

# XML: Data the Way You Want It

---

Michael Edwards  
Developer Technology Engineer  
Microsoft Corporation

October 31, 1997

## Contents

Introduction  
Is HTML Broken?  
What Is XML and Where Did It Come From?  
    What Does XML Look Like?  
    An XML Example: Channel Definition Format  
    Defining XML Data Structure  
What Can You Do With XML?  
    XML Vocabularies  
    XML Development Tools  
Summary

## Introduction

The World Wide Web Consortium (W3C) is defining a new standard for data formats called *Extensible Markup Language* (XML). If you've been around the Web very long, you know that HTML is pretty good at *displaying* Web-page content, but pretty bad at *describing* Web-page content as data so that it can be successfully manipulated over the Internet. Fortunately, that is what XML is all about -- providing a standard for defining your own markup tags and data structure so that data can be easily exchanged online.

XML provides the ability to augment HTML text with XML tags to make "smarter" Web pages and intranets. The XML tags let the browser know *about* the information that it's presenting, which enables all kinds of new capabilities:

- Imagine a Web where you can do real searches and *not* spend an hour sorting through 10,000 unrelated "hits" such as "Big Tall and Portly North Pole Fashions" on a search for pet shampoos. XML gives search engines the ability to discriminate between Bill (as in Gates) from bill (as in those annoying reminders that you owe somebody money).
- Imagine planning an evening on the town, using your laptop to integrate the address of a restaurant in your favorite online city guide into the traffic map application that tells you how to get there. XML lets people define data formats that can be seamlessly shared between unaffiliated applications.
- Imagine universal export and import formats supported by all your favorite applications. With this capability, a pizza delivery service could present their wares in a format that all client-side ordering software could understand? All sorts of things become possible with a universal standard for describing data.

Since co-founding the XML Working Group of the World Wide Web Consortium (W3C) over a year ago, Microsoft has been actively involved in advancing XML through draft specifications to a final recommended specification. And, just as Microsoft's Dynamic HTML offers fully compliant implementations of the W3C's HTML 4.0 and Document Object Model specifications, Internet Explorer 4.0 includes compliant support for XML data formats. For example, if you are creating Active Channel™ sites for Internet Explorer 4.0, you are using

Channel Definition Format (CDF), an XML data format for implementing push technology on the Web.

This article explains how you can use the XML features in Internet Explorer 4.0, how XML complements HTML, and what XML markup looks like. You'll also learn how XML data formats are being used today, and about the tools and technologies available for authoring and working with XML. You will be able to begin working with XML today using Internet Explorer 4.0, and take advantage of future developments.

## Is HTML Broken?

I've always believed in the common wisdom, "If it ain't broke, don't fix it" (unless I can find my hammer in which case I can "fix" anything). But really, XML doesn't replace HTML; in fact XML *complements* HTML by solving some key HTML limitations:

- **XML is designed to develop and extend new markup in a standard way.** There is no provision for authors to dynamically extend HTML in a standard fashion. To make a (standard) change or addition to HTML, you would have to endure a whole new round of standards battles.
- **XML is designed to specify precise rules for how XML markup is formed so development tools and browsers will already know how to parse it.** Ambiguity in HTML's tagging rules and structure adds up to lots of special-case code in HTML software. As a result, developers can't easily make use of new HTML features (and related technologies) without getting updated development tools, and users can't see these improvements until browsers are updated to support them. Special-case code and tools create problems because browsers don't know what to do when they come across these non-standard tags.
- **HTML is all about what content looks like, and XML is about describing the content so it can be manipulated online.** The problem with sending around text in HTML is that it doesn't tell you anything *about* the data. All the HTML markup tells you is how to show it. If you want applications to understand the data, you need XML.

In short, HTML is a markup language for presentation, and XML is a language for creating markup languages that describe structured data.

