# NET-ENABLE LEGACY APPLICATIONS WITH XML

Mark Wonsil

4M Enterprises, Inc.
4020 Albert Avenue
Royal Oak, MI 48073-6601

Phone: 248.288.1015
Fax: 248.288.6416
E-mail: wonsil@4m-ent.com

## Introduction

One of today's most popular requests made on Information Technology organizations around the world is to unlock the data maintained by legacy applications and make it available to other resources on the network. While it may be trivial to publish these resources on a local network, the Internet introduces many new technological challenges to the legacy application programmer. At the same time, one of today's most touted technological break-through is the Extensible Markup Language or XML. Its proponents claim it is the solution to everything from seamless business to business application integration to world peace. Can markup technology help the typical application programmer share legacy data across the network and even the Internet?

## A History of Markup Languages

For most people, their first exposure to markup was in school. Instructors added notes to homework assignments, usually in red ink, indicating errors in spelling or grammar. Markup is information added to a document to further describe or interpret that document. Proofreading markup tells the author of the document where the errors are. Many people are also familiar with presentation markup. This kind of markup tells a page formatter when to change fonts, to bold or underline. Content markup conveys a context to a portion of a document. The word 'June' by itself is ambiguous, but adding markup tells one if the author is talking about a person's name, a month in the year or a bug.

So, who gets the credit for the creation of content markup technology? Curiously, the credit may go to lawyers. More accurately, it goes to the lawyer's legal documents. In the 1960's, when Microsoft's Bill Gates was only a child, IBM was the darling of the U.S. Department of Justice. The DOJ accused IBM of using predatory and exclusionary practices to control the computer industry. IBM realized early on that taking on the U.S. Government in addition to its normal legal activity would generate a great deal of legal documentation. IBM assigned an employee to help organize and manage its legal documents. The employee's name was Charles Goldfarb. Because IBM hired many attorneys for their legal work, Mr. Goldfarb had to manage documents created by different word processors and computer systems. Mr. Goldfarb designed a system that stored the content of the legal document stripping out the proprietary information from each of the different computer systems. To Goldfarb's credit, he designed the system in such a way that it would work with other types of documents and not just legal ones. In 1969, Charles Goldfarb and two associates, Ed Mosher and Ray Lorrie, created a system whose initials are the same as its creators – GML. GML also stands for the Generalized Markup Language. In order for the system to work in multiple problem domains, GML followed three design goals: separate content from the presentation, provide for extensibility and create a facility to ensure a document's validity.

In 1986, the International Standards Organization released ISO 8879 based on the work of Goldfarb, Mosher and Lorrie as SGML – the Standardized Generalized Markup Language.

SGML is a standard that describes the structure of documents. It is a large standard taking well over 100 pages to describe. In SGML, the document that describes other documents is called a Document Type Definition or DTD. Soon, authors created DTD's for many other types of documents outside the legal arena. There are DTD's for book publishing, aircraft manuals and even recipes.

Besides legal documents, another unwieldy group of documents was the growing number of research documents at colleges, universities and research facilities around the world. Users exchanged these documents using the File Transfer Protocol (FTP) with the help of Gopher and Archie. In theory, SGML seemed to be an appropriate technology to aid in the sharing of these documents. However, SGML was so large and complex, it was difficult to implement in its entirety in a stand-alone program let alone for use in a network environment. So in 1989, Tim Berners-Lee and Anders Berglund at CERN, the European Nuclear Research Facility, used SGML to create the Hypertext Markup Language to help share these documents across a disparate computer base. HTML popularity is so great these days, few users even know or even care that it is an SGML application.

