Intermediate XSLT and XPath

Code: xml-xpath

Author and version

- Daniel K. Schneider
- Email: Daniel.Schneider@tecfa.unige.ch

- Version: 1.0 (modified 21/4/09 by DKS)

Prerequisites

- Editing XML (being able to use a simple DTD)
- Introductory XSLT (xsl:template, xsl:apply-templates and xsl:value-of)
- Know about the role of XPath with respect to XSLT

Availability

http://tecfa.unige.ch/guides/te/files/xml-xpath.pdf

SOME RIGHTS RESERVED

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Objectives

- Understand XPath expressions
- Learn some XSLT programming constructions (conditions and loops)
- Being able to cope with most XML to HTML transformations

Warning

- XSLT is a rather complex language
- I believe that one could distinguish four levels of difficulty:
  - Introductory (level 1)
  - These slides concerns Level 2 XSLT, i.e. XPath and XSLT conditional expressions, loops, etc.
  - Level 3 is advanced XPath expressions and more exotic XSLT instructions
  - Level 4 is functional programming with templates

Disclaimer

- There may be typos (sorry) and mistakes (sorry again)
- Please also consult a textbook !

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Contents

1. Introduction to XML Path Language 5
1.1 Definition and history 5
1.2 XSLT and XPath 6
2. The XPath Syntax and the document model 7
2.1 Xpath Syntax 7
2.2 The formal specification of an XML Path 8
2.3 The document model of XPath 9
2.4 Element Location Paths 10
Example 2-1:Extracting titles from an XML file 11
2.5 Attribute Location Paths 14
Example 2-2:Extract a list and also insert an html img link from an attribute 15
2.6 Location wildcards 18
2.7 XPaths with predicates 19
Example 2-3:Retrieve selected elements 21
2.8 XPath functions 23
Example 2-4:Computation of an average 25
Example 2-5:Find first names containing 'nat' 27
2.9 Union of XPaths 28
2.10 List of commonly used XPath expressions 29
3. XSLT reminder 30
Example 3-1:Dealing with pictures 30
4. Multiple templates for one element 32
5. Conditional XSLT and whitespaces 35
5.1 Simple if 35
Example 5-1:Display lists, the last element differently with xsl:if 35
5.2 If-then-else 38
Example 5-2:Animal colors with xsl:choose 38
6. Looping 40
Example 6-1:Translating database query output into an HTML table 40

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

6.1 Looping with a sort **42**
Example 6-2:Sort a participants list according to qualification 42
7. Moving on 44
Example 7-1:Display ingredients with increasing fonts 44
8. Next steps 46
8.1 Reading **46**
8.2 Next modules **46**
9. Homework: mini-project 5 47
9.1 Task **47**
9.2 Approximate evaluation grid **48**

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

1. Introduction to XML Path Language

1.1 Definition and history

- XPath is a language for addressing parts of an XML document
- In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans.
- XPath uses a compact non-XML syntax (to facilitate use of XPath within URIs and XML attribute values).
- XPath gets its name from its use of a path notation for navigating through the hierarchical structure of an XML document (similar to the Unix/Wi/Internet model of path names)

History

- XPath was defined at the same time as XSLT (nov 1999)
- Initally, it was developped to support XSLT and XPointer (XML Pointer Language used for XLink, XInclude, etc.)

Specification

url: **XPath 1.0** **http://www.w3.org/TR/xpath** (nov 1999)

