Theme 5: XPath

Functions for XSLT

Introduction

- XPath is a language used to **reference parts** of an XML document

- It’s seldom used by itself; mostly it’s used in conjunction with languages like XSLT and XQuery.

Introduction

- You know how to create and apply templates with XSLT.

- When you **create** a template...
- …a **pattern** shows what nodes the template applies to.

- When you **apply** a template…
- …an **expression** selects the node set to process.

Introduction

- **Expressions** are…
- …**logical arrangements** of language elements…
- …that are **interpreted** and **evaluated** by the XSLT processor…
- …to return a value

Introduction

- You write patterns and expressions using **XPath**.
- **XPath** = **X**ML **Path** Language.
- XPath is a language for **selecting** nodes and node sets.
- XPath is also used to further **process** node sets...
- ...to return **values** instead of nodes.

Introduction

XPath has **built-in functions** to…

- …do math…
- …process strings…
- …and test conditions in an XML doc.

- We will look at these functions in detail later.

XPath

- XPath is used to point into XML and select parts.
- XPath was designed to be embedded and used by other languages…
- …in particular by XSLT, XLink, and XPointer, and later by Xquery…

Locating Nodes

Locating Nodes

- XPath uses **location paths** to find a node or node set.

- **Reminder:** A **node** is an individual piece of the XML doc.

- A location path uses **relationships**…
- ...to describe the location of a node or node set...
- ...relative to a given node.

The XML Node Tree

- XPath considers the XML doc as a **node tree**.
  - **Reminder:** It is a hierarchical tree structure of nodes.

- Every node in the tree is in some way related to another.

- At the top of the tree is the **root** (or **document**) **node**.

The XML Node Tree

IMPORTANT:

- The **root node** represents the **document itself**…
- …and it is the **parent node** of the **root element node**.

- The root element node can have several **child nodes**.
- These child nodes can have their own child nodes, etc.
- Child nodes with the same parents are **sibling nodes**.

The XML Node Tree

- **Descendant nodes** are…
    - …a node’s child nodes…
    - …and its children’s child nodes, etc.

- **Ancestor nodes** are…
    - …a node’s parent node…
    - …and its grandparent nodes, etc.

The XML Node Tree

- You can access any nodes from any other nodes…
- ...if you know the relationship between the two.

- **There are two kinds of location paths:**
    - Relative location paths
    - Absolute location paths

Relative Location Paths

- A **relative location path** consists of…
  - …a sequence of **location steps**.
  - …separated by a forward slash (**/**).

```
step/step/step
```

- A **location step**…
  - …selects node(s) relative to the **current node**…
  - …then each node in **that** set…
  - …is the current node for the next step, etc.

Absolute Location Paths

- An **absolute location path** consists of…
  - …a forward slash (**/**)…
  - …optionally followed by a relative location path.

```
/step/step/step
```

- A forward slash by itself selects the XML’s **root node**.

Location Paths

- If the slash is followed by **a relative location path**…
- …the **current** node for the first step is the root node.

- Using absolute or relative depends on the circumstance.

- You'd usually want nodes relative to the current node…
- …so relative location paths are more common.

Determining the Current Node

- In XSLT, you'll often specify…
- …what node the processor should process next…
- …with respect to what is being processed now.

- The node currently being processed = the **current node**.

Determining the Current Node

- **How to determine the current node:**
  - By default, the current node is the one specified by…
  - …the current template's **match** attribute.

  - When an **xsl:apply-templates** executes, the current node…
  - …becomes the one matched by the applied template…

Determining the Current Node

- **How to determine the current node (continued):**
  - …and when the processor returns to the calling template…
  - …the current node reverts back to that template’s **match**.
  - When an **xsl:for-each** starts, the current node changes…
  - …to the one specified by its **select** attribute**.**

Determining the Current Node

- **How to determine the current node (continued):**
  - At the end of the **xsl:for-each**, the current node…
  - …reverts back to what it was before the instruction occurred.

- If the processor is processing nodes in a set, one by one…
- …the current node changes to each node, one by one.

Creating Absolute Location Paths

- Absolute location paths do not rely on the current node.

```
/root/container/.../parent/node
```

- If you only want the root node, stop after the first **/**.
- If you want the root element, replace **root** with its name.
- To get a child of root, replace **container** with its name.

Creating Absolute Location Paths

- To get a child, grandchild, etc. of **container**...
- ...continue down the hierarchy, child level by child level...
- ...using the child names and separating levels using **/**.

- Continue until you reach the **parent** of the node(s) you want (replace **parent** with the parent's name).

