XSLT, continued

Sebastian Rahtz
July 2001

Recap 1

We have met the following XSL basic controls:

```
<xsl:stylesheet>
<xsl:template match="...">
<xsl:apply-templates select="...">
<xsl:value-of select="...">
<xsl:text>
<xsl:if test="...">
<xsl:choose>
<xsl:for-each select="...">
```

Recap 2

... with the following ‘extras’

```
<xsl:sort>
<xsl:number>
id()
count()
sum()
last()
```

Other important commands

`<xsl:element>`: make a new element
`<xsl:attribute>`: add an attribute and value
`<xsl:comment>`: make a comment
`<xsl:processing-instruction>`: make a processing instruction
`<xsl:copy>`: copy the current element
`<xsl:copy-of>`: copy a (sub)tree
`<xsl:message>`: put out a message
`<xsl:key>`: construct a lookup table

More functions

- `document`: (*string*)
- `generate-id`: (*string*)
- `position`: ()
- `name`: (*node*)
- `concat`: (*string*, *string*)
- `contains`: (*string*, *string*)
- `substring-before`: (*string*, *string*)
- `substring-after`: (*string*, *string*)
- `string-length`: (*string*)
- `translate`: (*node*,*string*, *string*)
- `normalize-space`: (*string*)

Example of element/attribute creation

```
<xsl:template match="hi">
  <xsl:comment>this was a hi element</xsl:comment>
  <xsl:message>doing a hi now </xsl:message>
  <xsl:element name="span">
    <xsl:attribute name="class">
      <xsl:value-of select="@rend"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

(test1.xsl)

Copying input to output

Contrast

```
<xsl:template match="*">
  <xsl:copy><xsl:apply-templates/></xsl:copy>
</xsl:template>
```

with

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

(test2.xsl, test3.xsl)

A useful catchall template

```
<xsl:template match="p">
    <p><xsl:apply-templates/></p>
  </xsl:template>

  <xsl:template match="*">
   <span style="color: red">
     <xsl:text>&lt;</xsl:text>
     <xsl:value-of select="name(.)"/>
     <xsl:text>&gt;</xsl:text>
   </span>
   <xsl:apply-templates/>
   <span style="color: red">
     <xsl:text>&lt;/</xsl:text>
     <xsl:value-of select="name(.)"/>
     <xsl:text>&gt;</xsl:text>
   </span>
  </xsl:template>
```

(test4.xsl)

.. but what about attributes?

```
<xsl:template match="*">
   <span style="color: red">
     <xsl:text>&lt;</xsl:text>
     <xsl:value-of select="name(.)"/>
     <xsl:for-each select="attribute::*">
       <xsl:text> </xsl:text>
       <xsl:value-of select="name(.)"/>
       <xsl:text>=’</xsl:text><xsl:value-of se-
lect="."/>
       <xsl:text>’</xsl:text>
     </xsl:for-each>
     <xsl:text>&gt;</xsl:text>
   </span>
....
```

(test5.xsl)

String functions

```
  <xsl:template match="hi[@rend='upper']">
    <xsl:value-of se-
lect="translate(.,'abcdefghijklmnopqrstuvwxzy',
                          'ABCDEFGHIJKLMNOPQRSTU-
VWXZY')"/>
  </xsl:template>

  <xsl:template match="xref">
    <span style="color:red">
       (<xsl:value-of select="substring-
before(@url,'://')"/> protocol)
    </span>
    <span style="color:green">
       <xsl:value-of select="substring-
after(@url,'://')"/>
    </span>
  </xsl:template>
```

(test6.xsl)

Using `generate-id()`:

```
<xsl:template match="/">
    <p> <xsl:for-each select=".//p">
      <a href="#para-{generate-id()}">para
        <xsl:number level="any"/>, </a>
      </xsl:for-each>
    </p>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="p">
    <p> <a name="para-{generate-id()}"/><xsl:apply-
templates/></p>
  </xsl:template>
```

(test7.xsl)

Using keys

```
<xsl:key name="elements" match="*" use="name(.)"/>

  <xsl:template match="/">
    <xsl:for-each select="key('elements','hi')">
      <xsl:sort select="."/>
      <xsl:value-of select="."/>:
    </xsl:for-each>
  </xsl:template>
```

(test8.xsl)

The `document` function

Using `document` to pull in another file and process it:

