



Web Application Development: XSLT and XPath

Week 8

Content

- Introduction of XSLT
- XSLT Transformation
- XPath
- XSLT functions
- Examples

XSL

- In 1998, the W3C developed the Extensible Style sheet Language (XSL)
- XSL is composed of three parts:
 - XSL-FO (Extensible Style sheet Language – Formatting Objects) is used to implement page layout and design
 - XSLT (Extensible Style sheet Language Transformations) is used to transform XML content into another presentation format
 - XPath is used to locate information from an XML document and perform operations and calculations upon that content

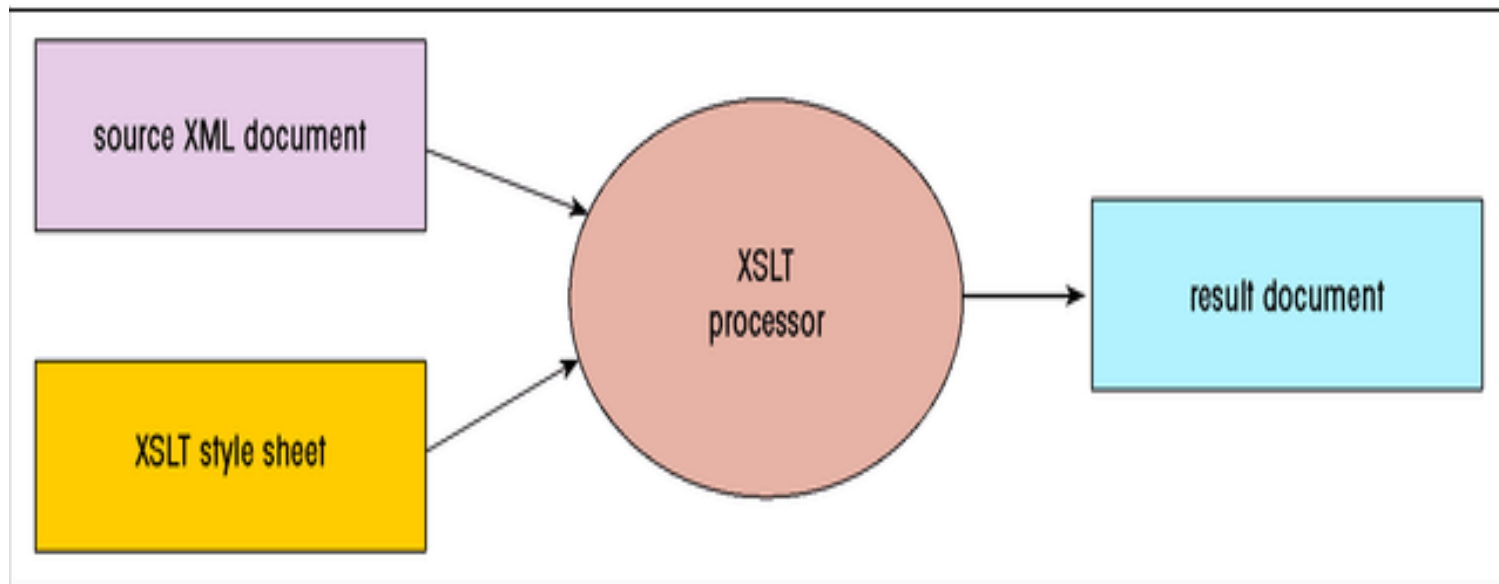
XSLT

- 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
- namespace for XSLT instructions:

<http://www.w3.org/1999/XSL/Transform>
- XSLT is a *declarative* language: the rules (templates) indicate *what* to do but do not have control over how each step of processing occurs
- Need to use *XPath* to perform transformation – to identify parts of XML document we wish to output in another way
- Use context node in travelling through tree
- XSLT's operation is data-oriented and not code-driven

Generating a Result Document

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



XSLT vs. XML DOM

- So far, we have processed XML by using the DOM API
- This has allowed us to extract information from an XML document, and then insert it into an HTML document
- This direct manipulation can be tedious
- With XPath, XSLT allows us to easily locate a fragment of an XML document at a time, and then transform it to part of another document in the format of HTML, XML or text

hotels.xml

```
<?xml version="1.0" standalone="no" ?>
<hotels>
  <hotel>
    <city>Paris</city>
    <name>La Splendide</name>
    <type>Budget</type>
    <price>100</price>
  </hotel>
  <hotel>
    <city>London</city>
    <name>The Rilton</name>
    <type>Luxury</type>
    <price>300</price>
  </hotel>
  <hotel>
    <city>Paris</city>
    <name>Marriott Rive Gauche</name>
    <type>Luxury</type>
    <price>350</price>
  </hotel>
</hotels>
```

```
<hotel>
  <city>New York</city>
  <name>The Imperial</name>
  <type>Standard</type>
  <price>150</price>
</hotel>
<hotel>
  <city>Paris</city>
  <name>Passy Eiffel</name>
  <type>Standard</type>
  <price>240</price>
</hotel>
</hotels>
```

Paris.xsl

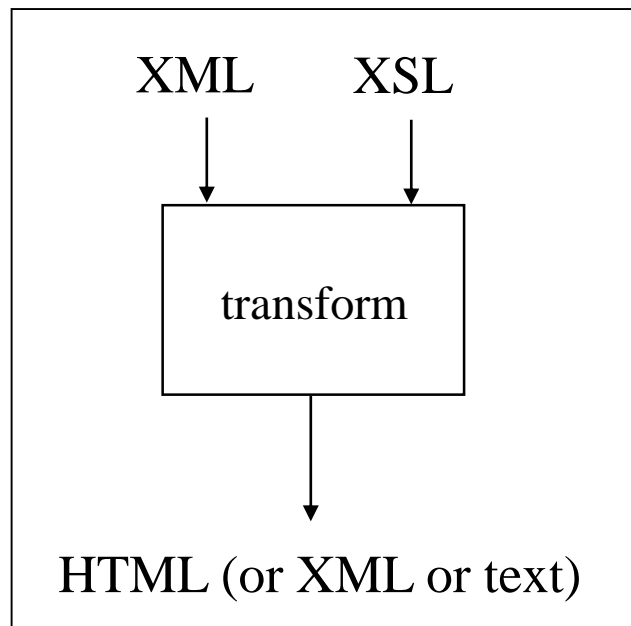
```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" version="4.0" />
  <xsl:template match="/">
    <table border="1">
      <xsl:for-each select="//hotel[city='Paris']">
        <tr><td><xsl:value-of select="name"/></td>
          <xsl:choose>
            <xsl:when test="type='Budget'">
              <td style="background:red"><xsl:value-of select="price"/></td>
            </xsl:when>
            <xsl:when test="type='Luxury'">
              <td style="background:lightblue"><xsl:value-of select="price"/></td>
            </xsl:when>
            <xsl:otherwise>
              <td><xsl:value-of select="price"/></td>
            </xsl:otherwise>
          </xsl:choose></tr>
      </xsl:for-each>
    </table>
    <br />Total: <xsl:value-of select="count(//hotel[city='Paris'])"/>
    <br />Average Price: <xsl:value-of select="sum(//hotel[city='Paris']/price) div count(//hotel[city='Paris'])"/>
  </xsl:template>
</xsl:stylesheet>
```