- Used by XSLT 1.0

url: **XPath 2.0** **http://www.w3.org/TR/xpath20/** **(Jan 2007)**

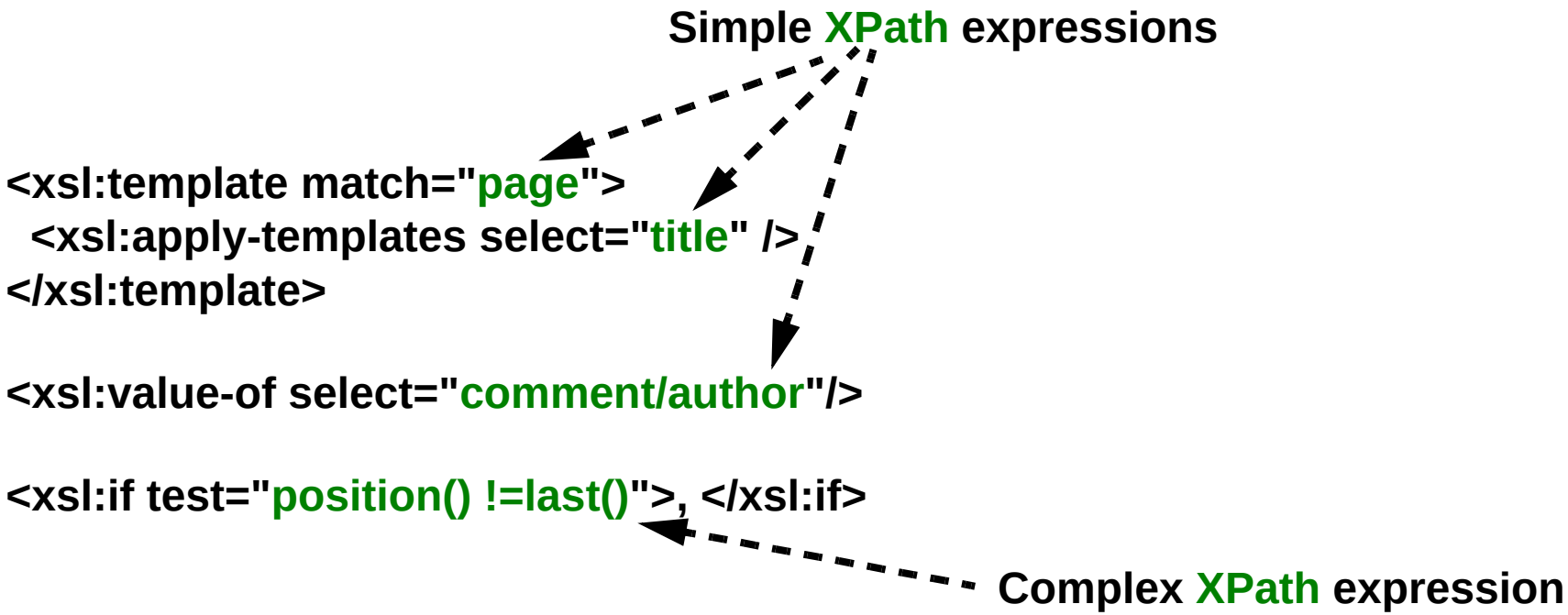
url: **XPath 2.0 Functions and Operators** **http://www.w3.org/TR/xquery-operators/**

- XPath 2.0 is a superset of XPath 1.0
- Used by XSLT 2.0 and XQuery ... and other specifications

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

1.2 XSLT and XPath

- Each time a given XSLT instruction needs to address (refer to) parts of an XML document we use XPath expressions.
- XPath expressions also can contain functions, simple math and boolean expressions
- Typically, XPath expressions are used in **`match`**, **`select`** and **`test`** attributes



© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2. The XPath Syntax and the document model

2.1 Xpath Syntax

Primary (relatively simple) XPath expressions:

Location Path [tests]

Axis / Node Test [Zero or more predicates]
(where to look for) / (tag/attribute name) [... further tests ...]

//child::solutions/child::item[attribute::val="low"]
//solutions/item[@val="low"]

Result of an Xpath

- Can be various data structures, e.g. sets of nodes, a single node, a number, etc ....

There are two notations for location paths

1. abbreviated (less available options)
   - e.g. **`para`** is identical to **`child::para`**
2. unabbreviated (not presented in these slides !!)
   - e.g. "**`child::para`**" would define the **`para`** element children of the current context node.

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.2 The formal specification of an XML Path

- is very complex, i.e. has about 39 clauses and is very difficult to understand
- Some expressions shown here are beyound the scope of this class, don't panic !

Here are the first few clauses expressed in EBNF syntax:

```
[1] LocationPath::= RelativeLocationPath | AbsoluteLocationPath
[2] AbsoluteLocationPath ::= '//' RelativeLocationPath?
                           | AbbreviatedAbsoluteLocationPath
[3] RelativeLocationPath ::= Step | RelativeLocationPath '//' Step
                           | AbbreviatedRelativeLocationPath
[4] Step ::= AxisSpecifier NodeTestPredicate* | AbbreviatedStep
[5] AxisSpecifier ::= AxisName '::' | AbbreviatedAxisSpecifier
[6] AxisName ::= 'ancestor' | 'ancestor-or-self' | 'attribute' | 'child' |
'descendant' | 'descendant-or-self' | 'following' | 'following-sibling' |
'namespace' | 'parent' | 'preceding' | 'preceding-sibling' | 'self'
[7] NodeTest ::= NameTest | NodeType '(' ')'
                          | 'processing-instruction' '(' Literal ')'
[8] Predicate ::= '[' PredicateExpr ']'
[9]PredicateExpr ::= Expr
etc.
```

Ignore this page ....

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.3 The document model of XPath

- XPath sees an XML document as a tree structure
- Each information (XML elements, attributes, text, etc.) is called a **node**
  - this is fairly similar to the W3C DOM model an XML or XSLT processor would use

Nodes that XPath can see:

- root node
  - ATTENTION: The root is not necessarily the XML root element. E.g. processing instructions like a stylesheet declaration are also nodes.
- Elements and attributes
- Special nodes like comments, processing instructions, namespace declarations.

Nodes XPath can’t see:

- XPath looks at the final document, therefore can’t see entities and document type declarations....

The XML context

- What a given XPath expression means, is always defined by a given XML context, i.e. the current node in the XML tree

... Advance warning:

The rest of this chapter will be quite boring (and it only covers XPath essentials ....)