## What Is XML and Where Did It Come From?

XML is the universal language for describing and exchanging data on the Web. XML was born from an existing international standard called Standard Generalized Markup Language (SGML). SGML was created about 30 years ago in an effort to define a markup language for representing textual information. XML leverages the great work that went into creating SGML (and related standards) by identifying a subset specifically targeted for the Web. Hence, XML is a much smaller and simpler language that is often referred to as containing "20% of the complexity and 80% of the functionality" of SGML. Thus, like SGML, XML is a facility for creating new markups that provides a file format and data structure for representing data.

## What Does XML Look Like?

XML looks a lot like HTML. If you can read HTML, you will have little trouble reading XML. XML uses the same symbols to delineate markup. However, the actual XML elements might seem unfamiliar. The main thing to remember is that XML does not define any elements of its own, except the ones that are used to define new elements. Thus, XML is *self-describing*: A small set of built-in XML elements are used to define any new set of elements (and their hierarchical structure) that are contained in an XML document. This self-describing nature makes XML a wonderful transport mechanism for data because you can define a new set of elements that describe a certain data set, and include that description with the marked-up data.

In addition to creating stand-alone XML documents, you can integrate XML data with HTML using the data binding features of Internet Explorer 4.0. But I am getting ahead of myself; let's first take a walk down XML lane and look at an XML example.

## An XML Example: Channel Definition Format

Let's take a look at an XML sample that I modified from the Internet Client SDK. This Active Channel sample is based on the CDF specification that Microsoft submitted to the W3C. CDF is a vocabulary that is based on the latest XML work in the W3C. Here's a CDF file that contains XML markup:

```
<?xml version="1.0"?>
<CHANNEL HREF="http://www.microsoft.com/xml/articles/sample/homepage.htm"
  BASE="http://www.microsoft.com/xml/articles/sample/">
  <TITLE>Sample "Color" Channel</TITLE>
  <ABSTRACT>This channel contains a red, green, and blue page for
viewing.</ABSTRACT>
  <LOGO HREF="logo_big.gif" STYLE="IMAGE-WIDE"/>
  <LOGO HREF="logo_med.gif" STYLE="IMAGE"/>
  <SCHEDULE STARTDATE="1997-09-23">
    <INTERVALTIME DAY="1" />
    <EARLIESTTIME HOUR="2" />
    <LATESTTIME HOUR="6" />
  </SCHEDULE>
  <ITEM HREF="page1.htm">
    <LOGO HREF="red.gif" STYLE="ICON"/>
    <TITLE>The Red Page</TITLE>
    <ABSTRACT>This is the abstract description for the red page.</ABSTRACT>
  </ITEM>
  <ITEM HREF="page2.htm">
    <LOGO HREF="green.gif" STYLE="ICON"/>
    <TITLE>The Green Page</TITLE>
    <ABSTRACT>This is the abstract description for the green page.</ABSTRACT>
  </ITEM>
  <ITEM HREF="page3.htm">
    <LOGO HREF="blue.gif" STYLE="ICON"/>
    <TITLE>The Blue Page</TITLE>
    <ABSTRACT>This is the abstract description for the blue page.</ABSTRACT>
  </ITEM>
  <ITEM HREF="scrnsave.htm">
    <USAGE VALUE="ScreenSaver"></USAGE>
  </ITEM>
</CHANNEL>
```

CDF-specific tags enable a publisher to offer automatic delivery of information (called *channels*) from their Web servers to Web users. As you might infer from the CDF element names, the data includes a schedule for automatic downloads, indications for newly updated content and branded logos, and a topic hierarchy. For example, if you are running Internet Explorer 4.0, you can install the INetSDK sample just by clicking here (button .gif). The HTML source code for this button shows how HTML and XML (in this case, CDF-specific tags) can complement each other:

```
<A HREF="http://www.microsoft.com/xml/articles/sample/sample.cdf" ALT="Add  
Active Channel example">  
<IMG SRC="/xml/articles/sample/ch_chbtn.gif" BORDER="0">  
</A>
```

Because the URL in this example has a .CDF filename extension, Internet Explorer 4.0 expects to find XML markup in the CDF format. Thus, the above CDF example is copied straight out of the SAMPLE.CDF file referred to in this HTML anchor.

