Chapter 8: XSLT

Chapter 8 Objectives

- How XSLT can be used to convert XML for presentation or restructure XML for business-to-business data interchange.
- How XSLT differs from conventional procedural languages.
- An XSLT transformation is described in terms of a source document and a result document. However, under the hood, the transformation taking place is a source tree (which uses the XPath data model) to a result tree (which also uses the XPath data model).
- How the elements that make up an XSLT stylesheet are used. For example, you look at how to use the xsl:value-of element to retrieve values from the source tree which is being transformed, and look at the xsl:copy and xsl:copy-of elements which, respectively, shallow copy and deep copy nodes from the source tree.
- How to use XSLT variables and parameters.
- The new features of XSLT 2.0 and how they make transformations easier.

What is XSLT?

Extensible Stylesheet Language Transformations

XML-Based

It's like a XML conversion toolkit

Restructuring XML

- Data exchange
- Create new resulting documents

Many more

Presenting XML Content

- Typically, HTML or XHTML

Not limited to these

How Does an XSLT Processor Work?

- Takes an XML document, applies a stylesheet and outputs a new XML HTML document
- Not all XML editors do XSLT processing
- May require additional software

- Saxon
- XML Copy Editor

Procedural versus Declarative Programming

- Java and JavaScript is procedural
  - Tell computer what you want step by step
- SQL and XSLT are declarative
  - Tell computer what you want to achieve
- Also, XSLT is a
  - Template-based language
  - Functional language

Example: want to create certain output for every chapter element

```
<xsl:template match="Chapter">
 <!-- The content of the <xsl:template> element defines what is to
    be added -->
 <!-- to the result tree. -->
</xsl:template>
```


Foundational XSLT Elements

Example: People.xml and People.xslt

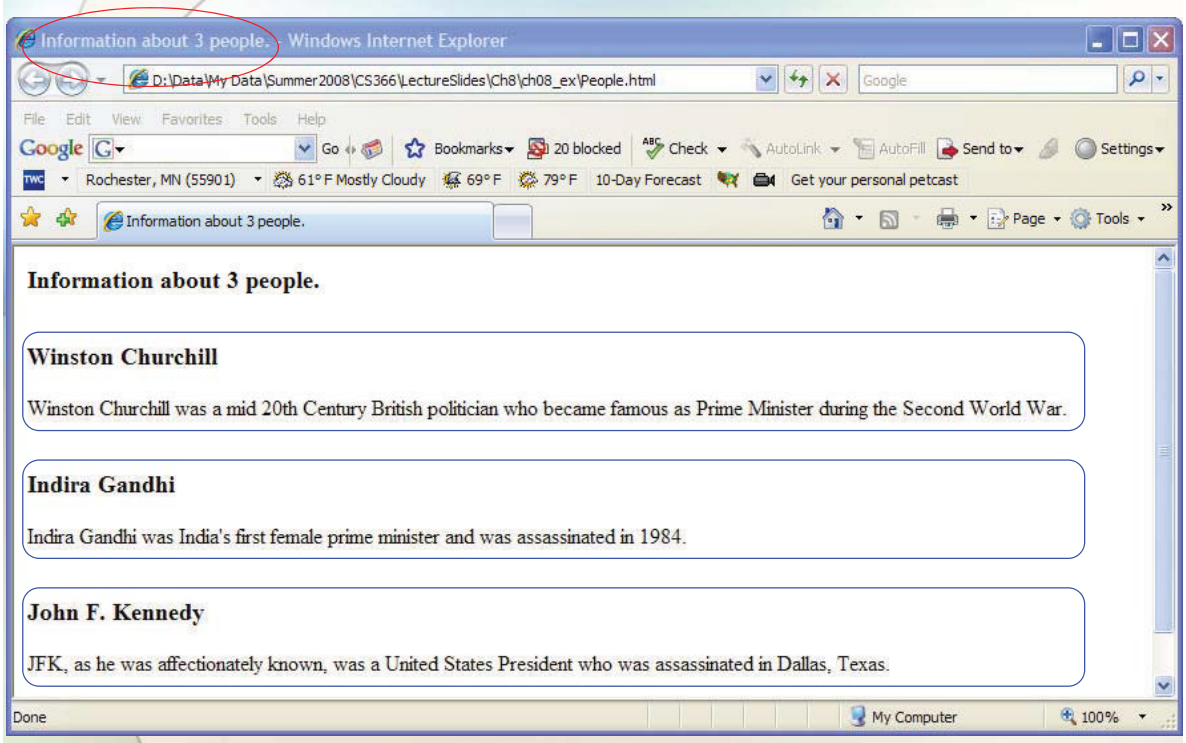
```
<People>
  <Person>
   <Name>Winston Churchill</Name>
   <Description>Winston Churchill was a mid 20th century British politician who
 became famous as Prime Minister during the Second World War.</Description>
  </Person>
  <Person>
   <Name>Indira Gandhi</Name>
   <Description>Indira Gandhi was India's first female prime minister and was
 assassinated in 1984.</Description>
  </Person>
  <Person>
   <Name>John F. Kennedy</Name>
   <Description>JFK, as he was affectionately known, was a United States president
 who was assassinated in Dallas, Texas.</Description>
  </Person> </People>
```