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.4 Element Location Paths

- We present a few expressions for locating nodes
- This page is not complete and uses abbreviated syntax

Document root node: returns the document root (which is not necessarily the XML root!)

```
Syntax: /
```

Direct child element:

```
Syntax: XML_element_name
```

Direct child of the root node:

```
Syntax: /XML_element_name
```

Child of a child:

```
Syntax: XML_element_name/XML_element_name
```

Descendant of the root:

```
Syntax: //XML_element_name
```

Descendant of a node:

```
Syntax: XML_element_name//XML_element_name
```

Parent of a node:

```
Syntax: ../
```

Un far cousin of a node:

```
Syntax: ../../XML_element_name/XML_element_name/XML_element_name
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Example 2-1: Extracting titles from an XML file

An XML document (xpath-jungle.xml is used throughout the XPath chapter)

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle.xml

- We only show part of the document

```
!-- XML fragment -->
<project>
 <title>The Xpath project</title>
 ......
 <problems>
  <problem>
   <title>Initial problem</title>
    <description>We have to learn something about Location Path</description>
    <difficulty level="5">This problem should not be too hard</difficulty>
  </problem>
  <problem>
    <title>Next problem</title>
    <description>We have to learn something about predicates</description>
    <difficulty level="6">This problem is a bit more difficult</difficulty>
  </problem>
 </problems>
</project>
```

Task

- We would like to get a simple list of problem titles

  (XSLT Templates on next slide)

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

(1) XSLT template for project XML root element (file: xpath-jungle-1.xsl)

```
<xsl:template match="/project">
   <html>
     <body bgcolor="#FFFFFF">
       <h1><xsl:value-of select="title" /></h1>
       Here are the titles of our problems:
       <ul>
       <xsl:apply-templates select="problems/problem" />
      </ul>
     </body>
   </html>
 </xsl:template>
```

- The XPath of the "match" means: applies to **project** element node, descendant of root node
- Execution **context** of this template is therefore the element "**project**"
- xsl:apply-templates will select a rule for descendant "**problem**".

(2) XSLT template for the problem element

```
<xsl:template match="problems/problem">
  <li><xsl:value-of select="title" /></li>
</xsl:template>
```

- This second rule will be triggered by the first rule, because problems/problem is indeed a descendant of the project element

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

(3) Result HTML

```
<html>
   <body bgcolor="#FFFFFF">
      <h1>The Xpath project</h1>
      Here are the titles of our problems:
      <ul>
         <li>Initial problem</li>
         <li>Next problem</li>
      </ul>
   </body>
</html>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.5 Attribute Location Paths

Find an attribute of a child element of the current context

```
Syntax: @attribute_name
```

Example:

```
@val
```

Find attributes of an element in a longer location path starting from root

```
Syntax: /element_name/element_name/@attribute_name
```

Example:

```
/project/problems/solutions/item/@val
```

Find attributes in the whole document

```
Syntax: //@attribute_name
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Example 2-2: Extract a list and also insert an html img link from an attribute

url: **http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-2.xsl**

url: **http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-2.xml**

XML fragment

```
<participants>
  <participant>
    <FirstName>Daniel</FirstName>
    <qualification>8</qualification>
    <description>Daniel will be the tutor</description>
    <FoodPref picture="dolores_001.jpg">Sea Food</FoodPref>
  </participant>
  <participant>
    <FirstName>Jonathan</FirstName>
    <qualification>5</qualification>
    <FoodPref picture="dolores_002.jpg">Asian</FoodPref>
  </participant>
  <participant>
   <FirstName>Bernadette</FirstName>
   <qualification>8</qualification>
    <description>Bernadette is an arts major</description>
  </participant>
 .......
```

Task

- Display a list of First Names plus their food preferences

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

XSLT (File xpath-jungle-2.xsl)

- The first rule will just select all participants and create the list container (ul)

```
<xsl:template match="/">
   <html>
     <body bgcolor="#FFFFFF">
       <h1>What do we know about our participants ?</h1>
       Here are some food preferences:
       <ul>
       <xsl:apply-templates select=".//participant" />
       </ul>
     </body>
   </html>
 </xsl:template>
```

- The second rule will display names of participants and launch a template for FoodPref
- Note: Not all participants have a FoodPref element. If it is absent it will just be ignored.

```
<xsl:template match="participant">
  <li><xsl:value-of select="FirstName"/>
  <xsl:apply-templates select="FoodPref"/>
  </li>
</xsl:template>
```

- This rule displays the text (contents) of FoodPref and then makes an HTML img tag

```
<xsl:template match="FoodPref">
  prefers <xsl:value-of select="."/>.
  <img src="{@picture}"/> <br clear="all"/>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Parts of the result:

```
<h1>What do we know about our participants ?</h1>
    Here are some food preferences:
    <ul>
       <li>Daniel prefers Sea Food.
          <img src="dolores_001.jpg"><br clear="all"></li>
       <li>Jonathan
          prefers Asian.
          <img src="dolores_002.jpg"><br clear="all"></li>
       <li>Bernadette</li>
       <li>Nathalie</li>
    </ul>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.6 Location wildcards

- Sometimes (but not often!), it is useful to work with wildcards
- You have to understand that only **one** rule will be applied/element. Rules with wildcards have less priority and this is why "your rules" are applied before the system default rules...

Find all child nodes of type XML element

```
Syntax: *
```

Find all child nodes (including comments, etc.)

