The X-Factor: Implications for Internet Programming Today and Tomorrow

Roy A. Boggs
Computer Information Systems, Florida Gulf Coast University
Ft. Myers, Florida, 33965, USA

Abstract

Internet programming is leaving the simple and sometime loose programming path of the HTML era. Newer structures are more disciplined, browser and platform independent, and demand a new set of skills and programming knowledge. Using a simple HTML table, the following pages present an overview of the transformations that will soon be used to display this table and its contents. Structures, with X- names, from (X)HTML through XML, XSL(T) and XSL(FO), are presented in programming examples. The result is a demonstration, based upon examples, of where the X-Factor is taking Internet programming. Implications for practice and pedagogy are then summarized at the end of the paper.

Keywords: Internet Programming, Internet Development, XML, XHTML, XSL

WHERE ARE WE GOING?

There is hardly a discipline that is changing and advancing as fast as Internet design, development and management. A multidirectional process moving back and forth between disciplines and Internet structures has brought this about. Everyone is affected or influenced in some form by material available on the Internet. At the same time, Internet development finds itself faced with an ever-increasing myriad of new structures, newer versions of browsers with increased capabilities, more complex platforms, faster delivery systems, and demands that everything respond equally quickly and equally as well.

HTML alone won't do it anymore. Even if current versions of some browsers respond to weak code in a favorable manner, this situation is not likely to continue very much longer, this is especially true if a set of web pages is to enjoy widespread use. The Internet world is moving, or being pulled, into assimilating a set of recommendations that is designed to free Internet web pages from constrictions imposed by browsers, scripts, and platforms.

These recommendations are developed and shepherded by the World Wide Web Consortium: W3C (http://www.w3.org/). All of the various pieces of the current and the new can be found there. The information is often overly technical. However, it is possible to cull tendencies and directions. It is also at times in this process useful to take one example and follow the recent developments. This is what is done in the following review – as sort of an answer to the question: where are we going?

The various programs and examples are intended for those with an interest in Internet programming and who have some experience working in the Internet environment. They present points of reference and nothing more. Those with only a passing interest in Internet development may read the text, leaving the examples for those who prefer reading code. For this version, complete examples are given. Examples assume current browsers, or any plug-ins for older browsers

FROM HERE TO THERE

The ultimate goal of the review below is to show the direction the X-Factor is taking Internet programming. The X-Factor includes those programming and data structures beginning with the letter X: (X)HTML,

XML, XSL(T), XSL(FO), etc. However, it will be seen in the examples that HTML code is not being eliminated. Quite the opposite, it is being given a disciplined hierarchical structure and then wrapped in constructs that makes it **language-, browser-, and platform-independent**.

The first key word in this process is '*deprecation*'. Older tags, and some tags currently enjoying widespread usage, are being discouraged and will ultimately be discontinued. Three other important key words are '*well-formed*', '*valid*' and '*namespace*'. 'Well-formed' indicates that a document's structure satisfies all of the rules that make it highly structured and predictable. A document that is not 'well-formed' will not be displayed. 'Valid' indicates that it satisfies all of the rules for a particular document, specifying which elements and attributes are allowed or required and in what format. Documents that are not 'valid' may not be displayed until corrections are made. 'Namespace' indicates a collection of related element names. They are often distinctively marked (for example: xsl:name, fo:name) to prevent collisions between similar names in related documents.

The review begins with HTML, and then passes through (X)HTML to XML and XSL. There are other topics that might have been included, such as XPATH. However, the main topic is XML. It is the main path. First, from the beginning.

BASIC HTML

Employee Directory	
Smith, James	James@OurFirm.com
Jones, Jill	Jill@OurFirm.com

The table above contains the caption 'Employee Directory' along with two data instances of 'name' and 'email' for each of two employees. The caption contains a larger and bolder font. A table is used here as the example because tables represent basic units for creating and displaying documents, and for managing the display of objects. They may well displace frames, especially since frames are regarded as special structures in (X)HTML; and they may themselves eventually become replaced by Cascading Style Sheets (CSS) positioning elements and '*.inc' files.

