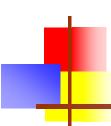


XSLT





- XSLT stands for Extensible Stylesheet Language Transformations
- XSLT is used to transform XML documents into other kinds of documents--usually, but not necessarily, XHTML
- XSLT uses two input files:
 - The XML document containing the actual data
 - The XSL document containing both the "framework" in which to insert the data, *and* XSLT commands to do so



Very simple example

File data.xml:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="render.xsl"?>
<message>Howdy!</message>
```

File render.xsl:

The .xsl file

- An XSLT document has the .xsl extension
- The XSLT document begins with:

Contains one or more templates, such as:

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

And ends with:

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



Finding the message text

- The template <xsl:template match="/"> says to select the entire file
 - You can think of this as selecting the root node of the XML tree
- Inside this template,
 - <xsl:value-of select="message"/> selects the message child
 - Alternative Xpath expressions that would *also* work:
 - ./message
 - /message/text() (text() is an XPath function)
 - ./message/text()

Putting it together

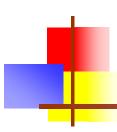
The XSL was:

- The <xsl:template match="/"> chooses the root
- The <body> <html><body> <html><body> is written to the output file
- The contents of message is written to the output file
- The </h1> </body></html> is written to the output file
- The resultant file looks like:

```
<html><body>
<h1>Howdy!</h1>
</body></html>
```

How XSLT works

- The XML text document is read in and stored as a tree of nodes
- The <xsl:template match="/"> template is used to select the entire tree
- The rules within the template are applied to the matching nodes, thus changing the structure of the XML tree
 - If there are other templates, they must be *called* explicitly from the main template
- Unmatched parts of the XML tree are not changed
- After the template is applied, the tree is written out again as a text document



Where XSLT can be used

- With an appropriate program, such as Xerces, XSLT can be used to read and write files
- A server can use XSLT to change XML files into HTML files before sending them to the client
- A modern browser can use XSLT to change XML into HTML on the client side
 - This is what we will mostly be doing in this class
- Most users seldom update their browsers
 - If you want "everyone" to see your pages, do any XSL processing on the server side
 - Otherwise, think about what best fits your situation



- Internet Explorer 6 best supports XML
- Netscape 6 supports some of XML
- Internet Explorer 5.x supports an *obsolete* version of XML
 - IE5 is not good enough for this course
 - If you *must* use IE5, the initial PI is different (you can look it up if you ever need it)

xsl:value-of

- <xsl:value-of select="XPath expression"/> selects the contents of an element and adds it to the output stream
 - The select attribute is required
 - Notice that xsl:value-of is not a container, hence it needs to end with a slash
- Example (from an earlier slide):

```
<h1> <xsl:value-of select="message"/> </h1>
```

xsl:for-each

- xsl:for-each is a kind of loop statement
- The syntax is

```
<xsl:for-each select="XPath expression">
    Text to insert and rules to apply
</xsl:for-each>
```

Example: to select every book (//book) and make an unordered list () of their titles (title), use:

```
<xsl:for-each select="//book"><xsl:value-of select="title"/> </xsl:for-each>
```

Filtering output

You can filter (restrict) output by adding a criterion to the select attribute's value:

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

This will select book titles by Terry Pratchett

Filter details

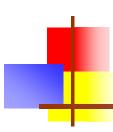
Here is the filter we just used:

```
<xsl:value-of
select="title[../author='Terry Pratchett']"/>
```

- author is a sibling of title, so from title we have to go up to its parent, book, then back down to author
- This filter requires a quote within a quote, so we need both single quotes and double quotes
- Legal filter operators are:

```
= != < &gt;
```

Numbers should be quoted, but apparently don't have to be



But it doesn't work right!

Here's what we did:

- This will output and for every book, so we will get empty bullets for authors other than Terry Pratchett
- There is no obvious way to solve this with just xsl:value-of



- xsl:if allows us to include content if a given condition (in the test attribute) is true
- Example:

```
<xsl:for-each select="//book">
    <xsl:if test="author='Terry Pratchett'">
        <xsl:value-of select="title"/>

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

This does work correctly!

xsl:choose

- The xsl:choose ... xsl:when ... xsl:otherwise construct is XML's equivalent of Java's switch ... case ... default statement
- The syntax is:

 xsl:choose is often used within an xsl:for-each loop

xsl:sort

- You can place an xsl:sort inside an xsl:for-each
- The attribute of the sort tells what field to sort on
- Example:

```
     <xsl:for-each select="//book">
          <xsl:sort select="author"/>
          <xsl:value-of select="title"/> by
                <xsl:value-of select="author"/> 
                 </xsl:for-each>
```