```
Syntax: node()
```

Find all element attributes

```
Syntax: @*
```

Find all text nodes

```
Syntax: text()
```

FYI: The system built-in (default) rules rely on wildcards

This rule applies to the document root and all other elements (see 2.9 “Union of XPaths” [28])

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

Text and attribute values are just copied (see

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


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.7 XPaths with predicates

- A predicate is an ***expression that can be true or false***
- A predicate is appended within [...] to a given location path and it will ***refine results***
- More than one predicate can be appended to and within (!) a location path
- Expressions can contain mathematical or boolean operators

Find element number N in a list

```
Syntax: XML_element_name [ N ]
/project/participants/participant[2]
/project/participants/participant[2]/FirstName
```

Find elements that have a given attribute

```
Syntax: XML_element_name [ @attribute_name ]
```

Find elements that have a given element as child

```
Syntax: XML_element_name [ XML_element_name ]
//participant[FoodPref]
```

Mathematical expressions

- Use the standard operators, except **`div`** instead of `/`")

  ```
  Syntax: + - * div mod
  ```

- mod is interesting if you want to display a long list in table format

  ```
  5 mod 2 returns 1, "7 mod 2" and "3 mod 2" too
  ```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Boolean operators (comparison, and, or)

- List of operators (according to precedence)

```
<=, <, >=, >
=, !=
and
or
```

Examples

- Return all exercise titles with a note bigger than 5.

```
//exercise[note>5]/title
```

- Find elements that have a given attribute with a given value

```
Syntax: XML_element_name [ @attribute_name = 'value']
```

```
//solutions/item[@val="low"]
```

```
Syntax:
```

- Example XSLT template that will match all item elements with val="low".

```
<xsl:template match="//item[@val='low']">
  <xsl:value-of select="." />
</xsl:template>
```

Note: Usually expression also contain functions (see 2.8 "XPath functions" [23])

- Return last five elements of a list

```
author [(last() - 4) <= position()) and (position() <= last())]
```

- Return all **`Participant`** nodes with contents of **`FirstName`** bigger than 7 characters:

```
"//Participant[string-length(FirstName)>=8]"
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Example 2-3: Retrieve selected elements

The XSLT stylesheet (file xpath-jungle-3.xsl)

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-3.xsl

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-3.xml

```
<xsl:template match="/">
  <html>
    <body bgcolor="#FFFFFF">
      <h1>Retrieve selected elements</h1>
      Here is the name of participant two:
      <ul><li><xsl:value-of select=".//participant[2]/FirstName"/></li></ul>
      Here are all participant's firstnames that have a food preference:
      <ul><xsl:apply-templates select=".//participant[FoodPref]"/></ul>
      Here are all items that have a value of "high"
      <ul><xsl:apply-templates select=".//item[@val='high']"/></ul>
    </body>
  </html>
</xsl:template>

<xsl:template match="participant">
  <li><xsl:value-of select="FirstName"/></li>
</xsl:template>

<xsl:template match="item">
  <li><xsl:value-of select="."/></li>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

HTML result

```
<html>
   <body bgcolor="#FFFFFF">
      <h1>Retrieve selected elements</h1>
      Here is the name of participant two:
      <ul>
         <li>Jonathan</li>
      </ul>
      Here are all participant's firstnames that have a food preference:

      <ul>
         <li>Daniel</li>
         <li>Jonathan</li>
      </ul>
      Here are all items that have a value of "high"
      <ul>
         <li>Register for a XSLT course and do exercices</li>
         <li>Register for a XPath course and do exercices</li>
      </ul>
   </body>
</html>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.8 XPath functions

- XPath defines a certain number of functions
  - You can recognize a function because it has "()".
- Functions are programming constructs that will return various kinds of informations, e.g.
  - true / false
  - a number
  - a string
  - a list of nodes
- It is not obvious to understand these ....
- There are restrictions on how you can use functions (stick to examples or the reference)

last()

last() gives the number or nodes within a context

position()

position() returns the position of an element with respect to other children for a parent

count(node-set)

count gives the number of nodes in a node set (usually found with an XPath).

starts-with(string, string)

returns TRUE if the second string is part of the first and starts off the first
//Participant[starts-with(Firstname,'Berna')]"

contains(string, string)

returns TRUE if the second string is part of the first
//Participant[contains(FirstName,'nat')]

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

string-length(string)

returns the length of a string

number(string)

transforms a string into a number

sum(node-set)

computes the sum of a given set of nodes.
If necessary, does string conversion with number()

round(number)

round a number, e.g. 1.4 becomes 1 and 1.7 becomes 2

translate(string1, string2, string3)

translates string1 by substituting string2 elements with string3 elements

(See next slides for examples ....)

