COM506 Professional Web Services Development

Practical A5: Transforming XML with XSL Stylesheets

Aims

- To explain the purpose of XSL stylesheets
- To demonstrate how to connect an XSL stylesheet to an XML data source
- To demonstrate the transformation of XML data
- To illustrate sorting of XML data on pre-set criteria
- To demonstrate selection of a subset of XML data for presentation on various criteria
- To present dynamic switching of XSL stylesheets depending on user interaction

Contents

A5.1 XSL – Stylesheets for XML data

Extensible Stylesheet Language (**XSL**) is a document format that specifies how XML data should be interpreted and displayed. Using XSL, we take a document specified in one format (XML) and create a document specified in another format (e.g. HTML). The relationship between XML and XSL is a similar one to that between HTML and CSS.

We will examine the capabilities of XSL by applying it to the XML file *senators.xml* that contains details for each of the 100 members of the United States Senate.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="senators.xsl" ?>

<senators>

  <member>
    <member_full>Akaka (D-HI)</member_full>
    <last_name>Akaka</last_name>
    <first_name>Daniel K.</first_name>
    <party>D</party>
    <state>HI</state>
    <address>
       141 HART SENATE OFFICE BUILDING WASHINGTON DC 20510
    </address>
   <phone>(202) 224-6361</phone>
   <email>
      http://www.akaka.senate.gov/email-senator-akaka.cfm
   </email>
   <website>http://www.akaka.senate.gov</website>
   <class>Class I</class>
   <bioguide_id>A000069</bioguide_id>
  </member>