- Replace **node** with the name of the node(s) you want.

Creating Absolute Location Paths

- You can also use predicates on absolute paths.
- Other shortcuts (e.g. *****) also apply to absolute paths.
- Replaced the relative path to absolute…

```
names/name[@status=‘hidden']
```

```
/character/class/names/name[@status=‘hidden']
```

Creating Absolute Location Paths

- Now, each time the **names** template is applied...
- ...instead of selecting the **name** of the **current node**...
- ...it will select the **name** of the **first names** every time.

- In this case, that's probably not what you wanted!

Selecting Nodes

Referring to the Current Node

- If you want to refer to the current node in a location path…
- …use a single full stop (**.**).

- For example, to output the value of the current node, use:

```
<xsl:value-of select=“.” />
```

Selecting a Node’s Children

- Refer to a current node’s child element…
- …by simply using its name in the location path.

- To get the value of a current node child, **surname**:

```
<xsl:value-of select=“surname” />
```

Selecting a Node’s Children

- Select the current node’s grandchild, great grandchild, etc...
- …by separating hierarchy levels using a forward slash (**/**).

- To get the **age** child of the current node’s **person** child:

```
<xsl:value-of select=“person/age” />
```

Selecting a Node’s Children

- Use an asterisk (*****) to refer to all children of a node.

- To get the **age** child of any current node children:

```
<xsl:value-of select=“*/age” />
```

Selecting a Node’s Parent or Siblings

- Use two full stops (**..**) to refer to the current node’s parent.
- Like this:

```
<xsl:value-of select=“..” />
```

Selecting a Node’s Parent or Siblings

- Get the current node’s sibling by going up one level…
- …then referring to the sibling by name (child of parent).

- So to find the current node’s sibling, **cheese**:

```
<xsl:value-of select=“../cheese” />
```

Selecting a Node’s Attributes

- Use the at sign (**@**) to signify that a node is an **attribute**.

- To get the current node’s attribute, **id**…
- …and then get the **person** child’s **num** attribute:

```
<xsl:value-of select=“@id” />
<xsl:value-of select=“person/@num” />
```

Selecting All Descendants

- To select all descendants of a node, use **//**.

- If the entire path is just **//**...
- ...you'll select all descendants of the **root node**.

- If you precede it with a full stop (**.//**)...
- ...you'll select all descendants of the **current node**.

Selecting All Descendants

- End a path with **//** to select...
- ...all descendants of the node(s) the **last step** selected.

- Follow a **//** with a **node name** or steps to a node...
- ...to select only the descendants with that name.

- E.g. to select all **name** elements in the doc:

```
//name
```

Conditional Selection

Conditionally Selecting Nodes

- Selecting an entire node set is not always precise enough.
- Use **predicates** to test a condition and create a subset.
- A **predicate** is a Boolean expression (i.e. true or false).
- Predicates can compare values, do math…
- …test for existence, contain functions, etc.

Conditionally Selecting Nodes

- Use block brackets (**[]**) to define a predicate.

- **Examples:**

  Select only the **name** children of the current node that have a **status** attribute whose value is **not** “hidden”:

```
name[@status!=‘hidden’]
```

Conditionally Selecting Nodes

Select only the **name** children of the current node that have a **status** attribute (regardless of its value):

```
name[@status]
```

Select only the last **name** child of the current node that has a **status** attribute whose value is “hidden” :

```
name[@status=‘hidden’][position()=last()]
```

Conditionally Selecting Nodes

Select all attributes (regardless of their names) of the last **name** child of the current node:

```
name[last()]/@*
```

- A location step can have more than one predicate.
- Adding **position()=** is optional.
- We’ll talk more about XPath functions later.

XPath Functions

XPath Functions

- When selecting a node set with a location path...
- ...you may not want or need all data in the node set.

- You can use **functions** to apply additional logic...
- ...to return only the data you need.

- When retrieving the string value of a node...

XPath Functions

- ...you can use **functions** on the string before it outputs.
- We are currently using **XPath 1.0**.
- Functions unique to XPath 2.0 will not work in XPath 1.0!

Comparing Two Values

- A common test is to compare one value with another.

- **The syntax (within an XPath expression):**

```
nodeSet operator ( nodeSet OR literalValue )
```

- Replace **operator** with one of the following:

```
=      !=      &gt;      &gt;=      &lt;      &lt;=
```

Comparing Two Values

- If **literalValue** is a string, enclose it in single quotes.

