XSLT 1.0 Reference

Purpose: A reference to the XSL Transformation language XSLT. XSLT relies on Xpath. XSLT and XPATH are collectively known as XSL.

Document group: Software Development Approach for XML

Sources:

Author: John Bentley

Template

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  version="1.0">

 <xsl:output omit-xml-declaration="yes" />

 <xsl:param name="nowLocalFriendly">Wed, 16 Jun 1999 11:02 (+1000 AUS
Eastern Standard Time)</xsl:param>

    <!-- Match: Match: The document root (not the root element)  -->
  <xsl:template match="/">
<!-- Keep the closing text tag indented flush for output formatting -->
    <xsl:text disable-output-escaping="yes">&lt;!DOCTYPE html PUBLIC "-
//w3c//dtd xhtml 1.1//en"
"http://www.w3.org/tr/xhtml11/dtd/xhtml11.dtd"&gt;
</xsl:text>
    <xsl:copy>
      <xsl:apply-templates select="*|comment()"/>
    </xsl:copy>
  </xsl:template>

  <!--  Summary:   Identity transformation
  -->
  <xsl:template match="*">
    <xsl:element name="{name(.)}" >
      <xsl:for-each select="@*">
          <xsl:attribute name="{name(.)}" >
                <xsl:value-of select="."/>
          </xsl:attribute>
      </xsl:for-each>
      <xsl:apply-templates />
    </xsl:element>
  </xsl:template>

  <!--
      Match:   Elements with empty content, explicitly named,
                and regardles of their namespace.
      Example Output:
                <br />
                <meta name="author" content="John Bentley" />
                <cool:myEmptyTag cool:feeling="nice" />
```

```
  -->
  <xsl:template match=
                   "*[local-name()='br']
                   | *[local-name()='meta']
                   | *[local-name()='link']
                   | *[local-name()='myEmptyTag']">
  <xsl:text disable-output-escaping="yes">&lt;</xsl:text>
  <xsl:value-of select="name(.)" />
  <xsl:for-each select="@*" >
    <xsl:text> </xsl:text>
    <xsl:value-of select="name(.)" />="<xsl:value-of
select="."/>"</xsl:for-each>
  <xsl:text disable-output-escaping="yes"> /&gt;</xsl:text>
  </xsl:template>

  <xsl:template match="comment()">
   <xsl:copy /><xsl:text>&#10;</xsl:text>
  </xsl:template>
  <!-- *********** Website specific bellow here ******************* -->

  <!-- Make a match whether the element belongs to a namespace
       or belongs to no namespace -->
  <xsl:template match="*[local-name()='menu']" xml:space="preserve">
          <div class="menu" >
            <a href="index.htm" class="menuCommand">Home</a>
            <br />
            <a href="news.htm" class="menuCommand">News</a>
            <br />
          </div>
  </xsl:template>

  <xsl:template match="*[local-name()='websiteLastUpdated']"
xml:space="preserve" >
              <div class="date" style="text-align: center;">
                Web Site Last Updated:  <xsl:value-of
select="$nowLocalFriendly" />
                <br />
                <br />
              </div>
  </xsl:template>
</xsl:stylesheet>
```

Processing

General

A template rule is invoked when the specified node, as expressed in an XPath pattern, is matched.

```
<!-- match the h1 elements with a class attribute of 'special, wack it in a
table, and do no more with respect to this template -->
<xsl:template match="h1[@class='special']" >
  <tr style="background-color: mediumblue">
    <td><xsl:value-of select="name(.)" /></td>
    <td><xsl:value-of select="." /></td>
  </tr>
</xsl:template>
```

Select patterns return a node set (or value). Match patterns return a boolean. That is, they test against a particular node.

```
<!-- For the element with an id attribute of 'pMessage' out put results of
template the apply any other templates to children that are text, comments,
elements, processing instructions, or attributes -->
<xsl:template match="*[@id='pMessage']" >
  <tr class="specific">
    <td><xsl:value-of select="name(.)" /></td>
    <td><xsl:value-of select="." /></td>
  </tr>
  <xsl:apply-templates select="node()|@*" />
</xsl:template>
```

If there are multiple matches of a node, the template rule is applied to all of them.

