Advanced Techniques in XSLT
Jean-Michel Hufflen

To cite this version:
Jean-Michel Hufflen. Advanced Techniques in XSLT. Biuletyn GUST, 2006, pp.69–75. hal-00663119

HAL Id: hal-00663119
https://hal.science/hal-00663119
Submitted on 26 Jan 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Advanced Techniques in XSLT

Jean-Michel HUFFLEN
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

This talk focus on some advanced techniques used within XSLT, such as sort procedures, keys, interface with identifier management, and priority rules among templates matching an XML node. We recall how these features work and propose some examples, some being related to bibliography styles. A short comparison between XSLT and nbst, the language used within MLaTeX for bibliography styles, is given, too.

Keywords XSLT, sort, keys, invoking rules, nbst.

Introduction

This didactic demonstration of XSLT — the language of transformations used for XML texts: XML nodes are matched by XSLT templates, in which case the corresponding rule is invoked — follows the more general introduction given at the BachoTeX 2005 conference [2]. We use the xsltproc program [10], belonging to the libxml2 library [9]. Both are running on Windows and Linux, but our demonstration is performed on the latter.

The XML texts used hereafter are already given in [3, Fig. 1 & 2]: a Polish song (Płonie ognisko) and two bibliographical entries using DocBook. We assume that readers can understand the behaviour of simple stylesheets written in XSLT. Basic knowledge of XPath [11], the language used to address parts of an XML document — in particular within XSLT programs — is required, too. So we propose an introduction to some advanced techniques within XSLT, some representative examples being given. As we will see, some of these techniques can be put into action within bibliography styles. We begin with the use of sort procedures and XSLT keys. Then we show how XSLT manages identifiers, as members of the ID type, and how a template is chosen among several ones whose match attribute matches an XML node. We end with a short comparison between XSLT and nbst, the language used within MLaTeX for bibliography styles.

Sorting — Using keys

About the behaviour of XSLT programs, we think that the best approach is to view elements such as xsl:apply-templates and xsl:for-each as producers of XML nodes, whereas xsl:template elements can be viewed as node consumers. When

---

* Title in Polish: XSLT — zaawansowane techniki.
1 XML — eXtensible Stylesheet Language Transformations.
2 eXtensible Markup Language. Readers interested in an introductory book to this formalism can refer to [5].
3 A progressive coursebook is [8].
4 New Bibliography STyles.
5 MultiLingual BiTeX.
an `xsl:template` element is invoked, it consumes the node matched by the value of the `match` attribute, and this node can have been produced by an `xsl:apply-templates` element of another template.

We know how to use the `xsl:sort` element if we keep this *modus operandi* in mind. We cannot physically change the order of appearance within a node list resulting from an XPath expression\(^6\), since side-effects do not exist in XSLT. The solution is to consider that a sort order is to be put *between*

\(^6\) Within the *official* terminology [11, § 4.1], such lists are called `node sets`, but the order is relevant, that is why we prefer using the word `list`.
the producer and the consumers [12, § 5.4]. The producer gives us a node list, this list is sorted before being sent to the corresponding consumer of each member of this list. For example, the following templates sorts all the verse elements, by increasing order, regarding the number of their characters, regardless of stanzas:

```<xsl:template match="body">
  <xsl:apply-templates select="stanza/verse"/>
  <xsl:sort select="string-length(string())" data-type="number"/>
</xsl:template>
```

some variants about parameterising the xsl:sort template being possible [12, § 10]. If several sort orders are used, each order sorts the elements viewed as equal by the previous sort statement. As an example, let us look at a bibliography w.r.t. DocBook [3, Fig. 2] and consider the biblioentry elements having an author child. To use authors’ last names as a primary sort key, first names as a secondary sort key, and middle name as a last sort key, just write:

```<xsl:apply-templates select="author/firstname"/>
<xsl:sort select="author/othername[role='mi']"/>
```

Let us go back to Plonic ognisko; if verse elements are given default behaviour, that is, displaying text nodes inside these elements, a program using the template given above displays verses, sorted w.r.t. the number of their characters. But the verses appear, in turn, without any other information. If we want to group the lines being n-character long and display them after a header, then do the same for (n+1)-character long lines, and so on... we cannot do that efficiently with the xsl:sort template. For such an application, the best technique consists of using the inverse images of a function, this function being called key w.r.t. XSLT’s terminology. The complete program illustrating the use of keys for this application is given in Fig. 1. By the way, we can see how information is passed to (resp. received by) a template by means of xsl:with-param (resp. xsl:param) elements. A simple container for a value — that is, a variable in the sense of programming languages — is built by an xsl:variable element. Last but not at least, this example shows that a template (e.g., by-char-nb) may be called recursively.

