XSL
XSL is a standard that consists of three parts:
- **XPath** (navigation in documents)
  - XPath was taught in the DB course, so it will not be taught
- **XSLT** (transformation of documents)
- **XSLFO** (FO for formatting objects)
  - This is a rather complex language for typesetting (i.e., preparing text for printing)
  - It will not be taught

XSLT
- XSLT is a language for transforming XML documents into other XML documents
  - For example, XHTML, RSS, KML, GML, MathML, WML
  - Can also transform XML to text documents, e.g., SQL programs
- An XSLT program is itself an XML document (called an **XSL stylesheet**) that describes the transformation process for input documents

XSLT Processors
- DTD
- XML
- XSL
- XSLT Processor
- XHTML
- WML
- Text

Web Pages – The Whole Picture
- Knowledge
- XSL
- XSLT
- XML
- Doc. Structure
- CSS
- Style
- Data
- Layout
- Web Page
Applying XSL Stylesheets to XML

There are several ways of applying an XSL stylesheet to an XML document:
- Directly applying an XSLT processor to the XML document and the XSL stylesheet
- Calling an XSLT processor from within a program
- Adding to the XML document a link to the XSL stylesheet and letting the browser do the transformation
- The resulting XHTML document is shown instead of the original XML

Processing XSL in Java

You can use the XALAN package of Apache in order to process XSL transformations:

```
java org.apache.xalan.xslt.Process
-IN myXmlFile.xml
-XSL myXslFile.xsl
-OUT myOutputFile.html
```

How Does XSLT Work?

- An XSL stylesheet is a collection of templates that are applied to source nodes (i.e., nodes of the given XML)
- Each template has a match attribute that specifies to which source nodes the template can be applied
- Each source node has at a template that matches it
- The current source node is processed by applying a template that matches this node
- When processing a node, it is possible (but not necessary) to recursively process other nodes, e.g., the children of the processed node
- The XSLT processor processes the document root (/)
<?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>cd catalog</title></head>
      <body><h1>This is a cd catalog!</h1></body>
    </html>
  </xsl:template>
</xsl:stylesheet>

<catalog>
  <cd country="UK">
    <title>Dark Side of the Moon</title>
    <artist>Pink Floyd</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Space Oddity</title>
    <artist>David Bowie</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Aretha: Lady Soul</title>
    <artist>Aretha Franklin</artist>
    <price>9.90</price>
  </cd>
</catalog>

In XALAN, automatically added to <head>

Examples of Match Attributes
- match="cd",
  All elements with tag name cd
- match="//cd",
  match="/catalog//artist"
  All matches of the absolute XPath
- match="cd/artist"
  All artist nodes that have a cd parent
- match="catalog//artist"
  All artist nodes that have a catalog ancestor
- match="cd[@country='UK']/artist"
Default Templates

- XSL provides implicit built-in templates that match every element and text nodes
  ```xml
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:value-of select="."/>
  </xsl:template>
  ```

- Templates we write always override these built-in templates (when they match)

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

In XALAN and in IE (it yields an empty page in Firefox)

```xml
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="cd[@country='UK']">
    <h2>A cd!</h2>
  </xsl:template>
</xsl:stylesheet>
```

Does not match the context node

```xml
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates select="/catalog/cd[@country='UK']/artist"/>
  </xsl:template>
</xsl:stylesheet>
```
The Most Frequently Used Elements of XSL

- `<xsl:value-of select="xpath"/>`
  - This element extracts the value of a node from the node list located by `xpath`

- `<xsl:for-each select="xpath"/>`
  - This element loops over all the nodes in the node list located by `xpath`

- `<xsl:if test="cond"/>
  `<xsl:if test="xpath"/>
  etc.
  - This element is for conditional processing

Example 1

Example 2

catalog2.xsl

catalog3.xsl
The \texttt{<xsl:sort>} Element

- The \texttt{<xsl:sort>} element is used to sort the list of nodes that are looped over by the \texttt{<xsl:for-each>} element.
- Thus, the \texttt{<xsl:sort>} must appear inside the \texttt{<xsl:for-each>} element.
- The looping is done in sorted order.

Setting Values in Attributes

- The \texttt{<xsl:value-of>} element cannot be used within attribute value.
- However, we can insert \texttt{expressions} into attribute values, by putting the expression inside curly braces ({}).
- Alternatively, we can use \texttt{<xsl:element>} in order to construct XML elements.