```
<!-- For every element put it's name inside a paragraph with the letters
'XXX' at the beginning then process the children of the matched element -->
<xsl:template match="*" >
  <p>XXX<xsl:value-of select="name(.)" /></p>
  <xsl:apply-templates  select="node()|@*" />
</xsl:template>
```

If a pattern specified in an XSLT style sheet that does not match any node in the source document, the associated template rule is ignored.

When the XSLT processor begins to process the XSLT tree, the processor looks for the template rule that points to the document root (not the root element) in the source tree. It begins there.

```
<!-- Match the document root -->
<xsl:template match="/" xml:space="preserve" ><html>
  <head>
        <link type="text/css" rel="styleUheet" href="Temp.css" />
  </head>
  <body>
    <table >
      <xsl:apply-templates select="*"/>
    </table>
  </body>
</html>
  </xsl:template>
```

Note that if you don't explicitly code a match="/" template then the built in template for the document root will be invoked.

You can use either <xsl:for-each> or <xsl:apply-templates>, to achieve the same effect of iterating over a node set.

```
<!-- This XML file ... -->
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="portfolioIterationApplyTemplates.xslt"?>
<portfolio xmlns:dt="urn:schemas-microsoft-com:datatypes"
xml:space="preserve">
  <stock exchange="nyse">
    <name>zacx corp</name>
    <symbol>ZCXM</symbol>
    <price dt:dt="number">28.875</price>
  </stock>
  <stock exchange="nasdaq">
    <name>zaffymat inc</name>
    <symbol>ZFFX</symbol>
    <price dt:dt="number">92.250</price>
```

```
  </stock>
  <stock exchange="nasdaq">
    <name>zysmergy inc</name>
    <symbol>ZYSZ</symbol>
    <price dt:dt="number">20.313</price>
  </stock>
</portfolio>

<!-- ... with either portfolioIterationFor.xslt or .. -->
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE BORDER="2">
          <TR>
            <TD>Symbol</TD>
            <TD>Name</TD>
            <TD>Price</TD>
          </TR>
          <xsl:for-each select="portfolio/stock">
            <TR>
              <TD><xsl:value-of select="symbol"/></TD>
              <TD><xsl:value-of select="name"/></TD>
              <TD><xsl:value-of select="price"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>

<!-- ... portfolioIterationApplyTemplates.xslt ... -->
<?xml version='1.0' ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE BORDER="2">
          <TR>
            <TD>Symbol</TD>
            <TD>Name</TD>
            <TD>Price</TD>
          </TR>
            <xsl:apply-templates select="portfolio/stock" />
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="portfolio/stock">
    <TR>
      <TD><xsl:value-of select="symbol" /></TD>
      <TD><xsl:value-of select="name" /></TD>
      <TD><xsl:value-of select="price" /></TD>
```

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

<!-- yields the same result: -->

Symbol	Name	Price
ZCXM	zacx corp	28.875
ZFFX	zaffymat inc	92.250
ZYSZ	zysmergy inc	20.313

Recursion

Because a `<section>` element can contain other `<section>` elements recursively, the template rule below will be applied recursively.

```
<xsl:template match="section">
  <DIV>
    <xsl:apply-templates />
  </DIV>
</xsl:template>
```

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Recurse through the elements:

```
  <xsl:template match="/">
    <html>
      <head><title></title>
        <link rel="stylesheet" href="JohnStyle.css" />
        <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema" />
      </head>
      <body>
        <table>
          <xsl:apply-templates />
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="*">
    <tr>
      <td>
        <xsl:value-of select="name()" />
      </td>
      <td>
        <xsl:value-of select="." />
      </td>
    </tr>
    <xsl:apply-templates select="*"/>
  </xsl:template>
```

Apply-templates

The following tells the XSL Processor to "go to each child **node** of the current context and execute the template rule for the node"

```
<xsl:apply-templates />
```

The following tells the XSL Processor to "go to each child **element** node of the current context and execute the template for the node."

```
<xsl:apply-templates select="*"/>
```

Apply-templates defaults to `select="node()"`.

