It started with XSL and ended up with XSLT, XPath, and XSL-FO.

It Started with XSL

XSL stands for E**X**tensible **S**tylesheet **L**anguage.

The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.

CSS = Style Sheets for HTML

HTML uses predefined tags, and the meaning of each tag is **well understood**.

The <table> tag in HTML defines a table - and a browser knows **how to display it**.

Adding styles to HTML elements are simple. Telling a browser to display an element in a special font or color, is easy with CSS.

XSL = Style Sheets for XML

XML does not use predefined tags (we can use any tag-names we like), and therefore the meaning of each tag is **not well understood**.

A <table> tag could mean an HTML table, a piece of furniture, or something else - and a browser **does not know how to display it**.

XSL describes how the XML document should be displayed!

XSL - More Than a Style Sheet Language

XSL consists of three parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents

This Tutorial is About XSLT

The rest of this tutorial is about XSLT - the language for transforming XML documents.

XSLT is a language for transforming XML documents into XHTML documents or to other XML documents.

XPath is a language for navigating in XML documents.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML / XHTML
- XML / XML Namespaces
- XPath

If you want to study these subjects first, find the tutorials on our Home page.

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT = XSL Transformations

XSLT is the most important part of XSL.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

If you want to study XPath first, please read our XPath Tutorial.

How Does it Work?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT is a W3C Recommendation

XSLT became a W3C Recommendation 16. November 1999.

All major browsers have support for XML and XSLT.

Mozilla Firefox

Firefox supports XML, XSLT, and XPath from version 3.

Internet Explorer

Internet Explorer supports XML, XSLT, and XPath from version 6.

Internet Explorer 5 is **NOT** compatible with the official W3C XSL Recommendation.

Google Chrome

Chrome supports XML, XSLT, and XPath from version 1.

Opera

Opera supports XML, XSLT, and XPath from version 9. Opera 8 supports only XML + CSS.

Apple Safari

Safari supports XML and XSLT from version 3.

Example study: How to transform XML into XHTML using XSLT.

The details of this example will be explained in the next chapter.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>.

Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

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

or:

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

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".

Start with a Raw XML Document

We want to **transform** the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

Viewing XML Files in Firefox and Internet Explorer: Open the XML file (typically by clicking on a link) - The XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

Viewing XML Files in Netscape 6: Open the XML file, then right-click in XML file and select "View Page Source". The XML document will then be displayed with color-coded root and child elements.

Viewing XML Files in Opera 7: Open the XML file, then right-click in XML file and select "Frame" / "View Source". The XML document will be displayed as plain text.

View "cdcatalog.xml"

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

View "cdcatalog.xsl"

Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
```

```
  <year>1985</year>
 </cd>
.
.
</catalog>
```

If you have an XSLT compliant browser it will nicely **transform** your XML into XHTML.

View the result

The details of the example above will be explained in the next chapters.

An XSL style sheet consists of one or more set of rules that are called templates.

A template contains rules to apply when a specified node is matched.

The <xsl:template> Element

The <xsl:template> element is used to build templates.

The **match** attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

Ok, let's look at a simplified version of the XSL file from the previous chapter:

Example

```
<?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>
 <body>
 <h2>My CD Collection</h2>
 <table border="1">
  <tr bgcolor="#9acd32">
   <th>Title</th>
   <th>Artist</th>
  </tr>
```

```
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

Example Explained

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="ISO-8859-1"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The **<xsl:template>** element defines a template. The **match="/"** attribute associates the template with the root of the XML source document.

The content inside the <xsl:template> element defines some HTML to write to the output.

The last two lines define the end of the template and the end of the style sheet.

The result from this example was a little disappointing, because no data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:value-of>** element to select values from the XML elements.

The <xsl:value-of> element is used to extract the value of a selected node.