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Example 2-4: Computation of an average

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-4.xsl

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-4.xml

- We would like to compute the average of participant's qualifications

```
<participant><FirstName>Daniel</FirstName>
              <qualification>8</qualification>   </participant>
```

The XSLT stylesheet (file xpath-jungle-4.xsl**)**

- We compute the sum of a node-set and then divide by the number of nodes

```
<xsl:template match="/">
    <html>
      <body bgcolor="#FFFFFF">
        <h1>Qualification level of participants</h1>
        Average is
        <xsl:value-of select="sum(.//participant/qualification) div
                              count(.//participant/qualification)"/>
      </body>
    </html>
  </xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

HTML result

```
<html>
   <body bgcolor="#FFFFFF">
      <h1>Qualification level of participants</h1>
      Average is
      5.75
   </body>
</html>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Example 2-5: Find first names containing 'nat'

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-5.xsl

url: http://tecfa.unige.ch/guides/xml/examples/xpath/xpath-jungle-5.xml

The XSLT stylesheet (file xpath-jungle-5.xsl)

```
<xsl:template match="/">
 <html>
  <body bgcolor="#FFFFFF">
  <h1>Do we have a "nat" ?</h1>
  First Names that contain "nat":
  <ul>
  <xsl:apply-templates select=".//participant[contains(FirstName,'nat')]"/>
  </ul>
  First Names that contain "nat" and "Nat":
  <ul>
   <xsl:apply-templates select=".//participant[contains
                          (translate(FirstName,'N','n'),'nat')]"/>
   </ul>
  </body>
 </html>
</xsl:template>

<xsl:template match="participant">
  <li><xsl:value-of select="FirstName"/></li>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.9 Union of XPaths

- Union Xpaths combine more than one XPath (and all the resulting nodes are returned).
- A typical example is the default rule which means that the template matches either the root element (i.e. "/" or just any element),

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

- Often this is used to simplify apply-templates or even templates themselves. E.g. the following rules applies to both "description" and "para" elements.

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


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

2.10List of commonly used XPath expressions

Syntax element	(Type of path)	Example path	Example matches
name	child element name	project	***<project> ..... </project>***
/	child / child	project/title	<project>***<title> ... </title>***
		/	(root element)
//	descendant	project//title	<project><problem>***<title>....</title>***
		//title	<root>...***<title>..</title>*** (any place)
*****	"wildcard"	*/title	<bla>***<title>..</title> and*** <bli>***<title>...</title>***
|	"or operator	title|head	<title>...</title> ***or*** <head> ...</head>
		|/|@	All elements: root, children and attributes
.	current element	.	
../	parent element	../problem	<project>
@attr	attribute name	@id	<xyz id="test">...</xyz>
element/@attr	attribute of child	project/@id	***<project id="test" ...> ... </project>***
@attr='value'	value of attribute	list[@type='ol']	***<list type="ol"> ..... </list>***
position()	position of element in parent	position()	
last()	number of elements within a context	last() position()!=last()	

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

3. XSLT reminder

- In most situations, writing simple XSLT rules will do
- Do not attempt to use looping constructs etc. when you don't have to
- There is no special "magic" for dealing with images, links, stylesheets etc. Simply:
  - look at your XML
  - figure out how to translate into equivalent HTML (or whatever you are translating into)

Example 3-1: Dealing with pictures

XML file with picture file names (file images.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet href="images.xsl" type="text/xsl"?>
<page>
 <title>Hello Here are my images</title>
 <list>
   <image>dolores_001.jpg</image>
   <image>dolores_002.jpg</image>
   <image>dolores_002.jpg</image>
   <image2>scrolls.jpg </image2>
   <image2>scrolls.jpg </image2>
   <image3 source="dolores_002.jpg">Recipe image</image3>
 </list>
 <comment>Written by DKS.</comment>
</page>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

XSLT stylesheet (file images.xsl)

```
<xsl:template match="list">
   Apply templates for image elements:
   <xsl:apply-templates select="image"/>
   This will only insert the first "image2" element contents it finds:
   <p> <img src="{image2}"/> </p>
   And another template for a tag image3 element (with an attribute)
   <xsl:apply-templates select="image3"/>
</xsl:template>
```

- This rule will insert the content of the image element into the value of src="".

```
<xsl:template match="image">
  <p> <img src="{.}"/> </p>
</xsl:template>
```

- This rule will insert the value of the source attribute into the value of src and also insert the contents of the the image3 element.

