TUTORIAL 6

WORKING WITH XSLT AND XPATH

THE HISTORY OF XSL

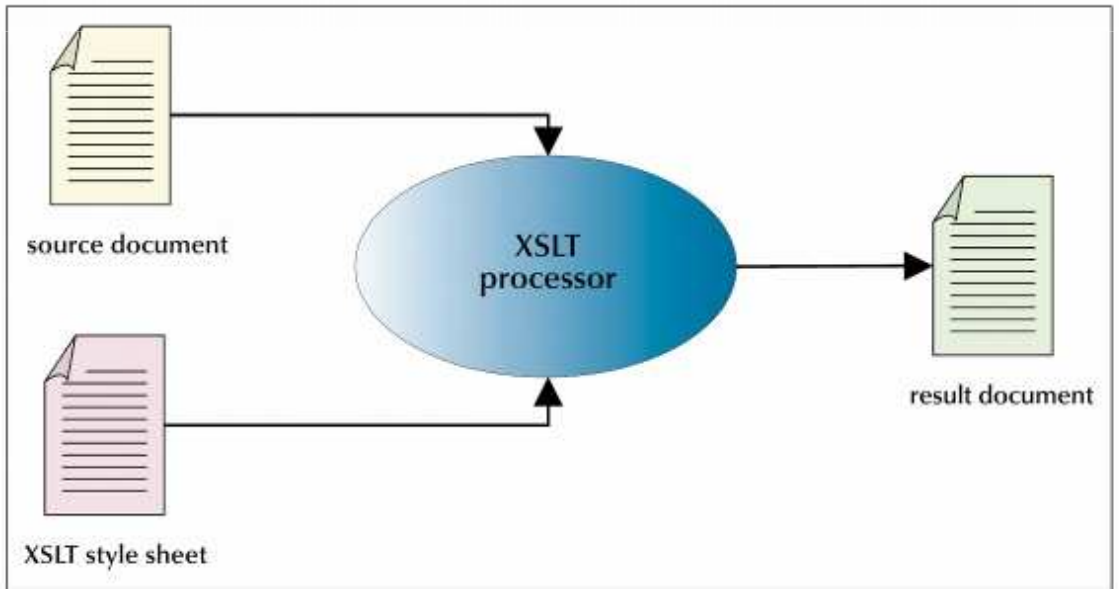
- In 1998, the W3C developed the Extensible Style sheet Language, or XSL
- XSL is composed of two parts:
  - XSL-FO (Extensible Style sheet Language – Formatting Objects) - *layout of paginated documents*
  - XSLT (Extensible Style sheet Language Transformations) – to transform contents of an XML document into another document format

INTRODUCING XSLT STYLE SHEETS AND PROCESSORS

- An XSLT style sheet contains instructions for transforming the contents of an XML document into another format
- An XSLT style sheet document is itself an XML document
- An XSLT style sheet document has an extension .xsl

GENERATING A RESULT DOCUMENT

- An XSLT style sheet converts a source document of XML content into a result document by using the XSLT processor



INTRODUCING XSLT STYLE SHEETS AND PROCESSORS

- The transformation can be performed by a server or a client
- In a server-side transformation, the server receives a request from a client, applies the style sheet to the source document, and returns the result document to the client
- In a client-side transformation, a client requests retrieval of both the source document and the style sheet from the server, then performs the transformation, and generates the result document

CREATING AN XSLT STYLE SHEET

- To create an XSLT style sheet, the general structure:

  <?xml version =“1.0”>

  <xsl:stylesheet version = “1.0”

