XML Basics (part 1)

By icarus

Table of Contents

Deconstructing The Silver Bullet

Unless you've spent the past few years down a rabbit hole, you've already heard about XML, the W3C's effort to create an extensible toolkit to store and manage different types of data. By defining a set of rules to organize collections of data, and then developing a set of technologies that can work with these organized collections, XML is a serious attempt to simplify the task of data management.

From the release of the first working draft of the XML specification in 1998, all the way through to its current incarnation, XML has been the subject of mass media hysteria, with technology pundits and business leaders alike proclaiming its virtues. XML, they say, is the silver bullet, the magic elixir that will cure all of humanity's woes...and help you make a profit in the bargain. Don't wait, they say; get on the XML bandwagon now and your customers will thank you for it.

Don't believe them.

XML is no silver bullet; it's a tool. A tool whose benefits lie primarily in the manner of its usage. Used correctly, XML and its related technologies can, indeed, make your life simpler, your processes more efficient, and perhaps even fatten your bottom line. Used incorrectly, it's simply a toy, a beautiful thing that you admire for a while and then go back to work. If you don't understand the basic concepts and principles of XML, and how to apply them to your requirements, your ability to exploit it will be, at best, limited.

Over the course of this series of articles, I will be focusing on core XML concepts − elements, attributes, namespaces, entities and the like − in the hope of offering a starting point to XML novices. In case you already know most of this stuff, fear not − at a later date, I will also be discussing the applications of XML and its related technologies − data exchange, transformations, linkages and more − together with more focused discussions of new and upcoming XML technologies.

This introductory article will discuss the origins and design goals of XML, the basic rules of XML markup, and how to use elements and attributes in an XML document. And class is now in session.

A Little History

XML, or Extensible Markup Language, is not new. In fact, it's a subset of SGML, the Standardized General Markup Language, modified for use on the Web. SGML was originally developed by Goldfarb, Mosher, and Lorie at IBM in 1969, as a way to structure legal documents; it has evolved over time into an international standard for representing textual data in system−independent format. Since SGML is overly complex for the requirements of the Web, XML has evolved as a modified (read: simpler) version of SGML, adapted specifically for use on the Web.

You might be thinking to yourself: isn't there already a universal language for the Web called HTML? And you'd be right to wonder...

While HTML is great for putting together Web pages, it doesn't offer any way to describe the data contained within those pages. As a formatting language, it doesn't offer any mechanism to define data structures within the document, thereby limiting its usefulness. The fact that it understands a limited set of tags − and even that frequently depends on which browser you're using − reduces its flexibility and makes it difficult to extend its usefulness to other applications.

XML was designed to avoid these disadvantages by creating a markup language which would be simple yet flexible, easy to use yet powerful enough to offer a variety of different applications. Briefly, the original design goals for XML (as stated in the W3C's XML 1.0 Recommendation) were: XML should be simple and easy to use.

XML should support a variety of different applications, by allowing users to develop their own markup.

XML documents should precisely follow certain formally−defined rules and principles.

XML documents should be human−legible and reasonably clear.

The Big Picture

They say that the whole is greater than the sum of its parts...and nowhere is this seen more clearly than with XML and its ancillary technologies. Over the past year and a half, the XML universe has grown by leaps and bounds to include many new technologies, most with hard−to−remember acronyms. Here's a quick list of the important ones, and how they fit into the larger picture:

XML Schema: XML Schema makes it possible to define the structure and format of "classes" of XML documents, providing more advanced features than those offered by the regular Document Type Definition (DTD). Find out more about it at http://www.w3.org/XML/Schema

XLink: XLink is a specification for linking XML data structures together, in much the same way as the hyperlinks available in HTML...although XLink allows for far more sophisticated types of links, including simultaneous links to more than one resource. Find out more about it at http://www.w3.org/XML/Linking

XPointer: XPointer is a specification for navigating the hierarchical tree structure of an XML document, and referencing elements, attributes and other data structures within the document. Find out more about it at http://www.w3.org/XML/Linking