```
<xsl:template match="image3">
  <p> <img src="{@source}"/><xsl:value-of select="."/> </p>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

4. Multiple templates for one element

- XSLT allows to define multiple rules for the same XPath by using the "mode" attribute
- This construct is frequently used to produce table of contents

Parts of the XML file (file douglas-recipes.xml)

url: http://tecfa.unige.ch/guides/xml/examples/xpath/douglas-recipes.xml

```
<list>
  <recipe id="r1">
    <recipe_name>TACOS</recipe_name>
    <meal>Dinner</meal>
    <ingredients>
      <item>3 taco shells</item>
      <item>1 pkg hamburger 100g</item>
      ........
    </ingredients>

    <directions>
      <bullet>Oven on at 180 degrees.</bullet>
      ......
      <bullet>Cut up lettuce.</bullet>
    </directions>
  </recipe>

  <recipe id="r2">
 .... </list>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

XSLT fragment (file douglas-recipes.xsl)

url: http://tecfa.unige.ch/guides/xml/examples/xpath/douglas-recipes.xsl

```
<xsl:template match="list">
  <html xmlns="http://www.w3.org/1999/xhtml">
   <head> <title>Recipes</title> </head>
   <body bgcolor="#ffffff">
    <h1>Recipes</h1>
    <h2>Contents</h2>
    <p> <xsl:apply-templates select="recipe" mode="toc"/> </p>
    <xsl:apply-templates select="recipe"/>
   </body> </html>
</xsl:template>
```

- We define 2 rules for "recipe": One will have a mode="toc"
- The next rule will create the table of contents.
  - It will use the recipe_name as link
  - It will use the id attribute to create an internal href link

```
<xsl:template match="recipe" mode="toc">
  <p><a href="#{@id}"><xsl:value-of select="recipe_name"/></a></p>
</xsl:template>
```

- This rule is almost "normal", except that we insert a name anchor

```
<xsl:template match="recipe">
  <h1><a name="{@id}"><xsl:value-of select="recipe_name"/></a> </h1>
  <xsl:apply-templates select="meal"/>
  <xsl:apply-templates select="ingredients"/>
  <xsl:apply-templates select="directions"/> </xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

HTML result

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>Recipes</title>
   </head>
   <body bgcolor="#ffffff">
      <h1>Recipes</h1>
      <h2>Contents</h2>
      <p>
         <p> <a href="#r1">TACOS</a> </p>
         <p> <a href="#r2">VOL AU VENT (one person)</a> </p>
      </p>
      <hr/>
      <h1>
         <a name="r1">TACOS</a>
      </h1>
      <p> Type: Dinner</p>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

5. Conditional XSLT and whitespaces

- Although many conditionals can be expressed with XPath and according selection rules, XSLT provides 2 typical programming constructs to handle "if" and "if-then-else" situations.
- When you use XPath functions like position(), you have to be very careful about whitespaces

5.1 Simple if

Syntax for xsl:if:

```
<xsl:if test = "boolean_expression">
 <!-- Content or xsl instructions here -->
</xsl:if>
```

Example 5-1: Display lists, the last element differently with xsl:if

- Warning: This only works in standards compliant browsers if you either eliminate whitespaces from your XML file or if you add the following XSL instruction at the beginning:

  `Syntax: <xsl:strip-space elements="element_name element_name ...."/>`

  `<xsl:strip-space elements="ingredients"/>`

XML (file http://tecfa.unige.ch/guides/xml/examples/xpath/ingredient-list.xml)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="ingredient-list.xsl" type="text/xsl"?>
<ingredients>
  <item>3 taco shells</item>
  <item>1 pkg hamburger 100g</item>
  <item>Taco mix</item><item>Lettuce</item></ingredients>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

XSL (file http://tecfa.unige.ch/guides/xml/examples/xpath/ingredient-list.xsl)

```
 <xsl:template match="/ingredients">
  <html xmlns="http://www.w3.org/1999/xhtml">
   <head> <title>Ingrediant list</title> </head>
   <body bgcolor="#ffffff">
     Here is a list of recipe ingrediants:
    <p> <xsl:apply-templates select="item"/>. </p>
     Here is a list of recipe ingrediants, this time numbered:
    <p> <xsl:apply-templates select="item" mode="numb"/>. </p>
   </body>
  </html>
</xsl:template>

<xsl:template match="item">
  <xsl:value-of select="."/>
  <xsl:if test="position()!=last()"> <xsl:text>, </xsl:text> </xsl:if>
</xsl:template>

<xsl:template match="item" mode="numb">
  (<xsl:value-of select="position()"/>)
  <xsl:value-of select="."/>
  <xsl:if test="position()!=last()"> <xsl:text>, </xsl:text> </xsl:if>
</xsl:template>

</xsl:stylesheet>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

HTML results

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>Ingrediant list</title>
   </head>
   <body bgcolor="#ffffff">
     Here is a list of recipe ingrediants:
    <p>3 taco shells, 1 pkg hamburger 100g, Taco mix, Lettuce. </p>
     Here is a list of recipe ingrediants, this time numbered:
    <p>
  (1) 3 taco shells,
  (2) 1 pkg hamburger 100g,
  (3) Taco mix,
  (4) Lettuce. </p>
    </body>
</html>
```