HTML stands for Hypertext Markup Language. A basic HTML set of code for this table would be as follows. The various <tags> structure show the data are to be *displayed*.

```
<html>
<head>
  <style type="text/css">
      table, td {border-color: black;}
      caption {font-size: 16pt; font-weight: bold;}
  </style>
</head>
<body>
 <table border>
  <caption>Employee Directory</caption>
  <tr>
    <td>Smith, James</td>
    <td>James@OurFirm.com</td>
  </tr>
  <tr>
    <td>Jones, Jill</td>
    <td>Jill@OurFirm.com</td>
  <tr>
 </table>
</body>
</html>
```

```
<table border =1
 <caption> Employee Directory
 <tr>
   <td>Smith, James
   <td>James@OurFirm.com

 <tr>
   <td>Jones, Jill
   <td>Jill@OurFirm.com
```

The code on the left can be executed on all current browsers. It is 'well-formed'. The hierarchical structure is maintained. For the sake of the examples below, a CSS <style> tag is included. The code on the right will run on *some* browsers. It is not 'well-formed'. For example, closing tags are omitted. This can be problematic for maintenance, especially when debugging code. However, it is more problematic

when the amount of browser code needed to recover from sloppy programming is considered. 'Well-formed' documents can be executed safely using current browsers and demand less browser code.

HTML also permits the use of scripting languages, such as VBscript and JavaScript, and more freedom in manipulating and displaying objects. The choice of a scripting language is important. Some scripting languages, which offering greater ease of use and more opportunities for creating web pages, are platform dependant. Scripting languages do not all execute on the same platforms. Some tags, such as <layer>, are also browser dependant. In some environments, use of scripts often implies multiple versions of the same web page. This quickly becomes a costly and time-consuming endeavor.

(X)HTML-BASIC

(extensible)e Hypertext Markup Language ((X)HTML) represents a major step in resolving some of these problems. It is considered an intermediate step between HTML and XML, at least until more browsers fully support XML. (X)HTML adds to HTML code an XML declaration that, if 'well-formed', ensures that it is not browser dependant and supports W3C recommendations. By definition an (X)HTML document is an XML document. Proper validation includes such items as no empty tags <meta … />, all tags in lowercase <td>, all tags properly closed </td>, all defaults eliminated <table border="border">, etc.

The code for the BASIC HTML table above has been rewritten to (X)HTML standards. It is thus considered to be (current) browser independent.

```
<?xml version="1.0" encoding="iso-8859-1"?>
   <!DOCTYPE  html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>
Basic XHTML Format
</title>
<!--
Date:    1/01/02
-->
<meta name="author" content="Roy A. Boggs" />
<meta name="keywords" content="XHTML" />
<meta name="description" content="Basic XHTML Format" />

<style type="text/css">
   table, td {border-color: black;}
   caption {font-size: 16pt; font-weight: bold;}
</style>
</head>

<body>
 <table border>
   <caption>Employee Directory</caption>
   <tr>
      <td>Smith, James</td>
      <td>James@OurFirm.com</td>
   </tr>
   <tr>
      <td>Jones, Jill</td>
      <td>Jill@OurFirm.com</td>
   <tr>
 </table>
</body>
</html>
```

This is an XML (eXtensible Markup Language) document and must by definition be 'well-formed'. 'iso-8859-1' encoding is simply used here to ensure a larger character set. The DOCTYPE identifies the document as an (X)HTML document that is *strictly* 'well-formed'. The choices are 'strict', 'transitional'

and 'frames'. 'Strict' is preferable unless it is deemed necessary to use a tag even though it will eventually be deprecated or unless a <style> tag does not yet function as expected. (X)HTML code can be validated at: http://validator.w3.org. Styles can be validated at: http://jigsaw.w3.org/css-validator.

BASIC XML

The first step is to provide data that are 'well-formed' and 'valid'. There might be several steps to the process. A small data base file containing the two entries for the Employee Directory table serves as data source. This example is from a Microsoft Access database.

name	email
Smith, James	James@OurFirm.com
Jones, Jill	Jill@OurFirm.com

The language used to pull the data from the database given here is for simplicity VBscript running under ASP conventions.

```
Print #1, "<?xml version=1.0 encoding = iso-8859-1 ?>"
Print #1, "<employee_directory>"
Do Until datEmployee.Recordset.EOF
Print #1, "<employee>"
Print #1, "<name>" & lrs.Name & "</name>"
Print #1, "<email>" & lrs.Email & "</email>"
Print #1, "</employee>"
datEmployee.Recordset.MoveNext
Loop
Print #1, "</employee_directory>"
```

The result is a stand-alone XML file that could just as well have been entered by hand. It is important to note once again that the XML file must not only be 'well-formed' but also 'valid'. That might mean here that each employee must have a name, and perhaps it might also mean that an email address could be optional – or, the email address might be required. At any rate, the structure must reflect definite rules to be considered valid.

xmlEmployees.xml

```
<? xml version="1.0" encoding = "iso-8859-1" ?>
<employee_directory>
 <employee>
  <name>Smith, James</name>
  <email>James@OurFirm.com</email>
 </employee>
 <employee>
  <name>Jones, Jill</name>
  <email>Jill@OurFirm.com</email>
 </employee>
</employee_directory>
```

The data now exist within *descriptive* tags. The tags make visible the content of the data items. They do not dictate how the data is to be *displayed*. Because the xml-file has been produced from a database via a programming language, the data it contains is expected to be 'well-formed' and 'valid'. Should the data came from a less reliable source, 'Schemas' are constructed to ensure proper data structures. To be valid, XML data must reflect schema definitions. A simple example might be as follows.

