

Introduction to XML

Why XML?

What is it that gets developers excited about XML, and makes them want to use it? XML is not dependent on any particular operating system; it's designed to survive IT changes such as a move from Microsoft NT to Linux. It allows you to be able to repurpose data quickly without moving to a new architecture or language. And it allows you to build your own applications. It's also an open standard; you don't need to pay anyone anything to use it; no special programs are required to produce xml (just a text editor), and no single company controls it.

XML lets you create domain-specific markup languages (called XML applications.) Some that already exist are WML (Wireless Markup Language), CML (Chemical Markup Language), MathML (Math Markup Language) and VoxML (by Motorola, Voice Markup Language for phone systems.) You may even be familiar with SVG which is an XML application for delivering scalable vector graphics (thus the name) to web browsers. You also see XML used a great deal for configuration and customisation files.

These are very specialized markup languages that address specific needs. These needs could not be adequately met by HTML. XML is also an excellent format for data interchange. It is non-proprietary and easy to read and write. XML can be authored just like HTML – in a simple text editor. Furthermore, XML files are merely ASCII files. This means not only is it human readable, but it has a very small footprint. This makes it ideal to be sent back and forth between applications. XML is ideal for large and complex documents because the data is structured. You can place requirements on tags, integrate data from multiple sources and show or hide data depending on user actions.

In the real world companies have been saving millions of dollars by using XML. Dell created custom ordering pages for some 27,000 B2B customers and used XML to update each customer's internal procurement system as well as Dell's order system each time the customer made a purchase. That system eliminates the need for the customer's workers to enter the order information by hand on their own systems after they ordered at Dell. Dell estimates that some of its largest accounts will see savings of as much as \$4 million annually.

Cisco uses an XML-based system for its product ordering (which is build-to-order) which allows order information to be communicated directly to its own manufacturing plants and its entire supply chain of distributors and other assembly partners. With their system in place, this process takes a matter of minutes, cutting product lead time from an average of four to six weeks to two to three weeks. Cisco claims to save an estimated \$175 million annually from this project.

What is XML?

XML stands for Extensible Markup Language. However, XML is not just "another" Markup Language to compete with HTML. HTML defines a fixed set of tags that describe a fixed number of set elements, and this is a limiting factor.

XML is a *meta-markup language* – in other words, XML *defines* markup languages. You make up the tags you need as you go along. Although the tags must adhere to some general principles, you are free to make tags for what you need at will. For example, if you are working with your CD collection and you want to describe artist, title and song, you can create tags for each of these without having to try to fit the data into paragraphs, ordered lists, and so on.

For example:

```
<CD>
  <BAND>Rush</BAND>
  <TITLE>Signals</TITLE>
</CD>
```

XML is a set of rules for defining significant tags that identify different parts of a document in a custom built markup language. The markup language may have been authored publicly, by you or by another individual. A good example of a custom markup language (called an XML application) is WML (Wireless Markup Language.) WML is defined by XML, and is of itself what is called *well-formed* and also *valid* XML. WML pertains only to wireless devices and was authored by a public organization.

Lastly, XML markup describes a document's *structure* and *meaning*. It does not describe its presentation to the user – formatting, fonts, and so on. In other words, XML contains tags that say what is in the document, not what the document looks like. The presentation of an XML document is handled using style sheets and/or XSL (Extensible Style Language.)

If you are thinking this sounds something like XHTML – you are on the right track! XHTML uses some of the same rules that XML does. For example, every tag must be closed – this comes from XML. XHTML is meant to be able to be "parsed" by a XML parser so that the information in the XHTML document can be interchanged in a similar manner as XML.

What is "Well Formed" XML?

XML is a little more rigid in a sense that it demands you write *well formed* code. Well formed code requires you to close all tags, have a root element and quote attribute values. While on the surface it seems this rigidity makes for more work, the truth is XML simply demands you write good solid code.

A Rules Summary for a Well Formed XML Document

1. The XML declaration must begin the document
(`<?xml version="1.0" standalone="yes"??>`)
2. Elements that contain data must have both start and end tags
3. Elements that do not contain data and use only a single tag must end with a `/>`
(i.e. `
`)
4. The document must contain exactly one element that completely contains all other elements (known as the "root" element. Just like `<html>` is the root element of a HTML document)
5. Elements may nest but may not overlap
6. Attribute values must be quoted
7. The characters `<` and `&` may only be used to start tags and entity references respectively
8. The only entity references which appear are `&`, `<`, `>`, `'`, and `"`. (Entity references encode special characters – such as an apostrophe – so they are readable by any application. If you've authored HTML, you have probably already seen these too. You simply can't use any other entities than the 5 in the list).