Output Document

```
<table border="1">
<tr>
<td>La Splendide</td><td style="background:red">100</td>
</tr>
<tr>
<td>Marriott Rive Gauche</td><td style="background:lightblue">350</td>
</tr>
<tr>
<td>Passy Eiffel</td><td>240</td>
</tr>
</table>
<br>Total: 3<br>Average Price: 230
```

XSLT Transformation

- Xalan transform tool: A free XSL processor, implemented in Java, from Apache (<http://www.apache.org/>)
- Put a stylesheet PI at the top of your XML document and use browsers



hotels.xml

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet type="text/xsl"
  href="Paris.xsl"?>
<hotels>
  <hotel>
    <city>Paris</city>
    <name>La Splendide</name>
    <type>Budget</type>
    <price>100</price>
  </hotel>
  <hotel>
    <city>London</city>
    <name>The Rilton</name>
    <type>Luxury</type>
    <price>300</price>
  </hotel>
```

```
<hotel>
  <city>Paris</city>
  <name>Marriott Rive Gauche</name>
  <type>Luxury</type>
  <price>350</price>
</hotel>
<hotel>
  <city>New York</city>
  <name>The Imperial</name>
  <type>Standard</type>
  <price>150</price>
</hotel>
<hotel>
  <city>Paris</city>
  <name>Passy Eiffel</name>
  <type>Standard</type>
  <price>240</price>
</hotel>
</hotels>
```

XSLT Transformation in Ajax

- Assume we transform XML to HTML, we can do it on the client, or on the server
- Client-side
 - ☐ here we essentially send the XML from the server to the client
 - ☐ pick it up in the responseXML property of an XHR object
 - ☐ transform it to HTML on the client, using XSLT
 - ☐ there are browser differences
 - ☐ place as required
- Server-side
 - ☐ transform XML to HTML on the server
 - ☐ send the HTML to the client
 - ☐ pick up as text in responseText property of XHR object
 - ☐ place as required

Performing a Transformation in IE

- load the XML file and XSL file into appropriate ActiveX document objects
- Apply a particular method that is available for the XML ActiveX document object, passing the XSL ActiveX document object as parameter; the output is the desired HTML (or XML or text)

Performing a Transformation in IE

IETransform.htm

```
<html>
<head>
  <title>IE Transformation</title>
  <script type="text/javascript"
    src="IETransform.js">
  </script>
</head>
<body onload="Transform()">
<span id="example"></span>
</body>
</html>
```

IETransform.js

```
function Transform()
{
  //Load XML and XSL
  var xml = loadXMLDoc("hotel.xml");
  var xsl = loadXMLDoc("Paris.xsl");
  //Transform
  var transform = xml.transformNode(xsl);
  //Write to screen
  var spantag = document.getElementById("example");
  spantag.innerHTML = transform;
}
```

Load XML

```
function loadXMLDoc(filename)
{
  if (window.ActiveXObject) // for IE
  {
    xhttp = new ActiveXObject("Msxml2.XMLHTTP");
  }
  else // for Firefox, google chrome
  {
    xhttp = new XMLHttpRequest();
  }
  xhttp.open("GET", filename, false);
  try {xhttp.responseType = "msxml-document"} catch(err) {} // Helping IE
  xhttp.send("");
  return xhttp.responseXML;
}
```

Firefox

- load the XML file and XSL file into XML DOM document objects
- Create an XSLT processor object
- Load the XSL DOM Document object in as the “program” for the XSLT processor object
- Send the XML DOM document object in as the input to the loaded XSLT processor, and get the transformed HTML as the output

Performing a Transformation in Firefox

FFTransform.htm

```
<html>
<head>
  <title>Firefox Transformation</title>
  <script type="text/javascript"
    src="FFTransform.js">
  </script>
</head>
<body onload="Transform()">
<span id="example"></span>
</body>
</html>
```

FFTransform.js

```
function Transform()
{
  var xsltProcessor = new XSLTProcessor();
  //Load XSL and XML
  var xsltProcessor = new XSLTProcessor();
  var xslStylesheet = loadXMLDoc(" Paris.xsl ");
  xsltProcessor.importStylesheet(xslStylesheet);
  var xml = loadXMLDoc("hotel.xml");

  //Transform
  var fragment = xsltProcessor.transformToFragment(xml,
document);
  document.getElementById("example").appendChild(fragment);
}
```

Server using PHP

essentially the same as client-side Firefox:

- load the XML file and XSL file into XML DOM document objects
- Create an XSLT processor object
- Load the XSL DOM Document object in as the “program” for the XSLT processor object
- Send the XML DOM document object in as the input to the loaded XSLT processor, and get the transformed HTML as the output

Performing a Transformation on Server

PHPTransform.htm

```
<html>
<head>
  <title>PHP Transformation</title>
  <script type="text/javascript"
    src="PHPTransform.js">
  </script>
</head>
<body onload="Transform()">
<span id="example"></span>
</body>
</html>
```

PHPTransform.js

```
var xHRObj = false;
if (window.XMLHttpRequest)
{ xHRObj = new XMLHttpRequest();}
else if (window.ActiveXObject)
{ xHRObj = new ActiveXObject("Microsoft.XMLHTTP");}

function Transform()
{ xHRObj.open("GET", "PHPTransform.php", true);
  xHRObj.onreadystatechange = getData;
  xHRObj.send(null);
}

function getData()
{ if ((xHRObj.readyState == 4) &&(xHRObj.status == 200))
  { var spantag =
    document.getElementById("example").innerHTML =
    xHRObj.responseText;
  }
}
```

Performing a Transformation on Server

