Introduction To XML

By Gayathri Gokul

2003−08−28

Printed from DevShed.com

URL: http://www.devshed.com/Server_Side/XML/XMLTutorial/

The Future Of The Web...

It’s a fact of life that XML, which stands for eXtensible Markup Language, is invading our lives more and more as programmers, and I feel this is a “good thing”. The reason being XML has the ability to cross all sorts of boundaries, and probably the only chance we have to obtain a truly independent, cross−platform data transfer format.

Data integration and data exchange is one of the key features in XML that will have wide usage in the future which means that applications would be able to exchange data. All large and small players in the computer industry are adopting XML. Hence XML is and will be a sure key enabler for e−business. Sound’s quite intriguing and fascinating isn’t it? Then read this article to understand why and how.

You may think in this strange world where standards come and go like seasons, even a single standard are fragmented by companies seeking competitive advantage. XML however, appears to be different. The technologies that dominate the market for data storage are pre−relational and relational databases that manage traditional data types like numbers and text. But with the advent of XML into the market, we have amazing new products that can manage and process *"diligent data"*, which means self−describing data. Relational systems will always be recommended for storing text and numbers, but if communication and processing are the priorities, XML is the technology of choice.

How is ASP and XML related? If you are using ASP to create a web site, then chances are pretty high that you are using a database to store data. XML is another format for storing data and being increasing used.

We’ve already seen XML support be introduced into Internet Explorer and ADO, but the support is far from full. The trouble stems from the fact that ADO and IE are evolving at different rates. While it is true that browser support is limited (although W3C's Amaya browser as well as JUMBO browser supports it), in the near future all browsers are expected to fully support XML, and ADO and IE albeit would be more integrated. This shouldn't give you the impression that XML is useless, until a browser supports it. XML isn't about display, it’s about structure.

XML is gaining universal acceptance and is sure to stay in future. Parsing algorithms and tools continue to improve as more and more people come to know the long−term benefits of moving their data to XML.

Technical gurus claim that XML is revolutionizing the way we communicate, spread and exchange information across the Web and within Intranets. This powerful language will help you do things with your data that you never before thought possible.

Preparing Yourself For The Future!

What Is Extensible Markup Language: Before we can learn what XML is, it’s a good idea to clarify what markup language is. Because, *language* is probably the wrong term. It is not a language in the sense that Visual Basic or C++ are languages, but it’s a set of rules that define how text documents should be marked up. Now we have to define what we mean by **marked up**. Marking up a document is the process of identifying certain areas of a document as having special meaning. In short markup language is just a set of rules that define how we add meaning to areas of a document.

The main difference between XML and HTML XML takes a different view from HTML, although it still uses tags, It is not a replacement for HTML. XML and HTML were designed with different goals. The major difference is that XML is designed to describe the structure of text, not how it should display. In short, ***XML was designed to carry data, to describe data and to focus on what data is***. On the other hand ***HTML was designed to display data and to focus on how data looks***. To put it in a line we can say HTML is about displaying information, while XML is about describing information. Let us take a simple example 1: Save it as .html

```
<body>

Hello !!

<h1> Welcome to the Web Age. </h1>

This is normal text. <b> while this is bold text.</b>

</body>
```

When you load the above html file into a browser, then it displays the above tags as though it were formatted document. Which will looks something like below

```
Hello!!
Welcome To The Web Age
This is normal text. While this is bold text.
```

However if the above text defines an XML document, then the tag don't mean anything. Say you load same file with .html extension (example 1) into the browser like IE but now with a .xml file suffix. Internet Explorer interprets this XML and displays it for us which looks like this

```
 <body>

Hello !!

<h1> Welcome to the Web Age. </h1>

This is normal text. <b> while this is bold text.</b>

</body>
```

But notice it's done nothing with the XML – just displayed it as such (except indents the tags to give a structured look). The browser knows how to interpret HTML and display it with all formatting. The browser also knows how to interpret XML, but in this case since XML tags don't imply formatting nothing is done and the tags are displayed as such.