## Issues with HTML

At the birth of HTML, a great deal of the content was still text. Users had character-based terminals and they used the Lynx command line browser to browse the web. It was the job of the webmaster to markup content and it was the job of the browser to render the document. There was a clear distinction of duties. The main purpose of the markup was simple presentation effects and navigation. The navigation markup was a powerful tool that gave users an efficient way to jump to other documents including any supporting graphics. A user could download a graphics file and then view it with a graphics program on a graphics terminal. Eventually, there was the idea to display the graphics inline with the text. The birth of the graphical browser changed the face of the World Wide Web forever. In the new graphical environment, presentation started to take precedence over content. Browser developers added so many extensions so quickly that HTML 3.0 never became a standard. Webmasters invented clever schemes to control the presentation of documents. The distinction of duties between content and presentation blurred.

As the number of documents on the web exploded, so did the need to organize all of these documents. Organizations created search engines that would "crawl" around the Web and catalog all of the content. Users of the web soon found that searching the web became more of an art than a science. The problem was that the search engines just cataloged words with out regard to context. A search on the word "Clinton" would bring up cities, rivers, businesses, the President and invariably a few pornographic sites. If webmasters marked up every city with the name Clinton, for example: <city>Clinton</city>, search engines and other programs would have a better understanding of the context.

Unmarked content mixed with presentation makes maintaining the content more difficult. A webmaster can take months designing a web site. Within a year, it is not uncommon to redesign that very same site. Often this means starting from scratch because it is too time consuming to remove the current presentation markup. Management wants to distribute the same content on other media like CD-ROM or print. Webmasters are also redesigning web sites to handle accessibility issues, better international support and wireless access. This usually means starting again from scratch. Along the way, if the content changes, one must go back to each presentation media and make changes.

## The Extensible Markup Language

The goal of the XML working group was to address these shortcomings of HTML and in February of 1998, the World Wide Web Consortium released the XML 1.0 recommendation. This recommendation is sometimes called a foundation standard. A foundation standard tends to be

more general and vague as it allows the building of other standards on top of it. Some believe it is this vagueness that has caused the unprecedented hype over XML as developers project their promising solutions on the new unknown technology. At just over thirty pages, XML is much smaller than its older sibling but its goals of content separation from presentation, extensibility and document validity are similar. In addition, the group had other goals, XML should be compatible with SGML to leverage existing tools and lightweight for use on the network.

Many have written books on XML and there are plenty of sources to learn the details of XML. (See the reference section for some helpful resources.) Figure 1 shows a simple XML document. The first line of the document is the prolog. It tells the parser basic information about the document. In this case, the prolog says this document conforms to version 1.0 of the XML recommendation. An XML document is text only and it is made up of elements. An element consists of a start tag, optionally some content and an end tag. Start tags may also have attributes. The hpworld2000 element has an attribute named start. All attribute content *must* be in quotes. Every start tag *must* have an end tag. The interex tag is a special notation for an element with no content. All XML documents have one and only one root element. In this case, it is the hpworld2000 element. All XML documents must be well formed. A well-formed document is one where all elements are properly nested. This means each end tag closes the closest start tag. It is also worth noting that XML is case sensitive. The sample document would not be well formed if the final tag was </HPWORLD2000>.

```
<?xml version=”1.0” ?>
<hpworld2000 start=”9/9/2000”>
  <event>HP World Conference</event>
  <event>Expo</event>
  <event>HP e3000 World</event>
  <event>eSolutions World</event>
  <event>ASP Conference</event>
  <interex/>
</hpworld2000>
```

**Figure 1**

Like SGML, an XML document may have a Document Type Definition (DTD). There are entire books devoted to the creation of DTD’s but here are some highlights. A DTD can be internal, located in the document or held externally in another file. It is important to know that a DTD only tells the parser a valid sequence of the elements. A DTD does not in any way enforce any rules on the content itself. (There is a standard in process called XSchema that addresses content validation.) A DTD also defines the allowed element names and the use of attributes for those elements. If a document is well formed and it follows the constraints of the DTD, it is said to be valid. In addition to element order, the DTD may also include entities. An entity is a string that represents a character, another string or resource. A general entity is similar to a constant in popular computer languages. One good use for an entity is to hold a company name. In today’s corporate environment, an employee may start the day working for Hewlett-Packard and by the end of the day be working for Agilent. An entity may include a properly formatted file into your document too.