  xmlns:xsl =“http://www.w3.org/1999/XSL/Transform”>

  Content of the style sheet

  </xsl:stylesheet>

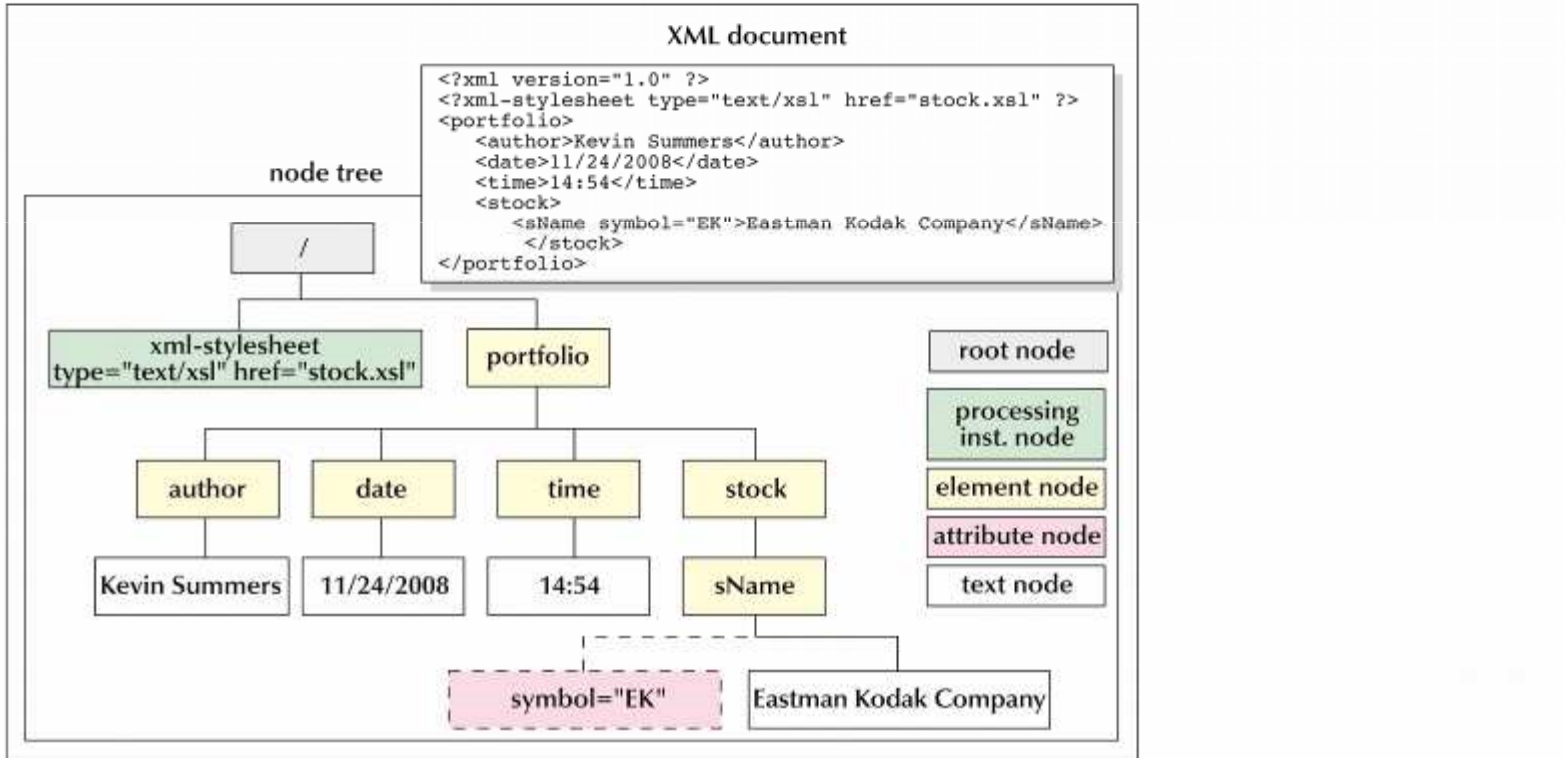
  The <xsl:stylesheet> tag can be substituted for the <xsl:transform> tag

WORKING WITH DOCUMENT NODES

- Under XPath, each component in the document is referred to as a node, and the entire structure of the document is a node tree
- The node tree consists of the following objects:
  - the source document itself
  - comments
  - processing instructions
  - namespaces
  - elements,
  - element text
  - element attributes

NODE TREE EXAMPLE

A sample node tree **Figure 6-7**



WORKING WITH DOCUMENT NODES

- At the top of the node is the root node
- A node that contains other nodes is called a parent node, and the nodes contained in the parent are called child nodes
- Nodes that share a common parent are called sibling nodes
- Any node below another node is referred to as a descendant of that node

WORKING WITH DOCUMENT NODES

- Nodes are distinguished based on the object they refer to in the document
- A node for an element is called an element node
- The node that stores element attributes is called an attribute node