Command: java -jar saxon9he.jar -o outfile infile.xml infile.xslt
Or use Kernow

Foundational XSLT Elements

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform“ version="1.0">
 <xsl:template match="/">
  <html>
   <head>
    <title>Information about
    <xsl:value-of select="count(/People/Person)" /> people.</title>
   </head>
   <body>
    <h3>Information about
    <xsl:value-of select="count(/People/Person)" /> people.</h3>
    <br />
    <xsl:apply-templates select="/People/Person" />
   </body>
  </html>
 </xsl:template>
 <xsl:template match="Person">
  <h3><xsl:value-of select="Name" /></h3>
  <p><xsl:value-of select="Description" /></p>
  <br />
 </xsl:template>
</xsl:stylesheet>
```


Foundational XSLT Elements



The <xsl:stylesheet> Element

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


The <xsl:template> Element

```
<xsl:template match="/">
 <html>
  <head>
   <title>Information about
<xsl:value-of select="count(/People/Person)" /> people.</title>
  </head>
  <body>
  <h3>Information about
<xsl:value-of select="count(/People/Person)" /> people.</h3>
  <br />
    <xsl:apply-templates select="/People/Person" />
  </body>
 </html>
</xsl:template>
```

: output directly

The <xsl:apply-templates> Element

```
<xsl:apply-templates select="/People/Person" />
```

```
<People>
 <Person>
  <Name>Winston Churchill</Name>
  <Description>Winston Churchill was a mid 20th Century British politician who
 became famous as Prime Minister during the Second World War.</Description>
 </Person>
 <Person>
  <Name>Indira Gandhi</Name>
  <Description>Indira Gandhi was India's first female prime minister and was
 assassinated in 1984.</Description>
 </Person>
 <Person>
  <Name>John F. Kennedy</Name>
  <Description>JFK, as he was affectionately known, was a United States
    President
 who was assassinated in Dallas, Texas.</Description>
 </Person>
</People>
```


The <xsl:apply-templates> Element

```
<xsl:template match="Person">
 <h3><xsl:value-of select="Name" /></h3>
 <p><xsl:value-of select="Description" /></p>
 <br />
</xsl:template>
```

The <xsl:value-of>

```
<html>
 <head>
  <title>Information about <xsl:value-of select="count(/People/Person)" />
 people.</title>
 </head>
 <body>
 <h3>Information about <xsl:value-of select="count(/People/Person)" />
 people.</h3>
```

Result

```
<title>Information about 3 people.</title>
```


The <xsl:copy> Element

- Copies a node
- Does not copy descendants
- Does not copy attributes
- Use <xsl:copy-of> for deep copy

<xsl:copy> Example

Example: transform elements to attribute using **<xsl:attribute>** element

Persons.xml

```
<Persons>
 <Person>
  <FirstName>Jill</FirstName>
  <LastName>Harper</LastName>
 </Person>
 <Person>
  <FirstName>Claire</FirstName>
  <LastName>Vogue</LastName>
 </Person>
 <Person>
  <FirstName>Paul</FirstName>
  <LastName>Cathedral</LastName>
 </Person>
```

Want …