  <!--other <member> definitions -->

</senators>
```

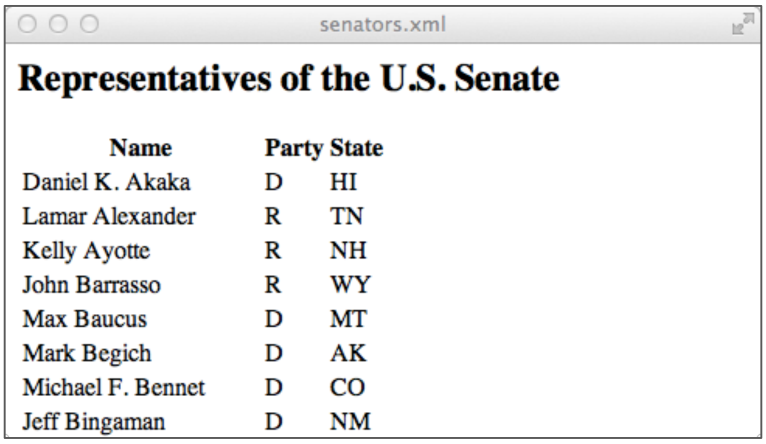
Note the **`<?xml-stylesheet ?>` processing instruction** that tells the browser to transform the XML data using the XSL stylesheet found at the location specified.

Most of the elements in the XML file are self-explanatory, but we will identify two in particular for further explanation:

- **`<party>`** represents the political party to which the Senator belongs. Potential values here are 'D' (Democrat), 'R' (Republican) or 'IN' (Independent).
- **`<state>`** is the state that the Senator represents. Each of the 50 states returns two Senators, with each state identified by a 2-letter code, e.g. 'TX' (Texas), 'FL' (Florida), 'HI' (Hawaii), etc.

In order to represent the information contained in the XML file as a web page, we specify an XSL file to be used to transform the XML data to HTML. Here, we specify *senators.xsl* as the transformation document.

Do it now!
Load *senators.xml* into a web browser. See how specifying the transformation document results in the XML data being displayed in as web page as illustrated in Figure A5.1.



Representatives of the U.S. Senate

Name	Party	State
Daniel K. Akaka	D	HI
Lamar Alexander	R	TN
Kelly Ayotte	R	NH
John Barrasso	R	WY
Max Baucus	D	MT
Mark Begich	D	AK
Michael F. Bennet	D	CO
Jeff Bingaman	D	NM

Figure A5.1. XSL Stylesheet

If we examine the source of the XSL transformation document, we can trace the process by which this web page is generated.

Firstly, note that the XSL file is itself a valid XML document, conforming to all of the rules of XML introduced in Practical A1. The file consists of a combination of HTML and XSL processing directives that describe how the XML data should be extracted and inserted into the HTML.

In this simple example, 4 XSL elements are used as described below.

- **`<xsl:template>`** is used to describe how to transform particular nodes from the source (XML) document to the result (HTML) document. The template is applied to all source elements that are specified in the `match` attribute. In this example, we specify that all elements should be included, by matching with the document root (`/`).
- **`<xsl:for-each>`** is used to iterate through all of the elements that match the term defined in the `search` attribute. Here, we specify that we want to carry out a particular formatting action to each **`<member>`** element in turn.
- **`<xsl:value-of>`** retrieves the data content of a node. Here, it is used 4 times to extract the first name, last name, party and state of a given **`<member>`** element.

- **`<xsl:text>`** is used to insert some text information into the output stream. Remember that the XSL file must be valid XML, so all content must be specified within some tag structure. Here, we use **`<xsl:text>`** to insert a space between the first name and last name values.

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

<xsl:template match="/">
  <html>
  <body>
  <h2>Representatives of the U.S. Senate</h2>
    <table>
      <tr>
        <th>Name</th>
        <th>Party</th>
        <th>State</th>
      </tr>
      <xsl:for-each select="senators/member">
            <tr>
              <td><xsl:value-of select="first_name"/>
                  <xsl:text> </xsl:text>
                  <xsl:value-of select="last_name"/></td>
              <td><xsl:value-of select="party"/></td>
              <td><xsl:value-of select="state"/></td>
            </tr>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Do it now!
Add a new column to the output table that contains the Senator's phone number.

Research Task!
Use online resources to discover how to present the Senator's website URL as a clickable hyperlink

Try it now!
Revisit *stock.xml* from Practical A2 and create *stock.xsl* to generate a structured display of all elements.

Hint: We can select the value of an element's attribute using **`<xsl:value-of>`** as above, except that we prefix the attribute name with '**`@`**'
e.g. **`<xsl:value-of select="@id" />`**

A5.2 Incorporating CSS Information

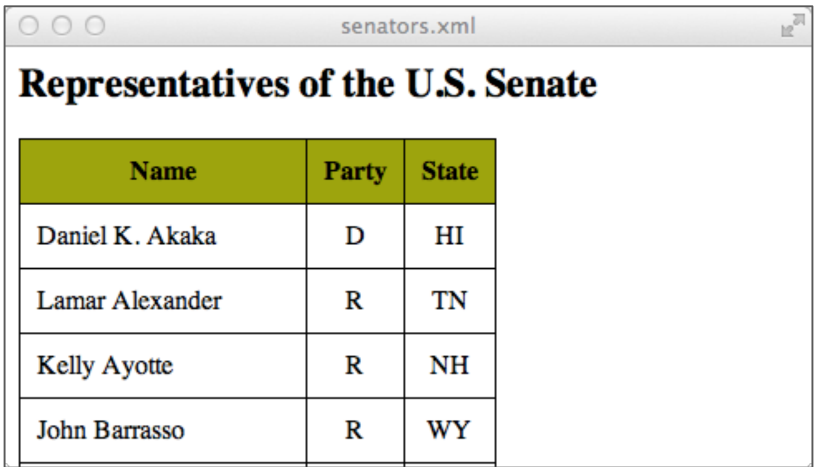
So far, we have described the output of the XSL transformation as an HTML document. However, the output document can contain additional JavaScript, CSS or any of the other elements that we can normally include within an HTML specification.

See, for example, the following extract from *senatorsWithCSS.xsl*, which uses embedded CSS to create a more visually pleasing layout for the output document.

```
<xsl:template match="/">
  <html>
  <head>
    <style type="text/css">
      table { border-collapse:collapse }
      th, td { border: solid 1px #000; padding:10px }
      .heading { background-color: #9a0 }
      .center { text-align:center }
    </style>
  </head>
  <body>
  <h2>Representatives of the U.S. Senate</h2>
    <table>
      <tr class="heading">
        <th>Name</th>
        <th>Party</th>
        <th>State</th>