As you can see, you are not forced to do anything that would not be considered good coding practice. If a document is not well formed, most attempts to read or render it will fail.

What is "Valid" XML?

Valid XML adheres to a specific DTD (Document Type Definition – see "What is a DTD?" below) of a defined markup language (such as WML.) The parser/processor will check the XML sent to it against the DTD to make sure that the rules specified are followed. If they are not, it is declared invalid. A well formed XML document may not be a valid one (it may not be attached to a DTD or it may have errors, thus being invalid) however, a valid XML document will always be well formed.

What is a DTD?

A DTD (Document Type Definition) provides a list of the elements, attributes, notations and entities contained in a XML document, as well as their relationships to one another. DTDs specify a set of rules for the structure of a document. For example, a DTD may indicate that a <ARTICLE> element may have exactly one <BYLINE> element, and it may or may not have a <IMAGE> or a <FOOTNOTE> element. The DTD accomplishes this with a list of markup declarations for particular elements, entities, attributes and notations.

DTDs can be part of the XML file or linked externally. For the types of XML applications you will see at Scient, probably all DTDs will be linked externally. External DTDs can be shared by different documents and Web sites and provide a means to agree upon and adhere to markup standards set by a group or organization. For example, on the Sageo project, the content for the site was being written by Hewitt employees. In order to make sure that all content met fixed and agreed upon structure, the content writers authored their work in XML using a WYSIWYG editor. This editor would only allow them to use the tags that were set forth in the DTD. Style sheets took care of the presentation to the user. XMLSpy is probably the most well known of this type of editor (see http://www.xmlspy.com/download_spy_enterprise.html).

DTDs can also help ensure that different people and programs can read each other's files. If a domain specific organization were to agree on a single DTD for their documents, they could be assured that everyone in the organization could read and understand each document. Because the DTD defines exactly what is and is not allowed to appear inside a document, the DTD establishes a standard for the elements that viewing and editing software must support.

The DTD also allows you to see the structure of a document without actually viewing the data. This makes it easy for you to swap in and out different styles and formatting into the structure without destroying or having to manipulate the data. And this structure can be manipulated by JavaScripts, CGIs, servlets, databases and other programs – again – separate from the data.

The separation of structure and presentation is very important in data exchange. You want the data to be able to be quickly parsed and digested by the application. Having presentation code in your data slows this process down. On the flip side, it's much more difficult to keep up and author your presentation with all that data in there. This is the same concept behind keeping XHTML and CSS separate. The same benefits apply.

Lastly, you can also use DTDs (Document Type Definitions) to define glossary entities that insert boilerplate text. In other words, it's a quick and easy way to place in static text without worrying about human error in reproducing it each time. This way you know that data entry clerks are adhering to the format that you need, and you can migrate data to and from relational and object databases without fear of data corruption. Although the freedom you have in making up your own tags and structure in the document of XML is great, a lot of the real power and extensibility in using XML lies within what you can do with DTD.

Next time, I'll talk about authoring DTDs. Don't panic; it's easy!

XML Authoring

Introduction

As stated above, XML is authored just like HTML. You can use the most basic text editor. XML can also be created on the fly by another application. For example, a database could accept some data and then have a calculation convert the field to XML.

After all the actions that are to be performed on the XML document are complete, it will be sent to a parser, or processor. The parser reads the document and verifies that it is well formed. If it is not well formed, it will display an error but it will *not* attempt to fix it or render it. The parser may also check to see that the document is valid, though this is not required.

Most modern browsers contain parsers (IE 6 and the Mozilla to name two.) They can parse the XML, and if it has no errors then the XML code will be displayed in the browser window. If the author has attached a CSS or XSL style sheet, then you will see the data presented as formatted, just like you would see an HTML page. However, other programs can receive the data. A database can receive XML or a Java program may be launched – XML is very flexible in this manner.

Hello XML

Writing XML code is very straightforward:

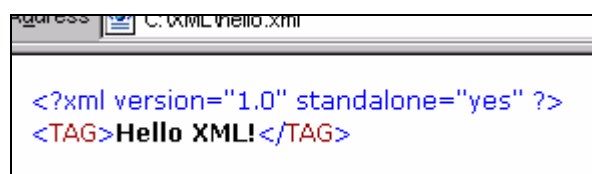
```
<?xml version="1.0" standalone="yes"?>
<TAG>
Hello XML!
</TAG>
```

All XML documents start off with a processing instruction (shown by the tag set `<? ?>`). The processing instruction above says this is an XML document, what version of XML it is, and that it is a standalone document (it is not using a DTD.) Adding a tag and content is quite simple. What is different from HTML is that you're using tags that you have defined or have been defined in a DTD, and that you must write well formed code as described earlier.