The <xsl:value-of> Element

The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation:

Example

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

Example Explained

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

The result of the transformation above will look like this:

The result from this example was also a little disappointing, because only one line of data was copied from the XML document to the output. In the next chapter you will learn how to use the **<xsl:for-each>** element to loop through the XML elements, and display all of the records.

The <xsl:for-each> element allows you to do looping in XSLT.

The <xsl:for-each> Element

The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

Example

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

Filtering the Output

We can also filter the output from the XML file by adding a criterion to the select attribute in the <xsl:for-each> element.

<xsl:for-each select="catalog/cd[artist='Bob Dylan']">

Legal filter operators are:

- = (equal)

- != (not equal)
- < less than
- > greater than

Take a look at the adjusted XSL style sheet:

Example

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

The <xsl:sort> element is used to sort the output.

Where to put the Sort Information

To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file:

Example

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

Note: The **select** attribute indicates what XML element to sort on.

The <xsl:if> element is used to put a conditional test against the content of the XML file.

The <xsl:if> Element

To put a conditional if test against the content of the XML file, add an <xsl:if> element to the XSL document.

Syntax

```
<xsl:if test="expression">
  ...some output if the expression is true...
```

</xsl:if>

Where to Put the <xsl:if> Element

To add a conditional test, add the <xsl:if> element inside the <xsl:for-each> element in the XSL file:

Example

```
<?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>
 <body>
 <h2>My CD Collection</h2>
 <table border="1">
  <tr bgcolor="#9acd32">
   <th>Title</th>
   <th>Artist</th>
  </tr>
  <xsl:for-each select="catalog/cd">
   <xsl:if test="price &gt; 10">
    <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="artist"/></td>
    </tr>
   </xsl:if>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

Note: The value of the required **test** attribute contains the expression to be evaluated.

The code above will only output the title and artist elements of the CDs that has a price that is higher than 10.

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

The <xsl:choose> Element

Syntax

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>
</xsl:choose>
```

Where to put the Choose Condition

To insert a multiple conditional test against the XML file, add the <xsl:choose>, <xsl:when>, and <xsl:otherwise> elements to the XSL file:

Example

```
<?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>
  <body>
```

```
 <h2>My CD Collection</h2>
 <table border="1">
  <tr bgcolor="#9acd32">
   <th>Title</th>
   <th>Artist</th>
  </tr>
  <xsl:for-each select="catalog/cd">
  <tr>
   <td><xsl:value-of select="title"/></td>
   <xsl:choose>
    <xsl:when test="price &gt; 10">
     <td bgcolor="#ff00ff">
     <xsl:value-of select="artist"/></td>
    </xsl:when>
    <xsl:otherwise>
     <td><xsl:value-of select="artist"/></td>
    </xsl:otherwise>
   </xsl:choose>
  </tr>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

The code above will add a pink background-color to the "Artist" column WHEN the price of the CD is higher than 10.

Another Example

Here is another example that contains two <xsl:when> elements:

Example

```
<?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>
 <body>
 <h2>My CD Collection</h2>
 <table border="1">
  <tr bgcolor="#9acd32">
   <th>Title</th>
   <th>Artist</th>
  </tr>
  <xsl:for-each select="catalog/cd">
  <tr>
   <td><xsl:value-of select="title"/></td>
   <xsl:choose>
    <xsl:when test="price &gt; 10">
     <td bgcolor="#ff00ff">
     <xsl:value-of select="artist"/></td>
    </xsl:when>
    <xsl:when test="price &gt; 9">
     <td bgcolor="#cccccc">
     <xsl:value-of select="artist"/></td>
    </xsl:when>
    <xsl:otherwise>
     <td><xsl:value-of select="artist"/></td>
    </xsl:otherwise>
   </xsl:choose>
  </tr>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

The code above will add a pink background color to the "Artist" column WHEN the price of the CD is higher than 10, and a grey background-color WHEN the price of the CD is higher than 9 and lower or equal to 10.

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

The <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.