XSL and XSLT: The Extensible Stylesheet Language (XSL) makes it possible to apply presentation rules to XML documents, and convert − or transform − them from one format to another. Find out more about it at http://www.w3.org/Style/XSL/

XHTML: The next version of HTML, XHTML combines the precision of XML markup with the easy−to−understand tags of HTML to create a more powerful and flexible language. Find out more about it at http://www.w3.org/MarkUp/

XForms: XForms offers a way to improve the current crop of HTML−based forms by separating the function of the form from its appearance, thereby making it possible to easily adapt a form for display on a variety of devices and systems. Find out more about it at http://www.w3.org/MarkUp/Forms/

XML Query: The XML Query effort is focused on creating a specification that makes it possible to query one or more XML document(s) and generate usable result data (in much the same way as SQL is used to retrieve database records.) Find out more about it at http://www.w3.org/XML/Query

XML Encryption: XML Encryption is a means of encrypting and decrypting XML documents, so as to secure it against unauthorized usage. Find out more about it at http://www.w3.org/Encryption/2001/

The list of XML−related technologies keeps increasing, and you should always refer to the W3C's Web site at http://www.w3.org/ for the latest information.

The Hammer And The Chisel

Before beginning any development effort with XML, you should make sure that you have the right development environment and tools.

The first − and most important − development tool is the XML editor. Since XML is a set of rules which allow for the description of textual data, XML documents can be created with any text editor (just like HTML.) On a UN*X system, both vi and emacs can handle XML documents, while Notepad remains one of my favourites under Windows. If you prefer something a little more user−friendly, take a look at XMLSpy, a powerful and full−featured XML editor, at http://www.xmlspy.com, or XMetaL at http://www.xmetal.com/

Both Microsoft Internet Explorer 5.0 and Netscape Navigator 6.0 come with built−in XML support, and can read and display an XML document in a hierarchical tree view. Since most systems come with either or both of these installed, you don't need to look very far if you need a tool to simply display an XML document. In addition to these, both Opera and the W3C's Amaya browser now have support for XML documents.

It should be noted at this point that since one of the primary purposes of XML is to describe data − not present it − browser support is not an essential requirement for XML usage. Since XML is an open standard, it can be used to package data into structures that are easily transferable from one system to another. Consequently, you don't need to constrain yourself to a browser to validate XML data − James Clark's expat parser, at http://www.jclark.com/xml/expat.html, and Tim Bray's Lark parser, at http://www.textuality.com/Lark/, will both do the job for you.

In addition to the general−purpose tools listed above, there are a huge number of specialized little programs floating around the Web. As this series narrows its focus, I'll be identifying the tools most suited for specific applications; however, if you can't wait, drop by http://www.garshol.priv.no/download/xmltools/, a frequently−updated list of free XML software, and download to your heart's content.

Lights, Camera, Action!

OK, enough with the background − let's get our hands dirty. Consider the following XML document:

```
<?xml version="1.0"?>

<review>

<genre>Action</genre>

<title>X−Men</title>

<cast>
<person>Hugh Jackman</person>
<person>Patrick Stewart</person>
<person>Ian McKellen</person>
<person>Famke Janssen</person>
</cast>

<director>Bryan Singer</director>

<duration>104</duration>

<year>2000</year>

<body>Every once in a while, Hollywood takes a comic−book
hero, shoots him
on celluloid, slaps in a few whiz−bang special effects and
stands back to
see the reaction. Sometimes the results are memorable
(<title>Superman</title>, <title>Spiderman</title>,
<title>Flash
Gordon</title>) and sometimes disastrous
(<title>Spawn</title>, <title>The
Avengers</title>). Luckily, <title>X−Men</title> falls into
the former
category − it's a clever, well−directed film that should
please both
comic−book aficionados and their less well−read
cousins.</body>

<rating>4</rating>

</review>
```

As you can see, an XML document, like an HTML document, is simply an ASCII text file. This specific text file contains a recipe, broken up into different sections; each section is further "marked up" with descriptive tags to precisely identify the type of data contained within it.

An XML document may be either "well−formed" or "valid".