Note that XML is case-sensitive. If you do upper case it has to stay that way. It's kind of personal preference and depends really on what you are doing and how readable it is.

Viewing XML

In order to view your code, you can simply open the file in your browser. Since this code has no style sheet attached, you should see this:



XML naked and unashamed in the browser

Color coded code is displayed when you are not using a style sheet, and when no errors are reported. If this document had not been well formed you would have gotten an error instead. Not a helpful error mind you – but an error nevertheless.

Assigning Meaning to XML Tags

XML tags can have two types of meaning: structure and semantics. The structure of Hello XML above is quite simple: a root tag with one piece of content. Semantics has to do with the meaning of the tag. <TAG> means little to a person who might look at this document. If I renamed it <GREETING>, it makes more sense – Hello is a greeting. Semantic meaning exists outside the document and is more subjective. Thus tags such as <AUTHOR> and <PLANET> have semantic meaning – much more than <P> or <DD> in HTML -speak.

Adding a Style Sheet to XML

To add a style sheet to our Hello example, we simply add another processing instruction. This instruction is very similar to how you would use a style sheet in HTML:

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="stylesheet.css"?>
<TAG>
Hello XML!
</TAG>
```

Your "Hello XML!" text will now be formatted according to stylesheet.css. If you load it into your browser you will see this:



XML, resplendently styled in the browser.

It is just as simple as adding a style sheet to HTML. However, note that you can not include inline styles or declare styles within the XML document – it must be done externally. This includes classes

More XML Code

XML is based on a containment model. XML elements either contain text or other elements (called children) but rarely both. Before you start thinking that is going to constrain you, realize there are many ways to organize data, and more data you have the more ways you have to organize it. Let's look at a small, three cd collection in xml:

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="cds.css"?>

<CD_COLLECTION>
  <CD>
    <BAND>Rush</BAND>
    <TITLE>Signals</TITLE>
  </CD>
```

```
<CD>
  <BAND>UFO</BAND>
  <TITLE>Lights Out</TITLE>
</CD>
<CD>
  <BAND>King's X</BAND>
  <TITLE>Tapehead</TITLE>
</CD>
</CD_COLLECTION>
```

Note the first two processing instructions, we say that this is a version 1.0 xml standalone document (it has no attached DTD) and we say we are using the cds.css style sheet. There is only one root element - `<CD_COLLECTION>`. You can never have more than one root element, and everything else must be a child of that element. You have probably noticed we have semantic names for our tags to give them meaning, and the style sheet will enhance the meaning and present the data to the user.

Attached is this style sheet:

```
CD_COLLECTION {display:block; font-size: 24px; font-weight: bold;}
CD {display:block; font-size: 18px;}
TITLE {display:block; font-size: 14px;}
```

When viewed through IE we see this:



XML, with funky styling, on the IE catwalk.

Not exactly beautiful, but you can see how the data and presentation are handled separately. More will be said about presentation in the next section.

Attributes and Elements

You may be asking why we couldn't have made the title of the CD an attribute of `<BAND>`, or even `<BAND>` and `<TITLE>` an attribute of `<CD>`. Well you could have done that. In fact, depending on what the goal of the project is making those two elements attributes might be the better way to go, but we don't know enough about the project to be sure. It will not always be a black and white choice of when to use attributes or elements, generally you will use what suits the application. However you can take these facts into account:

- Attributes do not hold structure well (elements can have substructure, attributes cannot)
- Elements allow you to include meta-meta-data (information about the information about the information – crazy I know but at times it may be necessary.)

- Not everyone always agrees on what is and is not meta-data
- Elements are more extensible in the face of future changes - elements are easier to manipulate than attributes, because you have to do more work to get the attribute of an element in XSLT.

Conclusion

A couple years ago when the Internet was flying, XML was the "next big thing." While it can be debated whether or not it became what everyone thought it would, the truth is that XML pervades any serious online endeavour, whether it is in server configuration files or a new fangled system sending data to a legacy system.

Furthermore, now that you understand XML a little better, you should understand XHTML a little better. The promise of XHTML is that it will eventually take advantage of all the benefits of XML. This is in fact what the W3C has in mind:

"This Recommendation specifies an abstract modularization of XHTML and an implementation of the abstraction using XML Document Type Definitions (DTDs). This modularization provides a means for subsetting and extending XHTML, a feature needed for extending XHTML's reach onto emerging platforms."
(<http://www.w3.org/TR/xhtml-modularization/>)

Who will make the next breakthrough on the web? You never can tell, but you can bet that XML (and probably XHTML) will be involved!

If you have comments about this article, or any issues you would like to see covered in this JavaScript column, please send them to Tom at pixelmech@yahoo.com