---

Managing identifiers

[3, Fig. 1] is a good use of attributes being types ID and IDREF. If we would like to display this song on a Web page, we may build an XSLT program whose output is an (X)HTML file, this program including the templates given in Fig. 2. The default behaviour for resume elements causes no display since this element is empty. To display the contents of such a stanza, XSLT provides the id function, that returns the node labelled by its argument:

```<xsl:template match="resume">
  <xsl:apply-templates select="id(@ref)/verse"/>
</xsl:template>
```

provided that this argument is an attribute being IDREF type: XSLT processors check that by looking into the DTD associated with the XML text. That is why using this function can be heavy. Such identifiers can be automatically generated within an XSLT program, by means of the generate-id function [12, § 12.4]. For example, this template allows a HTML hyperlink to point to each stanza:

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

When this function is applied, either it has already been applied to the same node — in which case the previous result is retained — or a new identifier is created. Using both key and generate-id functions

---

7 The solution suitable for both one author or several ones carried out by an authorgroup element is given in Fig. 3.

---

8 (X)Extensible HyperText Markup Language. [4] is a good introduction to these two languages, the ‘official’ documents being located at [13]. Let us recall that all the tags included in an XSLT program must conform to XML syntax, so XHTML tags must be used. Then the result is serialised according to HTML conventions [12, § 16.2].

9 Document Type Definition.
allow XSLT developers to create identifiers from keys partitioning a node set. The correspondence is one-to-one if the key is bijective. Otherwise, only the first member of an inverse image is labelled. As an example, if we adapt the program given in Fig. 1 to yield HTML pages, each group of same-sized lines can be accessed by a hyperlink put as follows:

```xml
<p> <!-- Corresponds to '[*]...[**]'. -->
  <a name="generate-id($buffer)"/>
  <xsl:apply-templates select="$buffer"/>
</p>
```

Managing priority

Let us look at [3, Fig. 2] and recall that the three kinds of titles are not to be displayed the same way when this bibliography is listed [3, § 1]. Within such an example, this means that a template devoted to processing title elements should be able to deal with a context information, the name of the parent bibliography, bibliographyentry, biblioset) in such a case. A better solution is to use three different templates, having different priorities.

The rules governing priority among XSLT templates are given in [12, § 5.5]. To sum up what is useful in most practical cases, let us say that only one template is invoked if several ones with the same priority match the same node.10 The priority may be managed explicitly by a namesake attribute: in practice, such an attribute is useful for specifying default rules with lower priority, or disabling imported templates by overriding them by other templates with higher priority. If the priority is not put explicitly, it is computed from the patterns used within match attributes: several-step patterns supersede one-step patterns. As an example, let us consider the XSLT program given in Fig. 3 and 4, which builds a LATEX environment — thebibliography — from a bibliography expressed in DocBook. The template matching title elements — one step — is viewed as a default template for such elements and is used for books’ titles in practice. It is superseded by two-step templates — the length of the corresponding XPath expression (step1/step2) — when a title element is a child of a bibliography or biblioset element. Likewise, a constraint also counts for a step (within a XPath expression like step1,[step2]), so a template just matching othername elements would be superseded by the last template.

nbst vs XSLT

A comprehensive of differences between XSLT and nbst — close but not identical to XSLT — has already come out in [1, App. C]. Here are the differences related to the abovementioned points.

- nbst does not provide equivalent elements for:
  - xsl:apply-imports [12, § 5.6],
  - xsl:import [12, § 2.6.2].

To implement the behaviour of these XSLT elements, use priority and mode attributes.

- The XSLT function generate-id is replaced by a function generate-newly with three arguments. generate-newly(s1,s2,ns) returns a unique string associated with the first node of the node list ns — if ns is not expressed, this function considers the current node — like generate-id. If s1 is not empty, it is used as result’s prefix. If s2 is not empty, it must be a format for numbers [12, § 7.7] and is used to generate result’s suffixes. If both s1 and s2 are empty, this function behaves like the generate-id function from XSLT.

Let us give an example with a ‘References’ section built by MiBibTEX w.r.t. an ‘alpha’ bibliography style, that is, labels are like [Donaldson 1982]. If the symbol whose print name is Donaldson 1982 has not been used yet, the expression:

```
generate-newly("Donaldson 1982","a")
```

returns this string itself. Otherwise, it builds Donaldson 1982a, Donaldson 1982b, . . . until it reaches a ‘new’ symbol. So this function is very useful to build this kind of keys, including the generation of several labels for the same author and year. As an other example:

```
generate-newly("Donaldson 1982","i")
```

appends Roman lower-case digits and generates Donaldson 1982i, Donaldson 1982ii, . . . in turn.