```
<?php
    // load XML file into a DOM document
    $xmlDoc = new DOMDocument('1.0');
    $xmlDoc->formatOutput = true;
    $xmlDoc->load("hotels.xml");
    // load XSL file into a DOM document
    $xslDoc = new DomDocument('1.0');
    $xslDoc->load("Paris.xsl");
    // create a new XSLT processor object
    $proc = new XSLTProcessor;
    // load the XSL DOM object into the XSLT processor
    $proc->importStyleSheet($xslDoc);
    // transform the XML document using the configured XSLT processor
    $strXml= $proc->transformToXML($xmlDoc);
    // echo the transformed HTML back to the client
    echo ($strXml);
?>
```

PHPTransform.php

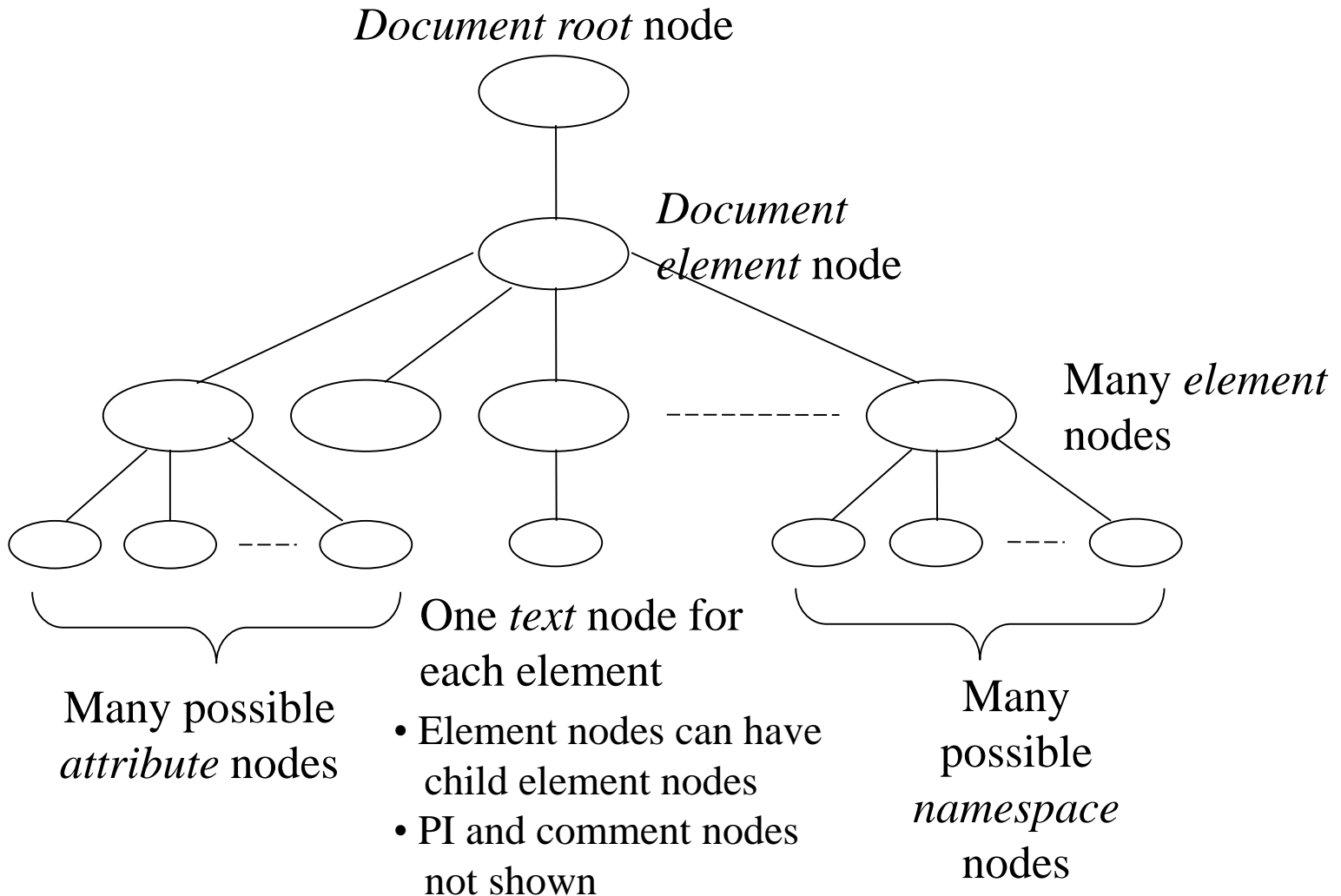
XPath

- a technology used for locating parts of an XML document for addressing or navigating to those parts
- be used with XSLT, XQuery
- essentially about moving through the *XPath tree* of a document, testing nodes to determine if they are relevant
- use a *location path* to locate a node or set of nodes in movement
- use *absolute location path* starting at root node or *relative location path* from context node
- W3C standard recommendation - <http://www.w3.org/TR/xpath>

XPath Data Model

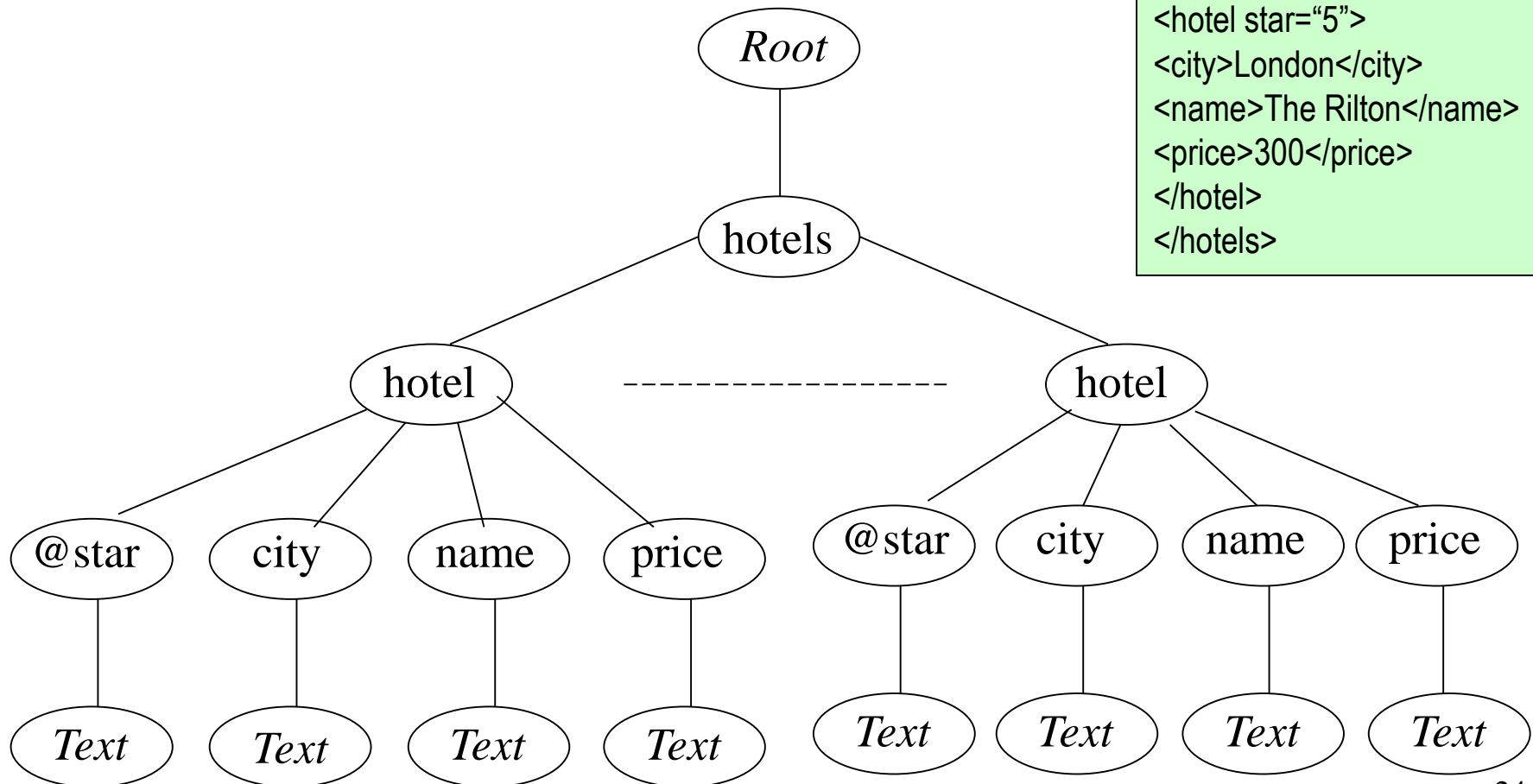
- A hierarchy, the *XPath tree*, is constructed by the parser from the contents of a document
- Tree is made from various nodes representing parts of the document
- There are 7 node types
 1. Document root
 2. Element
 3. Attribute
 4. Text
 5. Namespace
 6. Processing instruction
 7. Comment

XPath Data Model



XPath Tree for the Example