## The XML Files

Those are the foundation concepts. Here are a few of the standards that build on the XML recommendation.

### *Content Referencing – XPath and XPointer*

One of the current issues with HTML is that in order to reference a portion of a document, the developer must have write access to that document in order to place an anchor (<a>…</a>) at the point of interest. XPath addresses this weakness. Remember the example file and the definition of being well formed, it can easily be seen how one could represent an XML document as a tree. XPath exploits the tree structure of XML documents. Each node on the tree has its own type. It could be an element node, an attribute node or a text node for example.

The fundamental building block of XPath is the expression. An expression evaluates to one of four distinct types: A collection of nodes, a boolean value, a number or a string. XPath supports the most straightforward method of element access which is found in a typical file system. The developer can list all elements from the root or reference a relative location. XPath also supports direct element access, pattern matching and tree navigation. In direct element access, the application can jump directly to an element that has an ID type attribute. One of the rules of the ID attribute type is that the value of the ID element *must* be unique within the document. Pattern matching returns a group of nodes that matches a selection. In tree navigation, the program can step from sibling to sibling, parent to child or child to parent. It is also possible to inspect the type of nodes along the way.

The XPointer standard builds on XPath. Its purpose is to enhance document fragment access from a Universal Resource Indicator (URI). In HTML, one uses the "#" fragment identifier to load a document and then point to the named anchor. With XPointer, it is possible to return just a portion of the document. What is even more useful, is that it is possible to reference a document or portions of a document without modifying the document. XPath and XPointer are higher-level foundation specifications that other standards use.

## *Content Linking*

XLink defines how to set up links among documents. SGML has a similar standard called HyTime, the Hypermedia/Time Based Structuring Language. HTML links are very limited. They are one way links to a hard-coded location in the target document. XLink provides several new capabilities. First, links may be in-line or out-of-line. A file can hold all of the links. This is possible because of the XPath and XPointer standards mentioned above. Second, it is possible to set up a choice of links from a single point in the document. In the context of a browser, a user would click on a link and a popup menu of potential destinations appears. One destination may be the next step in a tutorial, another may be a definition and another may take the user to a quiz on the topic just covered. Third, XLink can control what happens when a user clicks a link. Today, web masters use scripting languages to open a document in a new browser window. But in XLink, the link determines if the browser opens a new window or replaces the current screen with the linked document.

## *Content Presentation*

The Extensible Stylesheet Language, XSL, is the standard for formatting XML documents. SGML developers use DSSSL, Document Style Semantics and Specification Language to perform the same function for SGML documents. An understanding of Cascading Style Sheets, CSS, will help in the understanding of XSL. XSL breaks down a page into different formatting objects. Among them, there are page objects, block objects, line objects and character objects. The web master can assign properties like font weight, size, color and position to these objects. Unlike CSS, one has the capability to transform the original document before applying the formatting. The developer has the power to create a table of contents, a user's manual and an administrators guide all from the same original XML document. The capability to transform a document became so complex however it was more manageable to put transformations into its own recommendation.

## *Content Transformation*

XSLT, XSL Transformations, is a specification that transforms XML documents into other XML documents. In is not designed to be a general-purpose transformation tool, rather it is meant to rearrange and filter XML documents before formatting. XSLT is a declarative language, so it is a different way of thinking for most legacy programmers. Once again, there are enough books on the capabilities of XSLT. XSLT contains many functions that allows the developer to select nodes by attributes, elements and text values and you can even reorder the nodes. A web master can also build templates of commonly used transformations for reuse. Developers can use XSLT to

transform XML into (X)HTML, WML or for use in other applications. These applications will parse the document to use the data as it see fit.

## Parsing XML Documents

The typical way applications access an XML document is with an XML parser. Most parsers are generic tools meant to work in many situations. A parser can be a validating or non-validating. A validating parser will check the document against its DTD. Parsers usually belong to one of two different processing models: the Document Object Model (DOM) and the Simple API for XML (SAX).