```
<!-- The following are equivalent -->
<xsl:apply-templates  />
<xsl:apply-templates  select="node()" />
```

Recall that node() tests only for: elements, text, comment, processing-instruction.

When the XSLT processor begins to process the XSLT tree, the processor looks for the template rule that points to the document root (not the root element) in the source tree. The only way it can then move to another template is via an apply-template instruction contained within the document root template rule.

```
<xsl:template match="/" />

<!-- This can't match anything because the above document root template
closes off further traversal.
<xsl:template match="//*" >
  <xsl:apply-templates />
</xsl:template>
```

Processing then proceeds from template rule to template rule ONLY VIA apply-templates.

An `apply-templates` rule applies only to the immediate children of the context node not its grandchildren nor further descendants (at least not directly).

An `apply-templates` rule applies only to the immediate children of the context node not further descendants (at least not directly).

```
<!-- Source Document -->
<?xml version='1.0' ?>
<?xml-stylesheet type="text/xsl" href="TypeRecurs.xslt"?>
<html  xml:lang="en" >
  <head>
    <title>Trying to Remove User Namespace in XSLT Output</title>
    <!-- A comment up the top -->
    <meta name="author" content="john bentley" />
  </head>
  <body>
    <h1>Some Heading</h1>
    <p id="p1">My first paragraph.</p>
  </body>
</html>

<!-- XSLT Stylesheet -->
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">
```

```
  <xsl:output method="xml"
    omit-xml-declaration="yes"
     />

  <xsl:template match="/" xml:space="preserve" ><html>
  <head>
        <link type="text/css" rel="stylesheet" href="Temp.css" />
  </head>
  <body>
    <table >
      <xsl:apply-templates select="node()|@*"/>
    </table>
  </body>
</html>
  </xsl:template>

  <xsl:template match="html" xml:space="preserve">
    <tr class="specific">
      <td><xsl:value-of select="name(.)" /></td>
      <td>Matched with "html" template</td>
    </tr>
    <!-- Select only the immediate element children of html -->
    <xsl:apply-templates  select="*" />
  </xsl:template>

  <xsl:template match="*" xml:space="preserve">
    <tr class="element">
      <td><xsl:value-of select="name(.)" /></td>
      <td>Matched with "*" template</td>
    </tr>
    <!-- No apply templates here so stop processing -->
  </xsl:template>
</xsl:stylesheet>
```

Output:

html	Matched with "html" template
head	Matched with "*" template
body	Matched with "*" template

Only the immediate children of html, head and body, are output.

Traversal Order

Traversal of the nodes via recursion proceeds as deep as possible along a branch to the leaf before moving back up ONLY ONE LEVEL to process the next child and thence as deep as possible along that branch.

We can change the above stylesheet to allow processing of all the elements, along every branch, all the way to a leaf. We just change the element template above to become:

```
<xsl:template match="*" xml:space="preserve">
  <tr class="element">
    <td><xsl:value-of select="name(.)" /></td>
  </tr>
  <xsl:apply-templates  select="*" />
</xsl:template>
```

This allows us to examine the order in which elements are processed.

We now might have a source file that contains this tree fragment:

```
<body>
  <p>
    <em>Emphatic
      <i>and italicised paragraph.</i>
    </em>
    <b>Some bold.</b>
  </p>
  <span>More writing</span>
</body>
```

The interesting part of the output is as follows:

body
p
em
i
b
span

Observe that after the leaf <i> it is <b> that is processed NOT

Conflict Resolution

The more specific match patterns override (have a higher priority than) the more general ones.

Patterns at the BOTTOM of the table override those at the top:

	What	Examples	Priority
More Specific ↓	Templates with lower import precedence		It depends.

	Patterns that just tests for nodes with particular types	`*`
		`@*`
		`node()`
		`comments()`
		`text()`
		`processing-instruction()`
			-0.5

	Pattern that tests for a node with a particular type and an expanded-name with a particular namespace URI	`user:*` `@user:*`	-0.25

	The most common kind of pattern (Pattern that tests for a node with a particular type and a particular expanded-name)	`cool`
		`@cool`
		`user:cool`
		`@user:cool`
			0

	Patterns more specific than the most common kind of pattern.	`cool[@shirt]`	0.5

	Patterns where the `priority` attribute of the `template` element is higher.	`<xsl:template priority="1" match="node()|@*"`	user specified (> 0.5)

If more than one template rule matches a given node in the source tree, MSXML gives the highest priority to the template rule that matches the most specific pattern.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

If templates have the same priority then the last in the stylesheet wins.

Technically any templates competing after the above conflict resolution should throw an error. Parsers are permitted to, and in practice do, give final priority to the competing template last in the style sheet.

Templates later in the style sheet override earlier templates if they select the same node.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

For readability order your templates in the order that they will be processed (as far as possible). Usually this will be general to specific as you go down the stylesheet.

Built-In Template rules

If a pattern specified in an XSLT style sheet that does not match any node in the source document, the associated template rule is ignored. If there are no matches, the processor uses one of the built-in template rules to decide how to handle the node.