```
<hotels>  
<hotel star="3">  
  <city>Paris</city>  
  <name>La Splendide</name>  
  <price>100</price>  
</hotel>  
  
... ..  
<hotel star="5">  
  <city>London</city>  
  <name>The Rilton</name>  
  <price>300</price>  
</hotel>  
</hotels>
```



How Does XPath Operate?

- We know that XML content is in hierarchy
- Need to locate various parts of this hierarchy in order to use them
- Start at a *context node*, e.g., **hotels**
- From context node, proceed in one of several directions in the hierarchy – these directions are called *axes*
- Get to desired node through a number of *location steps* involving a *location path*
- Example of location path: **/hotels/hotel/name**
- Need to determine applicable type of node using a *node test*, e.g., Is the node a name element?

How Does XPath Operate?

- Location step has three aspects:
 - Axis
 - Node test
 - Zero or more predicates
- Syntax of *unabbreviated* location step:
<axis>::<nodetest><predicate>
- Example of unabbreviated location step
parent::hotel[attribute::star="5"]
- Selecting **hotel** elements from the **hotels** element:
/child::hotels/child::hotel or */hotels/hotel* (*abbreviated form*)
- Location steps separated by "/"

Axes

- child – context node's children, excluding attribute and namespace nodes
- parent – context node's parent (if node is not document element)
- descendant – context node's descendants, excluding attribute and namespace nodes
- ancestor – context node's parent, its parent's parent, etc.
- self – context node!
- descendant-or-self – context node + descendants
- ancestor-or-self – context node + ancestors
- attribute – attribute nodes of context (element) node
- namespace – namespace nodes of context (element) node
- following-sibling
- preceding-sibling

Node Tests

- If interested in selecting a particular element or attribute, just name that element or attribute.
- Wildcard (*) can be used to select all nodes of context node (depending upon axis used)
 - ☐ child::* selects all child *element* nodes of context node
 - ☐ attribute::* selects all *attribute* nodes of context node
- Other tests:
 - ☐ text() selects text nodes
 - ☐ comment() selects comment nodes
 - ☐ processing-instruction() selects PI nodes
 - ☐ node() selects all nodes, of all types

Node tests

```
<Staff>
  <Staff_member>
    <Name>Alan Colman</Name>
    <Email uni="true">acolman@it....</Email>
    <Phone>8771</Phone>
  </Staff_member>
  ... ..
```

```
<Staff_member>
  <Name>Yun Yang</Name>
  <Email uni="true">yyang@it....</Email>
  <Phone>8752</Phone>
</Staff_member>
</Staff>
```

Context node

Staff
 Staff_member
 Staff_member (of Alan)
 Staff_member
 Phone (of Yun)
 Staff_member (either)
 8752
 Phone (of Alan)
 Email (of Alan)
 Staff

Expression

child::Staff_member
 child::Staff_member
 child::*
 child::text()
 child::text()
 parent::*
 parent::*
 attribute::*
 attribute::uni
 descendant::Email

Returns ...

Staff_member elements
 Empty node set
Name, Email, Phone (of Alan)
 Empty node set
 8752
Staff
Phone (of Yun)
 Empty node set
uni (of Alan)
 Both **Email** elements

Predicates

- Predicates filter the returned *nodeset*
- Predicates are presented in square brackets, e.g.,
 - [attribute::star="5"]
 - [child::price < 150]
- Predicates consist of expressions built from XPath functions
- There are four main categories of XPath functions for use in predicates:
 1. Nodeset functions
 2. Boolean functions
 3. Number functions
 4. String functions

Abbreviated Location Steps

Axis	Abbreviation	Unabbreviated equivalent
Child	hotel	child::hotel
	*	child::*
	/price	child:: / child::price
	hotel[last()]	child::hotel[last()]
Self	.	self::node()
Parent	..	parent::node()
Attribute	@star	attribute::star
	@*	attribute::*
descendant-or-self		
	./city	self::node()/descendant-or-self ::node() /child::city
	hotels//name	child::hotels/descendant::name

Relative and Absolute XPath Expressions

- Relative XPath expressions start from the context node
 - `hotel[@star="5"]/price`
 - `./city`
- Absolute XPath expressions start from the root node "/", i.e., using "/" at start of expression denotes root node as context node, e.g.:
 - `/`
 - `/hotels//city[../hotel/@star="3"]` or `/hotels/hotel[@star="3"]/city`
 - `//hotel/name`

//

Selects nodes in the document from the current node that match the selection no matter where they are

Nodesets

- In */hotels/hotel/city*, *hotel* elements form a nodeset, and each *hotel* is a context node for any further location step */city*
- If there was a predicate for *hotel* , such as:

`hotel[position() = last()]`

this would return a nodeset of one node

- Some Nodeset functions
 - `count(node-set)`: Number of nodes in any nodeset
 - `count(//hotel)`
 - `last()`: Number of last node in context nodeset
 - `position()`: Number of context node in context nodeset

XPath Operators and Other Functions

- Arithmetic operators: + | - | * | div | mod
- Logical comparison operators: = | != | < | <= | > | >=
- Logical functions: not | true | false | and | or
- Some number functions
 - sum(node-set)
 - sum(/hotels/hotel/price) div count(/hotels/hotel)
 - ceiling(expr), floor(expr), round(expr)
- Some string functions
 - concat(string, string, ...)
 - contains(string1, string2), substring(string, offset, range)
 - substring-before(string1, string2), substring-after(string1, string2)
 - starts-with(string1, string2), string-length(string)

How does XSLT operate?

- Use context node in travelling through tree
- The processor works with tree *through ever-changing context*
- Templates are used to identify the fragments for transformation
- Several templates can be created in a stylesheet
- Processor could be asked to apply these templates to the context
→ each template's expression checked against the context so that one of them will match
- Matching template's instructions are executed for context
- The original XML document is unaffected by any transformation

XSLT Elements

1. Constructing stylesheets
2. Controlling output
3. Generating output
4. Generating markup
5. Iteration
6. Conditionals

1: <xsl:stylesheet>

- Document element of stylesheet is: <xsl:stylesheet> (or <xsl:transform>)
- <xsl:stylesheet> indicates:
 1. this is an XSLT stylesheet
 2. which XSLT namespace is used
 3. version of XSLT
- That is:

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

1: <xsl:template>

- All stylesheets have templates for their operation
- Stylesheets typically have a template matching the root node, i.e:

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