        <!-- continued... -->
```

Here, we embed the CSS rules into the **`<head>`** of the template for the output document and refer to the rules in the corresponding tags, just as we would for standard HTML. We could equally include a link to an external CSS file.

Do it now!
Change the XSL link in *senators.xml* to point to the new XSL document. Re-load *senators.xml* into the browser and verify that you can see the effect of the CSS rules as illustrated in Figure A5.2 below.



Representatives of the U.S. Senate

Name	Party	State
Daniel K. Akaka	D	HI
Lamar Alexander	R	TN
Kelly Ayotte	R	NH
John Barrasso	R	WY

Figure A5.2. Applying CSS

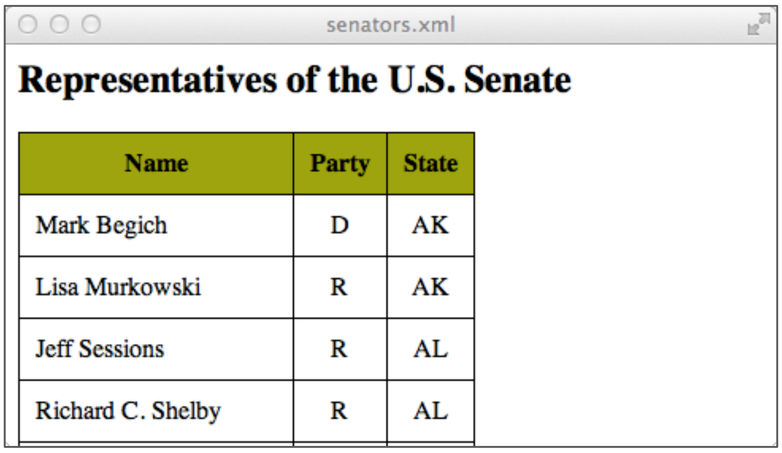
Try it now!
Modify *stock.xsl* so that it includes CSS information to create an attractive layout

A5.3 Specifying the Order of Presentation

So far our examples have simply extracted the data from the XML file in the order in which it is encountered. XSL also enables us to sort the data according to one or more of the data values in the XML. In *senatorsSorted.xsl*, we use the **`<xsl:sort>`** element to specify that the senators should be sorted by the state they represent.

```
<xsl:for-each select="senators/member">
   <xsl:sort select="state" />
     <tr>
       <td><xsl:value-of select="first_name"/>
           <xsl:text> </xsl:text>
           <xsl:value-of select="last_name"/></td>
       <td class="center">
           <xsl:value-of select="party"/></td>
       <td class="center"><xsl:value-of select="state"/></td>
     </tr>
</xsl:for-each>
```

Do it now!
Change the XSL link in *senators.xml* to point to the new XSL document. Re-load *senators.xml* into the browser and verify that you can see the effect of the CSS rules as illustrated in Figure A5.3 below.



Representatives of the U.S. Senate

Name	Party	State
Mark Begich	D	AK
Lisa Murkowski	R	AK
Jeff Sessions	R	AL
Richard C. Shelby	R	AL

Figure A5.3. Sorting the XML data

The **`<xsl-sort>`** element also has a number of optional attributes that can configure the sorted order. The two most useful are `order` which takes values of "**`ascending`**" (the default) or "**`descending`**", and **`data-type`** which takes values of "**`text`**" (the default" or "**`number`**".

For example, to sort a collection of **`<student>`** elements by descending order of **`<averageMark>`** we might have

```
<xsl:for-each select="student">
   <xsl:sort select="averageMark"
             order="descending" data-type="number" />

    <!-- other presentation instructions -->

</xsl:for-each>
```

Sometimes we may want to have a multi-value sort, where we sort initially on one element, and then on a second element where the first element values are equal. This is accomplished by simply using multiple **`<xsl:sort>`** elements in the order in which the sort values are to be applied.

For example, to sort our list of senators by **`<state>`** and then by **`<lastname>`**, we would use

```
<xsl:for-each select="senators/member">
   <xsl:sort select="state" />
   <xsl:sort select="lastname" />

