

XSLT

What is XSL?

- XSL stands for Extensible Stylesheet Language
- CSS was designed for styling HTML pages, and can be used to style XML pages
- XSL was designed specifically to style XML pages, and is much more sophisticated than CSS
- XSL consists of *three* languages:
 - XSLT (XSL Transformations) is a language used to transform XML documents into other kinds of documents (most commonly HTML, so they can be displayed)
 - XPath is a language to select parts of an XML document to transform with XSLT
 - XSL-FO (XSL Formating Objects) is a replacement for CSS

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

How does it work?

- The XML **source document** is parsed into an XML **source tree**
- You use XPath to define **templates** that *match* parts of the source tree
- You use XSLT to *transform* the matched part and put the transformed information into the **result tree**
- The result tree is output as a **result document**
- Parts of the source document that are not matched by a template are typically copied unchanged

Very simple example

- File `data.xml`:

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

- File `render.xsl`:

```
<?xml version="1.0"?>  
<xsl:stylesheet version="1.0"  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  <!-- one rule, to transform the input root (/) -->  
  <xsl:template match="/">  
    <html><body>  
      <h1><xsl:value-of select="message"/></h1>  
    </body></html>  
  </xsl:template>  
</xsl:stylesheet>
```

The .xsl file

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

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

```
<xsl:template match="/">
  <html><body>
    <h1><xsl:value-of select="message"/></h1>
  </body></html>
</xsl:template>
```
- The `<xsl:template match="/">` chooses the root
- The `<html><body> <h1>` 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

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:value-of

```
<xsl:value-of select="XPath expression" />
```

Remarks: The <xsl:value-of> element inserts a text string representing the value of the first element (in document order) specified by the select attribute.

If the XPath expression returns more than a single node, the <xsl:value-of> element returns the text of the first node returned.

If the node returned is an element with substructure, <xsl:value-of> returns the concatenated text nodes of that element's subtree with the markup removed (like the data() function).

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:

```
<ul>
```

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

```
    <li> <xsl:value-of select="title"/> </li>
```

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

```
</ul>
```

Filtering output

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

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

- 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:
= != < >
– Numbers should be quoted, but apparently don't have to be

But it doesn't work right!

- Here's what we did:

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

```
  <li>
```

```
    <xsl:value-of
```

```
      select="title[../author=' Terry Pratchett ' ]"/>
```

```
  </li>
```

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

- 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

- **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'">
    <li>
      <xsl:value-of select="title"/>
    </li>
  </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>  
  <xsl:when test="some condition">  
    ... some code ...  
  </xsl:when>  
  <xsl:otherwise>  
    ... some code ...  
  </xsl:otherwise>  
</xsl:choose>
```
- **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:

```
<ul>
```

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

```
    <xsl:sort select="author"/>
```

```
    <li> <xsl:value-of select="title"/> by
```

```
      <xsl:value-of select="author"> </li>
```

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

```
</ul>
```

- 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 `<pre>` element in HTML
 - Since XML defines only five entities, you cannot readily put other entities (such as ` `) in your XSL
 - ` ` almost works, but ` ` is *visible* on the page
 - Here's the secret formula for entities:
`<xsl:text disable-output-escaping="yes"> </xsl:text>`

Using XSL to create HTML

- Our goal is to turn *this*:

```
<?xml version="1.0"?>
<library>
  <book>
    <title>XML</title>
    <author>Gregory Brill</author>
  </book>
  <book>
    <title>Java and XML</title>
    <author>Brett McLaughlin</author>
  </book>
</library >
```

- Into HTML that displays something like *this*:

Book Titles:

- XML
- Java and XML

Book Authors:

- Gregory Brill
- Brett McLaughlin

- Note that we've grouped titles and authors separately

Desired HTML

```
<html>
  <head>
    <title>Book Titles and Authors</title>
  </head>
  <body>
    <h2>Book titles:</h2>
    <ul>
      <li>XML</li>
      <li>Java and XML</li>
    </ul>
    <h2>Book authors:</h2>
    <ul>
      <li>Gregory Brill</li>
      <li>Brett McLaughlin</li>
    </ul>
  </body>
</html>
```

Red text is data extracted from the XML document

White text is our HTML template

We don't necessarily know how much data we will have

All of books.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/
    XSL/Transform">
<xsl:template match="/">
<html>
  <head>
    <title>Book Titles and Authors</title>
  </head>
  <body>
    <h2>Book titles:</h2>
    <ul>
      <xsl:for-each select="//book">
        <li>
          <xsl:value-of select="title"/>
        </li>
      </xsl:for-each>
    </ul>
```

```
<h2>Book authors:</h2>
  <ul>
    <xsl:for-each
      select="//book">
      <li>
        <xsl:value-of
          select="author"/>
      </li>
    </xsl:for-each>
  </ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XQuery + HTML

```
<html>
  <head>
    <title>Book Titles and Authors</title>
  </head>
  <body>
    <h2>Book titles:</h2>
    <ul>
      {
        for $x in doc("books.xml")/library/
book/title
        return <li>{data($x)}</li>
      }
    </ul>
```

```
<h2>Book authors:</h2>
  <ul>
    {
      for $x in doc("books.xml")/
library/book/author
      return <li>{data($x)}</li>
    }
  </ul>
</body>
</html>
```


Creating tags from XML data

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

Creating tags--solution 2

- Suppose the XML contains

```
<name>Dr. AAA's Home Page</name>
<url>http://www.ece.rutgers.edu/~aaa</url>
```
- An **attribute value template** (AVT) consists of braces { } inside the attribute value
- The content of the braces is replaced by its value
- Example:

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

```
<a href="http://www.ece.rutgers.edu/~aaa">
  Dr. AAA's Home Page</a>
```

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

- `<xsl:template match="/">`
 `<html> <body>`
 `<xsl:apply-templates />`
 `</body> </html>`
 `</xsl:template>`
- `<xsl:template match="tableOfContents">`
 `<h1>Table of Contents</h1>`
 `<xsl:apply-templates select="chapterNumber" />`
 `<xsl:apply-templates select="chapterName" />`
 `<xsl:apply-templates select="pageNumber" />`
 `</xsl:template>`
- 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>`
 `<author>Gregory Brill</author>`
 `</book>`

- `<xsl:template match="/">`
 `<html> <head></head> <body>`
 `<xsl:value-of select="/book/title"/>`
 `<xsl:apply-templates select="/book/author"/>`
 `</body> </html>`
 `</xsl:template>`

```
<xsl:template match="/book/author">  
  by <i><xsl:value-of select="."/></i>  
</xsl:template>
```

With this line:
XML by *Gregory Brill*

Without this line:
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:call-template`

- Example call:

```
<xsl:call-template name="doOneType">  
  <xsl:with-param name="header" select="'Lectures'"/>  
  <xsl:with-param name="nodes" select="//lecture"/>  
</xsl:call-template>
```

Single quotes inside double quotes make this a string

- Example template:

```
<xsl:template name="doOneType">  
  <xsl:param name="header"/>  
  <xsl:param name="nodes"/>  
  ...body of template...  
</xsl:template>
```

This parameter is a typical XPath expression

- Parameters are matched up by *name*, not by position