```
.....
```

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

- <xsl:template> has four attributes, we will look at only one of them – match
- Value of match attribute is an XPath expression
- Expression is compared with context, if matched, instructions associated with template are executed, i.e., template is executed
- A template contains two types of content:
 - 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

1: <xsl:apply-templates>

- Once matched with root-matching template, need to be able to execute other templates
- Executing these templates is something like function calls in procedural language
- Use <xsl:apply-templates> to execute templates in stylesheet
- Requests processor to match templates in stylesheet against current context node
- e.g.: suppose XML is:

<Subject_outline>

 <Extensions>

 Extensions will only ... compassionate grounds.

 <VeryImportant>Extensions must be ... the due date

 </VeryImportant>

 </Extensions>

</Subject_outline>

1: <xsl:apply-templates>

- Stylesheet is:

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

```
<xsl:template match="VeryImportant">  
  <EM><xsl:value-of select="."/></EM>  
</xsl:template>
```

- The stylesheet says that once the *Extensions* element is located, apply the templates defined for the context node (*Extensions*) or its child nodes
- However, this is not like a function call in that there can be more than one template that matches the context node – all matching templates are executed

1: <xsl:apply-templates>

- With select attribute

```
<xsl:template match="//Extensions">  
  <xsl:apply-templates select = "VeryImportant"/>  
</xsl:template>
```

```
<xsl:template match="VeryImportant">  
  <EM><xsl:value-of select="."/></EM>  
</xsl:template>
```

- The stylesheet says that once the *Extensions* element is located, apply the template defined for the child node *VeryImportant* of the context node (*Extensions*)

https://www.w3schools.com/xml/tryxslt.asp?xmlfile=catalog&xsltfile=catalog_apply

2: <xsl:output>

- <xsl:output> is child element of <xsl:stylesheet> used to say something about the type of output produced

- Some attributes of <xsl:output>:

method	Generate xml, html or text
version	Version of xml or html
encoding	Encoding of xml or html
omit-xml-declaration	yes/no values
standalone	yes/no values
doctype-public	Provides URI of DOCTYPE's identifier
doctype-system	Provides sys. identifier of DOCTYPE

- If method="xml" used, well-formed *fragment* must be generated

2: <xsl:output> Example

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

```
<xsl:stylesheet version="1.0"
```

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

```
<xsl:output
```

```
  method="xml"
```

```
  version="1.0"
```

```
  encoding="UTF-16"
```

```
  standalone="yes"/>
```

3: <xsl:value-of select>

- Templates will contain instructions for moving through the tree and generating output
- Use <xsl:value-of select="*XPath expression*"> to return value of node where this value is a string
- If node has sub-elements, values of all descendants and that node are output
- If a nodeset returned, value of first node in set is output

```
<xsl:template match="person">
  <p> <xsl:value-of select="given-name"/> <xsl:text> </xsl:text>
    <xsl:value-of select="family-name"/>
  </p>
</xsl:template>
```

<xsl:apply-templates select=“*pattern*”>

- The xsl:apply-templates element (without the select attribute) tells the XSL Processor to apply the template rules to all children (in document order)
- The xsl: apply-templates element can have a select attribute that tells the XSL Processor to process only the child element that matches “*pattern*”.

```
<xsl:template match="Member">  
  <xsl:apply-templates select="Name"/>  
  <xsl:apply-templates select="Phone[ @type='work']"/>  
</xsl:template>
```

4: Generating markup

- 5 XSLT elements exist for generating markup, particularly for XML:

<element name=...>	name of element
<attribute name=...>	name of attribute
<text>	content of a text node
<processing-instruction name=...>	target of PI
<comment>	content of comment

Creating an Element / Attribute

investment.xml

```
<?xml version="1.0"?>
<investment>
  <type>stock</type>
  <name>Microsoft</name>
  <price type="high">100</price>
  <price type="low">94</price>
</investment>
```

investment.xsl

newformat.xml

```
<?xml version="1.0"
  encoding="UTF-8"?>
<stock name="Microsoft"
  high="100" low="94"/>
```

investment.xsl

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="investment">
    <xsl:element name="{type}">
      <xsl:attribute name="name" ><xsl:value-of
        select="name"/></xsl:attribute>
      <xsl:for-each select="price">
        <xsl:attribute name="{@type}" ><xsl:value-of
          select="."/></xsl:attribute>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

5: <xsl:for-each>

- There is one instruction for iteration: **<xsl:for-each>**
- Recall that XSLT is declarative, the **<xsl:for-each>** instruction does not work in the same way as a procedural for loop statement
- It operates on a nodeset by simply going through each node of the set applying the template's instructions

6: <xsl:if test>

- **<xsl:if test>** is like if statement in procedural language, however has no else/else if statements!!
- test attribute's value is the XPath expression being checked for true/false
- Simply, if the expression evaluates to true, then the statements found between the **<xsl:if test>**'s start and end tags are executed

6: <xsl:choose>

- Similar to switch in C/C++ or Select Case in VB/VBScript
- **<xsl:choose> ... </xsl:choose>** is used to enclose all elements used in conditional
- **<xsl:when test>** is just like **<xsl:if test>**
- After a **<xsl:when test>** is found to be true, the instructions within the **<xsl:when> ... </xsl:when>** tags are executed, and processor then leaves **<xsl:choose>**
- Optional **<xsl:otherwise>** provides default instructions

FitnessCenter.xml