Same logic would apply e.g. to building references

```
<xsl:template match="reference">
  <xsl:apply-templates
       select="author|ref|title|edition|publisher|pubPlace|publicationYear"/>.
 </xsl:template>

 <xsl:template match="author|edition|publisher|pubPlace|publicationYear">
  <xsl:apply-templates/>
  <xsl:if test="position() !=last()">, </xsl:if>
 </xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

5.2 If-then-else

Syntax:

```
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>

<xsl:when test = "boolean-expression">
  <!-- Content: template -->
</xsl:when>

<xsl:otherwise>
  <!-- Content: template -->
</xsl:otherwise>
```

Example 5-2: Animal colors with xsl:choose

XML (file http://tecfa.unige.ch/guides/xml/examples/xpath/animals.xml)

```
<example>
 <title>Animals with colors</title>
 <list>
  <animal color="black">Panther</animal>
  <animal color="complicated with spots">Panther</animal>
  <animal color="white">Polar bear</animal>
  <animal color="green">Frog</animal>
  <animal>Cow</animal>
 </list>
</example>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

XSLT template for animal (file http://tecfa.unige.ch/guides/xml/examples/xpath/animals.xsl)

```
<xsl:template match="animal">
<li>
  <xsl:choose>
    <xsl:when test="@color='black'">
    <p style="color:black;font-weight:bold;"> <xsl:value-of select="."/> </p>
    </xsl:when>

    <xsl:when test="@color='green'">
    <p style="color:green;font-weight:bold;"> <xsl:value-of select="."/> </p>
    </xsl:when>

    <xsl:otherwise>
     <p style="color:cyan"> <xsl:value-of select="."/> </p>
    </xsl:otherwise>
  </xsl:choose>
</li>
</xsl:template>
```

HTML (some)

```
<li><p style="color:black;font-weight:bold;">Panther</p> </li>
<li><p style="color:cyan">Panther</p></li>
<li><p style="color:cyan">Polar bear</p></li>
<li><p style="color:green;font-weight:bold;">Frog</p></li>
<li><p style="color:cyan">Cow</p></li>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

6. Looping

- Most of the time you do "looping" with templates (as in all previous examples)
- There is a xsl:for-each looping construct. It is useful to write more compact code, to use it with sorting and functional programming constructs (not explained here).

  Syntax: **xsl:for-each select="XPath"**

Example 6-1: Translating database query output into an HTML table

XML (file http://tecfa.unige.ch/guides/xml/examples/xpath/rowset.xml)

- Typical database query output may look like this
- E.g. survey data will present as a series of rows of the same length
- We would like to translate each ROW into a table row and each child as a table cell

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="rowset.xsl" type="text/xsl"?>
<page>
  <ROWSET>
   <ROW><id>1</id><login>test</login><fullname>Joe Test </fullname>
    <food>3</food><work>4</work><love>5</love><leisure>2</leisure></ROW>
   <ROW><id>2</id><login>test2</login><fullname>Janine Test </fullname>
    <food>3</food><work>4</work><love>6</love><leisure>2</leisure></ROW>
  .......
  </ROWSET>
</page>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

XSLT (file http://tecfa.unige.ch/guides/xml/examples/xpath/rowset.xsl)

```
<xsl:template match="ROWSET">
    <table border="2" cellspacing="1" cellpadding="6">
    <tr><th>id</th>
        <th>Login</th>
        <th>Full Name</th>
        <th>food</th><th>work</th><th>love</th><th>leisure</th>
     </tr>
     <xsl:for-each select="ROW">
      <tr>
      <td><xsl:value-of select="id"/></td>
      <td><xsl:value-of select="login"/></td>
      <td><xsl:value-of select="fullname"/></td>
      <td><xsl:value-of select="food"/></td>
      <td><xsl:value-of select="work"/></td>
      <td><xsl:value-of select="love"/></td>
      <td><xsl:value-of select="leisure"/></td>
      </tr>
     </xsl:for-each>
    </table>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

6.1 Looping with a sort

- xsl:sort can sort a list according to several criteria, can be used more than once

```
<xsl:sort
  select = "Xpath"
  data-type = { "text" | "number" }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } />
```

Example 6-2: Sort a participants list according to qualification

XML fragment (file: http://tecfa.unige.ch/guides/xml/examples/xpath/participants.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet href="participants.xsl" type="text/xsl"?>
 <participants>
  <participant>
    <FirstName>Daniel</FirstName>
    <qualification>8</qualification>
    <description>Daniel will be the tutor</description>
    <FoodPref picture="dolores_001.jpg">Sea Food</FoodPref>
  </participant>
  <participant>
    <FirstName>Jonathan</FirstName>
    <qualification>5</qualification>
    <FoodPref picture="dolores_002.jpg">Asian</FoodPref>
  ......
  </participant>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Part of XSLT (file: http://tecfa.unige.ch/guides/xml/examples/xpath/participants.xsl)

```
<xsl:template match="participants">
    <table border="2" cellspacing="1" cellpadding="6">
    <tr><th>Qualification</th>
        <th>First Name</th>
        <th>Description</th>
        <th>Food Picture</th>
     </tr>
     <xsl:for-each select="participant">
       <xsl:sort select="qualification"/>
      <tr>
      <td><xsl:value-of select="qualification"/></td>
      <td><xsl:value-of select="FirstName"/></td>
      <td><xsl:value-of select="description"/></td>
       <td><xsl:if test="FoodPref/@picture">
           <img src="{FoodPref/@picture}"/></xsl:if></td>
      </tr>
     </xsl:for-each>
    </table>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

7. Moving on

- XSLT is a full programming language, but further XSLT constructs are out of scope for this class.....
- Just for your information, I present a small example that demonstrates function calls.

Example 7-1: Display ingredients with increasing fonts

XML (file: http://tecfa.unige.ch/guides/xml/examples/xpath/ingredient-list2.xml)