```
<!-- The built-in template rules -->
<xsl:template match="*|/">
    <xsl:apply-templates />
</xsl:template>

<xsl:template match="*|/" mode="m">
    <xsl:apply-templates mode="m"/>
</xsl:template>

<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="processing-instruction()|comment()"/>
```

Built in templates lie invisibly at the **top** of the document according to conflict resolution rules.

Syntax

Variables and Parameters

A XSLT variable cannot change their value until they go out of scope.

A XSLT parameter can change their value during the run of the style sheet.

XSLT variables and parameters are global or local.

XSLT variables and parameters are referred to with $.

Variable example

```
<xsl:template match="weather">
   <H1>Weather Readings</H1>
   <xsl:for-each select="day">
      <H2>As of <xsl:value-of select="@date"/></H2>
      <xsl:for-each select="locale">
         <xsl:variable name="placename">
             <xsl:choose>
                 <xsl:when test="@place='location1'">Midtown</xsl:when>
                 <xsl:when test="@place='location2'">Northeast</xsl:when>
                 <xsl:when test="@place='location3'">Airport</xsl:when>
                 <xsl:otherwise>[Unknown Locale]</xsl:otherwise>
             </xsl:choose>
         </xsl:variable>
         <H3><xsl:value-of select="$placename"/></H3>
      </xsl:for-each>
   </xsl:for-each>
</xsl:template>
```

To pass a parameter from one template rule to another, use the <xsl:with-param> element as a child of an <xsl:apply-templates> or <xsl:call-template> element.

```
<xsl:call-template name="some_named_template">
    <xsl:with-param name="param1" select="@some_attrib"/>
</xsl:call-template>
```

Sorting

When sorting be careful to specify the data type for numeric values.

```
<xsl:apply-templates>
  <xsl:sort select="price" data-type="number" />
</xsl:apply-templates>
```

Sorting on calculated variables see:

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Include and Import

Any template rules **imported** into a style sheet have a lower import precedence than those that physically reside in that style sheet. Template rules **included** in a style sheet have the same import precedence as those in the including style sheet.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

All <xsl:import> elements must appear as the first top-level elements in the importing style sheet. By contrast, <xsl:include> elements can appear anywhere among other top-level elements in the including style sheet.

Copy

<xsl:copy /> copies the current node, including namespaces (if not defined at ancestor node), but not child nodes, nor child attributes.

```
 <!-- Source -->
<em style="color: blue" user:factor="12" xmlns:user="urn:space:nice">
  Emphatic
  <i>and italicised paragraph.</i>
</em>

<!-- Stylesheet -->
<xsl:template match="*[local-name()='em']" >
  <xsl:copy />
</xsl:template>

<!-- Output -->
<em xmlns:user="urn:space:nice" />
```

Content inside `<xsl:copy></xsl:copy>` determines the children and attributes that are copied.

