Advanced XSLT

Web Data Management and Distribution

Serge Abiteboul Ioana Manolescu Philippe Rigaux
Marie-Christine Rousset Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

June 23, 2010

Outline

1 Stylesheets revisited

2 XSLT programming

3 Complements

4 Reference Information

5 Beyond XSLT 1.0

The children elements of `xsl:stylesheet`

import	Import the templates of an XSLT program, with low priorities
include	Same as before, but no priority level
output	Gives the output format (default: `xml`)
param	Defines or imports a parameter
variable	Defines a variable.
template	Defines a template.

Also: `strip-space` and `preserve-space`, for, resp., removal or preservation of blank text nodes. Not presented here.

Stylesheet Output

```
<xsl:output
  method="html"
  encoding="iso-8859-1"
  doctype-public="-//W3C//DTD HTML 4.01//EN"
  doctype-system="http://www.w3.org/TR/html4/strict.dtd"
  indent="yes"
/>
```

- `method` is either `xml` (default), `html` or `text`.
- `encoding` is the desired encoding of the result.
- `doctype-public` and `doctype-system` make it possible to add a document type declaration in the resulting document.
- `indent` specifies whether the resulting XML document will be indented (default is `no`).

Importing Stylesheets

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

Templates are imported this way from another stylesheet. Their precedence is less that that of the local templates.
`<xsl:import>` must be the first declaration of the stylesheet.

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

Templates are included from another stylesheet. No precedence rule: works as if the included templates were local ones.

Template Rule Conflicts

- Rules from imported stylesheets are overridden by rules of the stylesheet which imports them.
- Rules with highest priority (as specified by the `priority` attribute of `<xsl:template>`) prevail. If no priority is specified on a template rule, a default priority is assigned according to the specificity of the XPath expression (the more specific, the highest).
- If there are still conflicts, it is an error.
- If no rules apply for the node currently processed (the document node at the start, or the nodes selected by a `<xsl:apply-templates>` instruction), built-in rules are applied.

Built-in templates

- A first built-in rule applies to the **Element** nodes and to the root node:

```
<xsl:template match="*|/">
  <xsl:apply-templates select="node()" />
</xsl:template>
```

  Interpretation: recursive call to the children of the context node.

- Second built-in rule: applies to **Attribute** and **Text** nodes.

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

  Interpretation: copy the textual value of the context node to the output document.

Exercise: what happens when an empty stylesheet is applied to an XML document?

Global Variables and Parameters

```
<xsl:param name="nom" select="'John Doe'" />

<xsl:variable name="pi" select="3.14159" />
```

Global parameters are passed to the stylesheet through some implementation-defined way. The `select` attribute gives the default value, in case the parameter is not passed, as an XPath expression.

Global variables, as well as local variables which are defined in the same way inside template rules, are immutable in XSLT, since it is a side-effect-free language.

The `select` content may be replaced in both cases by the content of the `<xsl:param>` or `<xsl:variable>` elements.

Outline

1 Stylesheets revisited

2 XSLT programming

3 Complements

4 Reference Information

5 Beyond XSLT 1.0

Named templates

```
<xsl:template name="print">
  <xsl:value-of select="position()"/>:
     <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="*">
 <xsl:call-template name="print"/>
</xsl:template>

<xsl:template match="text()">
 <xsl:call-template name="print"/>
</xsl:template>
```

Named templates play a role analogous to functions in traditional programming languages.

Remark

A call to a named template does not change the context node.

Parameters

```
<xsl:template name="print">
  <xsl:param name="message" select="'nothing'"/>

  <xsl:value-of select="position()"/>:
      <xsl:value-of select="$message"/>
</xsl:template>

<xsl:template match="*">
 <xsl:call-template name="print">
   <xsl:with-param name="message"
             select="'Element node'"/>
 </xsl:call-template>
</xsl:template>
```

Same mechanism for `<xsl:apply-templates>`.

- **param** describes the parameter received from a template
- **with-param** defines the parameter sent to a template

Example: computing *n*! with XSLT

```
<xsl:template name="factorial">
  <xsl:param name="n" />

  <xsl:choose>
    <xsl:when test="$n&lt;=1">1</xsl:when>
    <xsl:otherwise>
      <xsl:variable name="fact">
        <xsl:call-template name="factorial">
          <xsl:with-param name="n" select="$n - 1" />
        </xsl:call-template>
      </xsl:variable>
      <xsl:value-of select="$fact * $n" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Conditional Constructs: `<xsl:if>`

```
<xsl:template match="Movie">
  <xsl:if test="year &lt; 1970">
    <xsl:copy-of select="."/>
  </xsl:if>
</xsl:template>
```

`<xsl:copy-of>` makes a deep copy of a node set; `<xsl:copy>` copies the nodes without their descendant.

Remark

- An XSLT program is an XML document: we must use entities for $<$ and &.
- XSLT is closely associated to XPath (node select, node matching, and here data manipulation)

Conditional Constructs: `<xsl:choose>`