The Name Says It All…..EXtensible:

HTML has a fixed set of element types, for example TITLE, H1, H2, EM, BODY, P, HR, and so on. A valid HTML document must use these elements. It cannot invent its own. A browser vendor can of course add its own proprietary tags (for instance, Netscape's <BLINK> tag), but there is no standard way to introduce new element types. While this is not the case with XML. XML has the ability to define new elements, and furthermore, this can be done in the document itself. Thus, XML documents are in essence self−describing. Let's consider an example. Imagine that you are looking at an HTML document with the following table in it:

Visual Basic Programming	BTech	Eduwardo Wancelloti	Naven Gokul Anna
Java Programming	EEE	Robert John	Patricia Mary Sasi
ASP Programming	CSE	Devi Karun	Susan Guna

```
<table >

<tr>

<td> <i>Visual Basic Programming</i> </td>

<td> BTech</td>

<td> Eduwardo Wancelloti </td>

<td> Naven<br> Gokul <br> Anna </td>

</tr>
```

```
(and so on for each row)...

</table>
```

Although we as humans or programmers can make an educated guess, we really have no way of knowing what this data represents. Is "Visual Basic Programming" the name of a book? A course opted? Skill set required? It is impossible to say. And if it's a course (as appear to be), are there three named participants registered (Naven, Gokul, and Anna). Or just one named Naven Gokul Anna? XML fixes this. The corresponding information would be represented in XML like this:

```
<course>

<name> Java Programming </name>

<department> BTech </department>

<teacher> Eduwardo Wancelloti </teacher>

<student> Naven </student>

<student> Gokul </student>

<student> Anna </student>

</course>
```

The lesson here is tags can be anything you like, it is only how you use them that gives them meaning. Of course it is sensible to give meaningful names to start with. In the above example we have used XML to describe data, and we've used tag names that represent field names of the data. Thus in XML, information about the data structure can be preserved. It's standard text, so can easily transfer from machine to machine. It's not in proprietary format, so anyone can read it, and if the tags are named sensibly, the XML is self−describing. Furthermore, all information concerning presentation style has been removed from the document. We can say that this XML document evidently supports an element named "course," and the question you should be asking yourself is: But how am I supposed to know how to format this? (Notice that the HTML code contains a style directive for the text "Visual Basic Programming;" for example, the *element indicates italics.*

Proof Is In The Output

XML does not have a fixed number of tags or elements, as HTML does, but is extensible, as is SGML, allowing the document designer to define meaningful tags. XML represents a response to the inadequacy of both languages to meet typical information publishing needs in an era that includes global information networks, and conventional paper publishing. XML is designed as a slim SGML, better suited for software development, distribution on information networks, and for use on non−conventional computing systems. The virtue of XML will become clearer as the Internet expands and as information devices such as palm−held computers and cellular phones become increasingly popular. Now let us look at some terms, and different ways that XML can be laid.

Tag and Element:

We have used the name tag to identify some HTML such as and <h1>. An Element is a fully formed use of those tags. For example:

```
<b> Some Bold text and <I> italic text </I></b>.
```

This tag consists of two opening and closing tags and two elements: b and I . So an element comprises of a start tag, an end tag, and text it encloses, which can include other elements. This is of great significance because it introduces the concept of Well−formed XML. In which an openeing tag must have closing tags. This is very different from HTML where some tags like <IMG> and
 don t have closing tags.

If you are using XML it is possible that some fields might contain no data. In that case the tags would be empty. Empty tags in XML could be define in one of the two ways. First is with start and end tags, but no content: <TagName></TagName>

The second way is to use an opening tag, but put slash at the end.

```
<TagName/>
```

Another aspect of being well−formed is that XML tags are case sensitive, so opening and closin tags must match. This means the following is incorrect:

```
<TagName></tagname>
```

Starting From The Roots of XML…

Root Tags:
Another term to be aware of is the **Root Tag**. This is defined as the outer tag, and an XMl document can have only one root. For example:

```
<BookAuthors>

<Author>

<au_id>1001</au_id>

<au_name> Bill Gates </au_name>

</Author>

<Author>

<au_id>1002</au_id>

<au_name> Harry Potter</au_name>

</Author>

</BookAuthors>
```

There can be only one top−level tag in the above code <BookAuthors>. Say you add <BookAuthors> some <author>tags and another <Bookauthors> within the above tag, then it will become invalid.

The <?XML Tag>: This isn't a true XML tag but special tag indicating special processing instructions. The ***<?xml tag>*** should be the first line of each XML document. And can be used to identify the version and language information. It is also a place where you define the language used in your XML data. It is trivial that if your data contains characters that aren't a part of ASCII (standard english character set) you can specify the encoding used in your document by adding ***encoding*** attribute to the ?xml processing instruction as follows:

```
<?xml version=é.0  encoding= iso−8859−1  ?>
```

Attributes:
Like HTML, XML has Attributes to define the properties of elements, and they mus also be well−formed.