A DOM parser converts an XML document into a DOM Tree. As seen earlier, it is natural to represent XML documents as trees. This kind of parser will generally load the entire document into memory. Once there, the developer can traverse the tree, modify, add or remove nodes. After the modifications are complete, the parser can save the document back to a file. This gives the greatest flexibility to the developer, but it is not the easiest to use and loading many large documents may not fit into memory.

On the other hand, a SAX parser is an event driven parser. In this type of parser, the programmer registers functions with particular events. For example, there is a 'start document' event. If one registers a function with this event, the parser calls the routine at the start of the document. There are also events for start tags, end tags, and character data. This kind of parser tends to be fast and very memory efficient. Of course, it is a little more difficult to modify a document.

Many parsers tend to use object-oriented languages like C++ and Java. This is a problem for many legacy programmers that use Cobol or Fortran. But there is a way to for these non object-oriented languages to access XML files. Back in the early days of SGML, Unix was the operating system of choice. The Unix philosophy is to use small tools that are run in a sequence. Most of these tools, like grep, awk, wc, are record oriented in nature. Fortunately, there is a way to represent XML files in a record oriented format. It is based loosely on the ISO standard called the Element Structure Information Set or ESIS. A Python developer created a notation based on ESIS called PYX. In PYX, each record begins with a character that identifies what kind of data is on the line. Figure 2 shows the coding of the records and Figure 3 shows a portion of our original XML from figure 1. (For more information on the PYX notation, see the reference section.) With the XML file in this format, it would be fairly easy for Cobol or Fortran programmers to read and write these kinds of files.

| PYX | Meaning |
|---|---|
| ( | Start Tag |
| ) | End Tag |
| A | Attribute |
| - | Data |
| ? | Processing Instruction |

**Figure 2**

```
?xml version="1.0"
(hpworld2000
Astart "9/9/2000"
)hpworld2000
(event
-HP World Conference
)event
 …
)hpworld2000
```

**Figure 3**

## Barriers to the Net

There are at least three basic issues when trying to expose legacy data to the network: client support, transport method and data access.

Once an organization decides to expose legacy data to the net, the developer has to choose what kind of client will have access to the legacy data. A organization could write a custom client. This solution can be problematic in larger audiences like the Internet because maintaining and distributing multiple versions of the client software for multiple operating environments can

become a logistical nightmare. The organization could write a custom Java client but client-side Java will still has to reconcile the different Java Development Kit versions (1.01, 1.1, 1.2 etc.) and Java Virtual Machine implementation issues. For the broadest audience, many developers have turned to the Web browser and a lowest common subset of HTML.

As far as transporting the data, a programmer could write a custom protocol. But this solution adds more work and unknown protocols usually have a difficult time getting through firewalls. The most popular protocol on the net is the Hypertext Transfer Protocol (RFC 2616) or HTTP. HTTP is a request-response protocol. In this model, a client opens a connection with host and requests a resource. The host sends the resource and then closes the connection. This model scales well because network administrators can use load balancing technology with multiple servers to handle many requests. (Over time the HTTP protocol implemented features like persistent connections and caching to improve its efficiency.) The trouble with the request-response is the way it maintains state. In legacy programs a client, usually a terminal, presents the screens of information and the host "remembers" what the user entered on previous screens. The HTTP protocol does define a method to maintain state between requests – the controversial cookie. After fulfilling a request, the host may send some data to the browser in the form of a cookie and associate that cookie with a particular URL. The next time the browser sends a request to that URL, the browser includes the information in the cookie. The controversy is about privacy and security. In the earlier browsers, the scripting languages had the capability to browse the cookies on the user's computer. If a web site stored a customer's credit card number in a cookie, an unscrupulous developer could access that data. Even if the hacker could not read the data in the cookie, the browsing patterns of a user could be sold to marketing firms. Because of this, many users turn off or limit the use of cookies in their browser. There are ways to track state outside the HTTP protocol, but they have a small performance penalty.