    <!-- other presentation instructions -->

</xsl:for_each>
```

Do it now!
Experiment with the sort option on *senatorsSorted.xsl*

Try it now!
Revisit your *stock.xml* example and produce an XSL file that generates a list of stock items sorted in by price with the cheapest product first.

A5.4 Selecting a Subset of Elements

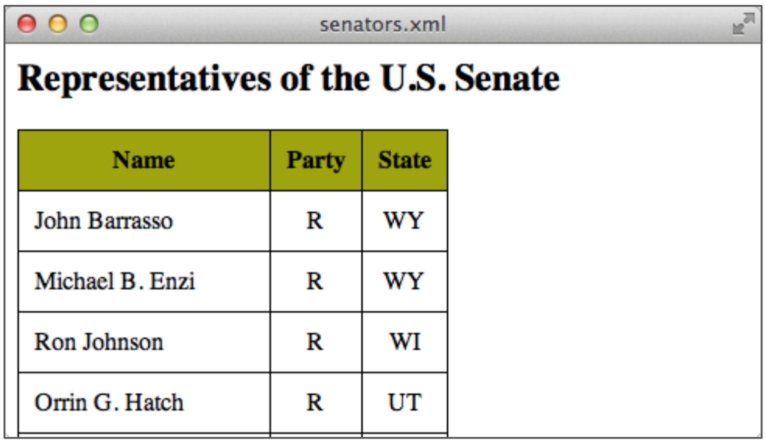
All of our examples so far have generated HTML pages that contain the entire contents of the XML database. Sometimes, however, we only require a certain subset of the data to be displayed. XSL provides three ways of achieving this.

A5.4.1 Filtering

Filtering involves providing additional information to the `select` attribute of the **`<xsl:for-each>`** element that describes the subset of data required.. For example, in *senatorsR.xsl*, we instruct the **`<xsl:for-each>`** element to only return those **`<member>`** elements where the **`<party>`** element is '**`R`**'.

```
   <xsl:for-each select="senators/member[party='R']">
     <xsl:sort select="state" order="descending"/>
       <tr>
         <td><xsl:value-of select="first_name"/>
             <xsl:text> </xsl:text>
             <xsl:value-of select="last_name"/></td>
         <td class="center"><xsl:value-of select="party"/></td>
         <td class="center"><xsl:value-of select="state"/></td>
       </tr>
   </xsl:for-each>
```

Do it now!
Change the XSL link in *senators.xml* to point to the new XSL document. Re-load *senators.xml* into the browser and verify that you can see the effect of the filter operator as illustrated in Figure A5.4 below.



Representatives of the U.S. Senate

Name	Party	State
John Barrasso	R	WY
Michael B. Enzi	R	WY
Ron Johnson	R	WI
Orrin G. Hatch	R	UT

Figure A5.4. Filtering XML data

Additional filter operations are also available including

Operator	Example	Meaning
`=`	`[party='R']`	Equal
`!=`	`[party!='R']`	Not equal
`&lt;`	`[price &lt; 10]`	Less than
`&gt;`	`[price &gt; 10]`	Greater than
`&lt;=`	`[price &lt;= 10]`	Less than or equal to
`&gt;=`	`[price &gt;= 10]`	Greater than or equal to
`or`	`[party='R' or state='TX']`	Or
`and`	`[party='R' and state='TX]`	And

Try it now!
Using *stock.xsl*, experiment with the filter operators and confirm their effect.

A5.4.2 Conditional Selection

Filtering allows us to select only a subset of the data set, but all elements are still subjected to the same formatting transformations. In contrast, the **`<xsl:if>`** element facilitates different formatting of different subsets of the data. For example, consider *senatorsFlorida.xsl*, which combines filtering with **`<xsl:if>`** to display only Republican Senators that represent the state of Florida.

```
<xsl:for-each select="senators/member[party='R']">
   <xsl:if test="state='FL'">
     <tr>
       <td><xsl:value-of select="first_name"/>
           <xsl:text> </xsl:text>
           <xsl:value-of select="last_name"/></td>
       <td class="center"><xsl:value-of select="party"/></td>
       <td class="center"><xsl:value-of select="state"/></td>
     </tr>
   </xsl:if>
</xsl:for-each>
```

Do it now!
Change the XSL link in *senators.xml* to point to the new XSL document. Re-load *senators.xml* into the browser and verify that you can see the effect of the **`<xsl:if>`** element. Try different conditions in the `test` attribute and verify their effect.

Multiple conditions can be expressed by simply using multiple **<xsl:if>** elements. For example to display Florida Senators on a red background and Texas Senators on a blue background, we might have the following code structure.