If you refer back to the CDF file example while you are reading about the details of XML data structure (next section), you'll find it contains only the markup. Additional information about the markup's valid syntax is usually inserted just before the markup (directly in front of the <CHANNEL> tag). The syntax information is not necessary in this file because the XML spec allows it to be implied from context (in this case the .CDF filename extension in an HTML hyperlink provides the context). You can review the spec that describes the rules for valid CDF syntax that Internet Explorer 4.0 automatically supplies while we go over Defining XML Data Structure.

## Defining XML Data Structure

The W3C has identified design goals for the XML standard. One of those goals is to define a precise structure for XML markup so it is easy to parse and straightforward to understand. As a result, the rules for constructing start tags, end tags, attribute *name=value* pairs, and so on, are spelled out very clearly in the (draft) XML syntax spec. But the basic XML syntax rules only determine whether XML markup is *well-formed*, that is, whether a piece of XML markup conforms to the basic rules for how all XML markup must be constructed. To determine whether an instance of XML markup is *valid*, you need to determine whether it conforms to the rules for that data type -- each XML data type has rules for what elements are allowed enabled and exactly how they are pieced together. To that end, XML documents have two parts: the *prolog* declares the element names, attributes, and construction rules of valid markup for that data type, and the *document instance* contains the actual markup. Thus, the XML markup contained in the document instance is valid only if it conforms to the rules laid out in the document prolog.

## Prolog

The prolog section identifies an XML document or code fragment, and includes the information needed to parse the data in the file. The prolog section includes several kinds of *declarations*, statements that define the construction rules for that data type. The XML declarations use processing instructions. The DOCTYPE declarations together describe what is called the *Document Type Definition* (DTD), the rules by which the data must abide to be valid. Element names and their valid hierarchical structure, together with their allowed attributes and entities (the ELEMENT, ATTLIST, and ENTITY declarations), are declared in the DOCTYPE processing instruction and define the complete set of XML markup that authors can use in the document. As a group, the declarations are referred to as internal or external, depending on

whether they are in the same file or in another file. The syntax for building DTDs from these declarations enables authors to describe documents with a high degree of structural complexity. Let's look at each kind of declaration and how to use it.

## XML declaration

The first line of the prolog is an *XML declaration*:

```
<?xml Version Encoding RequiredMarkupDeclaration?>
```

The `<?` and `?>` delimiters indicate a processing instruction for an XML parser. The *XML* processing instruction indicates an XML document, and includes:

- *Version*: the XML standard version (currently 1.0)
- *Encoding*: specifies the Unicode character set that is used in the document (very important for localization issues)
- *RequiredMarkupDeclaration* (RMD) indicates whether the browser needs to read the document type declarations in the prolog before it can parse the markup data.

## DOCTYPE declaration

Immediately following the XML declaration are the *document type declarations* that define this type of XML data. Sometimes an application can determine the data type from context (such as a filename extension), in which case document type declarations are optional because they are built in to the application or the it knows where to find the DTD information. In our CDF example file, the DOCTYPE declarations are not necessary. If declarations are present, the following format is observed:

```
<!DOCTYPE Name SYSTEM ExternalDeclarations [InternalDeclarations]>
```

The `<!` characters are an open delimiter for a document type declaration, and the DOCTYPE declaration contains these document type declarations:

- *Name* is a value that by convention matches the name of the root element for this document type. In the above CDF example, *Name* = *CHANNEL*. Because a valid XML document must have a single root element with all other elements as children, the name of that great-granddaddy root element specifies the overall document type. In other words, the value of *Name* defines the XML document type.
- *ExternalDeclarations* is a URL to a file containing any external declarations that specify the document's markup tags and their structure.
- *InternalDeclarations* are inline declarations specifying the document's markup tags and their structure.