```
<xsl:choose>
  <xsl:when test="$year mod 4">no</xsl:when>
  <xsl:when test="$year mod 100">yes</xsl:when>
  <xsl:when test="$year mod 400">no</xsl:when>
  <xsl:otherwise>yes</xsl:otherwise>
</xsl:choose>

<xsl:value-of select="count(a)"/>
<xsl:text> item</xsl:text>
<xsl:if test="count(a)>1">s</xsl:if>
```

`<xsl:otherwise>` is optional. There can be any number of `<xsl:when>`, only the content of the first matching one will be processed.

Loops

`<xsl:for-each>` is an instruction for looping over a set of nodes.
It is more or less an alternative to the use of
`<xsl:template>` / `<xsl:apply-templates>`.

- The set of nodes is obtained with an XPath expression (attribute `select`);
- Each node of the set becomes in turn the context node (which temporarily replaces the template context node).
- The body of `<xsl:for-each>` is instantiated for each context node.

$\Rightarrow$ no need to call another template: somewhat simpler to read, and likely to be more efficient.

The `<xsl:for-each>` element

Example (`<xsl:sort>` is optional):

```
<xsl:template match="person">
[...]
  <xsl:for-each select="child">
    <xsl:sort select="@age" order="ascending"
                 data-type="number"/>

    <xsl:value-of select="name" />
    <xsl:text> is </xsl:text>
    <xsl:value-of select="@age" />
</xsl:for-each>
[...]
</xsl:template>
```

`<xsl:sort>` may also be used as a direct child of an `<xsl:apply-templates>` element.

Variables in XSLT

A variable is a (*name*, *value*) pair. It may be defined

- Either as the result of an XPath expression

```
<xsl:variable name='pi' select='3.14116'/>
<xsl:variable name='children' select='//child'/>
```

- or as the content of the `<xsl:variable>` element.

```
<xsl:variable name="Signature">
  Franck Sampori<br/>
  Institution: INRIA<br/>
  Email: <i>franck.sampori@inria.fr</i>
</xsl:variable>
```

Remark

A variable has a scope (all its siblings, and their descendants) and cannot be redefined within this scope.

Outline

1 Stylesheets revisited

2 XSLT programming

3 Complements

4 Reference Information

5 Beyond XSLT 1.0

Other XSLT features

Many other instructions and functionalities. In brief:

Control of text output `<xsl:text>`, `<xsl:strip-space>`, `<xsl:preserve-space>`, and `normalize-space` function;

Dynamic creation of elements and attributes `<xsl:element>` and `<xsl:attribute>`.

Multiple document input, and multiple document output `document` function, `<xsl:document>` element (XSLT 2.0, but widely implemented in XSLT 1.0 as an extension function)

Generation of hypertext documents with links and anchors `generate-id` function

$\Rightarrow$ See the Exercises and projects.

Handling Whitespaces and blank nodes

Main rules

All the whitespace is kept in the input document, including blank nodes.
All the whitespace is kept in the XSLT program document, except blank nodes.

Handling Whitespace explicitly:

```
<xsl:strip-space elements="*" />
<xsl:preserve-space elements="para poem" />
```

`<xsl:strip-space>` specifies the set of nodes whose whitespace-only text child nodes will be removed, and `<xsl:preserve-space>` allows for exceptions to this list.

Dynamic Elements and dynamic attributes

```
<xsl:element name="{concat('p',@age)}"
  namespace="http://ns">
  <xsl:attribute name="name">
    <xsl:value-of select="name" />
  </xsl:attribute>
</xsl:element>
```

```
<person age="12">
  <name>titi</name>
</person>
```

$\longrightarrow$

```
<p12
  name="titi"
  xmlns="http://ns" />
```

Remark

The value of the `name` attribute is here an attribute template: this attribute normally requires a string, not an XPath expression, but XPath expressions between curly braces are evaluated. This is often used with literal result elements: `<toto titi="{$var + 1}"/>`.

Working with multiple documents

`document($s)` returns the document node of the document at the URL `$s`

Example: `document("toto.xml")/*`

Note: `$s` can by computed dynamically (e.g., by an XPath expression). The result can be manipulated as the root node of the returned document.

Unique identifiers