Special Characters:
XML has special set of characters, which cannot be used as normal string. They are &, <, >, and,. For example the following is invalid <book> Advanced ASP & ADO Concepts </book>

Schemas and DTD's:
As stated earlier XML tags don't actual mean anything and you can give any name, but how do you know what name to give and what sort of tags are allowed in a document. For this purpose you have to use either a ***Document Type Definition (DTD) or Schema***. Schema and DTDs are like flip side of the same coin. They both specify which element are allowed in a document, and can run ***well−formed XML*** document into ***Valid XML*** document. All this means is that XML as well as being correctly marked up (well−formed) it contains only allowed elements and attributes.

The Best Of Both The Worlds
We will discuss both DTDs and Schemas you can decide which to use. A DTD is a text file that defines the structure of an XML document, but DTDs isn't XML – it is completely separate syntax.

Let us look at a typical DTD – say you have Employees table in your database that had fields like emp_id, emp_lname, emp_fname, phone, address, zip. Then the XML document generated for this Employee from the database will look this below.

```
<!ELEMENT DOCUMENT (EMPLOYEE+)>
```

```
<!ELEMENT EMPLOYEE(emp_id, emp_lname, emp_fname, phone, address, zip)>

<!ELEMENT emp_id (CDATA)>

<!ELEMENT emp_lname(CDATA)>

<!ELEMENT emp_fname(CDATA)>

<!ELEMENT phone(CDATA)>

<!ELEMENT address (CDATA)>

<!ELEMENT zip (CDATA)>
```

This is actually quite simple it states that this document comprises zero or more employee elements. The plus sign on the end of EMPLOYEE says one or more. Each EMPLOYEE element is made up of six other elements. And each of these sub−elements contains character data (CDATA).

There are two real flaws with DTD:
" They are not XML as such
" You cannot specify data types such as integer, data and so on.

Because of this Microsoft proposed Schemas to the W3C. If we converted the above DTD into a Schema it will look something like this:

```
<Schema ID= EMPLOYEE >

<ELEMENT name =  emp_id />

<ELEMENT name =  emp_lname />

<ELEMENT name =  emp_fname />

<ELEMENT name =  phone />

<ELEMENT name =  address />

<ELEMENT name =  zip />

</Schema>
```

With the addition of data types display will look like below.

```
<Schema ID= EMPLOYEE >

<ELEMENT name =  emp_id  type =  string />

<ELEMENT name =  emp_lname  type =  string />

<ELEMENT name =  emp_fname  type =  string />

<ELEMENT name =  phone  type =  string />

<ELEMENT name =  address  type =  string />

<ELEMENT name =  zip  type =  string />

</Schema>
```

Namespaces:
One problem with XMl is you can give an element almost any name. Unfortunately this means there is good chance you'll pick the same name as someone else. This is where ***namespaces*** come in, as namespaces uniquely identifies the schema to which elements belong. Namespaces are added to XML document by defining the xmlns attributes in the root tag, which requires ***Uniform resource Identifier (URI).***

The Potential Of XML

Parts of an XML document:
The various building blocks of an XML document is shown in the document below. And this will be a recap to what we have already learnt.

The Prolog consisits of two internal blocks :
" The first block defines the **XML version declaration** which is optional. <?xml version = "1.0" ?>
" **DTD (document type definition) OR SCHEMA**

PROLOG
VERSION
DECLARATION
DTD OR SCHEMA

<ROOT>

BODY&..

</ROOT>

XML Style Sheets
XML documents, unlike HTML, contain absolutely no formatting directives. Instead, an XSL (XML style sheet) is applied to the XML document. Think of a style sheet as a pre−processor for the actual graphical flow−engine that creates the final layout. The style sheet contains functions and rules for every element type. The functions are written in Scheme (a programming language derived from LISP) and have access to the full structure of the XML document being processed. This means that elements can be processed conditionally, for example, based upon their parent−element type (or some other relevant factor).

It is beyond the scope of this document to discuss XSL style sheets. However, the concept of applying a style sheet as part of the server−side processing is in fact extremely interesting and relevant to the discussion vis−a−vis server−side XML processing. I like to think of style sheets as transformation engines, which are capable of transforming an XML document into something else−−for instance, a database record−−or an HTML document.

XML Parser
A parser is a piece of software that makes sure the XML document is valid or at least well−formed. Parsers are extremely complex to develop and so only experienced programmers will design one. Once a parser is developed, other programmers and Web designers can easily reuse it .A parser is part of XML that you will never see.

XML tags are custom−defined thus enabling the generation of domain specific markup languages for diverse fields such as vector graphics, mathematics, music and technical documentation. I hope my now you have got a fair idea of this most hyped up XML technology, we will explore more of this in the next part.