```
<?xml version="1.0"?>
<FitnessCenter>
  <Member id="1" level="platinum">
    <Name>Jeff</Name>
    <Phone type="home">555-1234</Phone>
    <Phone type="work">555-4321</Phone>
    <FavoriteColor>lightgrey</FavoriteColor>
  </Member>
  <Member id="2" level="gold">
    <Name>David</Name>
    <Phone type="home">383-1234</Phone>
    <Phone type="work">383-4321</Phone>
    <FavoriteColor>lightblue</FavoriteColor>
  </Member>
  <Member id="3" level="platinum">
    <Name>Roger</Name>
    <Phone type="home">888-1234</Phone>
    <Phone type="work">888-4321</Phone>
    <FavoriteColor>lightyellow</FavoriteColor>
  </Member>
</FitnessCenter>
```

PhoneList.xsl

```
<HTML>
  <HEAD>
    <TITLE>Phone List</TITLE>
  </HEAD>
  <BODY bgcolor="{/FitnessCenter/Member/FavoriteColor}">
    Member's phone numbers:
    <TABLE border="1" width="25%">
      <TR><TH>Name</TH><TH>Type</TH><TH>Number</TH></TR>
      <xsl:for-each select="/FitnessCenter/Member/Phone">
        <TR>
          <TD><xsl:value-of select="../Name"/></TD>
          <TD><xsl:value-of select="@type"/></TD>
          <TD><xsl:value-of select="."/></TD>
        </TR>
      </xsl:for-each>
    </TABLE>
  </BODY>
</HTML>
```

Sorting <xsl:sort>

- <xsl:sort> sorts the elements that you extract from the XML document
- This element can be used with either the <xsl:apply-templates> or the <xsl:for-each> element

```
<xsl:for-each select="/FitnessCenter/Member">  
  <xsl:sort select="Name" order="ascending"/>  
  <xsl:value-of select="Name"/>  
  <BR/>  
</xsl:for-each>
```

"For each Member, sort the Name elements"

Output:
David
Jeff
Roger

<xsl:variable>

- This element creates a variable to hold a value or node
- A variable is “write once, read many”
- A variable has a scope that starts where it is defined and extends to the end of the XSL element that it is nested within

Member's Home Phone Numbers:

```
<TABLE border="1" width="25%">
<TR><TH>Name</TH><TH><TH>Number</TH></TR>
  <xsl:for-each select="/FitnessCenter/Member">
    <xsl:variable name="name" select="Name"/>
    <xsl:for-each select="Phone[@type='home']">
      <TR>
        <TD><xsl:value-of select="$name"/></TD>
        <TD><xsl:value-of select="."/></TD>
      </TR>
    </xsl:for-each>
  </xsl:for-each>
</TABLE>
```

Coloring alternate rows

Member Names:

```
<TABLE border="1" width="25%">
  <xsl:for-each select="/FitnessCenter/Member">
    <TR>
      <xsl:if test="position() mod 2 = 0">
        <xsl:attribute name="bgcolor">yellow</xsl:attribute>
      </xsl:if>
      <TD><xsl:value-of select="Name"/></TD>
    </TR>
  </xsl:for-each>
</TABLE>
```

For each even row of the table, the TR value will be:

```
<TR bgcolor="yellow">
```

<https://mercury.swin.edu.au/wlai/lec8/FitnessCenter.xml>

count() and sum() functions

Number of members =

```
<xsl:value-of select="count(//Member)"/>
```

Membership Fee Revenue:

```
<xsl:value-of select="sum(//MembershipFee)"/>
```


When to use Curly Braces?

- “When I assign an attribute a value, when do I use curly braces and when do I not use them?”

Use curly braces for these attributes:

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

Example: ``

- the name attribute of `xsl:attribute`

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

- the name attribute of `xsl:pi`

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

- the name attribute of `xsl:element`

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

- the optional attributes of `xsl:sort`:

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

<xsl:call-template> with parameters

```
<xsl:template match="/">
  <HTML>
    <HEAD><TITLE>Fitness Center</TITLE></HEAD>
    <BODY>
      <xsl:call-template name="displayNameWithFont">
        <xsl:with-param name="fontFace" select="'Impact'"/>
        <xsl:with-param name="name"
          select="/FitnessCenter/Member[1]/Name"/>
      </xsl:call-template>
      <BR/>
    </BODY>
  </HTML>
</xsl:template>
<xsl:template name="displayNameWithFont">
  <xsl:param name="fontFace" select="'Braggadocio'"/> <!-- default font -->
  <xsl:param name="name"/>
  <FONT face="{ $fontFace }">
    <xsl:value-of select="$name"/>
  </FONT>
</xsl:template>
```

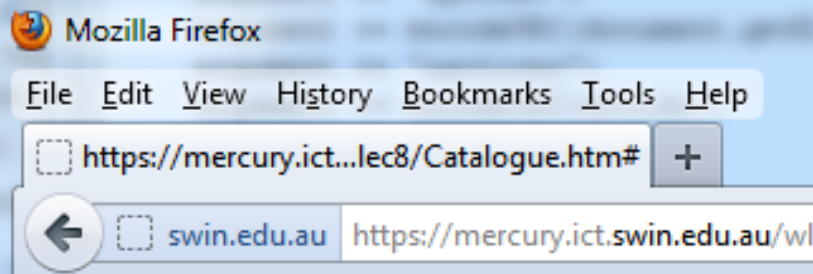
Returning a Value