```
<xsl:template match="*">
    <xsl:copy><xsl:apply-templates/></xsl:copy>
</xsl:template>

<xsl:template match="xptr[@type='include']">
  <xsl:for-each select="document(@url)/*">
    <xsl:copy><xsl:apply-templates/></xsl:copy>
  </xsl:for-each>
</xsl:template>
```

(test13.xsl)

Named templates, parameters and variables

`<xsl:template name="...">`: define a named template

`<xsl:call-template>`: call a named template

`<xsl:param>`: specify a parameter in a template definition

`<xsl:with-param>`: specify a parameter when calling a template

`<xsl:variable name="...">`: define a variable

Variables

```
<xsl:template match="p">
    <xsl:variable name="n">
      <xsl:number/>
    </xsl:variable>
    Paragraph <xsl:value-of select="$n"/>
    <a name="P{$n}"/>
       <xsl:apply-templates/>
  </xsl:template>
```

(test11.xsl)

Named templates

```
<xsl:template match="/div">
  <html>
   <xsl:call-template name="header">
    <xsl:with-param name="title" select="head"/>
   </xsl:call-template>
   <xsl:apply-templates/>
 </html>
</xsl:template>

<xsl:template name="header">
  <xsl:param name="title"/>
  <head>
    <title><xsl:value-of select="$title"/></title>
  </head>
</xsl:template>
```

(test12.xsl)

Top-level commands

`<xsl:import href="...">`: include a file of XSLT templates, overriding them as needed

`<xsl:include href="...">`: include a file of XSLT templates, but do not override them

`<xsl:output>`: specify output characteristics of this job

Some useful `xsl:output` attributes

```
method="xml | html | text"
encoding="string"
omit-xml-declation="yes | no"
doctype-public="string"
doctype-system="string"
indent="yes | no"
```

An identity transform

```
<xsl:output method="xml" indent="yes" encoding="iso-
8859-1"
    doctype-system="teixlite.dtd"/>

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

(test9.xsl)

A near-identity transform

```
<xsl:template match="*|@*|processing-instruction()">
 <xsl:copy>
 <xsl:apply-templates
    select="*|@*|processing-
instruction()|comment()|text()"/>
 </xsl:copy>
</xsl:template>

<xsl:template match="text()">
    <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="p">
  <para><xsl:apply-templates/></para>
</xsl:template>
```

(test10.xsl)

Generating plain text

```
<xsl:output method="text"/>
<xsl:template match="text()">
    <xsl:call-template name="split">
      <xsl:with-param name="text"
        select="concat(normalize-space(.),' ')"/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="split">
    <xsl:param name="text"/>
    <xsl:if test="not($text=”)">
      <xsl:value-of select="substring-
before($text,' ')"/><xsl:text>
</xsl:text>
      <xsl:call-template name="split">
        <xsl:with-param name="text" select="substring-
after($text,' ')"/>
      </xsl:call-template>
    </xsl:if>
</xsl:template>
```

(test14.xsl)

XSLT extensions

Although we will not be covering them here, most XSLT processors support various extensions, which will probably be formalized in version 2.0:

- The ability to create *multiple* output files from one input
- The ability to 'escape' to another language (eg Java) for special purposes
- The ability to turn results into input trees for further processing

TEI Stylesheets

A library of stylesheets for transforming TEI documents to either HTML or XSL Formatting Objects, at `http://www.hcu.ox.ac.uk/Stylesheets`, with a form-filling interface for the HTML at `http://www.hcu.ox.ac.uk/cgi-bin/tei/stylebear`

TEI Stylesheets (example)

An example of an importing stylesheet:

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
<xsl:import
 href="http://www.hcu.ox.ac.uk/TEI/Stylesheets/teihtml.xsl"/>

<xsl:variable name="splitLevel">1</xsl:variable>
<xsl:variable name="numberHeadings"></xsl:variable>
<xsl:variable name="topNavigationPanel">true</xsl:variable>

<xsl:variable name="bottomNavigationPanel">true</xsl:variable>

</xsl:stylesheet>
```