### *Data Access*

Many companies try to expose legacy data to the net using technology like ODBC or JDBC. These tools are made to make ad hoc requests and they return to the client exactly what is in the database. While this sounds like a good thing, the problem with this is that many legacy databases use special codes to represent dates or options. Some applications store dates as an integer that represents the number of days since a fixed date like January 1, 1970. A client may ask for an order policy code or payment terms code and get a '6' with no description. Now the client must interpret the data, which adds to maintenance costs. Another issue is that some programmers share fields because it is easier than making a database change. For example the first four characters or bits may stand for one item and the remaining characters or bits for another field. In order to perform updates on the database, the developer will have to duplicate this logic as well as any business logic at the client. Finally, some (but not all) ODBC drivers will not be able to read non-database files. Even if they do, these drivers may not be able to take advantage of features of your software that improve performance (like B-tree access or keyword searching).

Some people use a "screen scraping" technique to exchange data. This has the benefit of using the current business logic but it restricts one to the data available to the screen and is vulnerable to screen changes or difficult prompting patterns.

## An XML Server for Legacy Data

### *Design*

So how does an organization expose legacy data that will handle the fore-mentioned barriers? An XML data server. What kind of client will it support? Any client that can accept a text file through HTTP. Since there is no concern with presentation, the file is only marked-up text and this greatly increases the size of the audience. What transfer protocol would it use? It will use HTTP. But there is a problem with maintaining state in HTTP. How will the XML server handle

it? The best plan is to not handle it. The server will only distribute data. It is best to let frameworks like Active Server Pages, Java Server Pages and Application Servers maintain state. Finally, for data access, developers can use the same familiar tools used by legacy application programmers. Most legacy code already does the job of data access using native calls. All the developer needs to do is to format our responses in the PYX notation.

Borrowing from the object-oriented world, the XML data server would be based on the Model-View-Controller design pattern. In this pattern, there is data (model), the presentation (view) and the actions on the data (controller). The data model is based on the data that needs to be shared. It can be a purchase order, a medical claim, a bill of material, a service request or the current disk usage of a computer.

Here are the steps to build an XML server that shows the users currently logged on. The first step is to get the data. In a screen scraping solution, a program reads the results of the command that displays current users, formats it for display and sends it to the client. (For MPE/iX it is the SHOWJOB command, for HP/UX it is the who command and for NT it would be the NET USER command.) This gives the developer some basic information, usually the user's name, a login time and maybe a terminal ID. With the XML server, the developer can use operating system calls to find out more information. It could gather the current CPU time, remote IP address, current job step and the process tree. Instead of formatting the results for display, the XML server would just wrap tags around the data. To display the data in a browser, let a stylesheet convert the XML to (X)HTML.

The next step is to add some commands to the server. One command would be to list all users. Another command would be to list one user with all of the details. There could also be one command that sends a message to a user and another command that would log off the user.

With HTTP as the command interface, a typical command would look like:

http://myhost:32000/mpeixml?model=showjob&view=summary&command=list&user=all

Here, the XML server, listening on port 32000, will list all users in a summary format. The developer can add other commands and views and even new elements to the server without breaking any current functionality.

The basic flow of an XML server is:

```
Initialize

While not exit
  Wait for command on socket
  Parse request
  Gather Data
  Select & Sort Data according to view
  Tag the Data
  Send Response
End While

Cleanup
```

One way to build a very scalable solution is to use servlet technology like Apache JServ. A servlet would help solve several of our issues mentioned above. Servlets have a facility to maintain state, but they can also perform other vital services. First, a servlet would maintain a TCP/IP socket connection pool to our XML servers. This would provide quick access, as the requester would not have to wait for any program initialization. Second, the servlet would translate the XML to and from the PYX format. Third, the servlet would encode/decode the data

to make sure it is safe XML. For example, it would change all '<' characters to '<'. Fourth, it could pass the data through a stylesheet to create (X)HTML for a browser or WML if the device was wireless. Finally, the servlet would act as a proxy to request XML documents or document fragments from other systems. This would give the legacy applications access to databases on the network. For example, Oracle databases can send result sets in XML format using their XSQL pages technology.

