Crossroads
The ACM Student Magazine

XML: The Future Of The Web

by ***John B. Bedunah***

Introduction

Although **XML**, or **Extensible Markup Language**, is the new kid on the block as far as markup languages are concerned, it contains all the necessary ingredients to make the Web a powerful, versatile, and interactive medium. In short, XML is the future of the Web.

Markup Languages in a Nutshell

What is the difference between a markup language and a programming language?

Programming languages are dynamic. They process data through calculations such as sort, query, input, output, and render. In short, they input information, act upon it (in whatever ways the user needs), and produce results that a user can use. The problem is that if every single piece of information has to be hand coded, the program ends up being extraordinarily unwieldy and expensive to produce. Enter the idea of markup.

Markup languages are static. They do not process information. A document with markup can do nothing by itself. However, a programming language can easily process the information in markup for different uses. So essentially a markup language identifies similar units of information. In a sense, markup brings intelligence to a document so that applications can read and process them more effectively.

A Short History Of Current Computer Markup Languages

The Mother Of All Markup: SGML

In the 1960's, lawyers were wading through hundreds of cases to find the specific cases they needed. Furthermore, there was no way of merging these cases with those that had already been found and processed. Luckily, a bright young lawyer discovered a way to automate this process and save attorneys considerable time and money. Charles Goldfarb, who headed a research team (1969) at IBM at the time, along with Ed Mosher and Ray Lorie, produced what was then called GML (Generalized Markup Language). GML automated the law office and Charles Goldfarb went on to create SGML (Standard Generalized Markup Language, 1974) which automated documents in general [1].

Instant World Wide Communication: HTML

In 1989 Tim Berners-Lee, a scientist, took a page from Goldfarb's SGML to develop a hypertext document that could be linked to and read anywhere in the world by using computers. Berners-Lee's intention was to help scientists share information quickly from any location [2].

The Empowered Web: XML

HTML's phenomenal success was due partly to its simplicity. It was easy to learn and easy for web developers to implement. However, developers envisioned a more powerful web. The HTML document is inflexible, and it had not been designed to be interactive between server and client. In addition, the major browsers began to create their own extensions. HTML, as a world wide standard, weakened.

At this point, the **W3C**, the World Wide Web Consortium, took action. Backed by twenty years of experience with SGML and HTML, some members designed a new markup language that was tailored specifically for the Web. Extensible Markup Language, a subset of **SGML**, featured most of the power of SGML with many of the more obscure features left out. In other words, XML was streamlined for Web use.

XML Features And Benefits

XML features and benefits can be grouped into four major areas:

1. XML is extensible
2. XML has precise and deep structures
3. XML has developed two general document types
4. XML has powerful extensions

The Value Of Extensibility

Technically, SGML and XML are meta-languages. They can create their own markup languages. HTML is simply a markup language derived from SGML. Documents created from HTML adhere to a generic set of tags, more correctly called elements. With HTML, a document author will always be subject to using the same tag set. The author will always have access to a <p> tag indicating a paragraph. What is wrong with this? What if a company needed to define some of its own elements? HTML cannot do this. XML is different. It is extensible, which means it can create its own elements. Because XML is extensible, documents can be customized according to the kind of information that needs processing. Hence, if the discipline is chemistry, a document type might be created marking elements like <ATOMS>, <MOL> (for molecule), <BONDS>, and <FORMULA>.

If a business needed an "order" document, it could create elements such as <ORDER>, <CUSTOMERNAME>, <CUSTOMERADDRESS>, <PURCHASEAMOUNT>, and so forth. These elements seem similar to those found in a database and that is correct. In XML terms, it doesn't matter if the document is a book, a manual, or an order, it is a document because it contains information that can be processed. Extensibility gives the document creator the power to customize the document to fit the needs of the business, organization, or discipline.

Since HTML lacks extensibility, a document designer cannot add elements that would be very helpful in processing information and managing documents.

The Added Value Of Precise And Deep Structure

One main difference between XML, SGML, and HTML is a feature called tag minimization. Tag minimization means that certain elements may have tags left out. For instance, if you have a paragraph tag <p>, you are required to have the opening tag, but you may leave out the closing tag </p>. This is convenient for document authors. However, this makes it harder to write programs which process the document, and thus drives up programming costs. One intent of XML was to make document processing software easy to write. In XML, elements must be marked precisely by using both an opening and a closing tag. In XML our paragraph element must be tagged thus:

```
<p>This is a one sentence paragraph</p>
```

No ambiguity exists in XML markup, thus programmers have clear structures to work with and applications are easier to write and maintain. When easy-to-follow structures are combined with extensibility, documents become flexible and reuseable. Reuseable because the same document might output different information to different users depending on their information needs. A supplier may need different information than a customer although some of the information will be the same. Using XML as a content medium enables companies to customize information for web use.