Going on

Readers interested in XSLT stylesheets applied to DocBook documents can find interesting and real-sized examples on the CD-ROM associated with [6] and in [7].

10 According to [12, § 5.5], such a case is an error. In practice, most XSLT processors invoke a template and ignore others.

11 Besides, this XSLT stylesheet gives some examples of using modes [12, § 5.7] when several distinct procedures are to be applied to the same elements.

12 We use the ‘symbol’ word, rather than ‘identifier’, because whitespace characters cannot be used as first parameter, as shown by the example. ‘Symbol’ can be related to the definition used for Lisp dialects.
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" id="process-dbk" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="ISO-8859-1"/>
  <xsl:strip-space elements="*"/>
  <!-- Do not discard blank nodes inside some mixed elements. -->
  <xsl:preserve-space elements="title..."/>
  <!-- Do not discard blank nodes inside some mixed elements. -->
  <xsl:variable name="eol" select="'&#10;'"/>
  <xsl:template match="bibliography">
    <xsl:value-of select="concat('\documentclass{article}',$eol,\'\usepackage[T1]{fontenc}',$eol,\'\usepackage[latin1]{inputenc}',$eol,$eol,\'\title{')"/>
    <xsl:apply-templates select="title"/>
    <xsl:value-of select="concat('}',$eol,\'\author{}',$eol,\'\date{}',$eol,$eol,\'\begin{document}',$eol,$eol,\'\maketitle',$eol,$eol,\'\begin{thebibliography}{')"/>
    <xsl:apply-templates select="biblioentry[1]" mode="longest-label"/>
    <xsl:apply-templates select="biblioentry"/>
    <xsl:sort select="/author[1]/surname"/>
    <!-- The first author, even if it is specified within an authorgroup element, so we use the descendant axis of XPath. -->
    <xsl:sort select="/author[1]/firstname"/>
    <!-- use the descendant axis -->
    <xsl:sort select="/othername[role = 'mi']"/>
    </xsl:apply-templates>
    <!-- of XPath. -->
    <xsl:value-of select="concat('}',$eol,$eol,\'\end{thebibliography}',$eol,$eol,\'\end{document}',$eol,$eol)"/>
  </xsl:template>
  <xsl:template match="bibliography/title">
    <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>

Figure 3: Processing a bibliography expressed using DocBook.
Jean-Michel HUFFLEN

<xsl:template match="title">
  <xsl:text>"</xsl:text>
  <xsl:apply-templates/>
  <xsl:value-of select='concat("apos;apos;",$,eol)'/>
</xsl:template>

<xsl:template match="biblioset/title">
  <xsl:text>\emph{</xsl:text>
  <xsl:apply-templates/>
  <xsl:value-of select="concat('}.',$eol)"/>
</xsl:template>

<xsl:template match="biblioentry">
  <xsl:value-of select="concat('[',@xreflabel,']{',@id,'}')"/>
  <xsl:variable name="the-author" select="author"/>
  <xsl:choose>
    <xsl:when test="$the-author"><xsl:apply-templates select="$the-author"/></xsl:when>
    <xsl:otherwise><xsl:apply-templates select="authorgroup"/></xsl:otherwise>
  </xsl:choose>
  <xsl:value-of select="concat(':',$eol)"/>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="biblioset">
    <xsl:with-param name="seriesvolumns" select="seriesvolnums"/>
  </xsl:apply-templates> <!--
  . . . (The end of this template is skipped.)
  -->
</xsl:template>

<xsl:template match="author">
  <xsl:apply-templates select="firstname"/>
  <xsl:apply-templates select="othername"/>
  <xsl:apply-templates select="surname"/>
</xsl:template>

<xsl:template match="authorgroup">
  <xsl:variable name="the-authors" select="author"/>
  <xsl:apply-templates select="$the-authors[1]"/>
  <xsl:choose>
    <xsl:when test="count($the-authors) = 2">
      <xsl:text> and </xsl:text>
      <xsl:apply-templates select="$the-authors[2]"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="$the-authors[position() &gt; 1]" mode="within-authorgroup"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="author" mode="within-authorgroup">
  <xsl:text>, </xsl:text>
  <xsl:if test="position() = last()"><xsl:text> and </xsl:text></xsl:if>
  <xsl:apply-templates select="."/>
</xsl:template>

<xsl:template match="othername[@role = 'mi']">
  <xsl:apply-templates/>
</xsl:template>

<![-- ... (Other templates are skipped. The complete stylesheet is available on CD-ROM.) -->
</xsl:stylesheet>

Figure 4: Processing a bibliography expressed using DocBook (Figure 3 continued).
Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who has written the Polish translation of the abstract.

References