```
<Persons>
  <Person FirstName="Jill"
          LastName="Harper"/>
  <Person FirstName="Claire"
          LastName="Vogue"/>
  <Person FirstName="Paul"
          LastName="Cathedral"/>
</Persons>
```


<xsl:copy> Example

Persons.xslt

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

<xsl:template match="/">
 <Persons>
  <xsl:apply-templates select="/Persons/Person" />
 </Persons>
</xsl:template>

<xsl:template match="Person">
 <xsl:copy>
  <xsl:attribute name="FirstName"><xsl:value-of select="FirstName"/>
  </xsl:attribute>
  <xsl:attribute name="LastName"><xsl:value-of select="LastName"/>
  </xsl:attribute>
 </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Create 3 Person nodes

Add attribute's name and value

<xsl:copy> Example

Example: transform back to elements using **<xsl:element>** element

Persons2.xslt

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

<xsl:template match="Person">
 <xsl:copy>
  <xsl:element name="FirstName"><xsl:value-of select="@FirstName"/>
  </xsl:element>
  <xsl:element name="LastName"><xsl:value-of select="@LastName"/>
  </xsl:element>
 </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```


The <xsl:copy-of> Element

The xsl:copy-of element causes a **deep copy** to take place. In other words, a node together with all its attribute nodes and descendant nodes is copied to the result tree.

<xsl:copy-of> Example

Example: restructuring document, **<xsl:comment>** also used here

PurchaseOrder.xml

```
<PurchaseOrder>
 <From>Example.org</From>
 <To>XMML.com</To>
 <Address>
  <Street>234 Any Street</Street>
  <City>Any Town</City>
  <State>MO</State>
  <ZipCode>98765</ZipCode>
 </Address>
 <!-- Other purchase order information
       would go here. -->
</PurchaseOrder>
```

Want …

```
<Invoice>
  <From>XMML.com</From>
  <To>Example.org</To>
  <Address>
   <Street>234 Any Street</Street>
   <City>Any Town</City>
   <State>MO</State>
   <ZipCode>98765</ZipCode>
  </Address>
  <!--The rest of the Invoice would
       go here.-->
</Invoice>
```


PurchaseOrder.xslt

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

<xsl:template match="/">
 <Invoice>
  <xsl:apply-templates select="PurchaseOrder/To" />
  <xsl:apply-templates select="PurchaseOrder/From" />
  <xsl:apply-templates select="PurchaseOrder/Address" />
  <xsl:comment>The rest of the Invoice would go here.</xsl:comment>
 </Invoice>
</xsl:template>

<xsl:template match="To">
 <xsl:element name="From"><xsl:value-of select="." /></xsl:element>
</xsl:template>

<xsl:template match="From">
 <xsl:element name="To"><xsl:value-of select="." /></xsl:element>
</xsl:template>

<xsl:template match="Address">
 <xsl:copy-of select="." />
</xsl:template>

</xsl:stylesheet>
```

restructuring

deep copy

Using <xsl:output>

Influencing the output

default: <xsl:output method="xml" />

<xsl:output method="html" />

<xsl:output method="text" />

xhtml?

Not supported in XSLT 1.0 → use xml

Supported in XSLT 2.0

Conditional Processing: <xsl:if> Element

- Test whether a boolean is true or false
  - If true, then instantiated
  - If false, then nothing

Example: report characters with ages larger than 110

```
<Characters>
 <Character age="99">Julius Caesar</Character>
 <Character age="23">Anne Boleyn</Character>
 <Character age="41">George
Washington</Character>
 <Character age="45">Martin Luther</Character>
 <Character age="800">Methuselah</Character>
 <Character age="119">Moses</Character>
 <Character age="50">Asterix the Gaul</Character>
</Characters>
```

<xsl:if> Element Example

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

<xsl:template match="/">
 <html>
  <head>
   <title>Age check on Characters.</title>
  </head>
  <body>
  <h3>The recorded age is unusually high. Please check original data.</h3>
  <xsl:apply-templates select="/Characters/Character" />
  </body>
 </html>
</xsl:template>

<xsl:template match="Character">
 <xsl:if test="@age &gt; 110 " >
 <p><b><xsl:value-of select="." /></b> is older than expected. Please check if this
character's age, <b><xsl:value-of select="@age" /></b>, is correct.</p>
 </xsl:if>
</xsl:template>

</xsl:stylesheet>
```


Conditional Processing: <xsl:choose> Element

- When there are more than one conditions

Example: CharacterChoose.xslt

```
<xsl:template match="Character">
 <xsl:choose>
  <xsl:when test="@age &gt; 110 " >
   <p><b><xsl:value-of select="." /></b> - too high. Please check
    if this
   character's age, <b><xsl:value-of select="@age" /></b>, is
    correct.</p>
  </xsl:when>
  <xsl:otherwise>
   <p><b><xsl:value-of select="." /></b> - ok</p>.
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The <xsl:for-each> Element