Deep structures exist in SGML and XML because each document has a root element and all elements must be nested within other elements. Applications can use these structures to achieve sophisticated content management in documents. HTML suffers from loose structures which make it difficult for computers to process information effectively.

The Advantage Of A Two Document Specification

The two main XML document types are:

1. The Valid Document
2. The Well-Formed Document

In SGML a document is always defined by a Document Type Definition, usually referred to as a DTD. The DTD is the grammar of the document. The DTD defines specifically what kinds of elements, attributes, and entities may be in the document and what non-SGML data may be imported into the document. DTDs define the order as well the occurrence of elements. In long documents like books and technical manuals, the DTD can become quite complex. Each time the document is processed, the DTD must be accessed so that the document can be parsed (checked) for validity. A document that adheres to all the rules of the DTD as well as other rules of the specification is a valid document. XML, unlike HTML, is very strict about adhering to these rules. Break a rule and the results are an invalid document. Web pages abound with sloppily written HTML code. It looks okay on the page but when the page needs to be processed for any reason, it then becomes a difficult project to undertake. XML takes care of this at the outset by denying document authors the ability to break the rules of the specification.

In XML, as opposed to SGML, there can be another kind of document called "The Well-Formed Document". The well-formed document does not have to adhere to a DTD. Basically, it must adhere to two rules about structure. First, each element must have an open tag and a close tag. This means that even the most simple XML document must adhere to the concept of precise structure. Second, there must be one root element that contains all other elements. For example, if a document is a Book document the root element might be called the <Book> element. Elements like the <Chapter> element and the <Paragraph> must not appear outside of the root element. In other words, elements must nest properly.

Non-validating parsers are relatively easy to write and may be used on well-formed documents. Well-formed documents come in handy if a document is simple and doesn't really need a DTD, or when processing simple structures over the internet where bandwidth is a consideration. A well-formed document does not have the overhead of a complex DTD. The concept of well- formed will come in handy when transforming existing HTML documents to XML. One of the main steps in transformation will be to make sure all elements have start and end tags (i.e. the HTML document becomes well-formed).

XML Extensions: Linking And Style

Although HTML's simple linking mechanism has made it easy for Web use, it could be improved to provide more powerful linking capabilities. XML provides this in the form of **XLink**. Links can describe different kinds of resources as well as contain multiple links instead of one-directional links like HTML. HTML provides poor searching mechanisms within the document itself but XML provides XPointer which supports finding elements at whatever granularity the document designer needs.

In XML, content and style are separated. This makes content easier to process. To specify style in XML you use a style sheet and the **Extensible Stylesheet Language (XSL)** A single document can have more than one style sheet. If the document needs to be rendered in a slightly (or even vastly different) way, the style sheet is changed and not the whole document. Thus, different style sheets could be used by different users. If a user had trouble reading a normal font size, then a style sheet could provide a document with a larger font size.

The Problems With Documents And How To Solve Them

A document can be an unwieldy piece of writing. Problems result when users need specific units of information in a long complex document. The problem is not only about managing documents, but also about managing specific units of information within the document (or "content management"). XML solves this problem for electronic documents. Rather than a whole document, the document becomes separate units of useful information waiting to be accessed and processed.

Today's documents are often multi-user documents. Different users need access to different sets of information though some information may be common to both. For example, the marketing department may need a different set of information than technical writers. How do you accommodate both groups? Markup identifies segments of useful information so applications can process them. If the markup has extensibility, this information is easy to identify. Thus, <CustomerName> is exactly what it says it is, a customer's name.

Another problem is that a document can be expensive to produce. The traditional document was written for one audience and for one use. Further, once the document was printed, any major additions or changes would entail a major revised edition. And then the material could be outdated before the book or manual ended up in the hands of the user. If information is stored on computers as electronic documents, then information can be updated instantly. Further, output can be done in parts. If the user already had 98% of the information he needed then he would only need to output the remaining 2% -- much less expensive, more timely for the user. A markup technology like XML aids exactly this and helps solve the expense of document creation because it makes documents reuseable and extends the life of a document because updates are much cheaper.

Another problem with traditional documents is "what you first see, will probably be all you will see" because redoing the document in a different style or for a different medium is prohibitively expensive. Some users may need larger print or may need the information in audio. XML solves presentation needs through stylesheets. It can also solve medium problems by creating markup languages like Java Speech Markup Language (JSML).