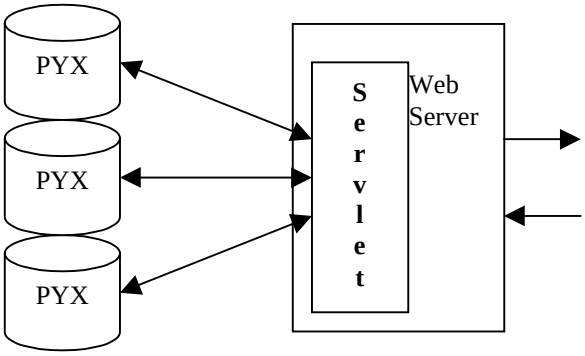


**Figure 4**

### *Using the server*

What can an XML server do? The first thing it does is it allows other programming languages like Perl, Python, Java, Visual Basic, Active Server Pages, Java Server Pages and even Javascript to have access to the data because all of these languages have support for XML. Many application servers, like Enhydra, could also access the data.

Decision support systems accepting XML data sources. Microsoft's Digital Dashboard technology is one of these systems. In fact, many application that can render HTML documents can use the XML server.

There has been some work done on using XML as a remote procedure mechanism. The first is XML-RPC and the other is the Simple Object Access Protocol or SOAP. The standards are still developing at this time. Both of these protocols let a developer access objects across the network. A SOAP server would be a controller in Model-View-Controller pattern.

With the release of the Java 2 Enterprise Edition (J2EE), XML plays many important roles. One of the most interesting roles is in the Java Message Service (JMS). JMS is a standard interface to messaging services. It allows Java to communicate with other, even proprietary, messaging systems like IBM's MQ*Series. Messaging services provide reliable asynchronous communication between applications. It has been said that a reliable transport service is needed before more companies put their business transactions on the Internet. Organizations can use JMS to reliably transport XML documents.

Some have argued that XML is the answer to seamless business-to-business (B2B) integration. While XML may provide a universal way to structure and distribute data, it does not solve many of the basic integration issues. There is the obvious problem of multiple element names for the same object, but there are still the issues of item identification, reconciling data structure differences, providing missing data, managing extra data and recognizing software organizational differences. XML provides a syntactical solution but alone it does not answer the semantic question, "What does this document mean?"

## Conclusion

By keeping content separate from the presentation and keeping a clear distinction of duties, there is a framework that will help the legacy programmer distribute data across the network. This solution lets the programmer focus on the business rules and not the idiosyncrasies of HTTP or (X)HTML. Using XML does not preclude the use of any other technology, but instead it helps extend the value of your current and future investments. There is really nothing magical about XML, it is its underlying philosophy that gives it its power. And now, even legacy programmers can access this power.

# References

## *Standards*

Hyper Text Transfer Protocol v. 1.1; RFC 2616
World Wide Web Consortium Recommendations: http://www.w3.org/TR/
SAX, Simple API for XML, http://www.megginson.com/SAX/

## *Web Sites*

World Wide Web Consortium: http://www.w3.org
Café Con Leche: http://www.metalab.unc.edu/xml
FineTuning.com: http://www.finetuning.com/
The XML Cover Pages: http://www.oasis-open.org/cover/
XML.COM: http://www.xml.com
XML Info: http://www.xmlinfo.com
PYX Notation: http://www.pyxie.org
Enhydra: http://www.enhydra.org
XML-RPC: http://www.xml-rpc.org

## *Publications*

Goldfarb, Charles, “The XML Handbook”, Prentice Hall, 1998
St. Laurent, Simon, “XML – Elements of Style”, McGraw-Hill, 2000
Harold, Elliotte Rusty, “The XML Bible”, IDG Books, 1999