- Allows all nodes in a node-set to be processed
- Not a loop through

Example: Objects.xml

```
<Objects>
 <Object name="Car">
  <Characteristic>Hard</Characteristic>
  <Characteristic>Shiny</Characteristic>
  <Characteristic>Has 4
wheels</Characteristic>
  <Characteristic>Internal Combustion
Engine</Characteristic>
 </Object>
</Objects>
```

```
    :
<xsl:apply-templates
  select="/Objects/Object" >

    :
<xsl:template match="Object">
 <ul>
 <xsl:for-each select="Characteristic">
  <li><xsl:value-of select="." /></li>
 </xsl:for-each>
 </ul>
</xsl:template>
    :
```


The <xsl:sort> Element

- Sort a node-set (not a loop through)
- Default order is ascending

Example: Objects2.xml & Objects.xslt (sort objects & their characteristics)

```
    :
<xsl:apply-templates select="/Objects/Object" >
  <xsl:sort select="@name" />
 </xsl:apply-templates>
    :
<xsl:template match="Object">
 <h3>Characteristics of <xsl:value-of select="@name" /></h3>
 <ul>
 <xsl:for-each select="Characteristic">
  <xsl:sort select="." order="descending" />
  <li><xsl:value-of select="." /></li>
 </xsl:for-each>
 </ul>
</xsl:template>
    :
```

The <xsl:sort> Element

Example: a different (recursive) approach without <xsl:for-each>

```
    :
<ul>
 <xsl:for-each select="Characteristic">
 <xsl:sort select="." order="descending" />
  <li><xsl:value-of select="." /></li>
 </xsl:for-each>
 </ul>
          :
```

Objects2.xslt

```
          :
<ul>
 <xsl:apply-templates select="Characteristic">
 <xsl:sort select="." order="descending" />
 </xsl:apply-templates>
 </ul>
</xsl:template>
          :
<xsl:template match="Characteristic">
  <li><xsl:value-of select="." /></li>
</xsl:template>
          :
```


XSLT Modes

Used to process certain nodes more than once.

Example: BegXML.xml and BegXML.xslt

```
<xsl:apply-templates select="/Book/Chapters/Chapter" mode="TOC" />
<xsl:apply-templates select="/Book/Chapters/Chapter" mode="fulltext" />

<xsl:template match="Chapter" mode="TOC" >
<xsl:template match="Chapter" mode="fulltext" >
```

XSLT Variables and Parameters

- <xsl:param> element
  - declare a local parameter
  - $para_name
  - can be passed from outside
- <xsl:variable> element
  - declare a variable (constant)
  - $variable

- To pass in a parameter from the command line:

```
java -jar saxon9he.jar -o Ages.html Ages.xml Ages.xsl
person="Peter"
```

(Ages.bat)

XSLT Variables and Parameters

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
<xsl:param name="person" />
<xsl:template match="/">
 <html>
  <head>
   <title>Finding an age using an XSLT parameter</title>
  </head>
  <body>
   <xsl:apply-templates select="/Ages/Person[@name=$person]" />
  </body>
 </html>
</xsl:template>
<xsl:template match="Person">
<p>The age of <xsl:value-of select="$person" /> is <xsl:value-of  select="@age"/>
    </p>
</xsl:template>
</xsl:stylesheet>
```

The <xsl:call-template> Element

- Like a function, could be **recursive**.
- Define

  <xsl:template name=" *TemplateName*">

  <!-- The template content goes here. -->

  </xsl:template>
- Call

  <xsl:call-template name=" *TemplateName*">
- With parameters

  <xsl:template name=" *TemplateName*">

  <xsl:with-param name =" *ParameterName*">

  <!-- The template content goes here. -->

  </xsl:template>

XSLT Functions

- **Partial list**
  - **document(): access the nodes in an external XML document**
  - **key(): returns a node-set using the index specified by an <xsl:key> element**
  - **format-number(): converts a number into a string**
  - **generate-id(): returns a string value that uniquely identifies a specified node**

Looking Forward to XSLT 2.0

The latest version of the XSLT 2.0 specific is located at http://www.w3.org/tr/xslt20/.

- Grouping in Version 2.0
- New data model
- W3C XML Schema data types
- New elements
- Non XML input
- Improved string handling
- Multiple outputs
- New functions

XSLT 2.0 Examples

- Grouping
  - contacts.xml and groupedContacts.xslt (groupedContacts.bat)
- Input from non-XML
  - config.ini and createConfig.xslt (nonXML.bat)
- Output to multiple files
  - contacts.xml and separatedContacts.xslt (separatedContacts.bat)