*Introduction to XSLT: scenarios and examples*

Laurent Romary

# General introduction to XSLT

* An XML application; a recommendation of the W3C: [XSL Transformations (XSLT) Version 3.0 W3C Recommendation 8 June 2017](https://www.w3.org/TR/xslt/)
* Based on XPath: addressing, selecting, functions
* Core principle: Input + XSLT => Output
* Usual usage: XML + XSLT => XML (e.g. XHTML)
* Other useful usages:
  + XML + XSLT => Text (e.g. generation of CSV outputs)
  + Text + XSLT => XML (e.g. structuring of existing text input)

# Making an XSLT stylesheet work

## Baseline XSLT structure

The following example is a complete and workable stylesheet to generate an HTML output from an XML document.

<**xsl:stylesheet** **exclude-result-prefixes**="xs"  
 **version**="2.0"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**xsl:output** **method**="html" **encoding**="UTF-8"/>  
</**xsl:stylesheet**>

Linking an XSLT document to an existing XML document in Oxygen: ‘Associate XSLT/CSS stylesheet’ (green pin).

Issues:

* The XSLT namespace: http://www.w3.org/1999/XSL/Transform
* <xsl:output>: importance of *@method* and *@encoding*
* Default XSLT behaviour: traversal of the XML tree structure (from parent to child) until text nodes are encountered and send to the output stream
* <xsl:template>: intercepting the default behaviour

## Intercepting the default behaviour

The <xsl:template> element provides a specific behaviour for any node fulfilling the constraint in *@match*

<**xsl:template** **match**="entry"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**p**>  
  <**xsl:apply-templates**/>  
 </**p**>  
</**xsl:template**>

Important: dealing with the namespace of the input document. Two possibilities:

* declaring the namespace with a prefix and using the prefix for all TEI elements:

<**xsl:stylesheet** **version**="2.0"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**xsl:output** **method**="html" **encoding**="UTF-8"/>  
 <**xsl:template** **match**="tei:entry">  
  <**p**>  
   <**xsl:apply-templates**/>  
  </**p**>  
 </**xsl:template**>  
</**xsl:stylesheet**>

* declaring a default namespace for XPath constructs by means of *@xpath-default-namespace*:

<**xsl:stylesheet** **version**="2.0"  
 **xpath-default-namespace**="http://www.tei-c.org/ns/1.0"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**xsl:output** **method**="html" **encoding**="UTF-8"/>  
 <**xsl:template** **match**="entry">  
  <**p**>  
   <**xsl:apply-templates**/>  
  </**p**>  
 </**xsl:template**>  
</**xsl:stylesheet**>

We adopt the latter in the rest of the tutorial.

Playing with further presentational aspects:

* Headwords in bold:

<**xsl:template** **match**="orth"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**b**>  
  <**xsl:apply-templates**/>  
 </**b**>  
</**xsl:template**>

* Examples in italics (Note: use of constraints in XPath):

<**xsl:template** **match**="cit[@type='example']/quote"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**i**>  
  <**xsl:apply-templates**/>  
 </**i**>  
</**xsl:template**>

* Putting parentheses around pronunciations:

<**xsl:template** **match**="pron"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**xsl:text**>[</**xsl:text**>  
 <**xsl:apply-templates**/>  
 <**xsl:text**>]</**xsl:text**>  
</**xsl:template**>

* Variant: dealing with cases where the pronunciation is already in square brackets:

<**xsl:template** **match**="pron[not(starts-with(normalize-space(.), '[') and ends-with(normalize-space(.), ']'))]"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**xsl:text**>[</**xsl:text**>  
 <**xsl:apply-templates**/>  
 <**xsl:text**>]</**xsl:text**>  
</**xsl:template**>

* + Logical operators: and and not
  + Removing leading and trailing whitespaces with normalize-space()
  + String functions in XSLT: start-with() and ends-with()

Dealing with senses: presenting sense as a new line, numbering senses when they do not have a *@n* attribute.