```
<xsl:template match="/">
  <HTML>
    <HEAD><TITLE>Fitness Center</TITLE></HEAD>
    <BODY>
      16 / 2 =
      <xsl:variable name="result">
        <xsl:call-template name="NumDiv2">
          <xsl:with-param name="N" select="16"/>
        </xsl:call-template>
      </xsl:variable>
      <xsl:value-of select="$result"/>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template name="NumDiv2">
  <xsl:param name="N"/>
  <xsl:value-of select="$N div 2"/>
</xsl:template>
```



Shopping Cart : from Lecture 7

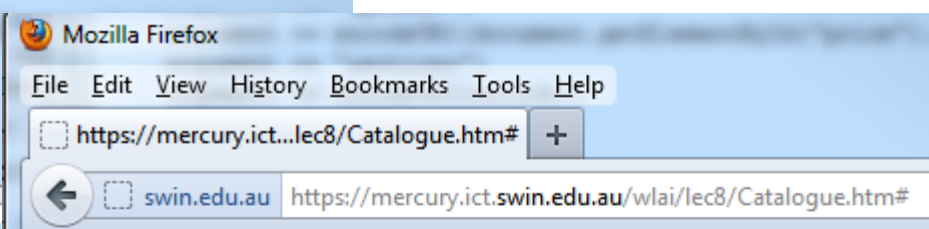
- Our previous work on this has all used direct manipulation of DOM objects, using the DOM API.
- In this Lecture we will explore the alternative techniques offered by XPath and XSLT
- Some changes
 - XML representation of the shopping cart received at client is now processed on the client using XPath and XSLT
 - Display in the client is now subject to CSS stylesheet; XSLT adds class tags so that formatting can be made using CSS
 - Uses POST rather than GET for main Ajax request



Book:Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: \$39.99

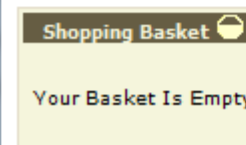
[Add to Shopping Cart](#)

Shopping Basket 		
Item	Qty	Price
Beginning ASP.NET with CSharp	2	\$79.98 
Total:		\$ 79.98



Book:Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: \$39.99

[Add to Shopping Cart](#)



Files

- Catalogue.htm
 - Home page for the system. Displays the user interface, and has place set up for display of shopping cart data
- ShoppingCart.js and xhr.js
 - JavaScript functions to control the client-server interaction
- Cartdisplay.php
 - PHP code to manage the shopping cart on the server
- Cart.xsl
 - XSLT stylesheet to extract and transform the cart data sent from the server
- Cart.css
 - CSS stylesheet to provide style information for display on the client
- begaspnet.jpg, button.jpg
 - Image files

Catalogue.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
  <link id="Link1" rel="stylesheet" href="Cart.css" type="text/css" />
  <script type="text/javascript" src="xhr.js"></script>
  <script type="text/javascript" src="ShoppingCart.js"></script>
</head>
<body>
  <form id="form1" method="post" action="cartdisplay.php"><br/>
    <br /><br />
    <b>Book:</b><span id="book">Beginning ASP.NET with CSharp</span><br />
    <b>Authors: </b><span id="authors"> Hart, Kauffman, Sussman, Ullman</span>
    <br /><b>ISBN: </b><span id="ISBN">0764588508</span>
    <br /><b>Price: </b><span id="price">$39.99</span>
    <br /><br />
    <a href="#" onclick="AddRemoveItem('Add');">Add to Shopping Cart</a>
    <br /><br />
    <span id="cart" ></span>
  </form>
</body>
</html>
```

ShoppingCart.js

```
var xHRObj = false;
xHRObj = createRequest(); // from xhr.js
function getBody(action)
{
    var argument = "book="; argument += encodeURIComponent(document.getElementById("book").innerHTML);
    argument += "&ISBN="; argument += encodeURIComponent(document.getElementById("ISBN").innerHTML);
    argument += "&authors="; argument += encodeURIComponent(document.getElementById("authors").innerHTML);
    argument += "&price="; argument += encodeURIComponent(document.getElementById("price").innerHTML);
    argument += "&action="; argument += encodeURIComponent(action);
    return argument;
}
function loadXMLDoc(filename) .....// see Slide 15

function getData()
{
    if ((xHRObj.readyState == 4) &&(xHRObj.status == 200))
    {
        if (window.ActiveXObject) // IE
        {
            //Load XML and XSL, then transform
            var xml = xHRObj.responseXML;
            var xsl = loadXMLDoc("Cart.xsl");
            var transform = xml.transformNode(xsl);
            var spanb = document.getElementById("cart");
            spanb.innerHTML = transform;
        }
    }
}
```


ShoppingCart.js (Cont'd)

else // Firefox

```
{ var xsltProcessor = new XSLTProcessor();  
  //Load XSL  
  var xsltProcessor = new XSLTProcessor();  
  var xslStylesheet = loadXMLDoc("Cart.xsl");  
  xsltProcessor.importStylesheet(xslStylesheet);
```

//Load XML

```
xmlDoc = xHRObject.responseXML;
```

//Transform

```
var fragment = xsltProcessor.transformToFragment(xmlDoc, document);  
document.getElementById("cart").innerHTML = new XMLSerializer().serializeToString(fragment);  
}
```

```
}
```

function AddRemoveItem(action)

```
{  
  var bodyofform = getBody(action);  
  xHRObject.open("POST", "cartdisplay.php", true);  
  xHRObject.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
  xHRObject.onreadystatechange = getData;  
  xHRObject.send(bodyofform);
```

Cartdisplay.php : as in Lecture 7

```
<?php
//session_register("Cart");
session_start();
header('Content-Type: text/xml');
?>
<?php // server side processing
$newitem = $_POST["book"]; // book name
$action = $_POST["action"]; // add or delete?
if ($_SESSION["Cart"] != "") // "cart" already exists
{
    $myCart = $_SESSION["Cart"];
    if ($action == "Add") // processing "Add" request
    {
        if ($myCart[$newitem] != "") // the cart already has
            // this book in it
        {
            $value = $myCart[$newitem] + 1;
            $myCart[$newitem] = $value; // add 1 to # copies
        }
        else // this is the first copy of this book to be
            // added (there may be other books)
        {
            $myCart[$newitem] = "1";
        }
    } // end case where we are processing an "Add"
```

```
    } // end case where we are processing an "Add"
    else // we are processing a "Remove" request
    {
        $myCart = ""; // nb – clearing the whole cart;
                        // ok if there is just one book in store
    }
} // end case where cart already exists with this item
else // the "cart" is NOT present – ie, we have no
    // books ordered
{
    $myCart[$newitem] = "1"; // we are sure that we
                            // are processing an "Add" in this case, as
                            // user had no chance to select "Delete"
} // end modification of cart
// copy the modified cart to the session variable,
// then convert to serialized XML and send to client
$_SESSION["Cart"] = $myCart;
ECHO (toXml($myCart));
```