- Use **and** or **or** to separate multiple comparisons:
  - If you use **and**, all conditions must be true before proceeding;
  - If you use **or**, only one needs to be true to proceed.

Testing the Position

- You can select a specific node in a set from its position.
- **The syntax (within an XPath expression):**

```
position() = n
```

```
current()
```

```
last()
```

- Replace **n** with the number that identifies…
- …the position of the node in the current node set.

Testing the Position

- If you use **position()=n** inside a **predicate**...
- ...you can shorten it to just **n**.

- E.g. to select the third **name** child of the current node:

```
name[3]
```

Multiplying, Dividing, Adding, Subtracting

- To test for more complicated conditions…
- …or to output calculated values…
- …include **arithmetic operations** to your expressions.

- **The syntax (within an XPath expression):**

```
nodeSet/number operator nodeSet/number
```

Multiplying, Dividing, Adding, Subtracting

- Replace **operator** with one of the following:

```
*    div    +    -
```

- Note that ***** and **div** are performed before **+** and **-**.

- You can also use **mod** to find the remainder of division.

Recap

- nodeSet operator (nodeSet or literalValue)

```
Names/name != 'David Bowie'
```

- position(), current(), last()

```
position()=6 OR name[6]
```

- nodeSet operator (nodeSet or number)

```
value[5] – value [4]
```

Counting Nodes

- Often you'll want to know how many nodes a set has.
- **The syntax (in an XPath expression):**

```
count(nodeSet)
```

- **You can optionally include predicates**

```
count( nodeSet [ . = ‘babypowder’ ])
```

Formatting Numbers

- XPath arithmetic uses floating point math.
- This can result in long output numbers.
- Use **format-number** to control the output format.
- **The syntax (in an XPath expression):**

```
format-number(number, 'pattern')
```

Formatting Numbers

- Replace **number** with a literal number or...
- ...an expression that results in a number.

- The **pattern** shows how you want the number formatted.
  - **#** in the pattern means the number must only appear if not 0.
  - **0** in the pattern means the number must always appear.
  - Use a full stop (**.**) to indicate the decimal point.
  - Use a comma (**,**) to separate groups of digits (e.g. 1,000,000).

Quick example

- Number 1000000
  - Format - ##,000,000 = 1,000,000
  - Format – 00,000,000 = 01,000,000
  - Format - ##,###,###.00 = 1,000,000.00
  - Format – 0#,0#,0#,0#.0# = ?

Rounding Numbers

- Use **ceiling** to round a number up.
- Use **floor** to round a number down.
- Use **round** to round a number to the nearest integer.
- We can do several arithmetic calculations...
- ...and use round function to round the answer, before output.

Extracting Substrings

- You can use functions to extract part of a string.

- **The syntax (in an XPath expression):**

```
substring-before(string, character)
substring-after(string, character)
```

- Replace **string** with a literal string or string expression.

Extracting Substrings

- Replace **character** with a character in the **string**.
  - If **-before**, the substring before the **character** is returned.
  - If **-after**, the substring after the **character** is returned.

- To extract a substring in the middle of a string, use:

```
substring(string, firstPos, numChars)
```

Changing the Case of a String

- You can change letters from upper- to lowercase and back.

- **To capitalise strings:**

```
translate(string, 'abcdefghijklmnopqrstuvwxyz',
'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
```

- Swap the last two parameters to change to lowercase.

Quick example

- String: Apple

```
Translate -> ‘abcdefghijklmnopqrstuvwxyz’
             ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’
                                                    =APPLE
Translate -> ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’
                        ‘abcdefghijklmnopqrstuvwxyz’
                                                    =apple
Translate -> ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’
                        ‘qwertyuiopasdfghjklzxcvbnm’
                                                    = ?
```

Totalling Values

- Use **sum** to add up all the values in a set of nodes.

- **The syntax (in an XPath expression):**

```
sum(nodeSet)
```

More XPath Functions

- **name(nodeSet)** returns the name of the set's first node.
- **name()** returns the name of the current node.
- **contains(str1, str2)** returns **true** if **str1** contains **str2**.
- **string-length(str)** returns the number of characters in the string.

More XPath Functions

- **normalize-space(str)** removes…
- …leading and trailing white space in the string…
- …and replaces sequences of white space with one space.

- **not(expression)** returns **true**…
- …if **expression** evaluates to **false**.

More XPath Functions

- Finally, you can use the vertical bar symbol (**|**)...
- ...to combine node sets.

- For example, **nodeSet1 | nodeSet2** will return...
- ...a node set that is a combination of both sets.

Theme 5: XPath

404

- END OF THEME 6