First off, because semantics (tag names) are left up to the author, and because the emphasis is on describing intellectual structure rather than appearance, every time you create a new set of semantics, you also have to create the rules for presenting data marked up using those semantics. XML tags have no inherent presentation attributes. Creating stylesheets to define how your XML document(s) should be presented represents added effort and therefore added resource expenditures. How much depends on the complexity of your presentation and your XML schema.

At present there are two competing standards for XML presentation stylesheets. One approach is XSL-FO (Extensible Stylesheet Language — Formatting Objects). This approach is unique to XML and the XSL-FO standard remains unfinished at present. Fortunately the second approach is to use CSS Language (Cascading Stylesheet Language) stylesheets to define presentation. Those of us who've found the built-in rules for HTML presentation insufficient are already using CSS to create better looking HTML web pages. HTML CSS authoring skills translate well when authoring XML CSS stylesheets. The CSS approach also seems to be favored by web browser and application vendors. As with HTML CSS stylesheets, XML CSS stylesheets may be applied to a whole class of XML documents -- assuming the members of the class were all created using the same semantics.

The other overhead expenditure inherent in XML is the process of actually defining the XML semantics to use for your particular application. Again, you can spend a lot of resources on this task or very little resources. If the semantics you define are simple and you're the only one who'll be authoring XML documents using your set of semantics, you may be able to keep it all in your head. As long as you follow the syntax rules of XML, your XML documents will be recognized as "well-formed" and applications like web browsers will deal with your XML documents just fine. On the other hand, information interchange is at the heart of most XML applications. At some point, you'll probably want to write down your tag names and the rules of your hierarchical structure. Doing so will allow other users to better understand your XML documents and will allow

sophisticated XML software applications to "validate" your documents. (This can help ensure consistency and quality control, especially during the authoring process itself.)

Again, there currently are two ways to document your XML semantics. One approach, inherited from SGML, is to create a DTD (Document Type Definition). A simple DTD for the above poem fragment XML example is given below. A DTD defines all allowed tag names, attribute names, and entity names in your semantic scheme. You also define "content models" which describe the allowed hierarchical structures of your XML documents. The DTD approach works, but it has drawbacks. In particular DTDs are not themselves "well-formed" XML documents. Also, the mechanisms for data typing and defining content models within DTDs are limited and imprecise.

```
<!ELEMENT     PoemFragment    (Stanza) >
<!ATTLIST     PoemFragment
      author              CDATA                    #REQUIRED >
<!ELEMENT     Stanza          (Sentence)+ >
<!ATTLIST     Stanza
        refNumber       ID                       #IMPLIED >
<!ELEMENT     Sentence                (Line)* >
<!ELEMENT     Line                    (#PCDATA) >
```

Illustrative DTD for PoemFragment Example

In the long-run DTDs will be supplanted by uniquely XML rule documents called XML Schemas. XML Schemas are well-formed XML documents and you write them to conform to a set of semantics called an XML Schema Language. Proposed XML Schema Languages allow for more precise data typing and content modeling than the XML DTD Language specifications. However, XML Schema documents aren't yet common because work continues on finalizing the definition of a universal XML Schema Language.

CLOSING THOUGHTS: So why go through all this? Well, as described above, there's the common-sense argument that HTML can only be stretched so far. A successor technology clearly is needed if the potential of the Web as a tool for business and enhanced global communication is to be realized. XML offers a promising migration path.

As well, there are specific benefits anticipated from XML in the areas of document management and document quality control, improved and more reliable information interchange, and enhanced search and retrieval functionality. There's also an expectation that the built-in extensibility of XML, in combination with it being a non-proprietary community standard, will give the format longevity. As more and more primary source material is published only in digital format, the potential long-term costs of archiving electronic documents loom large. Anything that would reduce the required refresh frequency of digital archives will pay great dividends in the long run. SGML, which has been a relatively stable international standard since 1986, has already shown many of the potential benefits of this approach to text encoding. It is now hoped that XML will translate SGML to the Web in a manner that preserves all the best features of SGML.

There remain uncertainties. XML requires a large up-front investment, arguably a much larger up-front investment than was required for HTML. Are document publishers and producers of data ready to make this investment? XML also means a loss of some control. While proprietary formats aren't perceived as best for the common good, they can be good for individual vendors of document management systems. Display, authoring, and storage formats have been an integral part of the product package in the past. Are vendors ready to forego their investments in proprietary formats to support a generic, non-proprietary format over which they have limited control? Doing so means their products will now be judged solely on how well they manipulate that generic format. It levels the playing field in a way that may not please all vendors and service providers. Some publishers also are concerned about the lack of control over appearance of materials published in XML. Rendering engines for XML are not yet as precise as rendering engines for competing appearance-oriented formats. Moreover, the end-user has the ability to override publisher-provided stylesheets, something with which not all publishers are yet comfortable. Finally, two years after its formalization as a Web standard, there remain very few discipline-specific XML tools. Microsoft Internet Explorer and the latest preview release of Netscape Navigator now support XML, and there are a number of generic XML tools available (both for authoring schemas and documents), but (for instance) there is not an XML product geared for a lawyer's use in day-to-day practice. The various communities are still defining discipline-specific schemas. Until that's done, discipline-specific tools must wait. Long-term success seems likely, but is not yet assured.