```
<!-- Stylesheet changed to -->
<xsl:template match="*[local-name()='em']" >
  <!-- Only attribute children specified as being copied accross -->
  <xsl:copy ><xsl:apply-templates select="@*" /></xsl:copy>
</xsl:template>

<xsl:template match="@*">
  <!-- we only neet to worry about copying attributes.
  <xsl:copy />
</xsl:template>

<!-- Output -->
<em style="color: blue" user:factor="12" xmlns:user="urn:space:nice" />
```

Copy-of

What `xsl:copy-of` copies depends on the object type returned by the select statement.

Select Expression Returns	What is copied.
Result tree fragment	That result tree fragment
Node-set	Copy all nodes in the set in document order. For each node in the set copy node and all descendants (including attributes, namespaces, and element children).
Boolean, String, number	A string representation.

xsl:copy-of example when node-set returned:

```
<!-- Source -->
<body>
  <p>
  <em style="color: blue" user:factor="12" xmlns:user="urn:space:nice">
    Emphatic
    <i>and italicised paragraph.</i>
  </em>
  <br />
  </p>
  <span>More writing</span>
  <div><code>for i = 1</code><samp>cool</samp></div>
</body>

<!-- Style Sheet -->
```

```
<xsl:template match="*[local-name()='body']" >
  <xsl:copy-of select="*[local-name()='p']
                     | *[local-name()='div']
                     | ./*/*[local-name()='samp']"/>
</xsl:template>

<!-- Output -->
<!-- the body element is replaced -->
<p>
  <em style="color: blue" user:factor="12" xmlns:user="urn:space:nice">
    Emphatic
    <i>and italicised paragraph.</i>
  </em>
  <br />
  </p>
<div><code>for i = 1</code><samp>cool</samp></div>
<samp>cool</samp>

<!-- Note the <samp> node in the node-set in the select expression. -->
```

xsl:copy-of example when string returned (conceptually identical for the return of number or boolean):

```
<!-- Source, as above -->

<!-- Stylesheet -->
<xsl:template match="*[local-name()='body']" >
  <body>
    <xsl:copy-of select="concat('very ', ./*/*[local-name()='samp'])"/>
  </body>
</xsl:template>

<!-- Output -->
<body>very cool</body>
```

Whitespace

To control white space, use the following XSLT constructs:

```
<xsl:preserve-space>
```

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

A single Line Feed in XML Makes for a new line.

```
<xsl:text>&#10;</xsl:text>
```

Also:

```
<xsl:template match="*[local-name()='menu']" xml:space="preserve">
...
</xsl:template>
```

How the XSLT processor treats white space in the document depends on the white space it receives from the Microsoft® Document Object Model (DOM) processor. The only way to set the `preserveWhiteSpace` property to True is to process the DOM through script or a programming language.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Because all of the content of an XML document is, by default, preserved, `<xsl:preserve-space>` is only useful when you have used `<xsl:strip-space>` to override the default behavior

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Techniques

Identity Transformation

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

    <xsl:output method="xml"/>

    <xsl:template match="*">
        <xsl:element name="{name(.)}">
            <xsl:for-each select="@*">
                <xsl:attribute name="{name(.)}">
                    <xsl:value-of select="."/>
                </xsl:attribute>
            </xsl:for-each>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

</xsl:stylesheet>
```

XSL With Namespaces

Identity Transformation with Namespaces

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

    <xsl:output method="xml"/>

    <xsl:template match="*">
      <xsl:element name="{name(.)}" namespace="{namespace-uri(.)}">
          <xsl:copy-of select="namespace::*" />
          <xsl:for-each select="@*">
              <xsl:attribute name="{name(.)}" namespace="{namespace-
uri(.)}">
                    <xsl:value-of select="."/>
              </xsl:attribute>
          </xsl:for-each>

          <xsl:apply-templates/>

      </xsl:element>
    </xsl:template>


</xsl:stylesheet>
```

Select elements within a namespace

```
<?xml version="1.0"?>
```

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

    <xsl:output method="html"/>

    <xsl:template match="/">
        <HTML>
            <BODY>
                <xsl:apply-templates/>
            </BODY>
        </HTML>
    </xsl:template>

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

 <!-- Works for elements and attributes that have a namespace with: an
explicit prefix; no prefix; or a default prefix -->
  <xsl:template match="*[local-name()='Member']">
    Your Name is: <xsl:value-of select="*[local-name()='Name']" /> <br />
      <xsl:for-each select="*[local-name()='Phone']">
         ------Your Phone is: <xsl:value-of select="." /> <br />
         ------Your Phone Type is: <xsl:value-of select="@*[local-
name()='type']" /> <br />
      </xsl:for-each>
    </xsl:template>

    <xsl:template match="text()">
         <!-- Do nothing -->
    </xsl:template>

</xsl:stylesheet>
```

Or

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

    <xsl:output method="html"/>

    <xsl:template match="/">
        <HTML>
            <BODY>
                <xsl:apply-templates/>
            </BODY>
        </HTML>
    </xsl:template>

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

    <xsl:template match="gym:Member">
         Your name is: <xsl:value-of select="gym:Name"/>
    </xsl:template>

    <!-- Do nothing -->
    <xsl:template match="text()" />
```

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

Debugging

Adding this template to the top of your style sheet will show you visually those elements that do not yet have specific templates associated with them.