A well−formed document is one which meets the specifications laid down in the XML recommendation − that it, it follows the rules for element and attribute names, contains all essential declarations, and has properly−nested elements.

A valid document is one which, in addition to being well−formed, adheres to the rules laid out in a document type definition (DTD) or XML Schema. By imposing some structure on an XML document, a DTD makes it possible for documents to conform to some standard rules, and for applications to avoid nasty surprises in the form of incompatible or invalid data.

DTDs are essential when managing a large number of XML documents, as they immediately make it possible to apply a standard set of rules to different documents and thereby demand conformance to a common standard. However, for smaller, simpler documents, a DTD can often be overkill, adding substantially to download and processing time.

Breaking It Down

Every XML document must begin with a declaration that states the version of XML being used; this declaration is also referred to as the "document prolog."

```
<?xml version="1.0"?>
```

This document prolog may also contain additional information, such as the document encoding, and whether the document is to be viewed in combination with external DTDs or other entities (as explained above, a DTD lays down the format for an XML document and can be used to verify whether or not it is valid.) Consequently, the document prolog can sometimes look like this,

```
<?xml version="1.0" standalone="yes" ?>
```

or this.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

The document prolog also contains a document type declaration, used to specify additional information about the document. This information typically includes the location of the DTD to use when validating the document (if there is one available), an optional list of entity declarations (more on this later), and the name of the root element of the document.

A document type declaration usually looks like this:

```
<!DOCTYPE rootElement DTDLocation
[
entityDeclarations
]
>
```

A possible document type declaration for the movie review above might look like this:

```
<!DOCTYPE review SYSTEM
"http://www.somedomain.com/review.dtd">
```

In this case, the document would be validated against the DTD located at http://www.somedomain.com/review.dtd

If entity declarations are present, this might be modified to read

```
<!DOCTYPE review SYSTEM "http://www.somedomain.com/review.dtd"
[
<!ENTITY html "Hypertext Markup Language">
<!ENTITY xml "Extensible Markup Language">

]
>
```

Entities will be discussed in detail in the second part of this article.

Simply Element−ary

The document prolog is followed by a series of "elements". An element, which is the basic unit of XML, consists of textual content (or "character data"), enhanced with descriptive tags (or "markup"). The boundaries of an element are defined by start and end tags, and may contain additional descriptive "attributes".

Here are some examples of XML elements:

```
<title>XML Basics</title>

<item>Nutcracker</item>

<dinosaur>Stegosaurus</dinosaur>
```

XML also allows for so−called empty elements − essentially, elements which have no content and therefore do not require a closing tag. Such elements are closed by adding a slash (/) to the end of their opening tag. For example,

```
<rule>Every sentence ends with a <period /></rule>
```

An element name must begin with a letter, optionally followed by more letters and numbers. For example,

```
<popeye>

<book>

<INCOME>
```

are all valid element names.

Element names are case sensitive − so

```
<me>
```

is different from

```
<Me>
```

or

```
<ME>
```

An element may contain only text,

```
<step>Garnish with lemon and chopped onions</step>
```

or a combination of text and other elements.

```
<sentence>The red <animal>wolf</animal> jumped over the blue
<vegetable>aubergine</vegetable></sentence>
```

In order to be well−formed, an XML document must contain at least one non−empty element. This outermost element is called the "root element" and, in turn, may contain other elements, nested in a hierarchical manner. In the first example above, the root element would be <review>...</review>.

Anyone For Chicken?