## Element declaration

To construct the elements that comprise a DTD you use an *element declaration*:

```
<!ELEMENT Name Content>
```

*Name*, of course, is the element's name, and is analogous to a unique data type. *Content* indicates other elements and character data that can be contained by the *Name* element type, and is constructed using its own *grammar*, that is, rules for how the content elements are structured. The grammar for the content in an element declaration can:

- Declare no content at all (an *empty* element). In this case, as with HTML, all the information is contained in the attributes themselves, and there is no need for an end tag to enclose character data. In the CDF example, the <LOGO/> element is empty (consisting only of an HREF attribute indicating a URL to the channel GIF).
- Declare content consisting only of other elements (to create a hierarchical or relational structure, for example), but no attributes or character data. Root elements such as <HTML> usually contain only other child elements.
- Declare a mixed content of character data (the stuff being marked up) and other element types.
- Declare that anything goes, that content can be any combination of other elements and character data with no structural constraints (other than being well-formed).

### Attribute list declaration

Because you're familiar with HTML you know that most elements have attributes. In XML, you can build your own attributes list in a DTD by using the attribute list declaration:

```
<!ATTLIST ElementName AttributeName AttributeType AttributeDefault>
```

*ElementName* is the name of the element to which the attribute information applies. Only one *ElementName* is used, but because any number of attributes may be applied to it, multiple attribute definitions can occur in an ATTLIST declaration. You can give each attribute a name, declare the type of value that may be associated with it, and whether the attribute must be present or can be implied from a default value supplied in the declaration. You can also indicate the attribute as a flag that should not include a value at all. The grammar for indicating the attribute's type is a little complicated, but lets you specify whether the attribute value is:

- A random string of characters
- A string consisting only of XML *name* characters: the set of alphanumeric characters (plus a few others such as ':') that can appear no more than once in the document (analogous to a unique ID value such as a CSS .CLASS name)
- One of a set of allowed strings enumerated in the attribute declaration
- An *entity reference* (a special character sequence that indicates a string substitution) provided in the DTD

### Entity declaration

You build the entities for a DTD using the entity declaration:

```
<!ENTITY Name Value>
```

Entity references declare replacement text for an escape sequence. The escape character used, as in HTML, is the ampersand. Entities can come in handy when you want to standardize boilerplate types of material. For example, if *Name* is **&MyTitle** and *Value* is **Master of my Destiny**, all occurrences of **&MyTitle** in character data would be replaced by **Master of my Destiny**. You can also declare an entity called a *character reference* to insert characters that cannot be typed on the keyboard of the authoring platform (you are probably familiar with character references from HTML).

### Document Instance

After the prolog section comes the *document instance*, the actual character data that is marked up with the elements declared in the DTD. Just as in HTML, this part of the document consists of character data that is delimited by various start and end tags (along with the appropriate attributes) that adhere to the order and structure of the DTD.

Now you might ask what XML markup *means*. XML enables you to define (in a standard way) the names of elements and how they are ordered and structured, but there is nothing in the standard about what the elements themselves **mean**, that is, we read <AUTHOR> and infer a person who wrote something, whereas XML only interprets <AUTHOR> in the context of how it's declared in the DTD. The meaning of <AUTHOR> is established in the documentation that people write, read, and interpret.

## What Can You Do With XML?

So where does the rubber meet the road when it comes to actually doing something productive with XML? XML's current usability is somewhat limited by its "newness"; it simply hasn't been around long enough. Even though the XML W3C Working Group is moving very quickly, new standards take some time to be defined before they're officially announced as a standard and broadly adopted. However, as you might have already surmised, XML is being used in a number of ways today.

XML markup provides a way to identify, exchange, and process any kind of data. The world has many terabytes of information distributed across a vast sea of incompatible information repositories. XML provides the mechanism for exchanging chunks of data with these entities in a mutually understood fashion. Because DTDs describe distinct collections of XML markup for different data sets, XML markup isn't a single conglomeration of elements where every new element adds to an ever-expanding list. Instead, DTDs enable the creation of an infinite number of distinct *vocabularies*.