```
  <xsl:template match="*">
    <span style="background-color:yellow">
      <xsl:attribute name="title">
        Warning:

        &lt;<xsl:value-of select="name()"/>&gt;
        <xsl:value-of select="."/>&lt;<xsl:value-of select="name()"/>&gt;

        does not match a specific template_rule in the xslt stylesheet.
      </xsl:attribute>
      <xsl:apply-templates/>
    </span>
  </xsl:template>
```

Add this template near the top of your style sheet so that it does not override templates that handle specific elements. This template will display a yellow background behind the content of any element without a specific template in existence. Running the mouse over the yellow areas will display a ToolTip showing those elements that must still be handled by templates.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSL Developer's Guide

XSL:Message

```
        <xsl:message>
            XSLT Processor: <xsl:value-of select="system-
property('xsl:vendor')"/>
        </xsl:message>
```

Initiating the Transformation

Transform Using a Stylesheet Processing Instruction from with the XML Document

1. Add a Processing instruction to the XML doc:

```
<?xml-stylesheet type="text/xsl" href="bookreview.xslt"?>
```

2. Open the XML Document in IE.

Any errors in the XML Or XSLT file will be reported in IE. Easy.

Transform using the DOM/Script with a XHTML Page, Client Side

Create a xml file without any need for a processing instruction (here "'TransformUsingDomRegion.xml") and an xslt file (here "'TransformUsingDomRegion.xslt"). Create the following Html File

```
<?xml version="1.0" encoding="UTF-8" ?>
<!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>Quarterly Sales, by Region</title>
    <!-- Note the following line enables intellisense in Visual
             Studio .NET -->
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
    <script src="TransformUsingDomReportErrors.js"></script>
    <script language='JScript'>
    <!--
      function transformIt() {
          // Associate the result tree object with any element(s) whose
          // id attribute is "transformedXML."
          var objResTree = document.all['transformedXML'];
          // Declare two new MSXML DOMDocument objects.
          var objSrcTree = new ActiveXObject('MSXML2.DOMDocument.4.0');
          // When True this Validates against a Schema or DTD.
          // When False it just checks for well-formedness.
          objSrcTree.validateOnParse = true;
          var objXSLT = new ActiveXObject('MSXML2.DOMDocument.4.0');
          objXSLT.validateOnParse = true;
          // Load the two DOMDocuments with the XML document and the
          // XSLT style sheet.
          objSrcTree.load('TransformUsingDomRegion.xml');

          objXSLT.load('TransformUsingDomRegion.xslt');
          // Error Handling
          if (objSrcTree.parseError.errorCode != 0)
            result = reportParseError(objSrcTree.parseError);
          else
          {
            if (objXSLT.parseError.errorCode != 0)
              result = reportParseError(objXSLT.parseError);
            else
            {
              try {
                result = objSrcTree.transformNode(objXSLT);
              }
              catch (exception) {
                result = reportRuntimeError(exception);
              }
            }
          }

          // Use the transformNode method to apply the XSLT to the XML.
          //result = objSrcTree.transformNode(objXSLT);
          // Assign the resulting string to the result tree object's
          // innerHTML property.
          objResTree.innerHTML = result;
          return true;
          }
    -->
    </script>
  </head>
  <body onload='transformIt()'>
    <div id='transformedXML'></div>
  </body>
</html>
```

Then create the following file, "TransformUsingDomReportErrors.js"