```
<xsl:for-each select="senators/member[party='R']">

   <xsl:if test="state='FL'">
     <tr class='red'>
         <!—display content on a red background -->
     </tr>
   </xsl:if>

   <xsl:if test="state='TX'">
     <tr class='blue'>
         <!—display content on a blue background -->
     </tr>
   </xsl:if>

</xsl:for-each>
```

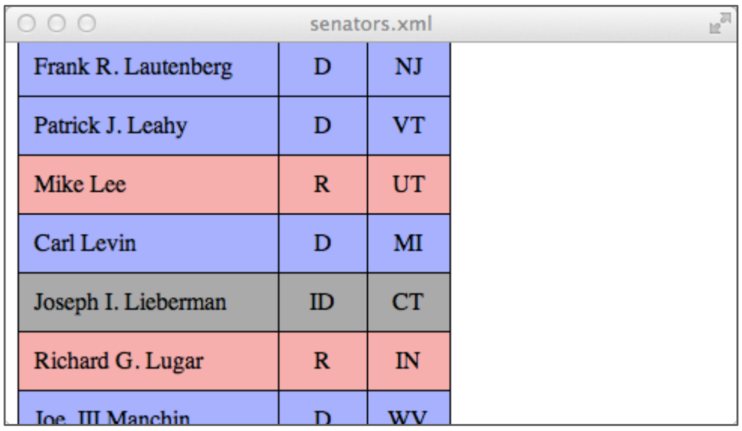
Try it now!
Modify *stock.xsl* so that only products with a high or low level of stock are highlighted. Where the stock level is <=10, the item details should be displayed in red text, but when the stock level is >= 100, the item details should be displayed in green text.

A5.4.3 Multi-value Conditional Tests

A significant limitation of the **`<xsl:if>`** element is the lack of a corresponding **`<xsl:else>`**. Where we require this functionality, we use the **`<xsl:choose>`** element as shown in *senatorsColourCoded.xsl*, which displays Republican Senators in red, Democratic Senators in blue, and any remaining Senators (usually independents) in a neutral grey.

```
<xsl:for-each select="senators/member">
    <xsl:choose>

      <xsl:when test="party='D'">
        <tr class="blue">
          <td><xsl:value-of select="first_name"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="last_name"/></td>
          <td class="center"><xsl:value-of select="party"/></td>
          <td class="center"><xsl:value-of select="state"/></td>
        </tr>
      </xsl:when>

      <xsl:when test="party='R'">
        <tr class="red">
          <td><xsl:value-of select="first_name"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="last_name"/></td>
          <td class="center"><xsl:value-of select="party"/></td>
          <td class="center"><xsl:value-of select="state"/></td>
        </tr>
      </xsl:when>

      <xsl:otherwise>
        <tr class="grey">
          <td><xsl:value-of select="first_name"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="last_name"/></td>
          <td class="center"><xsl:value-of select="party"/></td>
          <td class="center"><xsl:value-of select="state"/></td>
        </tr>
      </xsl:otherwise>

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

The **`<xsl:choose>`** element has a similar stricture to the Java/JavaScript/C/etc… **`switch`** construct. Individual cases are defined by **`<xsl:when>`** elements with **`<xsl:otherwise>`** used to “mop up” any remaining elements that do not fulfill any of the previous cases.



senators.xml

Frank R. Lautenberg	D	NJ
Patrick J. Leahy	D	VT
Mike Lee	R	UT
Carl Levin	D	MI
Joseph I. Lieberman	ID	CT
Richard G. Lugar	R	IN
Joe, III Manchin	D	WV

Figure A5.5. Multi-selection formatting

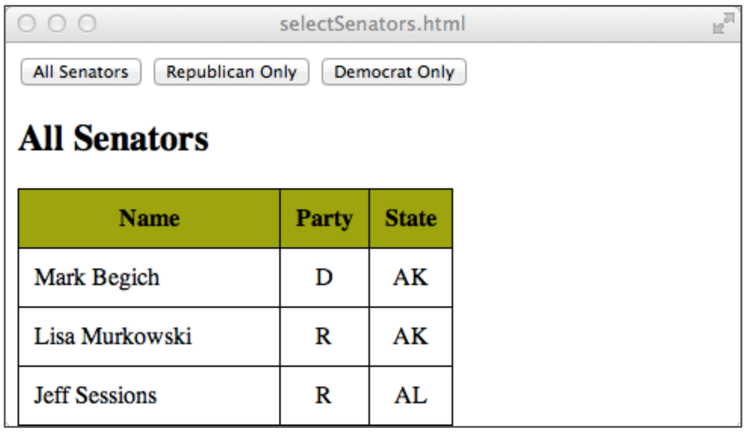
Do it now!
Change the XSL link in *senators.xml* to point to the new XSL document. Re-load *senators.xml* into the browser and verify that you can see the effect of the **`<xsl:choose>`** element. Try different conditions in the **`test`** attributes and verify their effect.