USING XPATH TO REFERENCE A NODE

- XPath provides the syntax to refer to the various nodes in the node tree
- The syntax is used by operation system to specify file pathnames
- The location of a node can be expressed in either absolute or relative terms
- XPath also does data extraction

RELATIVE PATHS

- With a relative path, the location of the node is indicated relative to a specific node in the tree called the context node

Figure 6-8 Relative path expressions

Relative path	Description
.	Refers to the context node
..	Refers to the parent of the context node
child	Refers to the child of the context node named *child*
child1/child2	Refers to the *child2* node, a child of the *child1* node beneath the context node
../sibling	Refers to a sibling of the context node named *sibling*
//descendant	Refers to a descendant of the context node named *descendant*

USING XPATH TO REFERENCE A NODE

- For absolute path, XPath begins with the root node, identified by a forward slash and proceeds down the levels of the node tree
- An absolute path: /child1/child2/child3/…
- To reference an element without regard to its location in the node tree, use a double forward slash with the name of the descendant node
- A relative path : //descendant

REFERENCING GROUPS OF ELEMENTS

- XPath allows you to refer to groups of nodes by using the wildcard character (*)
- To select all of the nodes in the node tree, you can use the path:

  //*

  The (*) symbol matches any node, and the (//)symbol matches any level of the node tree

  Example: /portfolio/stock/*

REFERENCING ATTRIBUTE NODES

- XPath uses different notation to refer to attribute nodes
- The syntax for attribute node is:

  @attribute

  where *attribute* is the name of the attribute

  Example: /portfolio/stock/name/@symbol

WORKING WITH TEXT NODES

- The text contained in an element node is treated as a text node
- The syntax for referencing a text node is:

  text()
- To match all text nodes in the document, use:

  //text()

CREATING THE ROOT TEMPLATE

- A template is a collection of elements that define how a particular section of the source document should be transformed in the result document
- The root template sets up the initial code for the result document

CREATING A TEMPLATE

- To create a template, the syntax is:

  <xsl:template match="node set">

  styles

  </xsl:template>

  –where *node set* is an XPath expression that references a node set from the source document and *styles* are the XSLT styles applied to those nodes

CREATING A ROOT TEMPLATE

- To create a root template, the syntax is:

  <xsl:template match="/">

  styles

  </xsl:template>

CREATING THE ROOT TEMPLATE

- A template contains two types of content: XSLT elements and literal result elements
  - XSLT elements are those elements that are part of the XSLT namespace and are used to send commands to the XSLT processor
  - A literal result element is text sent to the result document, but not acted upon by the XSLT processor

CREATING THE ROOT TEMPLATE EXAMPLE

literal result elements

```
<xsl:template match="/">
   <html>
   <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
   </head>
   <body>
      <h1>Hardin Financial</h1>
      <h2>Stock Information</h2>
   </body>
   </html>
</xsl:template>
```

SPECIFYING THE OUTPUT METHOD

- By default, the XSLT processor will render the result document as an XML file
- To control how the processor formats the source document, you can specify the output method using the <xsl:output *attributes*/> element – where attributes is the list of attributes that define the output format of the result document

ATTRIBUTS OF THE <XSL:OUTPUT/> ELEMENT

Attributes of the output element — Figure 6-12

Attribute	Description
method	Defines the output format using the value xml, html, or text
version	Specifies the version of the output
encoding	Specifies the character encoding
omit-xml-declaration	Specifies whether to omit an XML declaration in the first line of the result document (yes) or to include it (no)
standalone	Specifies whether a standalone attribute should be included in the output and sets its value (yes or no)
doctype-public	Sets the URI for the public identifier in the <!DOCTYPE> declaration
doctype-system	Sets the system identifier in the <!DOCTYPE> declaration
cdata-section-elements	Specifies a list of element names whose content should be output in CDATA sections
indent	Specifies whether the output should be indented to better display its structure (indentations are automatically added to HTML files without use of this attribute)
media-type	Sets the MIME type of the output

TRANSFORMING A DOCUMENT

- A browser with a built-in XSLT processor allows you to view the result document
- Alternatively, you can use XML Spy to create the result document as a separate file, and then view that file in your browser
- Most XSLT processors provide the capability to create the result document as a separate file