```
// Parse error formatting function
```

```
function reportParseError(error)
{
  var s = "";
  for (var i=1; i<error.linepos; i++) {
    s += " ";
  }
  r = "<font face=Verdana size=1><font size=1>XML Error loading <BR />'" +
      error.url + "'</font>" +
      "<P><B>" + error.reason +
      "</B></P></font>";
  if (error.line > 0)
    r += "<font size=2><XMP>" +
    "at line " + error.line + ", character " + error.linepos +
    "\n" + error.srcText +
    "\n" + s + "^" +
    "</XMP></font>";
  return r;
}
// Runtime error formatting function.
function reportRuntimeError(exception)
{
  return "<font face=Verdana size=2><font size=4>XSL Runtime Error</font>"
+
      "<P><B>" + exception.description + "</B></P></font>";
}
```

This includes Error Handling. This script calls MSXML Core Services 4.0. This must be installed.

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Transform using the DOM/Script with a XHTML Page, Server Side

```
<%@ LANGUAGE = JScript %>
<%
  // Set the source and style sheet locations here.
  var sourceFile = Server.MapPath("simple.xml");
  var styleFile = Server.MapPath("simple.xsl");

  // Load the XML.
  var source = Server.CreateObject("MSXML2.DOMDocument.4.0");
  source.async = false;
  source.resolveExternals = false;
  source.load(sourceFile);
  // Load the XSLT.
  var style = Server.CreateObject("MSXML2.DOMDocument.4.0");
  style.async = false;
  style.resolveExternals = false;
  style.load(styleFile);
  Response.Write(source.transformNode(style));
%>
```

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Changing Case

```
<xsl:variable name="upperCaseChars" select=" 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
"/>
<xsl:variable name="lowerCaseChars" select=" 'abcdefghijklmnopqrstuvwxyz'
"/>

translate("Hello", $upperCaseChars, $lowerCaseChars)
```

Date Comparison

Compare a specific Date/Time ('2004-06-15T18:30:00') with a mere date (2004-07-07) like this:

```
<xsl:variable name="now" select="translate('2004-06-15T18:30:00','-T:','')
" />
Now: <xsl:value-of select="$now" />

<xsl:for-each select="*[local-name()='meetings']/*[local-name()='meeting']
    [translate(@date, '-T:','') * 1000000
    &gt;=
    $now]">
...
```

MS XSL Extensions

To use Microsoft XSLT extensions use this in the root element: **xmlns:msxsl="urn:schemas-microsoft-com:xslt"**

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:msxsl="urn:schemas-microsoft-com:xslt">

<xsl:template match="text()"/>

</xsl:stylesheet>
```

Tips

Use curly braces for these attributes:

the attribute of a literal result element (where you literally type what should be output)

```
<a href="#{@id}">
```

the name attribute of xsl:attribute

```
<xsl:attribute name ="{@value}">
```

the name attribute of xsl:pi

```
<xsl:pi name ="{@value}">
```

the name attribute of xsl:element

```
<xsl:element name ="{@value}">
```

the optional attributes of xsl:sort:

```
<xsl:sort order ="{@value}">
  lang ="{@value}">
  data-type ="{@value}">
  case-order ="{@value}">
```

Epiphanies

Sources

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XSLT Developer's Guide

Microsoft

Microsoft XML Core Services (MSXML) 4.0 - XSLT Developer's Guide — Last Dated: 2003

ms-help://MS.MSDNQTR.2003FEB.1033/xmlsdk/htm/xsl_intro_7yw5.htm

MSDN 2003 > Microsoft XML Core Services (MSXML) 4.0 > XPath Developer's Guide

Microsoft

Microsoft XML Core Services (MSXML) 4.0 - XPath Developer's Guide — Last Dated: 2003

ms-help://MS.MSDNQTR.2003FEB.1033/xmlsdk/htm/xpath_devguide_overview_86gn.htm

Grosso and Walsh 2000 > XSL Concepts and Practical Use

Paul Grosso and Norman Walsh

XSL Concepts and Practical Use Version 1.5 — Monday, 12 June 2000

Last Update: Monday, 06 July 2000

http://www.nwalsh.com/docs/tutorials/xsl/xsl/slides.html

W3C 1999 > XSL Transformations 1.0 Recommendation

World Wide Web Consortium

XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999 — 16 Nov 1999

http://www.w3.org/TR/1999/REC-xslt-19991116

Document Licence