Managing Documents And Content

Documents contain **elements** and **entities** which can be reused however and whenever needed. Elements are the building blocks of the logical structure of the document and are important in managing content. Entities are the physical structure of the document. The **DOM** or Document Object Model is a way to manipulate the XML document, and it can be used with scripting languages like Javascript or VBScript. Using the DOM, content can be processed by using elements and attributes like objects

[3].

Information needs a write once, use anywhere method. XML meets this need. Once information is marked up in XML, it can be used as a middle tier between proprietary applications such as word processors.

Following is a sample document from a fictitious company that sells computer books on the internet.

```
<?xml version="1.0" standalone="no" encoding="UTF-8"?>
<DOCTYPE Order SYSTEM "Order.dtd">
<Order>
<Customer>
<CustomerName>
<LastName>Smith</LastName>
<FirstName>Steve</FirstName>
</CustomerName>
<CustomerAddress>
<Street>1279 Pecan Avenue</Street>
<City>Dallas</City>
<State>TX</State>
<Zip>78475</Zip>
</CustomerAddress>
</Customer>
<Book ISBN="0-14-820162-7" STOCKNUMBER="C1876">
<Title>XML in 30 Minutes</Title>
<Author>
<AuthorLast>Kramden</AuthorLast>
<AuthorFirst>Ralph</AuthorFirst>
</Author>
<Author>
<AuthorLast>Norton</AuthorLast>
<AuthorFirst>Ed</AuthorFirst>
</Author>
<BookPrice>$49.95</BookPrice>
<PayMethod Payment="MasterCard"/>
</Book>
```

```
        </Order>
```

Although this is a very simple document, it contains important information that can be processed easily because it is in XML. For instance, the online store might want to check what kind of books the customer has purchased in the past. Maybe the customer would like to be on a list where he will be automatically informed of any new books on XML or Javascript. By using Channel Definition Format **(CDF)**, an XML markup language, this is entirely possible and is being done currently. Perhaps the company would like to automate its purchasing system. Because the order document contains a book's ISBN and stocknumber in the form of attribute values, it will be easy to set up an automatic ordering system.

The beginning line states the XML version number which is presently "1," states that the DTD must be referenced because it does not "standalone," and lets the processor know what encoding scheme will be used which in this case is "UTF-8."

Different documents could be output to different departments by using XML. The shipping department might receive a shipping form, whereas the accounts receivable department would receive information on payment and amount.

Other elements that could be added to the above document are <ShippingInfo> for shipping information, <DateOrdered>, <DateShipped>, and <Carrier>.

How did we know what goes in the Order document? Everything that must go into a document is specified in the Document Type Definition or DTD.

```
        <!--Designed for an online bookstore specializing in computer books-->
        <!ELEMENT Order (Customer, Book+, PayMethod)>
        <!ELEMENT Customer (CustomerName, CustomerAddress)>
        <!ELEMENT CustomerName (LastName, FirstName)>
        <!ELEMENT LastName (#PCDATA)>
        <!ELEMENT FirstName (#PCDATA)>
        <!ELEMENT CustomerAddress (Street, City, State, Zip)>
        <!ELEMENT Street (#PCDATA)>
        <!ELEMENT City (#PCDATA)>
        <!ELEMENT State (#PCDATA)>
        <!ELEMENT Zip (#PCDATA)>
```

```
<!-- Book Information-->
<!ELEMENT Book (Title, Author+,BookPrice)>
<!ATTLIST Book
ISBN CDATA #REQUIRED
STOCKNUMBER CDATA #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (AuthorLast, AuthorFirst)>
<!ELEMENT AuthorLast (#PCDATA)>
<!ELEMENT AuthorFirst (#PCDATA)>
<!ELEMENT BookPrice (#PCDATA)>
<!ELEMENT PayMethod EMPTY/>
<!ATTLIST PayMethod
Payment (MasterCard | AmerExpress | Visa) #REQUIRED>
<!-- End DTD-->
```

Elements used within the document must be declared. The root element is Order. Every element must be contained within the root or Order element. Element content is placed in parentheses. Note the "+" sign. DTD's contain order and occurrence indicators. The "+" sign means that an element must occur once and may occur more than once. Some element may contain a "*" sign which means an element may occur 0 or more times. An example is an appendix in a book. It might not contain an appendix or it could have one or more. A "?" mark indicates an element could occur 0 or 1 times.

Notice that the Book element contains two attributes: *ISBN* and *StockNumber*. Usually attributes add information about the element and are often not seen in the final document but processed by the computer. Hence, an application could use the ISBN or StockNumber attributes to order a replacement book at the instant of each sale.

