Crash Course in XSLT

Beginning XSLT
Jan. 17, 2007
David Z. Maze

Last Time...

* XML syntax
* XML namespaces
* XPath

A Trivial Stylesheet

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <xsl:copy-of select="/"/>
  </xsl:template>
</xsl:stylesheet>
```


Making It Go

* Command-line processor, like xsltproc
* xsltproc stylesheet.xsl input.xml
* Java javax.xml infrastructure
* xml-stylesheet PI plus Web browser
* <?xml-stylesheet type="text/xml" href="foo.xsl"?>

What Do You Get Out?

* <xsl:output method="xml" indent="no"/>
* Possible output methods: xml, html, text
* XHTML is "xml"; default is either "xml" or "html" depending on name of root element
* Some other options, but probably bad form to depend on them

Templates

* <xsl:template match="foo:bar" mode="baz">
* Think "subroutine"
* Can take parameters, have local variables, contains XSLT "statements"
* Optional *mode* controls when the template is invoked
* match attribute is restricted XPath expression with child or attribute axes, // and | operators, and arbitrary predicates

Easy Ways To Get Output

* `<number>42</number>` (*literal result element*): non-XSLT XML (or text) is copied to output
* `<xsl:text>foo</xsl:text>`
* `<xsl:copy-of select="..."/>` copies entire selected subtree(s) (if any)
* `<xsl:value-of select="..."/>` evaluates expression, takes string value (discarding XML tags), and outputs that

Counting Things

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <counts>
      <count item="slides">
        <xsl:value-of
         select="count(//slide:slide)"
         xmlns:slide=
          "http://www.mit.edu/~dmaze/slide"/>
      </count>
      <count item="graphics">
        <xsl:value-of select="count(//svg:svg)"
         xmlns:svg="http://www.w3.org/2000/svg"/>
      </count>
    </counts>
  </xsl:template>
</xsl:stylesheet>
```


Applying Templates

* `<xsl:apply-templates select="*" mode="baz"/>`
* There's always a *context node*
* "Apply templates": find some set of other nodes by XPath expression; then invoke the matching template, but only if the mode matches
* Could do nothing if no nodes match
* Default for text is to copy to output

Matching Multiple Templates

```
<xsl:template match="xhtml:p"/>
<xsl:template match="xhtml:*"/>
```

* What do you do if multiple templates match?
* Top-level stylesheet first
* Highest priority (explicit `xsl:template/@priority`) wins
* Default priority: 0 for child/attribute full-name match; -0.25 for child/attribute namespace match (`pfx:*`); -0.5 for child/element kind match (`child::text()`); 0.5 for anything else
* Still a tie? "Error", but last one in stylesheet wins

HTML to XSL:FO

```
<p>Text with <i>italic</i> word</p>

<xsl:template match="p">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="i">
  <fo:inline font-style="italic">
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>

<fo:block>
 Text with
 <fo:inline font-style="italic">italic</fo:inline>
 word
</fo:block>
```


Modes

* Sometimes, just want to copy an element; other times, want to do full processing; other times...
* Can create/invoke template with named *mode*
* `<xsl:template match="..." mode="...">`
* `<xsl:apply-templates select="..." mode="..."/>`
* Defaults to no mode at all

Let's go backwards!

```
<xsl:template match="*">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates select="*[last()]"/>
  </xsl:copy>
  <xsl:apply-templates
   select="preceding-sibling::*[1]"/>
</xsl:template>
```


Named templates

* Templates can have *names* as well as match expressions
* `<xsl:call-template name="..."/>`
* Context node same as in caller

Let's go backwards! (again)

```
<xsl:template name="attr-children">
  <xsl:copy-of select="@*"/>
  <xsl:apply-templates select="*[last()]"/>
</xsl:template>
<xsl:template match="*" mode="child">
  <xsl:call-template name="attr-children">
</xsl:template>
<xsl:template match="*">
  <xsl:call-template name="attr-children">
  <xsl:apply-templates
    select="preceding-sibling::*"
    mode="child"/>
</xsl:template>
```


Attribute value templates (AVTs)

```
<xsl:template match="x">
  <x count="{count(child::*)}">
    <xsl:apply-templates/>
  </x>
</xsl:template>
```

* Any XPath expression can go inside curly braces in many contexts
* (Including attribute values in LREs)

