Introduction to XSLT

James Cummings

January 2010

What is the XSL family?

- XPath: a language for expressing paths through XML trees
- XSLT: a programming language for transforming XML
- XSL FO: an XML vocabulary for describing formatted pages

XSLT

The XSLT language is

- expressed in XML
- uses namespaces to distinguish output from instructions
- purely functional
- reads and writes XML trees

It was designed to generate XSL FO, but now widely used to generate HTML.

How is XSLT used? (1)

- With a command-line program to transform XML (eg to HTML)
  - Downside: no dynamic content, user sees HTML
  - Upside: no server overhead, understood by all clients
- In a web server *servlet*, eg serving up HTML from XML (eg Cocoon, XTF, Axkit)
  - Downside: user sees HTML, server overhead
  - Upside: understood by all clients, allows for dynamic changes

How is XSLT used? (2)

- In a web browser, displaying XML on the fly
  - Downside: many clients do not understand it
  - Upside: user sees XML
- Embedded in specialized program
- As part of a chain of production processes, performing arbitrary transformations

XSLT implementations

MSXML	Built into Microsoft Internet Explorer
Saxon	Java-based, standards leader, implements XSLT 2.0 (basic version free)
Xalan	Java-based, widely used in servlets (open source)
libxslt	C-based, fast and efficient (open source)
transformiix	C-based, used in Mozilla (open source)

What is a transformation?

Take this:

```
<persName>
  <forename>Milo</forename>
  <surname>Casagrande</surname>
</persName>
<persName>
  <forename>Corey</forename>
  <surname>Burger</surname>
</persName>
<persName>
  <forename>Naaman</forename>
  <surname>Campbell</surname>
</persName>
```

and make this:

```
<item  n="1">
  <name>Burger</name>
</item>
<item  n="2">
  <name>Campbell</name>
</item>
<item  n="3">
  <name>Casagrande</name>
</item>
```

A text example

Take this

```
<div type="recipe" n="34">
  <head>Pasta for beginners</head>
  <list>
    <item>Pasta</item>
    <item>Grated cheese</item>
  </list>
  <p>Cook the pasta and mix with the cheese</p>
</div>
```

and make this

```
<html>
  <h1>34:  Pasta for beginners</h1>
  <p>Ingredients:  Pasta Grated cheese</p>
  <p>Cook the pasta and mix with the cheese</p>
</html>
```

How do you express that in XSL?

```
<xsl:stylesheet
   xpath-default-namespace="http://www.tei-
c.org/ns/1.0" version="2.0">
 <xsl:template match="div">
  <html>
   <h1>
    <xsl:value-of select="@n"/>:
   <xsl:value-of select="head"/>
   </h1>
   <p>Ingredients:
   <xsl:apply-templates select="list/item"/>
   </p>
   <p>
    <xsl:value-of select="p"/>
   </p>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Structure of an XSL file

```
<xsl:stylesheet
   xpath-default-namespace="http://www.tei-
c.org/ns/1.0" version="2.0">
  <xsl:template match="div">
<!-- .... do something with div elements.....-->
  </xsl:template>
  <xsl:template match="p">
<!-- .... do something with p elements.....-->
  </xsl:template>
</xsl:stylesheet>
```

The `div` and `p` are *XPath expressions*, which specify which bit of the document is matched by the template.

Any element not starting with xsl: in a template body is put into the output.

The Golden Rules of XSLT

1. If there is no template matching an element, we process the elements inside it
2. If there are no elements to process by Rule 1, any text inside the element is output
3. Children elements are not processed by a template unless you explicitly say so:
4. `xsl:apply-templates select="XX"` looks for templates which match element "XX"; `xsl:value-of select="XX"` simply gets any text from that element
5. The order of templates in your program file is immaterial
6. You can process any part of the document from any template
7. Everything is well-formed XML. Everything!

Important magic

Our examples and exercises all start with two important attributes on <stylesheet>:

```
<xsl:stylesheet
   xpath-default-namespace="http://www.tei-
c.org/ns/1.0"  version="2.0">...
</xsl:stylesheet>
```

which indicates

1. In our XPath expressions, any element name without a namespace is assumed to be in the TEI namespace
2. We want to use version 2.0 of the XSLT specification. This means that we must use the Saxon processor for our work.

A simple test file

```
<text>
 <front>
  <div>
   <p>Material up front</p>
  </div>
 </front>
 <body>
  <div>
   <head>Introduction</head>
   <p rend="it">Some sane words</p>
   <p>Rather more surprising words</p>
  </div>
 </body>
 <back>
  <div>
   <p>Material in the back</p>
  </div>
 </back>
</text>
```

Feature: apply-templates

```
<xsl:stylesheet version="2.0"
   xpath-default-namespace="http://www.tei-c.org/ns/1.0">
  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:template match="TEI">
  <xsl:apply-templates select="text"/>
</xsl:template>
```

```
<xsl:template match="text">
  <h1>FRONT MATTER</h1>
  <xsl:apply-templates select="front"/>
  <h1>BODY MATTER</h1>
  <xsl:apply-templates select="body"/>
</xsl:template>
```

Feature: value-of

Templates for paragraphs and headings:

```
<xsl:template match="p">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>
<xsl:template match="div">
  <h2>
    <xsl:value-of select="head"/>
  </h2>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="div/head"/>
```

Notice how we avoid getting the heading text twice.

Why did we need to qualify it to deal with just <head> inside ?

More complex patterns

The match attribute of <template> can point to any part of the document. Using XPath expressions, we can find:

/	the root of document (*outside* the root element)
*	any element
text()	
name	an element called 'name'
@name	an attribute called 'name'

Example of complete path in <value- of>:

```
<xsl: value- of
   select="/TEI/teiHeader/fileDesc/titleStmt/title"/>
```

Example of context-dependent matches

Compare

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

with

```
<xsl:template match="div/head"> ...
</xsl:template>
<xsl:template match="figure/head"> ....
</xsl:template>
```

Priorities when templates conflict

It is possible for it to be ambiguous which template is to be used:

```
<xsl:template match="person/name">…
</xsl:template>
<xsl:template match="name">…
</xsl:template>
```

when the processor meets a <name>, which template is used?

Solving priorities

There is a priority attribute on <template>; the higher the value, the more inclined the XSLT engine is to use it:

```
<xsl:template match="name" priority="1">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="person/name" priority="2"> A name
</xsl:template>
```

Template priority generally

The more normal rule is that the most specific template wins.

```
<xsl:template match="*">
<!-- ... -->
</xsl:template>
<xsl:template match="tei:*">
<!-- ... -->
</xsl:template>
<xsl:template match="p">
<!-- ... -->
</xsl:template>
<xsl:template match="div/p">
<!-- ... -->
</xsl:template>
<xsl:template match="div/p/@n">
<!-- ... -->
</xsl:template>
```

Pushing and pulling

XSLT stylesheets can be characterized as being of two types:

push In this type of stylesheet, there is a different template for every element, communication via `<xsl:apply-templates>` and the overall result is assembled from bits in each template. It is sometimes hard to visualize the final design. Common for data-oriented processing where the structure is fixed.

pull In this type, there is a master template (usually matching `/`) with the main structure of the output, and specific `<xsl:for-each>` or `<xsl:value-of>` commands to grab what is needed for each part. The templates tend to get large and unwieldy. Common for document-oriented processing where the input document structure varies.

Attribute value template

What if we want to turn

```
<ref target="http://www.oucs.ox.ac.uk/">OUCS</ref>
```

into

```
<a href="http://www.oucs.ox.ac.uk/"/>
```

? What we cannot do is

```
<xsl:template match="ref">
  <a href="@target">
    <xsl:apply-templates/>
  </a>
</xsl:template>
```

This would give the @*href* attribute the value '@target'.

For example

Instead we use {} to indicate that the expression must be evaluated:

```
<xsl:template match="ref">
  <a href="{@target}">
    <xsl:apply-templates/>
  </a>
</xsl:template>
```

This would give the @*href* attribute whatever value the attribute @*target* has!

Feature: for-each

If we want to avoid lots of templates, we can do in-line looping over a set of elements. For example:

```
<xsl:template match="listPerson">
  <ul>
    <xsl:for-each select="person">
      <li>
        <xsl:value-of select="persName"/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

contrast with

```
<xsl:template match="listPerson">
  <ul>
    <xsl:apply-templates select="person"/>
  </ul>
</xsl:template>
<xsl:template match="person">
  <li>
    <xsl:value-of select="persName"/>
  </li>
</xsl:template>
```

Feature: `if`

We can make code conditional on a test being passed:

```
<xsl:template match="person">
 <xsl:if test="@sex='1'">
  <li>
   <xsl:value-of select="persName"/>
  </li>
 </xsl:if>
</xsl:template>
```

contrast with

```
<xsl:template match="person[@sex='1']">
 <li>
  <xsl:value-of select="persName"/>
 </li>
</xsl:template>
<xsl:template match="person"/>
```

The *@test* can use any XPath facilities.

Feature: choose

We can make a multi-value choice conditional on what we find in the text:

```
<xsl:template match="person">
  <xsl:apply-templates/>
  <xsl:choose>
    <xsl:when test="@sex='1'">(male)
    </xsl:when>
    <xsl:when test="@sex='2'">(female)
    </xsl:when>
    <xsl:when test="not(@sex)">(no sex specified)
    </xsl:when>
    <xsl:otherwise>(unknown sex)
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Feature: number

We can produce numbering based on the position of elements in the text.

1. Position within containing element:

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

2. Position within whole document:

```
<xsl:template match="p">
  <xsl:number level="any"/>
</xsl:template>
```

3. Position within an element further up the tree:

```
<xsl:template match="l">
  <xsl:number level="any" from="lg"/>
</xsl:template>
```

Creating new elements and attributes dynamically in the output

Instead of explicitly putting elements into the output, you can use <xsl:element> and <xsl:attribute>. So instead of

```
<a href="http://www.bbc.co.uk/">the BBC</a>
```

you can say

```
<xsl:element name="a">
  <xsl:attribute name="href">
    <xsl:text>http://www.bbc.co.uk/</xsl:text>
  </xsl:attribute>
  <xsl:text>The BBC</xsl:text>
</xsl:element>
```

which sometimes makes for more readable code. Note the use of <xsl:text> to control whitespace.

Summary

Now you can

1. Write templates which match any element or attribute
2. Pick out text from anywhere
3. Write code conditional on something in the text
4. Number output objects