If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

Look at the following XSL style sheet:

Example

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
```

```
  <br />
</xsl:template>

<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

</xsl:stylesheet>
```

Try it Yourself

If your browser supports it, XSLT can be used to transform the document to XHTML in your browser.

A JavaScript Solution

In the previous chapters we have explained how XSLT can be used to transform a document from XML to XHTML. We did this by adding an XSL style sheet to the XML file and let the browser do the transformation.

Even if this works fine, it is not always desirable to include a style sheet reference in an XML file (e.g. it will not work in a non XSLT aware browser.)

A more versatile solution would be to use a JavaScript to do the transformation.

By using a JavaScript, we can:

- do browser-specific testing
- use different style sheets according to browser and user needs

That is the beauty of XSLT! One of the design goals for XSLT was to make it possible to transform data from one format to another, supporting different browsers and different user needs.

XSLT transformation on the client side is bound to be a major part of the browsers work tasks in the future, as we will see a growth in the specialized browser market (Braille, aural browsers, Web printers, handheld devices, etc.)

The XML File and the XSL File

Look at the XML document that you have seen in the previous chapters:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

View the XML file.

And the accompanying XSL style sheet:

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title" /></td>
      <td><xsl:value-of select="artist" /></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
```

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

View the XSL file.

Notice that the XML file does not have a reference to the XSL file.

IMPORTANT: The above sentence indicates that an XML file could be transformed using many different XSL style sheets.

Transforming XML to XHTML in the Browser

Here is the source code needed to transform the XML file to XHTML on the client:

Example

```
<html>
<head>
<script>
function loadXMLDoc(dname)
{
if (window.XMLHttpRequest)
  {
  xhttp=new XMLHttpRequest();
  }
else
  {
  xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xhttp.open("GET",dname,false);
xhttp.send("");
return xhttp.responseXML;
}

function displayResult()
{
xml=loadXMLDoc("cdcatalog.xml");
xsl=loadXMLDoc("cdcatalog.xsl");
// code for IE
if (window.ActiveXObject)
  {
  ex=xml.transformNode(xsl);
  document.getElementById("example").innerHTML=ex;
  }
```

```
// code for Mozilla, Firefox, Opera, etc.
else if (document.implementation && document.implementation.createDocument)
  {
  xsltProcessor=new XSLTProcessor();
  xsltProcessor.importStylesheet(xsl);
  resultDocument = xsltProcessor.transformToFragment(xml,document);
  document.getElementById("example").appendChild(resultDocument);
  }
}
</script>
</head>
<body onload="displayResult()">
<div id="example" />
</body>
</html>
```

Try it Yourself

Tip: If you don't know how to write JavaScript, you can study our JavaScript tutorial.

Example Explained:

The loadXMLDoc() Function

The loadXMLDoc() function is used to load the XML and XSL files.

It checks what kind of browser the user has and loads the file.

The displayResult() Function

This function is used to display the XML file styled by the XSL file.

- Load XML and XSL file
- Test what kind of browser the user has
- If the user has a browser supporting the ActiveX object:
  - Use the transformNode() method to apply the XSL style sheet to the xml document
  - Set the body of the current document (id="example") to contain the styled xml document
- If the user has a browser that does not support the ActiveX object:
  - Create a new XSLTProcessor object and import the XSL file to it
  - Use the transformToFragment() method to apply the XSL style sheet to the xml document
  - Set the body of the current document (id="example") to contain the styled xml document

- Since not all browsers support XSLT, one solution is to transform the XML to XHTML on the server.
-

- # A Cross Browser Solution
- In the previous chapter we explained how XSLT can be used to transform a document from XML to XHTML in the browser. We created a JavaScript that used an XML parser to do the transformation. The JavaScript solution will not work in a browser that doesn't have an XML parser.
- To make XML data available to all kind of browsers, we must transform the XML document on the SERVER and send it as XHTML back to the browser.
- That's another beauty of XSLT. One of the design goals for XSLT was to make it possible to transform data from one format to another on a server, returning readable data to all kinds of browsers.
-