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:element name="employee_directory" type="employee"
/>
<xsd:complexType name="employee">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="email" type="xsd:string" />
   </xsd:sequence>
 </xsd:xomplexType>
</xsd:schema>
```

When simply run as an XML file ('xmlEmployee.xml") without a schema reference, the result is as follows. The use of tags with descriptive names should be noted.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <employee_directory>
- <employee>
  <name>Smith, James</name>
  <email>James@OurFirm.com</email>
  </employee>
- <employee>
  <name>Jones, Jill</name>
  <email>Jill@OurFirm.com</email>
  </employee>
  </employee_directory>
```

BASIC HTML AND XML FILES

It this point, the data can still be processed using a formal language, such as JavaScript. The following example demonstrates how this might work. The code is not trivial (it contains recursion) and requires some programming skill. The complete code is given for the sake of clarity. The input file (xmlDoc.load("xmlEmployee.xml")) retrieves the XML file given above. 'newHTML' is a variable that is built throughout the iterations to form the output. The CSS <style> tag is included as in the examples

above. It is here that tags for displaying data reside. The recursive function used to process data the items is 'buildTree(varName)'. The data are bound by code, (Node.parentNode.nodeName=="name"), where "name" contains the value of the corresponding XML tag. The result is the same Employee Directory table given at the beginning of this review.

```
<head>
<title>Roy A. Boggs: xml Javascript</title>

<style>
   table, td {border-color: black;}
   caption {font-size: 16 pt; font-weight: bold;}
</style>

<script language="JavaScript">
 var xmlDoc = new ActiveXObject("microsoft.xmldom")
 xmlDoc.async=false
 xmlDoc.load("xmlEmployee.xml")
 root=xmlDoc.documentElement
 newHTML=""

function start()
{
  newHTML+=("<table border>")
  newHTML+=("<caption>")
  newHTML+=("Employee Directory")
  newHTML+=("</caption>")
 buildTree(root)
  newHTML+=("</table>")
 content.innerHTML=newHTML
}

function buildTree(passedNode)
{
 var children = passedNode.childNodes.length

 for (var j=0; j<children; j++)
 {
  Node=passedNode.childNodes.item(j)

  if (!Node.hasChildNodes())
     {
          if (Node.parentNode.nodeName=="name")
          {
            newHTML+=("<tr><td>" + Node.text + "</td>")
          }
          if (Node.parentNode.nodeName=="email")
          {
            newHTML+=("<td>" + Node.text + "</td></tr>")
          }
     }
  buildTree(Node)
 }
}
</script>
</head>
<body onload="start()">
<span id="content"></span>
</body>
</html>
```

Here again, while the XML document is language and platform independent, the scripting process is language dependant and it is also platform dependant. As stated above, it takes programming skill to work with such structures. The answer is to side-step formal code and to apply XSL(T) templates. The coding is straightforward and independence is built-in.

XSL(T) AND XML FILES

The Extensible Stylesheet Language for Transformation (XSLT) transforms and renders XML documents into web pages. The possibilities for using XSLT are extensive and often complex. A possible XSLT file is as follows. It is named 'xmlXslEmployee.xsl'. The CSS <style> tag is enclosed as [CDATA …] to prevent it from being processed as part of the XSL namespace.

The line of code: <xsl:apply-templates select="employee_directory/employee"/> functions as a call to the code: <xsl:apply-templates select="employee"/>. <xsl:value-of select="name"/> pulls the value of the item, here 'name'. The resulting code is much more straightforward and simpler to apply.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
    <head>
      <title> Employee Directory</title>
<style>
 <![CDATA[
    table, td {border-color; black;}
    caption {font-size: 16pt; font-weight: bold;};
  ]]>
</style>
    </head>
    <body>
      <table border="1">
       <caption>Employee Directory</caption>
        <tr>
          <th>Name</th>
          <th>Email Address</th>
        </tr>
<xsl:apply-templates select="employee_directory/employee"/>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="employee">
<xsl:apply-templates select="employee"/>

      <tr>
        <td><xsl:value-of select="name"/></td>
        <td><xsl:value-of select="email"/></td>
      </tr>

