Theme 4

XSLT Part 2

Using XML data in output

Outputting Values

- You've seen how to write **literal elements**...
- ...as root template **instructions**...
- ...to output them to the result exactly as they appear.

- How can we **output** the text content of a source node?

- The text content of a node is a its **string value**.

Outputting Values

- Use **xsl:value-of** to retrieve a node's string value...
- ...and output it to the result.

- The string value will appear in the result...
- ...**exactly** where you place the **xsl:value-of**...
- ...in the result structure.

Outputting Values

- **The syntax (within a template):**

```
<xsl:value-of select="expression" />
```

- The **expression** selects a node set from the source xml document...
- ...whose string value should be output at this point.

- This method will later be used to retrieve **sets** of values.

Outputting Values

- The expression inside the **select** attribute...
- ...is a **path** to the node set we want.

file directory

- The root template applies to the root node of the source...
- ...so the path must go down the hierarchy from there.

- The path goes down the tree in a series of **steps**.

Outputting Values

- The path, **root/parent/parent/child**

  e.g. **character/class/names/name…**

- …returns a **node set** that contains:
  - First **name** nodes in the XML source…
  - …that have a parent node called **names**…
  - …which itself has a parent node called **class**…

Outputting Values

- You can add a **[condition]** to a path expression...
- ...to limit the node set it selects.

- For example, if we change the path to:

```
character/class/names/name[@active='true']
```

Outputting Values

- Note:
  The **xsl:value-of** command always outputs **only**...
- ...the string value of the **first** node in the selected set...
- ...**even** if the set contains more than one node.

only returns one thing

Outputting Values

- If that first node has child elements with text content...
- ...that node's text **and** that of its children are output.

- If the selected node set is empty, **nothing** is output.

- If the select **expression** evaluates to a **number** instead of a node set, the number is output as a string.

- If it evaluates to a Boolean, **true** or **false** is output (strings).

Generating Output Attributes

- **Consider the following scenario in an XSLT doc:**
  - You are transforming from XML to HTML.
  - You want to use XSLT to put a link in the result doc:

```
<a href="...">A link</a>
```

  - You want the value of **href** to be a value from the source.

Generating Output Attributes

- Assume that the source value is stored in a **url** element.
- **You CANNOT put** xsl:value-of **in an attribute value:**

```
<a href="<xsl:value-of select="url"/>">A link</a>
```

- How then do we get **url**'s string value into **href**?
- **Solution:** Generate the attribute with **xsl:attribute**.

Generating Output Attributes

- **Do this in the XSLT doc:**

```
<a>
        <xsl:attribute name="href">
            <xsl:value-of select="url" />
        </xsl:attribute>
        A link
</a>
```

- You can generate other output attributes in the same way.

Basic Computation

Looping Over Nodes

- Sometimes you want to act on **all selected nodes**...
- ...not only the first one.

- Use **xsl:for-each** to process each selected node...
- ...one after the other.

Looping Over Nodes

- **The syntax (within a template):**

```
<xsl:for-each select="expression">
        <!-- instructions -->
</xsl:for-each>
```

- Inside the **xsl:for-each**, we use **xsl:value-of** to…
- …output the string values of the child elements.

Looping Over Nodes

- **An important aside:**
  - If you write HTML code inside an XML doc...
  - ...such as HTML inside an XSLT style sheet...
  - ...that HTML must follow the XML grammar rules.

Processing Nodes Conditionally

- Instead of processing only the **first** node in a set...
- ...or processing **each** selected node one by one...
- ...you could process only nodes that satisfy a **condition**.

- Use **xsl:if** with a test expression to set such a condition.

Processing Nodes Conditionally

- **The syntax (within a template):**

```
<xsl:if test="expression">
          <!-- instructions -->
</xsl:if>
```

- The **expression** specifies a node set, string, or number.

- The **instructions** specify what should happen if...

Processing Nodes Conditionally

- **...**the node set, string, or number is **not empty**.
  - Or, **not equal to zero**, in the case of a number.

- With **xsl:if**, you can only test and react to **one condition**.

- Use **xsl:choose** to test for several conditions.

Adding Conditional Choices

- You can, for example, do one action if a condition is true...
- ...and another if it is false.

- In which we use **xsl:otherwise.**

```
<xsl:otherwise>
          <!-- instructions -->
</xsl:otherwise>
```

Adding Conditional Choices

- **The syntax (within a template):**

```
<xsl:choose>
        <xsl:when test="expression">
            <!-- instructions -->
        </xsl:when>
        <!-- zero or more additional xsl:when's -->
        <xsl:otherwise>
        <!-- what to do if none of the conditions are true -->
        </xsl:otherwise>
</xsl:choose>
```

Adding Conditional Choices

- In an **xsl:choose**, the **xsl:when** conditions are tested...
- ...starting from the first **xsl:when**.

- Once an **xsl:when**'s condition is found to be true...
- ...all subsequent **xsl:when**'s are ignored.

- Only the first true **xsl:when**'s instructions are performed.

Sorting Nodes Before Processing

- You already know that you can use **xsl:for-each**...
- ...to process each selected node one by one.

- By default, the nodes are processed from top to bottom...
- ...as depicted in the source doc (i.e. in **document order**).

bypass

- Use **xsl:sort** to process the nodes in a different order.

Sorting Nodes Before Processing

- **The syntax (as the first child of an** xsl:for-each**):**

```
<xsl:sort select="criteria"
        order="descending"
        data-type="text"
/>
        <!-- or data-type="number"-->
<!-- Add more xsl:sort's if you want -->
```

Sorting Nodes Before Processing

- Replace **criteria** with an **expression** that specifies...
- ...the node on which the selected nodes should be sorted.