VIEWING THE RESULT DOCUMENT IN A BROWSER

- Internet Explorer 6.0 contains built-in XSLT processor
- You can view the results of the transformation by opening the result document in the browser

CREATING AN HTML FILE IN XML SPY

Not used in the course

- One advantage of creating a separate HTML file is that it can be viewed in any Web browser
- You have to regenerate the HTML file every time you make a change to the source document, or the style sheet
- The XSLT processor adds one extra line to the document that provides additional information to the browser about the content of the document and its encoding

EXTRACTING ELEMENT VALUES

- To insert a node's value into the result document, the syntax is:
  - <xsl:value-of> select="e*xpression*" />
  - where *expression* is an expression that identifies the node from the source document's node tree
- If the node contains child elements in addition to text content, the text in those child nodes appears as well

INSERTING A NODE VALUE EXAMPLE

Figure 6-16 Displaying the date and time element values

```
<xsl:template match="/">
   <html>
   <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
   </head>
   <body>

      <div id="datetime"><b>Last Updated: </b>
         <xsl:value-of select="portfolio/date" /> at
         <xsl:value-of select="portfolio/time" />
      </div>

      <h1>Hardin Financial</h1>
      <h2>Stock Information</h2>
   </body>
   </html>
</xsl:template>
```

PROCESSING SEVERAL ELEMENTS

- To process a batch of nodes, the syntax is:

  <xsl:for-each select="*expression*" />

  styles

  </xsl:for-each>

  where *expression* is an expression that defines the group of nodes to which the XSLT and literal result elements are applied

PROCESSING SEVERAL ELEMENTS

Figure 6-20 Setting a style for each occurrence of the name element

replace the three lines to generate the h3 heading with a for-each statement

```
<div id="datetime"><b>Last Updated: </b>
   <xsl:value-of select="portfolio/date" /> at
   <xsl:value-of select="portfolio/time" />
</div>

<h1>Hardin Financial</h1>
<h2>Stock Information</h2>

<xsl:for-each select="portfolio/stock">
   <h3>
   <xsl:value-of select="sName" />
   </h3>
</xsl:for-each>
```

WORKING WITH TEMPLATES

- To apply a template in the result document, use the XSLT element
  - <xsl:apply-templates select="*expression*" />

    where *expression* indicates the node template to be applied

CREATING TEMPLATE EXAMPLE

Creating and applying a template Figure 6-22

```
<xsl:template match="/">
   <html>
   <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
   </head>
   <body>

      <div id="datetime"><b>Last Updated: </b>
         <xsl:value-of select="portfolio/date" /> at
         <xsl:value-of select="portfolio/time" />
      </div>

      <h1>Hardin Financial</h1>
      <h2>Stock Information</h2>

      <xsl:apply-templates select="portfolio/stock/sName" />

   </body>
   </html>
</xsl:template>

<xsl:template match="sName">
   <h3>
      <xsl:value-of select="." />
   </h3>
</xsl:template>
```

applying the name template

name template

USING THE BUILT-IN TEMPLATES

- Each node has its own built-in template.
- The built-in template for element nodes matches the document root and all elements in the node tree
- The built-in template for text nodes matches all text nodes and causes their values to appear in the result document
- For example, you can add the stock template to the style sheet

CREATING THE STOCK TEMPLATE EXAMPLE

Figure 6-23 Creating the stock template

INSERTING ATTRIBUTE VALUES

```
<xsl:template match="sName">
   <h3><a href=<xsl:value-of select="../link" />
    <xsl:value-of select="." />
   (<xsl:value-of select="@symbol" />)</a></h3>
</xsl:template>
```

Why an error occurs with this type of code?

REMEMBER – XML RULES ARE STRICT!