- # The XML File and the XSLT File
- Look at the XML document that you have seen in the previous chapters:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

- View the XML file.
- And the accompanying XSL style sheet:

```
<?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>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
```

```
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title" /></td>
      <td><xsl:value-of select="artist" /></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

- View the XSL file.
- **Notice that the XML file does not have a reference to the XSL file.**
- **IMPORTANT:** The above sentence indicates that an XML file could be transformed using many different XSL style sheets.
- ---

Transforming XML to XHTML on the Server

- Here is the ASP source code needed to transform the XML file to XHTML on the server:

```
<%
'Load XML
set xml = Server.CreateObject("Microsoft.XMLDOM")
xml.async = false
xml.load(Server.MapPath("cdcatalog.xml"))

'Load XSL
set xsl = Server.CreateObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(Server.MapPath("cdcatalog.xsl"))

'Transform file
Response.Write(xml.transformNode(xsl))
%>
```

- **Tip:** If you don't know how to write ASP, you can study our ASP tutorial.
- The first block of code creates an instance of the Microsoft XML parser (XMLDOM), and loads the XML file into memory. The second block of code creates another instance of the parser and loads the XSL file into memory. The last line of code transforms the XML document using the XSL document, and sends the result as XHTML to your browser. Nice!
- See how it works.
- Data stored in XML files can be edited from an Internet browser.
- ---

- # Open, Edit and Save XML
- Now, we will show how to open, edit, and save an XML file that is stored on the server.
- We will use XSL to transform the XML document into an HTML form. The values of the XML elements will be written to HTML input fields in an HTML form. The HTML form is editable. After editing the data, the data is going to be submitted back to the server and the XML file will be updated (this part is done with ASP).
- ---
- # The XML File and the XSL File
- First, look at the XML document that will be used ("tool.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tool>
  <field id="prodName">
    <value>HAMMER HG2606</value>
  </field>
  <field id="prodNo">
    <value>32456240</value>
  </field>
  <field id="price">
    <value>$30.00</value>
  </field>
</tool>
```

- View the XML file.
- Then, take a look at the following style sheet ("tool.xsl"):

```
<?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>
  <body>
  <form method="post" action="edittool.asp">
  <h2>Tool Information (edit):</h2>
  <table border="0">
    <xsl:for-each select="tool/field">
    <tr>
      <td><xsl:value-of select="@id"/></td>
      <td>
      <input type="text">
      <xsl:attribute name="id">
        <xsl:value-of select="@id" />
      </xsl:attribute>
      <xsl:attribute name="name">
        <xsl:value-of select="@id" />
```

```
    </xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="value" />
    </xsl:attribute>
    </input>
    </td>
   </tr>
   </xsl:for-each>
  </table>
  <br />
  <input type="submit" id="btn_sub" name="btn_sub" value="Submit" />
  <input type="reset" id="btn_res" name="btn_res" value="Reset" />
  </form>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

- View the XSL file.
- The XSL file above loops through the elements in the XML file and creates one input field for each XML "field" element. The value of the XML "field" element's "id" attribute is added to both the "id" and "name" attributes of each HTML input field. The value of each XML "value" element is added to the "value" attribute of each HTML input field. The result is an editable HTML form that contains the values from the XML file.
- Then, we have a second style sheet: "tool_updated.xsl". This is the XSL file that will be used to display the updated XML data. This style sheet will not result in an editable HTML form, but a static HTML table:

```
<?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>
  <body>
  <h2>Updated Tool Information:</h2>
  <table border="1">
    <xsl:for-each select="tool/field">
    <tr>
      <td><xsl:value-of select="@id" /></td>
      <td><xsl:value-of select="value" /></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
```

```
  </html>
</xsl:template>

</xsl:stylesheet>
```

- View the XSL file.
- ---
- # The ASP File
- The HTML form in the "tool.xsl" file above has an action attribute with a value of "edittool.asp".
- The "edittool.asp" page contains two functions: The loadFile() function loads and transforms the XML file for display and the updateFile() function applies the changes to the XML file:

```
<%
function loadFile(xmlfile,xslfile)
Dim xmlDoc,xslDoc
'Load XML file
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async = false
xmlDoc.load(xmlfile)
'Load XSL file
set xslDoc = Server.CreateObject("Microsoft.XMLDOM")
xslDoc.async = false
xslDoc.load(xslfile)
'Transform file
Response.Write(xmlDoc.transformNode(xslDoc))
end function

function updateFile(xmlfile)
Dim xmlDoc,rootEl,f
Dim i
'Load XML file
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
xmlDoc.async = false
xmlDoc.load(xmlfile)

'Set the rootEl variable equal to the root element
Set rootEl = xmlDoc.documentElement

'Loop through the form collection
for i = 1 To Request.Form.Count
  'Eliminate button elements in the form
  if instr(1,Request.Form.Key(i),"btn_")=0 then
    'The selectSingleNode method queries the XML file for a single node
    'that matches a query. This query requests the value element that is
```

```
    'the child of a field element that has an id attribute which matches
    'the current key value in the Form Collection. When there is a match -
    'set the text property equal to the value of the current field in the
    'Form Collection.
    set f = rootEl.selectSingleNode("field[@id='" & _
    Request.Form.Key(i) & "']/value")
    f.Text = Request.Form(i)
  end if
next

'Save the modified XML file
xmlDoc.save xmlfile

'Release all object references
set xmlDoc=nothing
set rootEl=nothing
set f=nothing

'Load the modified XML file with a style sheet that
'allows the client to see the edited information
loadFile xmlfile,server.MapPath("tool_updated.xsl")
end function

'If the form has been submitted update the
'XML file and display result - if not,
'transform the XML file for editing
if Request.Form("btn_sub")="" then
  loadFile server.MapPath("tool.xml"),server.MapPath("tool.xsl")
else
  updateFile server.MapPath("tool.xml")
end if
%>
```

- **Tip:** If you don't know how to write ASP, you can study our ASP tutorial.
- **Note:** We are doing the transformation and applying the changes to the XML file on the server. This is a cross-browser solution. The client will only get HTML back from the server - which will work in any browser.

If you are serious about XML, you will benefit from using a professional XML Editor.

XML is Text-based

XML is a text-based markup language.

One great thing about XML is that XML files can be created and edited using a simple text-editor like Notepad.

However, when you start working with XML, you will soon find that it is better to edit XML documents using a professional XML editor.

Why Not Notepad?

Many web developers use Notepad to edit both HTML and XML documents because Notepad is included with the most common OS and it is simple to use. Personally I often use Notepad for quick editing of simple HTML, CSS, and XML files.

But, if you use Notepad for XML editing, you will soon run into problems.

Notepad does not know that you are writing XML, so it will not be able to assist you.

Why an XML Editor?

Today XML is an important technology, and development projects use XML-based technologies like:

- XML Schema to define XML structures and data types
- XSLT to transform XML data
- SOAP to exchange XML data between applications
- WSDL to describe web services
- RDF to describe web resources
- XPath and XQuery to access XML data
- SMIL to define graphics

To be able to write error-free XML documents, you will need an intelligent XML editor!

XML Editors

Professional XML editors will help you to write error-free XML documents, validate your XML against a DTD or a schema, and force you to stick to a valid XML structure.

An XML editor should be able to:

- Add closing tags to your opening tags automatically

- Force you to write valid XML
- Verify your XML against a DTD
- Verify your XML against a Schema
- Color code your XML syntax

XSLT Summary

This tutorial has taught you how to use XSLT to transform XML documents into other formats, like XHTML.

You have learned how to add/remove elements and attributes to or from the output file.

You have also learned how to rearrange and sort elements, perform tests and make decisions about which elements to hide and display.

For more information on XSLT, please look at our XSLT reference.