</xsl:template>
</xsl:stylesheet>
```

An XML file is then used to process the file: xmlEmployee.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="xmlXslEmployee.xsl"  type="text/xsl"?>

<employee_directory>
 <employee>
    <name>James Smith</name>
    <email>james@OurFirm.com</email>
 </employee>
 <employee>
    <name>Smith, Jill</name>
    <email>Jill@OurFirm.com</email>
 </employee>
</employee_directory>
```

The addition of an XML processing instruction to the XML file above (<?xml-stylesheet href="xmlXslEmployee.xsl" type="text/xsl"?>) identifies the location of the XSLT file and is sufficient to produce the desired output with limited code via a process that will in effect be independent of current browsers, scripts, and platforms. (This is not always the case as some current browsers do yet fully support XSLT without downloading and installing relative modules.)

XSL(FO) AND XML FILES

A final and important step is an ability to render XML documents with extensive formatting: to a display, to a printer, or to a *.pdf file. The formatting objects for these processes are presented as Extensible Stylesheet Language Formatting Objects (XSL-FO). The following example expands on the XSLT code above and the XML remains the xmlEmployee.xml file.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">

<xsl:template match="/">
 <fo:root>
  <fo:layout-master-set>
   <fo:simple-page-master master-name="sampleTable"  margin="2.5cm 1.5cm" >
     <fo:region-body />
   </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-name="sampleTable">
    <fo:flow flow-name="xsl-region-body">
     <fo:block>
       <fo:block font-weight="bold" font-size="16pt" >Employee Directory </fo:block>
      <xsl:apply-templates select="employee_directory"/>
     </fo:block>
    </fo:flow>
  </fo:page-sequence>

 </fo:root>
</xsl:template>

<xsl:template match="employee">
    <fo:block>
       <fo:table width="300pt">
          <fo:table-column column-width="20pt" column-number="1"/>
          <fo:table-column column-width="20pt" column-number="2"/>
          <fo:table-body>
              <fo:table-row font-size="14pt">
                    <fo:table-cell border="0.5pt solid black" padding="4pt" >
                      <fo:block> <xsl:value-of select="name"/></fo:block>
                </fo:table-cell>
```

```
                <fo:table-cell border="0.5pt solid black" padding="4pt">
                    <fo:block>  <xsl:value-of select="email"/> </fo:block>
                </fo:table-cell>
            </fo:table-row>
          </fo:table-body>
      </fo:table>
 </fo:block>
</xsl:template>

</xsl:stylesheet>
```

The expected namespace is 'fo' and it encapsulates an extensive and sophisticated page-markup language. Current browsers do not yet fully support XSL-FO. However, these exist several freeware sources that will convert these documents. The desired table below is the same as that for the more complex, dependent processes above.

Employee Directory	
Smith, James	James@OurFirm.com
Jones, Jill	Jill@OurFirm.com

SOAP

The final step is to create a means of shipping XML documents freely across platforms. Simple Object Access Protocol (SOAP) is a 'protocol for exchanging information in a decentralized, distributed environment'. It is designed to permit XML formatted files to be exchanged independent of the sending or receiving platform. This step wraps the XML file in a SOAP envelope for shipping. The assumption, of course, is that the corresponding XSL(T/FO) file resides at the destination.

SOME IMPLICATIONS

What does one learn from this?

The first, and one of the most important lessons here is that HTML is not going away quickly. The initial step to Internet programming is still HTML. The second step is still a solid knowledge of the latest version of CSS. Basic Internet coding, such as the use of tables, continues with HTML structures, but all descriptive, and positioning elements are reserved for Cascading Style Sheets.

However, programming now comes in the form of XML structures. Implications suggest that from the beginning one should learn to create 'strict' (X)HTML documents. This means that CSS should be a part of initial programming efforts, and that the resulting code must be 'well-formed' and the document 'valid'. All CSS and (X)HTML documents then need to be validated. This is a must and a necessary step to ensure browser functionality.

The next steps lead from (X)HTML/CSS to XML and to XSL(T/FO). The learning progression is almost build-in. XML by itself will render web pages and the schemata will display data in various formats. Soon, (X)HTML and CSS may not be needed in an applications environment. (However, one can assume that quick coding in (X)HTML and CSS will be around for a while.)

XSL(T/FO) render web pages and *.rtf files respectfully. They represent a necessary final step in the learning process.

What is missing is the ability to *easily* access databases, using XSL, Xquery or XQL, to pull and format the data with the proper tags into an XML file. Until this happens scripting remains a necessary tool. So there is indeed more to come. The bottom line is still – as before – users must first know and understand their data and their data structures. Otherwise, the X-factor, however designed, may produce unintended consequences.