<**xsl:template** **match**="sense"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**br**/>  
 <**xsl:choose**>  
  <**xsl:when** **test**="@n">  
   <**xsl:value-of** **select**="@n"/>  
  </**xsl:when**>  
  <**xsl:otherwise**>  
   <**xsl:value-of** **select**="count(preceding-sibling::sense)+1"/>  
  </**xsl:otherwise**>  
 </**xsl:choose**>  
 <**xsl:text**>-</**xsl:text**>  
 <**xsl:apply-templates**/>  
</**xsl:template**>

Main features:

* Using axes: preceding-sibling::\*
* Arithmetic functions: count(), +
* Multiple tests using <xsl:choose> (with <xsl:when> and <xsl:otherwise>)

## Intercepting the root element

Setting up the global structure of the output document and selecting the objects to visualise:

<**xsl:template** **match**="/"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**html**>  
  <**body**>  
   <**xsl:apply-templates**/>  
  </**body**>  
 </**html**>  
</**xsl:template**>

In real life: this is usually the first template to implement.

## First notions of cherry picking

Showing the <body> only:

<**xsl:apply-templates** **select**="descendant::body"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>

Building up a simple HTML head:

<**head**  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**title**>  
  <**xsl:apply-templates** **select**="/TEI/teiHeader/fileDesc/titleStmt/title"/>  
 </**title**>  
</**head**>

Using oXygen to get the appropriate XPath expression.

# Sorting objects by means of <xsl:sort>

Using <xsl:sort> requires two main elements:

* A context that selects a group of nodes to be sorted (e.g. <xsl:apply-templates>)
* A sort key that can be extracted for each node of the group

Sorting all orthographic forms as a header to the output document (e.g. to carry out a diagnosis).

<**xsl:for-each** **select**="descendant::entry"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
 <**xsl:sort** **select**="form[1]/orth[1]"  
  **lang**="fr"/>  
 <**xsl:value-of** **select**="form[1]/orth[1]"/>  
 <**xsl:text**> - </**xsl:text**>  
</**xsl:for-each**>

# Dictionary check-ups

## Entries without forms

<**xsl:apply-templates** **select**="descendant::entry[not(form)]"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>

XPath and XSLT mechanisms in use:

* Complex XML tree traversal with <xsl:apply-templates>
* Testing the existence of a given child element (Does <entry> contain <form>?)
* XPtath function: not()

## Definition not embedded in a sense

The TEI lex 0 initiative recommends that all definitions (<def>) are actually part of an encompassing <sense>

<**xsl:apply-templates** **select**="descendant::def[not(ancestor::sense)]/ancestor::entry"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>

## Identifying duplicate entries

Looking for all the different forms in the dictionary and checking those for which there exist more than one entry.

XPath and XSLT mechanisms in use:

* XPtah function: distinct-values()
* Creating variables with <xsl:variable>
* Iterator XSLT construct: <xsl:for-each>
* Arithmetic operator: count() and >
* Loosing the context node within an iteration (hence the theDictionary variable

<**xsl:variable** **name**="theDictionary"  
 **select**="/"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>  
<**xsl:for-each** **select**="distinct-values(descendant::entry/form/orth)">  
 <**xsl:variable** **name**="theForm" **select**="."/>  
 <**xsl:if** **test**="count($theDictionary/descendant::entry[form/orth = $theForm])>1">  
  <**xsl:value-of** **select**="$theForm"/>  
  <**xsl:text**> - </**xsl:text**>  
 </**xsl:if**>  
</**xsl:for-each**>

## Different words used in definitions

XPath and XSLT mechanisms in use:

* XPtath iterator: for X in Y return Z with local variables
* XPath tokenise()
* Practical *@separator* attribute on <xsl:value-of>

<**xsl:variable** **name**="wordsInDef"  
 **select**="distinct-values(for $i in descendant::def return tokenize($i,' '))"  
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>  
<**xsl:value-of** **select**="$wordsInDef"  
 **separator**="; "/>

# Complementary remarks

* Favour a highly modular XSLT programming style to ensure conciseness and ease of maintenance.
* Therefore, always prefer <xsl:apply templates> to <xsl:for-each> when applicable
* When a stylesheet is not working, the main issue is usually a wrongly declared namespace. Pay attention to this when implementing an XML2XML scenario
* Take a regular look at the XPath and XSLT quick references ([Quick Reference Cards from Mulberry Technologies, Inc.](http://www.mulberrytech.com/quickref/)): be curious to discover new functions, at least to know they exist.