## XML Vocabularies

An XML vocabulary is defined by a specific DTD. Think of an XML *vocabulary* as the set of elements (words) and the rules for valid constructions of those elements (grammar) as defined by a particular DTD. The idea is that an inventor applies XML's declarative syntax to construct a DTD, and other folks make use of that DTD to create the indicated type of XML document (or *record*). For example, a DTD to specify drug allergies to include in medical records would be an XML application. But the *XML application* terminology is confusing because most of the online world thinks of an application as some type of computer program. Hence folks are starting to use the term *XML vocabulary* instead.

For example, when Microsoft created CDF, we invented a new XML vocabulary. Because the CDF vocabulary was created using a standards-based markup language, other people can easily use it. They will know how to construct and parse documents using the vocabulary, and they can read the documentation to interpret the meaning of the elements and their attributes. But the vocabulary itself is not a standard simply by having been defined in XML. It is usually advantageous for folks to agree on file formats, but with XML you don't have to. That is, no official standards body automatically states that the vocabulary embodies *the* definitive way of describing a particular data set. In essence, XML levels the playing field by providing a standards-based way to create file formats for any kind of data. If you find this subject particularly interesting, you may want to pursue the debate about XML namespaces on the xml-dev mailing list archive (it's searchable), and read about Open Software Description (OSD) extensibility using XML namespaces in this note submitted to the W3C.

Let's take a closer look at two XML vocabularies proposed by Microsoft that are being used today:

- The Microsoft Internet Explorer 4.0 site includes a CDF overview explaining how CDF is used to author a channel with Active Channel technology. Active Channel technology lets Web publishers personalize and streamline the delivery of information to their customers. With CDF, authors can specify which parts of a Web site's tree structure to download for a given information channel. It can also provide cues to schedule Web crawling to coincide with the actual content updates on the site, and more.

CDF was proposed because Microsoft needed an open file format to describe Web content. The format needed to be easily parsed and extended, and it needed to support international standards for character encoding and indicating online resources. Creating a new XML vocabulary as part of Internet Explorer 4.0's Webcasting infrastructure helped Microsoft provide the Active Channel technology of Internet Explorer 4.0 because we extended our Web browser in a standard way. You can learn more about Active Channel technology and CDF in the Internet Client SDK (go to Delivering Content to the Web) and the Active Desktop & Channels section of the SBN Workshop.

- As indicated in an overview on the Specs & Standards site, the Open Software Description (OSD) format is an XML vocabulary that describes software components, their versions, their underlying structure, and their relationships to other components. Developers can use OSD in conjunction with CDF to create a software distribution channel. As you might guess, software distribution channels make the process of updating software on the Web easier and safer for both developers and users. You can learn more about creating software distribution channels in the Internet Client SDK (go to Delivering Content to the Web; Authoring for New Delivery Mechanisms; Creating Software Distribution Channels). If you want to see how Internet Explorer 4.0 implements a software distribution channel to update itself, choose the Software Update option from Internet Explorer 4.0's Favorites menu. [If you want to see how the Microsoft SDK for Java uses OSD to provide software distribution, read about it on the Microsoft Java site \(link to http://mscominternal/java/sdk/20/tools/dubuildovr.htm#dubuildovr\\_0004000c00000000\).](http://mscominternal/java/sdk/20/tools/dubuildovr.htm#dubuildovr_0004000c00000000)

Because SGML came first, many data formats are currently based on SGML. For example, the banking industry has standardized on Open Financial Exchange (OFX) for exchanging financial data and instructions among financial institutions and client software. OFX enables banks to expose financial data in a single format that, as indicated in this press release, is supported by many companies, including CheckFree, Intuit, and Microsoft. OFX is great vocabulary for developers who write software that manages electronic financial transactions, because they have to learn just one file format. The mathematical community has also created an SGML vocabulary called Mathematical Markup Language (MML), and chemistry professionals have created the Chemical Markup Language (CML). Authors for many of the existing SGML vocabularies are now in the process of creating XML versions of their DTD so their markup can be used by XML-aware browsers and development tools.

In the coming months, you can expect to see a multitude of new vocabularies appear for broad areas like searching, filtering, electronic commerce, and other areas.