Let's look at another example:

```
<?xml version="1.0"?>

<recipe>

<name>Chicken Tikka</name>
<author>Anonymous</author>
<date>1 June 1999</date>

<ingredients>

<item>
<desc>Boneless chicken breasts</desc>
<quantity>2</quantity>
</item>

<item>
<desc>Chopped onions</desc>
<quantity>2</quantity>
</item>

<item>
<desc>Ginger</desc>
<quantity>1 tsp</quantity>
</item>

<item>
<desc>Garlic</desc>
<quantity>1 tsp</quantity>
</item>

<item>
<desc>Red chili powder</desc>
<quantity>1 tsp</quantity>
</item>

<item>
<desc>Coriander seeds</desc>
<quantity>1 tsp</quantity>
</item>

<item>
```

```
<desc>Lime juice</desc>
<quantity>2 tbsp</quantity>
</item>

<item>
<desc>Butter</desc>
<quantity>1 tbsp</quantity>
</item>
</ingredients>

<servings>
3
</servings>

<process>
<step>Cut chicken into cubes, wash and apply lime juice and
salt</step>
<step>Add ginger, garlic, chili, coriander and lime juice in a
separate
bowl</step>
<step>Mix well, and add chicken to marinate for 3-4
hours</step>
<step>Place chicken pieces on skewers and barbeque</step>
<step>Remove, apply butter, and barbeque again until meat is
tender</step>
<step>Garnish with lemon and chopped onions</step>
</process>

</recipe>
```

Since each markup tag has a name which describes the data contained within it, it becomes possible to break up an unorganized document into structured, atomic parts. In the example above, the <author> tag identifies the data contained within it to be the name of the recipe author, while the <desc> and <quantity> tags are used to identify ingredients and their respective quantities.

The textual content which appears between the opening and closing tags is referred to as "character data"...or, as the XML specification puts it, "all text that is not markup constitutes the character data of the document." Although this character data may contain alphanumeric characters or symbols, care should be taken to escape special characters like angle brackets and ampersands by replacing them with the corresponding hexadecimal representation or the strings

```
&lt;

&gt;
```

```
&amp;
```

respectively.

For example, while the following XML markup would generate an error,

```
to your left < is the yellow brick road
```

this would be absolutely fine.

```
to your left &lt; is the yellow brick road
```

Similarly, while

```
Barnes &Noble
```

would produce an error,

```
Barnes &amp; Noble
```

would have no trouble at all.

To Attribute Or Not To Attribute...

Elements can also contain attributes, which provide additional information about the element. Attributes are name−value pairs which appear within the start tag of an element and can be used to provide additional descriptive parameters or default values to the element. For example, the following XML snippet uses the attribute "sex" to provide additional data on the <person> element:

```
<cast>
<person sex="male">Hugh Jackman</person>
<person sex="male">Patrick Stewart</person>
<person sex="male">Ian McKellen</person>
<person sex="female">Famke Janssen</person>
</cast>
```

Attributes must always appear after the element name, and attribute names are case−sensitive. Attribute values must always be enclosed within quotation marks, and the same attribute should not be repeated twice within the same element. If your document is linked to a DTD, you can enforce rules on the types of values an attribute may and may not accept.

It should be noted that the line between attributes and elements is often very fine, since the two perform similar functions. For example, while it is perfectly valid for me to describe a

```
<person sex="male">Hugh Jackman</person>
```

I could achieve exactly the same effect by breaking the data down and assigning it to a series of elements.

```
<person>
<name>Hugh Jackman</name>
<sex>male</sex>
</person>
```

In other words − the decision as to whether to use an attribute or an element can sometimes be a tricky one, and needs to be made on a case−by−case basis. Most experts seem to agree that this is an implementation decision, and must be made keeping in mind the purpose for which the document is going to be used.

Valid reasons for using attributes over elements would include assigning an ID to a specific element,

```
<review id="6548450">...</review>
```

or describing characteristics of the element itself.

```
The <animal color="red">wolf</animal> jumped over the
<vegetable
color="blue">aubergine</vegetable>
```

If you need to restrict attribute values to some pre−defined options, you can use a DTD to specify a list of allowed and default values, thereby cutting down on the possibility of errors and incompatible data.

If you're interested in a detailed discussion and debate of this issue, you should make it a point to visit http://xml.coverpages.org/elementsAndAttrs.html, which has some interesting comments and opinions by experts in the field on this very topics.

That's about it for the moment. In the next article, I'm going to continue this discussion of basic XML concepts with a look at entities, namespaces, and processing instructions − so make sure that you don't miss that one. Until then...stay healthy!