INSERTING ATTRIBUTE VALUES

```
<xsl:template match="sName">
   <h3><a href="{../link}" />
    <xsl:value-of select="." />
   (<xsl:value-of select="@symbol" />)</a></h3>
</xsl:template>
```

Based on <elem attribute="{expresssion}">

Where elem=name of element; attribute=name of element's attribute; expression=Xpath for the value of the attribute

SORTING NODE SETS

- By default, nodes are processed in document order, by their appearance in the document
- To specify a different order, XSLT provides the <xsl:sort> element
- This element can be used with either the <xsl:apply-templates> or the <xsl:for-each> element

SORTING NODE SETS

- The <xsl:sort> comes with several attributes to control the sort
  - The select attribute determines the criteria under which the context node is sorted
  - The data-type attribute indicates the type of data
  - The order attribute indicates the direction of the sorting (ascending or descending)
  - The case-order – how to handle upper and lower cases

```
<xsl:apply-templates select="day".
        <xsl:sort data-type="number" order="descending" />
</xsl:apply-templates>
```

CREATING CONDITIONAL NODES

- XSLT supports two kinds of conditional elements:
  - <xsl:if>
  - <xsl:choose>
- To apply a format only if a particular condition is met , use the <xsl:if> element
- To test for multiple conditions and display different outcomes, use the <xsl:choose> element

CREATING CONDITIONAL NODES EXAMPLE

Inserting the five-day template Figure 6-37

```
<xsl:template match="today">
   <table>
   <tr>
      <th>Current</th>
      <th>Open</th>
      <th>High</th>
      <th>Low</th>
      <th>Volume</th>
   </tr>
   <tr>
      <td>
      <xsl:choose>
         <xsl:when test="@current &lt; @open">
            <img src="down.gif" />
         </xsl:when>
         <xsl:when test="@current > @open">
            <img src="up.gif" />
         </xsl:when>
         <xsl:otherwise>
            <img src="same.gif" />
         </xsl:otherwise>
      </xsl:choose>
      <xsl:value-of select="@current" />
      </td>
      <xsl:apply-templates select="@open" />
      <xsl:apply-templates select="@high" />
      <xsl:apply-templates select="@low" />
      <xsl:apply-templates select="@vol" />
   </tr>
   </table>
</xsl:template>
```

conditional style created for the @current attribute

@current attribute removed from the match list

```
<xsl:template match="@open|@high|@low|@vol|@date|@close">
   <td><xsl:value-of select="." /></td>
</xsl:template>
```

USING COMPARISON OPERATORS AND FUNCTIONS

Figure 6-36 Comparison operators

Operator	Description	Example
=	Tests whether two values are equal to each other	@symbol = "AA"
!=	Tests whether two values are unequal	@symbol != "AA"
<	Tests whether one value is less than another	day < 5
<=	Tests whether one value is less than or equal to another	day <= 5
>	Tests whether one value is greater than another	day > 1
>=	Tests whether one value is greater than or equal to another	day >= 1
and	Combines two expressions, returning a value of true only if both expressions are true	@symbol = "AA" and day > 1
or	Combines two expressions, returning a value of true if either expression is true	@symbol = "AA" or @symbol = "UCL"
not	Negates the value of the expression, changing true to false or false to true	not(day >= 1)

WORKING WITH PREDICATES

- Predicates are XPath expressions that test for a condition and create subsets of nodes that fulfill that condition
- The predicate can also indicate the position of the node in the node tree
- To select a specific position in the source document, use the position() function combined with any XPath expression

ADDING PREDICATES TO THE ROOT TEMPLATE EXAMPLE

Figure 6-39 Display stock information by categories

```
<xsl:template match="/">
   <html>
   <head>
      <title>Stock Information</title>
      <link href="stock.css" rel="stylesheet" type="text/css" />
   </head>
   <body>

      <div id="datetime"><b>Last Updated: </b>
         <xsl:value-of select="portfolio/date" /> at
         <xsl:value-of select="portfolio/time" />
      </div>

      <h1>Hardin Financial</h1>
      <h2>Stock Information</h2>

      <h2 class="category">Industrials</h2>
      <xsl:apply-templates select="portfolio/stock[category='Industrials']">
         <xsl:sort select="sName" />
      </xsl:apply-templates>

      <h2 class="category">Utilities</h2>
      <xsl:apply-templates select="portfolio/stock[category='Utilities']">
         <xsl:sort select="sName" />
      </xsl:apply-templates>

      <h2 class="category">Transportation</h2>
      <xsl:apply-templates select="portfolio/stock[category='Transportation']">
         <xsl:sort select="sName" />
      </xsl:apply-templates>

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

industrial stocks

utility stocks

transportation stocks