## XML Development Tools

With the promulgation of the XML standard, tool developers are now able to develop broad-based tools that span multiple data communities. Without a standard, it's the old chicken-or-the-egg problem -- software vendors don't want to spend a ton of money developing tools for a niche market, but it is very difficult for a market to become established without great tools and



broad support. Standards encourage a proliferation of tools by dramatically expanding the pool of potential users -- in this way, acceptable standards create a call to action.

So what kinds of tools do you need to develop in XML? Like the Web, XML tools fall into two main categories: tools for programmers and tools for authors. The programming tools generally take the form of visualization tools and software code libraries that can be used by authors to create and manipulate XML content. Generally, the software libraries come first. For example, Internet Explorer 4.0 includes an XML object model (that can be used from C, Java, or script) that tools developers can use as a foundation for creating their high-level XML visualization tools, or other XML development tools.

## **Parsing XML**

The tool that jump-starts all XML software development is an *XML parser*. That is because every XML application relies on a parser to process an XML document. Parsers take the form of a code library that exposes software interfaces to developers using higher-level languages such as C++ or Java. Using these interfaces, developers can access the structure of an XML document, enumerate its elements and their attributes, and play with stuff in the document prolog. A simple example would be an XML parser utility that checks for *well-formed* or *valid* documents, and serves as the XML equivalent of an HTML syntax (lint) checker.

Every XML development tool has an XML parser at its core, and the parsers are in turn based on some notion of an object model for an XML document. Currently, a group that includes Microsoft is working with the W3C to develop a XML object model standard. Other informal efforts are under way as well. For example, the xml-dev mailing list is working to define a Java-based API called XAPI-J. John Tigue, an independent developer, has been maintaining a page for this effort at <http://www.datachannel.com/xml/dev>. You can also download the Microsoft XML parser in Java (MSXML) and the Microsoft XML parser in C++. See the contents listing on the left for documentation on the XML object model used in Internet Explorer for manipulating CDF and OSD files. See the Tools section of our Tools & Samples area for a copy of Microsoft CDF Generator (it's in the Web Authoring bucket), which is a tool for authoring CDF files. For more general editing of any XML-based data you might try out our XML Notepad tool.

## **Authoring XML**

Equipped with the ability to parse XML documents, programmers can start building high-level tools that enable authors (and users) to create, edit, browse, and search XML documents. These tools range from general-purpose editors conversant in any XML vocabulary to vocabulary-specific applications.

Because they have been around a lot longer, authoring tools that support SGML (see the W3C summary page) are more plentiful than those currently available for XML. In fact, as far as I am aware, high-level Web-authoring tools have just begun to incorporate XML support (check out DynaBase from Inso Corporation and ADEPT by ArborText for two examples). Given the nascent nature of XML, this shouldn't be too surprising. It makes sense that authoring tools first exploit specific, existing XML vocabularies (such as CDF and OSD). As more and more XML vocabularies are developed, it will become economically feasible for tools vendors to expand their offerings to support any XML DTD. Given the current excitement and broad industry support for XML (as shown by the diversity of participants in the W3C XML Working Group), I am confident that XML will quickly become as ubiquitous as HTML.

There are already a number of vertical tools available for working with CDF. The Microsoft Channel Wizard walks you through the steps of building a channel (no understanding of CDF syntax, or any other technical details, is necessary). More advanced tools for generating CDF include the Microsoft CDF Generator discussed earlier, the Cold Fusion CDF Wizard, Bluestone Software's Sapphire/Web 4.0, and iNet Developer 2.0 by Pictorius. Another Microsoft offering, IIS 4.0, includes support for dynamically generating CDF code using server-side scripting. As a result, when browsers retrieve files with the .CDX extension, the default MIME type returned by the Web server will be the CDF MIME type (instead of HTML). And naturally the FrontPage 98 Beta also supports CDF. True to form, many other Internet-related Microsoft products are busily incorporating CDF capabilities.