```
<xsl:template match="Person">
  <h2 id="{generate-id(.)}">
    <xsl:value-of
      select="concat(first_name, ' ', last_name)"/>
  </h2>
</xsl:template>
```

`generate-id($s)` returns a unique identifier string for the first node of the nodeset `$s` in document order.

Useful for testing the identity of two different nodes, or to generate HTML anchor names.

Outline

1 Stylesheets revisited

2 XSLT programming

3 Complements

4 Reference Information

5 Beyond XSLT 1.0

XSLT 1.0 Implementations

Browsers	All modern browsers (Internet Explorer, Firefox, Opera, Safari) include XSLT engines, used to process `xml-stylesheet` references. Also available via JavaScript, with various interfaces.
libxslt	Free C library for XSLT transformations. Includes `xsltproc` command-line tool. Perl and Python wrappers exist.
Sablotron	Free C++ XSLT engine.
Xalan-C++	Free C++ XSLT engine.
JAXP	Java API for Transformation. Common interface for various JAVA XSLT engines (e.g., SAXON, Xalan, Oracle). Starting from JDK 1.4, a version of Xalan is bundled with Java.
System.Xml	.NET XML and XSLT library.
php-xslt	XSLT extension for PHP, based on Sablotron.
4XSLT	Free XSLT Python library.

References

- `http://www.w3.org/TR/xslt`
- *XML in a nutshell*, Eliotte Rusty Harold & W. Scott Means, O'Reilly
- *Comprendre XSLT*, Bernd Amman & Philippe Rigaux, O'Reilly

Outline

1 Stylesheets revisited

2 XSLT programming

3 Complements

4 Reference Information

5 Beyond XSLT 1.0

Limitations of XSLT 1.0

- Impossible to process a temporary tree stored into a variable (with `<xsl:variable name="t"><toto a="3"/></xsl:variable>`). Sometimes indispensable!
- Manipulation of strings is not very easy.
- Manipulation of sequences of nodes (for instance, for extracting all nodes with a distinct value) is awkward.
- Impossible to define in a portable way new functions to be used in XPath expressions. Using named templates for the same purpose is often verbose, since something equivalent to $y = f(2)$ needs to be written as:

```
<xsl:variable name="y">
  <xsl:call-template name="f">
    <xsl:with-param name="x" select="2" />
  </xsl:call-template>
</xsl:variable>
```

Extension Functions

XSLT allows for extension functions, defined in specific namespaces. These functions are typically written in a classical programming language, but the mechanism depends on the precise XSLT engine used. Extension elements also exist.

Once they are defined, such extension functions can be used in XSLT in the following way:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="http://exslt.org/math"
  version="1.0"
  extension-element-prefixes="math">

  ...

  <xsl:value-of select="math:cos($angle)" />
```

EXSLT

EXSLT (`http://www.exslt.org/`) is a collection of extensions to XSLT which are portable across some XSLT implementations. See the website for the description of the extensions, and which XSLT engines support them (varies greatly). Includes:

exsl:node-set solves one of the main limitation of XSLT, by allowing to process temporary trees stored in a variable.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:exsl="http://exslt.org/common"
  version="1.0" extension-element-prefixes="exsl">

  ...

  <xsl:variable name="t"><toto a="3" /></xsl:variable>

  <xsl:value-of select="exsl:node-set($t)/*/@a" />
```

date	library for formatting dates and times
math	library of mathematical (in particular, trigonometric) functions
regexp	library for regular expressions
strings	library for manipulating strings
...	

Other extension functions outside EXSLT may be provided by each XSLT engine.

XSLT 2.0

- W3C Recommendation (2007)
- Like XQuery 1.0, uses XPath 2.0, a much more powerful language than XPath 1.0:
  - Strong typing, in relation with XML Schemas
  - Regular expressions
  - Loop and conditional expressions
  - Manipulation of sequences of nodes and values
  - ...
- New functionalities in XSLT 2.0:
  - Native processing of temporary trees
  - Multiple output documents
  - Grouping functionalities
  - User-defined functions
  - ...
- All in all, XSLT 2.0 stylesheets tend to be much more concise and readable than XSLT 1.0 stylesheets.

Example XSLT 2.0 Stylesheet

Produces a list of each word appearing in a document, with their frequency.

(from *XSLT 2.0 Programmer's Reference*)

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <wordcount>
    <xsl:for-each-group group-by="." select=
      "for $w in tokenize(string(.), '\W+')
         return lower-case($w)">
      <word word="{current-grouping-key()}"
            frequency="{count(current-group())}"/>
    </xsl:for-each-group>
  </wordcount>
</xsl:template>

</xsl:stylesheet>
```

XSLT 2.0 Implementations

A few implementations.

SAXON Java and .NET implementation of XSLT 2.0 and XQuery 1.0. The full version is commercial (free evaluation version), but a GPL version is available without support of external XML Schemas.

Oracle XML Developer's Kit Java implementation of various XML technologies, including XSLT 2.0, XQuery 1.0, with full support of XML Schema. Less mature than SAXON.

References

- `http://www.w3.org/TR/xpath20/`
- `http://www.w3.org/TR/xslt20/`
- *XPath 2.0 Programmer's Reference*, Michael Kay, Wrox
- *XSLT 2.0 Programmer's Reference*, Michael Kay, Wrox