What is "#PCDATA"? It is the actual XML text or information. This is the information the user actually sees as output (text, etc.).

Attributes have types such as text, tokens, id, or enumeration. For instance, the PayMethod element has a Payment attribute that is an enumeration or listing of the possible pay methods. The customer must use either MasterCard, American Express, or Visa as a pay method. This is required information meaning the information must be included to have a valid document. Some attributes may have an *IMPLIED* value which means the user does not have to supply a value which instead may be supplied by an application. A value may also be *FIXED* so that a user must use the value and no other.

Lastly, an attribute may have a *DEFAULT* value. Elements and attributes are the logical structure of every XML document. A well thought out DTD leads to documents that can be processed efficiently and propagates effective business processes as a result.

One advantage of XML is that different encodings can be used. If a company engaged in international business, and it needed to use Japanese this would be no problem because XML uses **Unicode**. Think of Unicode as a world wide standard for writing almost any language known to man.

Attaining Physical Structure

One of the strengths of XML is that, like object oriented programming, the document can be organized into components. In the case of XML the physical structure or components are called *entities*. Entities, like elements and attributes, must be declared in the DTD. Entities can be any set of information like text or graphics.

An extremely helpful feature in XML (as well as in SGML) is the general entity. It is a very powerful text processing tool that can save a tremendous amount of time and money. General entities are symbolic representations of blocks of text. This lends itself to the replication of large blocks of text. For instance, attorneys often use large blocks of text in legal documents. This boilerplate text can be reproduced using a general entity. How is it done?

First the general entity must be declared in the DTD. Let's say an attorney uses a standard contract where he can just fill in the blanks with the proper names, etc.

Declare the entity in the DTD:

```
<!ENTITY SC "The party of the first part ,
______________________________, having agreed to said circumstances
on the 1st of April, 1998, and being cognizant of said condition did agree
to the setforth conditions.....">
```

The "SC" is merely a mnemonic to remember that it means "standard contract." The entity could have as easily be named:

```
<!ENTITY StandardContract "..........">
```

Now instead of typing this long statement into the actual instance of the document, the secretary inserts these keystrokes:

```
&sc;
```

When the application reads the &sc; entity it outputs:

> "The party of the first part, ____ , having agreed to said circumstances of the 1st of April, 1998, and being cognizant of said condition did agree to the setforth conditions...."

The entity can also be in an external file and referenced as such. The general entity is also very useful in changing text that is distributed throughout many documents in the computer's files. For instance, what if a company's address is spread over hundreds of documents in the computer system? Normally these addresses would have to be changed manually, a time consuming and costly process. With the use of the general entity the process is easy.

If the company address has been stored in an entity like this:

```
<!ENTITY CompanyAddress "1166 Camp Bowie Blvd. Ft. Worth, TX
76116">
```

Then, one change in the general entity will change every instance of the address throughout the system. The change will occur in the DTD as such:

```
<!ENTITY CompanyAddress "2552 Hulen St. Ft. Worth, TX 76107">
```

The new address now appears in all cases of the documents within the system. Note that this also means there will be no typing mistakes due to replication.

Another advantage of the entity comes when two or more people need to work on the same document. For instance, if several authors were working on the same book, the book could be divided into chapters by declaring them to be entities. That way authors could work on their individual chapters without needing the whole book. The chapter entity must be declared in the DTD:

```
        <ENTITY Ch1 SYSTEM "Ch1.xml">
```

To access Chapter 1 of the book:

```
        <Book>
        <Chapter>&Ch1;</Chapter>
        </Book>
```

Once the parser reads the Chapter 1 entity it retrieves the text from Chapter 1 and the author can work on it while another author may be working on Chapter 5.

Conclusions

Web users will benefit greatly from the features of XML. Dozens of markup languages are being created to help develop documents that will be useful to document users. Some already created are:

MathML	Mathematical Markup Language
CML	Chemical Markup Language
CDF	Channel Definition Format
OSD	Open Software Description
RDF	Resource Description Framework
MCF	Meta Content Framework
XML/EDI	Electronic Data Interchange

References

1
Floyd, Michael. A Conversation with Charles F. Goldfarb. *WEB Techniques* (Vol.3 Issue 11), pp.38-41, November, 1998.

2

Goldfarb , Charles F. and Prescod, Paul. *The XML Handbook*. Prentice-Hall,Inc, Upper Saddle River, NJ. 1998.

3

Pitts-Moultis,Natanya and Kirk, Cheryl. *XML Black Book*. The Corolis Group, Scottsdale, Arizona, 1999.