In the future, many application categories (such as databases, messaging, collaboration and productivity applications) will incorporate support for other XML vocabularies as they are defined. This will enable interoperability on the common data types used within an application category, as well as across application categories (for example, address information in a customer database can be easily shared with a PIM application or an e-mail client).

## **Presenting and transforming XML**

Capabilities for specifying the formatting of XML will be implemented as more advanced features of tools. The separation of content from presentation is a core design principle for XML. Because XML completely separates the notion of the markup from its intended visual presentation, authors can embed in structured data procedural descriptions of how to produce different data "views." This is an incredibly powerful mechanism for offloading as much user interaction as possible to the client computer, and also serves to reduce server traffic and speed browser response times.

The ability to specify how XML should be visually presented to the user addresses several author needs. First, authors have to be able to specify how XML data should look when it is presented in, say, a browser. Second, authors need to be able to specify alternate structures for XML data, that is, different ways the markup "tree" might be organized, depending on who the viewer is, or what they want to look at. Finally, there is a general need to translate between competing or overlapping XML vocabularies, and even non-XML proprietary file formats. For example, you might look to word-processing applications to read and write popular XML formats, or you might expect editors to provide a means of associating HTML tags with XML markup, and to embed XML into HTML.

For this to happen, a standard style language for XML needs to be recommended, or at least proposed. As you may already know, recommending a standard stylesheet language for XML is one of three deliverables specified by the W3C's XML Activity Statement. In August 1997, Microsoft and others made a proposal for eXtensible Style Language (XSL) to the W3C. The XSL proposal is also announced on our Specs & Standards site. XSL is derived from an existing international standard complementing SGML: the Document Style Semantics and Specification Language (DSSSL). So, just as XML is a subset of SGML to support Web-based information, the new style language for XML is a subset of DSSSL (I found this nice description of DSSSL by Paul Prescod). XSL is a great fit with XML because it's compatible with Cascading Style Sheets (CSS) and the script languages that many Web authors already know. So, if you're familiar with enhancing HTML presentation using CSS and script, you're smiling now.

Internet Explorer 4.0 is the world's only browser that includes a built-in generalized XML parser. This means you can write XML vocabularies today that can be utilized in Microsoft's shipping Internet browser. I think general support for displaying XML data directly in the browser will be the catalyst for an explosion of new XML development. This kind of capability

leverages the Web development expertise available today, and enables new XML vocabularies to be developed by *anybody*. And it will not require people to change the way they use the Web! You can get more information about XML support in Internet Explorer 4.0 in the Internet Client SDK (go to XML data source object and the XML Object Model).

## Summary

HTML revolutionized electronic document distribution and popularized a whole new information arena -- the Internet -- far faster than most people predicted. It certainly doesn't take a visionary to predict that the Web will continue to change our everyday lives. XML has started the journey toward a world where every conceivable category of information has an XML format that everybody can use and understand. After all, information is most useful when it is easy for everybody to access. And it all got started because the world was able to agree on a standard way to create new data types.

If you want more background information that will help you understand XML concepts and how XML came about, start with Robin Cover's SGML Web Page.

If you want to get more information about the current W3C XML activity, check out the SGML, XML, and Structured Document Interchange W3C site. Or, start with the W3C XML home page and read the draft proposals yourself! If you *really* want to start getting your fingernails dirty, you can read through the xml-dev mailing list archive (it is also searchable). Although discussions of the xml-wg (the current W3C-appointed decision-making body) and xml-sig (a group of experts who offer advice to xml-wg) mailing lists are confidential to W3C member organizations (and invited experts), anybody can join the xml-dev mailing list to gain perspective on the types of problems that are being addressed by all kinds of XML developers. You'll also find posts on xml-dev regarding various XML tools and technology demos released to the public. If you go down that path, be sure to check out Peter Murray-Rust's XML-DEV Jewels page.

If you would like to find out more about what we are doing with XML at Microsoft check out our XML home page. Also, stay tuned to the Site Builder Network because we will be posting more information about how you can use XML today with Internet Explorer 4.0.

This stuff is so cool I can start to imagine the day when the Web will actually reduce information overload instead of adding to it!