An Example

- In the following example, we add to each CD entitled \texttt{t} a link to the URL \texttt{/showcd.jsp?title=t}.

\begin{verbatim}
<xsl:template match="cd">
  <b><xsl:value-of select="artist"></b>:
  <a href="/showcd.jsp?title={./title}">
    <xsl:value-of select="title"/>
  </a>
</xsl:template>
\end{verbatim}
On XSL Code

- Typically, an XSLT program can be written in several, very different ways
  - Templates can sometime replace loops and vice versa
  - Conditions can sometimes be replaced with XPath predicates (e., in the select attribute) and vice versa
  - A matter of convenience and elegance

On Recursive Templates

- It is not always possible to avoid recursive templates
  - That is, use only the template of the root
- Suppose that we want to write an XSL stylesheet that generates a copy of the source document
  - It is rather easy to do it when the structure of the source XML document is known
- Can we write an XSL stylesheet that does it for every possible XML document?
  - Yes! (see next slide)

Generating Valid XHTML

- By default, the documents that XSL stylesheets generate are not valid XHTML
- Next, we will show how XSL stylesheet can be changed in order to generate valid XHTML

Identity Transformation Stylesheet
The Original XSL Example

```xml
<?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>cd catalog</title>
      </head>
      <body>
        <h1>This is a cd catalog!</h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

The Original Transformation Result

```xml
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>cd catalog</title>
</head>
<body>
  <h1>This is a cd catalog!</h1>
</body>
</html>
```

Modifying the XSL Example

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN" doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>cd catalog</title>
      </head>
      <body>
        <h1>This is a cd catalog!</h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

The Transformation Result

```xml
<head>
  <title>cd catalog</title>
</head>
<body>
  <h1>This is a cd catalog!</h1>
</body>
</html>
```

Some Other XSL Elements

- The `<xsl:text>` element inserts free text in the output
- The `<xsl:copy-of select="xpath">` creates a copy of the specified nodes (deep copy, i.e., copies the entire subtree)
- The `<xsl:copy select="xpath">` creates a copy of the specified nodes (does not copy children or attributes)
- The `<xsl:comment>` element creates a comment node in the result tree
- The `<xsl:variable>` element defines a variable (local or global) that can be used within the program
Example

- Transform this list of number to be
- Sorted
- Alternatingly red and blue

Using Variables and Functions

- Using names, templates can be called directly and serve as functions
- It is possible to use variables and function in XSL to perform computations
- Returned values of functions can be set to variables

<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="fib.xsl"?>
<val>
  8
</val>
Transforming XML to XHTML, in the Browser

- Use JavaScript
- Check what is the current browser before applying the transformation (different browsers use a slightly different DOM)
Calling Xalan in Java Programs

- The following example illustrates how to execute Xalan on given XML and XSL, from within a Java program

```java
cs 236369
Spring 2013
61

1. Generate a document builder
2. Parse the XML document

// Instantiate a DocumentBuilderFactory.
javax.xml.parsers.DocumentBuilderFactory dfactory =
        newInstance();

// Use the DocumentBuilderFactory to provide access to a DocumentBuilder.
javax.xml.parsers.DocumentBuilder dBuilder = dfactory.newDocumentBuilder();

// Use the DocumentBuilder to parse the XML input.
org.w3c.dom.Document inDoc = dBuilder.parse("foo.xml");

1. Generate a document builder
2. Parse the XML document

1. Generate a transformer
2. Create an empty DOM result
3. Apply the transformation
4. Get the output node

// Generate a Transformer.
javax.xml.transform.Transformer transformer =
    tFactory.newTransformer(new
    javax.xml.transform.stream.StreamSource("foo.xsl");

// Create an empty DOMResult object for the output.
javax.xml.transform.dom.DOMResult domResult =
    new
    javax.xml.transform.dom.DOMResult();

// Perform the transformation.
transformer.transform(new
    javax.xml.transform.dom.DOMSource(inDoc), domResult);

// Now you can get the output Node from the DOMResult.
org.w3c.dom.Node node = domResult.getNode();

// Generate a Transformer.
javax.xml.transform.Transformer transformer =
    tFactory.newTransformer(new
    javax.xml.transform.stream.StreamSource("foo.xsl");

// Create an empty DOMResult object for the output.
javax.xml.transform.dom.DOMResult domResult =
    new
    javax.xml.transform.dom.DOMResult();

// Perform the transformation.
transformer.transform(new
    javax.xml.transform.dom.DOMSource(inDoc),
    new StreamResult(new FileOutputStream("result.html")));"

To produce HTML, replace
```

Links

- W3C Recommendation: [http://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt)
- [http://www.w3schools.com/xsl/default.asp](http://www.w3schools.com/xsl/default.asp)