Cartdisplay.php (Cont'd)

```
function toXml($aCart) // return the contents of the shopping cart in XML format
{
    $doc = new DomDocument('1.0'); $cart = $doc->appendChild($doc->createElement('cart')); $total = 0;
    foreach ($aCart as $bkTitle => $qty)
    {
        $book = $cart->appendChild($doc->createElement('book'));
        $title = $book->appendChild($doc->createElement('Title'));
        $title->appendChild($doc->createTextNode($bkTitle));
        $authors = $book->appendChild($doc->createElement('Authors'));
        $authors->appendChild($doc->createTextNode($_POST['authors']));
        $isbn = $book->appendChild($doc->createElement('ISBN'));
        $isbn->appendChild($doc->createTextNode($_POST['ISBN']));
        $price = str_replace("$", "", $_POST['price']);
        $priceNode = $book->appendChild($doc->createElement('Price'));
        $priceNode->appendChild($doc->createTextNode($price));
        $quantity = $book->appendChild($doc->createElement('Quantity'));
        $quantity->appendChild($doc->createTextNode($qty));
        $total = $total + $price * $qty;
    }
    $totalNode = $cart->appendChild($doc->createElement('Total'));
    $totalNode->appendChild($doc->createTextNode($total)); $strXml = $doc->saveXML();
    return $strXml;
}
?>
```

```
<?xml version="1.0"?>
<cart>
  <book>
    <Title>Book Title</Title>
    <Author> xxxx </Author>
    <ISBN>0764588508</ISBN>
    <Price> 39.99 </Price>
    <Quantity> 2 </Quantity>
  </book>
  <Total>79.98</Total>
</cart>
```

What the XSL has to do

- The purpose of the XSL transform is to take the XML data in the shopping cart, that has been sent to the browser, and transform it into suitable HTML for display.
- The cart has data for several books. The data for each book is to be displayed in a similar way.
- There is also the need to display the summary data for the whole cart.
- The HTML should be given styling tags, so that it can be formatted according to a CSS linked to the displayed page.
- Set up a table to hold the shopping cart data
- Print the table heading
- If the cart is not empty, include a sub-heading, identifying the elements in the table
- For each book in the cart, include the details; include a button to press to delete that book from the cart
- If the cart is not empty, include a total row; else state that the cart is empty

Cart.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" version="4.0" doctype-public="-//W3C//DTD HTML 4.01//EN"
    doctype-system="http://www.w3.org/TR/html4/strict.dtd"/>
  <xsl:template match="/">
    <table id="shoppingcart"> <xsl:call-template name="DisplayCart"></xsl:call-template></table>
  </xsl:template>
  <xsl:template name="DisplayCart">
    <tr class="head"> <td colspan="4" align="center">Shopping Basket
    </img></td></tr>
    <xsl:if test="number(//book/Quantity)>0">
      <tr><td class="border">Item</td><td class="border">Qty</td><td class="border">Price</td>
        <td></td></tr>
    </xsl:if>
    <xsl:for-each select="//book">
      <tr><td class="border2" width="75px"><xsl:value-of select="Title"/></td>
        <td class="border2" align="center"><xsl:value-of select="Quantity"/></td>
        <td class="border2">$<xsl:value-of select="Price * Quantity"/></td>
        <td class="border2"><a href="javascript:AddRemoveItem('Remove');">
          <img src='button.jpg'/></a></td>
        </tr>
    </xsl:for-each>
```

```
<?xml version="1.0"?>
<cart>
  <book>
    <Title>Book Title</Title>
    <Author> xxxx </Author>
    <ISBN>0764588508</ISBN>
    <Price> 39.99 </Price>
    <Quantity> 2 </Quantity>
  </book>
  <Total>79.98</Total>
</cart>
```

Cart.xsl

```
<tr><td colspan='4' class="border2"></td></tr>
<xsl:choose>
  <xsl:when test="sum(//book/Quantity)>0">
    <tr><td colspan="2" class="border2">Total:</td>
      <td class="border">${xsl:value-of select="(//Total)"/}</td>
      <td class="border2"> </td>
    </tr>
  </xsl:when>
  <xsl:otherwise>
    <tr><td colspan = "4" class="border2">Your Basket Is Empty</td></tr>
  </xsl:otherwise>
</xsl:choose>
<tr><td colspan="4" class="border2"> </td></tr>
</xsl:template>
</xsl:stylesheet>
```

Cart.css

```
table { background-color: beige; border: 1px solid #e4ddca;}
```

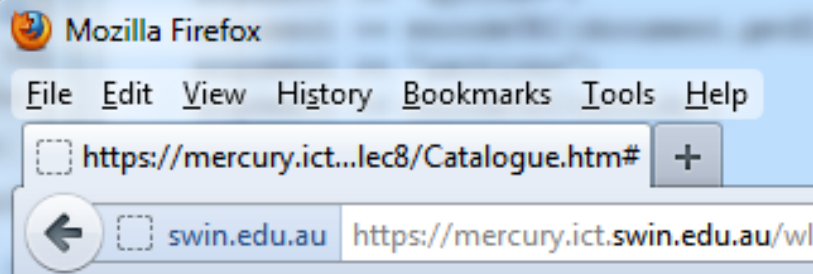
```
A IMG { border:0px;}
```

```
tr, td { font-family: Verdana; height:15px; font-size:75%;  
        margin:2px 0px 0px 0px; padding:0px 0px 0px 5px;}
```

```
.border { font-weight: bold; text-align:center;}
```



```
.border2 { background: beige;}
```

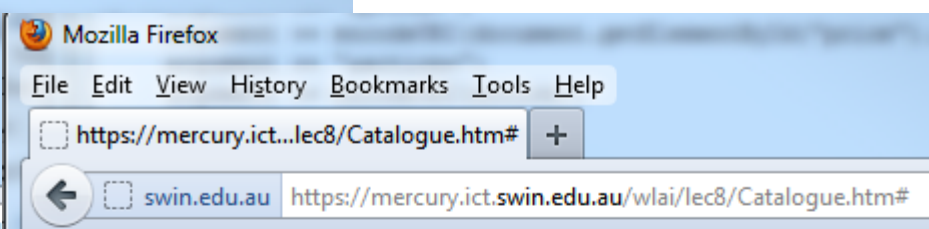
```
.head { font-family: Verdana; height:15px; vertical-align:middle;  
        background:#665D43; color: beige; font-weight: bold; }
```



Book:Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: \$39.99

[Add to Shopping Cart](#)

Shopping Basket 		
Item	Qty	Price
Beginning ASP.NET with CSharp	2	\$79.98 
Total:		\$ 79.98



Book:Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: \$39.99

[Add to Shopping Cart](#)



<http://mercury.swin.edu.au/wlai/Lab7/Catalogue1PHP.htm>

<https://mercury.swin.edu.au/wlai/Lec8Examples/Catalogue.htm#>

https://www.w3schools.com/xml/xsl_client.asp

<https://www.php.net/manual/en/xsltprocessor.transformtoxml.php>