```
<ingredients>  <item>3 taco shells</item><item>1 pkg hamburger 100g</item>
  <item>Taco mix</item> <item>50 g of cheese</item>  <item>1 tomato</item>
  <item>1 carot</item> <item>Lettuce</item></ingredients>
```

XSLT (file http://tecfa.unige.ch/guides/xml/examples/xpath/ingredient-list2.xsl)

```
<xsl:template match="/ingredients">
  <html>
  <head> <title>Ingrediant list</title> </head>
   <body bgcolor="#ffffff">
     Here is a list of recipe ingrediants:
     <xsl:for-each select="item">
       <xsl:call-template name="display_item">
         <xsl:with-param name="position" select="position()"/>
       </xsl:call-template>
     </xsl:for-each>
   </body>
  </html>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

```
<!-- you can change the value of these 2 parameters -->
 <xsl:param name="initial_size" select="5"/>
 <xsl:param name="multiplier" select="3"/>

<xsl:template name="display_item">
  <xsl:param name="position"/>
  <xsl:variable name="font_size"
                select="$initial_size + $position * $multiplier"/>
  <xsl:variable name="color">
    <xsl:choose>
      <xsl:when test="$position mod 3 = 0">
           <xsl:text>yellow</xsl:text></xsl:when>
      <xsl:when test="$position mod 3 = 1">
           <xsl:text>red</xsl:text>   </xsl:when>
      <xsl:when test="$position mod 3 = 2">
           <xsl:text>blue</xsl:text>  </xsl:when>
      <xsl:otherwise><xsl:text>green</xsl:text></xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <p style="color:{$color}; font-size: {$font_size}pt;">
     <xsl:value-of select="."/><br />
  [Pos = <xsl:value-of select="$position"/>,
   Multip. = <xsl:value-of select="$multiplier"/>,
   Font size = <xsl:value-of select="$font_size"/>] </p>
</xsl:template>
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

8. Next steps

8.1 Reading

Carey (pp. 297-317, 317-335)

Optional: 1 or 2 case problems

8.2 Next modules

- XML Schema

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

9. Homework: mini-project 5

Due: next Monday

9.1 Task

Create an XSLT transformation to (X)HTML.

- Create or reuse a DTD and a valid XML example file
- Translate this XML document with XSLT to a somewhat valid HTML or XHTML
- Write a report about the purpose and the architecture of the XSLT file.
- Make use of some more advanced XPath and XSLT constructs, e.g. filtering constructs or conditionals. Make sure to create something that is different from project 4.
- Bonus: Use CSS to style the HTML output
- Bonus: Produce really valid HTML or XHTML
- Take into account critique and self-critique for homework 4
- You may reuse materials from previous homework
  - I strongly suggest to base this project on homework 4 (XSLT)

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

9.2 Approximate evaluation grid

Minimal requirement:

Features	Expect a
Valid XML (DTD) and well-formed XSLT style sheet that does a translation	D
An XML contents that displays as (X)HTML in a web browser of your choice	C

You will get extra points for

Extra features	Extra points (+/- quality)
Inserted comments <!-- ... --> in the various files (XML, XSLT or CSS)	+
XML data organization that is appropriate for your domain	+
Complexity of style sheet (kinds of transformations)	+ ... ++
Ergonomic and nice presentation (style and function)	+ ... ++
Your result document is valid HTML or XHTML	+
A 1-2 page **user manual** that explains a user how to use your DTD and stylesheet	+ ... ++
A 1 page report that discusses your implementation	+ ... ++

- To get a B: Produce a useful (X)HTML document from valid XML
- To get an A:
  - Create a useful DTD, an ergonomic XSLT, a useful manual and a report
  - Alternatively: Create a really difficult XSLT and a good report

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

Submission format

- Electronic copies to be uploaded to the worlclassroom
  Please make sure to submit all elements. In addition, it is always good to use names like ex5_your_name.xml

```
your_name.xml
your_name.xsl
your_name.dtd
your_name.{doc|pdf|html} Chose the format you like (optional)
```


© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09

© Daniel. K. Schneider, Webster University Geneva - TECFA, University of Geneva 21/4/09