I don't know my name

* `<xsl:element name="..." namespace="...">`
* `<xsl:attribute name="..." namespace="...">`
* Contents of element/attribute used as contents of element/attribute
* Name and namespace attributes are both AVTs
* This is also useful for only maybe generating attributes
* Can't use xsl:attribute after first child has been output

Picking a namespace

```
<xsl:element name="..." namespace="...">
```

* **Is namespace present?** Use its value (possibly empty)
* **Does name have a prefix?** Use its namespace from the stylesheet
* **Otherwise?** Use the null namespace
* The generated element will have no prefix if in the null namespace, or otherwise the processor's choice of namespace (possibly the one in name)

Comments In The Output

```
<!-- Comment in the stylesheet -->
<xsl:comment>in the output</xsl:comment>
```


What if...

```
<xsl:if test="...">
  ...
</xsl:if>
```

* If/then, without an "else"
* `test` is any (boolean) XPath expression
* Contents are evaluated if `test` is true (non-empty)

But what if...

```
<xsl:choose>
  <xsl:when test="...">...</xsl:when>
  <xsl:when test="...">...</xsl:when>
  <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

* If first `xsl:when` is true, do that; otherwise, if second is true, do that, otherwise...
* Only way to have "else"

How many is many?

```
<xsl:attribute name="quantity">
  <xsl:choose>
    <xsl:when test="count(*)=0">none</xsl:when>
    <xsl:when test="count(*)=1">one</xsl:when>
    <xsl:when test="count(*)&lt;8">some</xsl:when>
    <xsl:otherwise>many</xsl:otherwise>
  </xsl:choose>
</xsl:attribute>

<xsl:if test="*">
  <xsl:attribute .../>
</xsl:if>
```


Variables

```
<xsl:variable name="..." select="..."/>
<xsl:variable name="...">...</xsl:variable>
```

* Defines a variable for all following things in the same template, plus their descendants
* No duplicate variables in the same template (might shadow globals)
* `select` form can have arbitrary XPath
* Otherwise create *result tree fragment* with root node and specified contents
* `<xsl:copy-of select="$variable"/>`
* No content? Defaults to empty string
* `select="/.."` for empty nodeset
* Get value later using XPath `$name`

How many is many?

```
<xsl:variable name="children"
   select="*[@on='yes']"/>
<xsl:variable name="count"
   select="count($children)"/>
<xsl:attribute name="quantity">
  <xsl:choose>
    <xsl:when test="$count=0">none</xsl:when>
    <xsl:when test="$count=1">one</xsl:when>
    <xsl:when test="$count&lt;8">some</xsl:when>
    <xsl:otherwise>many</xsl:otherwise>
  </xsl:choose>
</xsl:attribute>
```

Parameters

```
<xsl:param name="..." select="..."/>
<xsl:param name="...">...</xsl:param>
<xsl:apply-templates>
  <xsl:with-param name="..." select="..."/>
  <xsl:with-param name="...">...</xsl:with-param>
</xsl:apply-templates>
```

- Just like `xsl:variable`, but...
- Must appear first in template content
- `xsl:param`'s value is only the default value
- Callers may specify `xsl:with-param` to provide their own values

Backwards (yet again)

```
<xsl:template match="*">
  <xsl:param name="pos" select="count(*)"/>
  <xsl:variable name="item" select="*[$pos]"/>
  <xsl:element name="{local-name($item)}">
    <xsl:apply-templates select="$item"/>
  </xsl:element>
  <xsl:if test="$pos&gt;1">
    <xsl:apply-templates select=".">
      <xsl:with-param name="pos" select="$pos-1"/>
    </xsl:apply-templates>
  </xsl:if>
</xsl:template>
```


Globals

- You can define global `xsl:variable` and `xsl:param`
- Put them before the first template
- They can depend on input
- Global params can usually be set externally
- `xsltproc --param foo "'bar'"`

For-each

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

- Evaluates contents for each node in `select` node-set
- Context node changes

Sorting

```
<xsl:apply-templates>
  <xsl:sort select="@name"/>
</xsl:apply-templates
```

- Specify an order besides document order for `xsl:apply-templates` and `xsl:for-each`
- `order="ascending"` (or descending)
- `data-type="text"` (or number)
- Multiple sorts fine, first is "more important"

Modularizing

```
<xsl:include href="..."/>
<xsl:import href="..."/>
```

- Stylesheets you include can include other stylesheets
- Circular includes are illegal
- `xsl:include` is "the same stylesheet", `xsl:import` isn't

Next Time

* Gory details!
* XPath types, function library
* Extension elements and EXSLT