Try it now!
Modify *stock.xsl* so that products are sorted by descending **`<price>`** value. Items costing £10 or more should be displayed on a green background and products costing £1 or less should be displayed on a red background. All remaining products should be displayed on a yellow background.

A5.5 Dynamically Switching XSL stylesheets

So far in this practical, we have generated a large number of stylesheets to transform the same XML source in different ways. Sometimes, however, the appearance required depends on some interaction with the user. In such cases, we can generate a range of stylesheeets reflecting all of the possibilities and switch to the required view depending on the user's interaction with the application.

Examine *selectSenators.html*, which provides a set of buttons for the user to switch between a list of all Senators, a list of Democratic Senators and a list of Republican Senators.



All Senators

Name	Party	State
Mark Begich	D	AK
Lisa Murkowski	R	AK
Jeff Sessions	R	AL

Figure A5.6. Dynamically loading XSL stylesheets

When the user clicks one of the buttons, the following sequence is executed (refer to the code on the following page).

i. Call **`displayResult()`**, passing the identifier for the required stylesheet as a parameter
ii. Load the XML file containing the data to be transformed by the selected stylesheet into the variable `xml`
iii. Depending on the `flag` value passed as a parameter, load the required stylesheet into the variable **`xsl`**
iv. If the browser is Internet Explorer, generate the required output by the **`transformNode()`** method and send the result to the **`innerHTML`** attribute of the target page element
v. If the browser is non-IE, use it's **`XSLTProcessor`** object to perform the transformation and append the output to the empty target element,

Do it now!
Load *selectSenators.html* into a browser and verify its operation. Examine the source code and ensure that you can follow its execution path.

```
<html>
<head>
<script type="text/javascript">

function loadXMLDoc(url) {
  if (window.XMLHttpRequest) {
    xhttp=new XMLHttpRequest();
  } else {
      xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xhttp.open("GET",url,false);
  xhttp.send("");
  return xhttp.responseXML;
}

function displayResult(flag) {
  xml=loadXMLDoc("senators.xml");
  switch (flag) {
    case 'R' : xsl=loadXMLDoc("senatorsR.xsl"); break;
    case 'D' : xsl=loadXMLDoc("senatorsD.xsl"); break;
    default  : xsl=loadXMLDoc("senatorsA.xsl"); break;
  }

  if (window.ActiveXObject) { //IE
    ex=xml.transformNode(xsl);
    document.getElementById("senators").innerHTML=ex;
  } else if (document.implementation &&
             document.implementation.createDocument) { // others
      xsltProcessor=new XSLTProcessor();
      xsltProcessor.importStylesheet(xsl);
      resultDocument =
                 xsltProcessor.transformToFragment(xml,document);
      document.getElementById("senators").innerHTML="";
      document.getElementById("senators").
                                   appendChild(resultDocument);
  }
}
</script>
</head>

<body onload="displayResult('A')">

<div>
  <button onclick="displayResult('A')">All Senators</button>
  <button onclick="displayResult('R')">Republican Only</button>
  <button onclick="displayResult('D')">Democrat Only</button>
</div>

<div id="senators"></div>

</body>
</html>
```

Try it now!
Build a short tutorial of XSL stylesheets by dynamically switching between selected XSL sheets from this practical

Try it now!
Build a dynamic interface to the stock database that switches between a selection of views

A5.6 Further Information

- http://www.w3.org/Style/XSL/
  The Extensible Stylesheet Family – definition of the standard from W3C

- http://www.w3.org/TR/xslt
  Definition of XSL Transformations

- http://en.wikipedia.org/wiki/XSL
  Wikipedia definition of XSL

- https://www.w3schools.com/xml/xsl_intro.asp
  XSL tutorial from W3Schools

- http://nwalsh.com/docs/tutorials/xsl/xsl/slides.html
  XSL Concepts and Practical Use

- http://www.xfront.com/xsl.html
  Comprehensive XSL Tutorial (for download – contains Powerpoint presentations, demos and worked examples)