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
<?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
```

New Perspectives on XML, 2nd Edition
Tutorial 6
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

XML document

```xml
<?xml version="1.0" ?>
<!DOCTYPE portfolio SYSTEM "stock.dtd">
<portfolio>
  <author>Kevin Summers</author>
  <date>11/24/2008</date>
  <time>14:54</time>
  <stock>
    <sName>Eastman Kodak Company</sName>
  </stock>
</portfolio>
```

node tree

root node
- processing inst. node
  - element node
    - attribute node
      - text node

xml-stylesheet
type="text/xsl" href="stock.xsl"

portfolio

author
- Kevin Summers

date
- 11/24/2008

time
- 14:54

stock

symbol="EK"
- Eastman Kodak Company

New Perspectives on XML, 2nd Edition
Tutorial 6
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
• With a relative path, the location of the node is indicated relative to a specific node in the tree called the context node.

<table>
<thead>
<tr>
<th>Relative path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Refers to the context node</td>
</tr>
<tr>
<td>..</td>
<td>Refers to the parent of the context node</td>
</tr>
<tr>
<td>child</td>
<td>Refers to the child of the context node named child</td>
</tr>
<tr>
<td>child1/child2</td>
<td>Refers to the child2 node, a child of the child1 node beneath the context node</td>
</tr>
<tr>
<td>./sibling</td>
<td>Refers to a sibling of the context node named sibling</td>
</tr>
<tr>
<td>./descendant</td>
<td>Refers to a descendant of the context node named descendant</td>
</tr>
</tbody>
</table>
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.
• To create a template, the syntax is:

```xml
<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
• To create a root template, the syntax is:

```xml
<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

```xml
<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

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

```xml
<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>
```
• To process a batch of nodes, the syntax is:

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

*styles*

```xml
</xsl:for-each>
```

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

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

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

<h3>Hardin Financial</h3>
<h2>Stock Information</h2>

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

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

```
<html>
<head>
    <title>Stock Information</title>
    <link href="stock.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <div id="datetime">
        <b>Last Updated: </b> at
        <xs:when select="portfolio/time"/>
    </div>
    <h1>Hardin Financial</h1>
    <h2>Stock Information</h2>
    <xs:apply-templates select="portfolio/stock/sName"/>
</body>
</html>
```
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

```
<html>
  <head>
    <title>Hurdin Financial</title>
  </head>
  <body>
    <h2>Stock Information</h2>
    <xsl:apply-templates select="portfolio/stock" />
  </body>
</html>
```

```xml
<xsl:template match="stock">
  <div>
    <xsl:apply-templates select="sName" /> <p>
      <xsl:value-of select="description" /></p>
  </div>
</xsl:template>
```

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

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

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

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

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

<xsl:template match="@current">
  <td><xsl:value-of select="." /></td>
</xsl:template>
```

Conditional style created for the @current attribute.

Current attribute removed from the match list.
### Comparison operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Tests whether two values are equal to each other</td>
<td>@symbol = “AA”</td>
</tr>
<tr>
<td>!=</td>
<td>Tests whether two values are unequal</td>
<td>@symbol != “AA”</td>
</tr>
<tr>
<td>&lt;</td>
<td>Tests whether one value is less than another</td>
<td>day &lt; 5</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Tests whether one value is less than or equal to another</td>
<td>day &lt;= 5</td>
</tr>
<tr>
<td>&gt;</td>
<td>Tests whether one value is greater than another</td>
<td>day &gt; 1</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Tests whether one value is greater than or equal to another</td>
<td>day &gt;= 1</td>
</tr>
<tr>
<td>and</td>
<td>Combines two expressions, returning a value of true only if both expressions are true</td>
<td>@symbol = “AA” and day &gt; 1</td>
</tr>
<tr>
<td>or</td>
<td>Combines two expressions, returning a value of true if either expression is true</td>
<td>@symbol = “AA” or @symbol = “UCL”</td>
</tr>
<tr>
<td>not</td>
<td>Negates the value of the expression, changing true to false or false to true</td>
<td>not(day &gt;= 1)</td>
</tr>
</tbody>
</table>
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

```xml
<html>
<head>
<title>Stock Information</title>
</head>
<body>
  <div id="datetime">2023-04-01 12:00:00</div>
  <div>
    <h2>Hardin Financial</h2>
    <h2>Stock Information</h2>
    <h2 class="category">Industrials</h2>
    <xsl:for-each-select select="portfolio/stock[category='Industrials']">
      <xsl:apply-templates select="$/name"/>
    </xsl:for-each-select>
    <h2 class="category">Utilities</h2>
    <xsl:for-each-select select="portfolio/stock[category='Utilities']">
      <xsl:apply-templates select="$/name"/>
    </xsl:for-each-select>
    <h2 class="category">Transportation</h2>
    <xsl:for-each-select select="portfolio/stock[category='Transportation']">
      <xsl:apply-templates select="$/name"/>
    </xsl:for-each-select>
  </div>
</body>
</html>
```