 This example creates a list of titles and authors, sorted by author

xsl:text

- <xsl:text>...</xsl:text> helps deal with two common problems:
 - XSL isn't very careful with whitespace in the document
 - This doesn't matter much for HTML, which collapses all whitespace anyway (though the HTML source may look ugly)
 - <xsl:text> gives you much better control over whitespace; it acts like the element in HTML
 - Since XML defines only five entities, you cannot readily put other entities (such as) in your XSL
 - & amp; nbsp; almost works, but is visible on the page
 - Here's the secret formula for entities:

<xsl:text disable-output-escaping="yes">&nbsp;</xsl:text>



Creating tags from XML data

- Suppose the XML contains <name>Dr. Dave's Home Page</name> <url>http://www.cis.upenn.edu/~matuszek</url>
- And you want to turn this into Dr. Dave's Home Page
- We need additional tools to do this
 - It doesn't even help if the XML directly contains

 Dr. Dave's Home Page -- we still can't move it to the output
 - The same problem occurs with images in the XML

Creating tags--solution 1

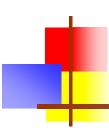
Suppose the XML contains

```
<name>Dr. Dave's Home Page</name>
<url>http://www.cis.upenn.edu/~matuszek</url>
```

- <xsl:attribute name="..."> adds the named attribute to the enclosing tag
- The *value* of the attribute is the *content* of this tag
- Example:

```
<a>>
    <xsl:attribute name="href">
        <xsl:value-of select="url"/>
        </xsl:attribute>
        <xsl:value-of select="name"/>
        </a>
```

Result: Dr. Dave's Home Page



Creating tags--solution 2

- Suppose the XML contains
 - <name>Dr. Dave's Home Page</name>
 <url>http://www.cis.upenn.edu/~matuszek</url>
- An attribute value template (AVT) consists of braces { } inside the attribute value
- The content of the braces is replaced by its value
- Example:

Result:

Modularization

- Modularization--breaking up a complex program into simpler parts--is an important programming tool
 - In programming languages modularization is often done with functions or methods
 - In XSL we can do something similar with xsl:apply-templates
- For example, suppose we have a DTD for book with parts titlePage, tableOfContents, chapter, and index
 - We can create separate templates for each of these parts

Book example

Etc.



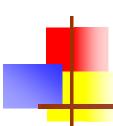
xsl:apply-templates

- The <xsl:apply-templates> element applies a template rule to the current element or to the current element's child nodes
- If we add a select attribute, it applies the template rule only to the child that matches
- If we have multiple <xsl:apply-templates> elements with select attributes, the child nodes are processed in the same order as the <xsl:apply-templates> elements



When templates are ignored

- Templates aren't used unless they are applied
 - Exception: Processing always starts with select="/"
 - If it didn't, nothing would ever happen
- If your templates are ignored, you probably forgot to apply them
- If you apply a template to an element that has child elements, templates are *not* automatically applied to those child elements



Applying templates to children

```
<book>
  <title>XML</title>
                                              With this line:
  <author>Gregory Brill</author>
                                         XML by Gregory Brill
 </book>
<xsl:template match="/">
  <html> <head></head> <body>
    <b><xsl:value-of select="/book/title"/></b>
    <xsl:apply-templates select="/book/author"/>
  </body> </html>
</xsl:template>
<xsl:template match="/book/author">
  by <i><xsl:value-of select="."/></i>
                                            Without this line:
</xsl:template>
                                                  \mathbf{XML}
```



Calling named templates

- You can name a template, then call it, similar to the way you would call a method in Java
- The named template:

```
<xsl:template name="myTemplateName">
    ...body of template...
</xsl:template>
```

• A call to the template:

```
<xsl:call-template name="myTemplateName"/>
```

Or:

```
<xsl:call-template name="myTemplateName">
    ...parameters...
</xsl:call-template>
```

Templates with parameters

Parameters, if present, are included in the content of xsl:template, but are the *only* content of xsl:calltemplate Single quotes inside double quotes make this a string Example call: <xsl:call-template name="doOneType"> <xsl:with-param name="header" select="Lectures"'/> <xsl:with-param name="nodes" select="//lecture"/> </xsl:call-template> Example template: This parameter is a <xsl:template name="doOneType"> typical XPath expression <xsl:param name="header"/> <xsl:param name="nodes"/> ...template body...refer to parameters as "\$header" and "\$nodes"

Parameters are matched up by name, not by position

</xsl:template>

Thoughts on XSL

- XSL is a programming language—and not a particularly simple one
 - Expect to spend considerable time debugging your XSL
- These slides have been an *introduction* to XSL and XSLT--there's a *lot more* of it we haven't covered
- As with any programming, it's a good idea to start simple and build it up incrementally: "Write a little, test a little"
 - This is especially a good idea for XSLT, because you don't get a lot of feedback about what went wrong
- I use jEdit with the XML plugin
 - I find it to be a big help, expecially with XML syntax
 - My approach is: write (or change) a line or two, check for syntax errors, then jump to IE and reload the XML file

