XSLT, continued

Sebastian Rahtz
February 2003

Recap

We have met the following XSL basic controls:

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

... with the following ‘extras’

```
<xsl:sort>
<xsl:number>
count()
sum()
XPath axes
qualified matches with []
```

Modes

You can process the same elements in different ways using modes:

```
<xsl:template match="/">
 <xsl:apply-templates select=".//div" mode="toc"/>
 <xsl:apply-templates/>
</xsl:template>

<xsl:template match="div" mode="toc">
 Heading <xsl:value-of select="head"/>
</xsl:template>
```

This is a very useful technique when the same information is processed in different ways in different places.

More functions

- `document`: (*string*)
- `generate-id`: (*string*)
- `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*)

A useful catchall template

The pattern * matches any element which does not have a more specific template, so

```
<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>
```

puts the names of all elements in red in the output. This shows you clearly which elements you have defined templates for.

.. 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 select="."/>
       <xsl:text>’</xsl:text>
     </xsl:for-each>
     <xsl:text>&gt;</xsl:text>
   </span>
....
```

String functions

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

(test4.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>
```

(test5.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>
```

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)

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.tei-c.org/Stylesheets, with a form-filling interface for the HTML at http://www.tei-c.org/tei-bin/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.tei-c.org/Stylesheets/teihtml.xsl"/>
<xsl:param
   name="splitLevel">1</xsl:param>
<xsl:param
   name="numberHeadings"></xsl:param>
<xsl:param
   name="topNavigationPanel">true</xsl:param>
<xsl:param
   name="bottomNavigationPanel">true</xsl:param>
</xsl:stylesheet>
```

What about the delivery problem?

We all have the same problem. We want documents which can be

- Put on the web, with a consistent style
- . . . with a different style next week
- . . . with a different style for colour-blind people
- Formatted nicely into a printable PDF file
- Printed out in Braille
- Indexed intelligently by next-generation web indexing robots
- Checked rigourously against syntactic constraints

. . . and all without any human intervention or maintenance cost.

And management too, please

- We want a complete document history and ability to revert to old versions
- We want to know who changed a file, and when, and why
- We want to have a workflow in which documents can be tested, approved, and published

In short, a *Content Management System*.
. . . and remember not to spend any money
. . . and keep it Open Source politically correct

Bits of a system, integrated or non-integrated

1. Storage, versioning and history
2. Authoring in some vocabulary
3. Content sanity (eg link validation)
4. Component assembly
5. **delivery via a web server**
6. Interface design (accessibility checking)

Key features of our XSLT system

- XSLT stylesheets can import others, and then override selected bits of them
- The local stylesheet can override at any level, right down to trapping character data
- Local stylesheet wrappers can be created using a web form
- CSS for decorating the HTML gives the designer a second bite at the cherry

Production system: HTML

Given a *vocabulary* and a *stylesheet*, we can

- Serve static HTML — too limited
- Supply XML and XSLT to the browser — not enough browser support yet
- Throw it all into a database and do the work on retrieval — only delays the problem
- Have the web server do the work — not perfectly efficient, but flexible

We need a sub-system on the web server.

AxKit summary

- Open source package by Matt Sergeant, running under Apache mod_perl, written in Perl and C
- Now part of Apache XML project
- Uses libxslt XSLT processor by Daniel Veillard (or Sablotron)
- Normal scripting/extension work is in Perl
- Same philosophy and setup as Apache Cocoon (Java)
- Works with a pipeline of transformations, which may be in XSLT or other languages
- Powerful features for choosing stylesheeting according to PI in source file, DTD, media, client, URL parameters, etc

Under active development at `http://www.axkit.org/`, currently at version 1.6

The Transformation Pipeline

- AxKit processes material in a *pipeline*
- The output of one stage is input of the next (passes DOM tree in memory normally)
- Processes in the pipeline can be XSLT transformations, Perl programs, or anything we care to define
- Flexible definition of how to start the pipeline (Providers — ie a file system)
- Processes map to processors by mime types

How to control AxKit

AxKit is controlled by Apache directives, eg in `httpd.conf` or `.htaccess`:

```
PerlModule AxKit
<Files "*.xml"> AxAddStyleMap text/xsl
Apache::AxKit::Language::LibXSLT AxAddRootProcessor
text/xsl /stylesheets/teihtml-oucs.xsl
TEI.2 AxAddRootProcessor text/xsl
/stylesheets/docbook/html/docbook.xsl article </Files>
```

AxKit config 2: allow for multiple styles

```
<AxStyleName #default> AxAddProcessor text/xsl
teihtml-oucs.xsl </AxStyleName> <AxStyleName printable>
AxAddProcessor text/xsl teihtml-oucs-printable.xsl
</AxStyleName> <AxStyleName text> AxAddProcessor
text/xsl teihtml-oucs-text.xsl </AxStyleName>
<AxStyleName css> AxAddProcessor text/xsl
teihtml-oucs-css.xsl </AxStyleName> <AxStyleName raw>
AxAddProcessor text/xsl teihtml-oucs-raw.xsl
</AxStyleName>
```

Choosing the style

By processing instruction `<?xml-stylesheet title="screen" href="index.xsl" type="text/xsl"?>`

By URL parameter `index.xml?style=screen`

By user agent

```
PerlSetVar AxUAStyleMap "netscape => Mozilla/4.77,\
   text => Lynx, \
   screen=>"
```

By cookie

Ways of specifying the stylesheet

By root element `AxAddRootProcessor text/xsl teihtml-oucs.xsl TEI.2`

By doctype `AxAddDocTypeProcessor text/xsl teihtml-oucs.xsl "-//TEI P4//DTD Main DTD Driver File//EN"`

By URI `AxAddURIProcessor text/xsl teihtml-oucs.xsl "text.*\.xml$"`

By DTD `AxAddDTDProcessor text/xsl teihtml-oucs.xsl /dtds/tei-oucs.dtd`