- The **order** attribute is optional and defaults to **ascending**.

- **descending** = high to low numbers, or Z to A.

- **ascending** = low to high numbers, or A to Z.

Sorting Nodes Before Processing

- The **data-type** attribute is optional and defaults to **text.**

- Set it to **number** if you're sorting numbers.
  - If you don't, the sorting will not occur as you expect.

- You can have multiple **xsl:sort**'s in an **xsl:for-each**.

- You can also have **xsl:sort**'s within other **xsl:sort**'s.

Using Templates

Templates

- **Reminder:**
  - Every XSLT style sheet has a **root template**…
  - …that applies to the source doc’s root node (**/**).
  - The processor starts by applying the root template…
  - …and stops processing at the end of the root template.

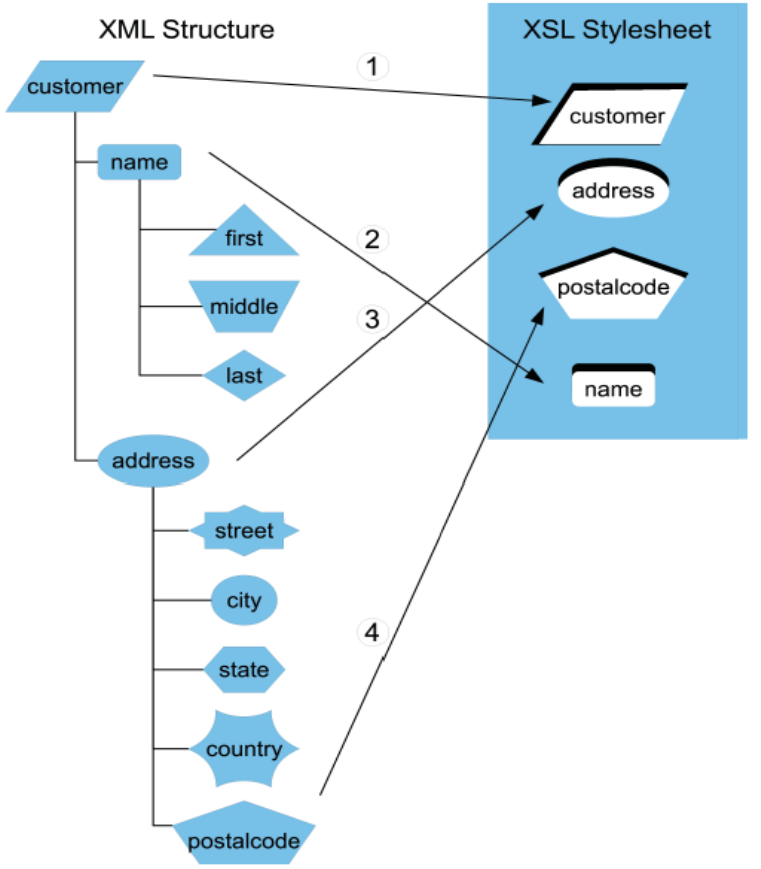
Templates

- Templates are the basic **working unit** of a stylesheet
- Templates provide **three type** of functions in XSLT
  - The [match] or [name] attribute (or both) tells the XSLT process when to use the template.
  - Inside the template, some **additional work** gets done
  - One or more instructions invoke other templates to process more of the XML document.

Creating and Applying Templates

- You can create more templates than just the root template.
  - We call them **sub-templates**.

- Like **functions** in a programming language...
- ...a **sub-template** contains a set of instructions...
- ...that can be reused multiple times in the XSLT.



Creating and Applying Templates

- To “run” a sub-template, you must **apply** it... <xsl:apply-templates select="
- ...from within another template (e.g. the root template).

- **Sub-template syntax (as a child of** xsl:stylesheet**):**

```
<xsl:template match="pattern">
        <!-- instructions -->
</xsl:template>
```

Creating and Applying Templates

- Replace **pattern** with an expression that...
- ...identifies node(s) from the source XML.

- Sub-templates are **like the root template**...
- ...except that they **do not apply** to the root node (**/**)...
- ...and they must be applied manually.

Creating and Applying Templates

- Use **xsl:apply-templates** to apply sub-templates.

- With it, you can control **where** and **when**...
- ...a sub-template's instructions will create output...
- ...in the result doc.

Creating and Applying Templates

- **The syntax (within any template):**

```
<xsl:apply-templates select="expression" />
```

- The **expression** identifies the node(s) in the source...
- ...for which the processor must find sub-templates...
- ...to apply where you placed **xsl:apply-templates**.

Creating and Applying Templates

- The processor reaches **xsl:apply-templates**…
- …looks to see if there are any such **name** source nodes…
- …then looks through the sub-templates in the XSLT…
- …until it finds one that applies to such **name** nodes…
- …then it executes the sub-template’s instructions…

Creating and Applying Templates

- ...at the point where the **xsl:apply-templates** was used...
- ...and once finished with the sub-template...
- ...the processor continues processing the root template...
- ...starting after the **xsl:apply-templates**.

Creating and Applying Templates

- If you use **xsl:apply-templates**...
- ...but you leave out the **select** attribute...
- ...all the **children** of the **current node** will be selected.

- (The **current node** is the node(s) matched by...
- ...the parent **xsl:for-each** or **xsl:template** instruction.)

Creating and Applying Templates

- If an **xsl:apply-templates cannot** find a sub-template...
- ...to apply to the selected node(s), it will instead...
- ...try to apply templates to the **selected node**(s) **children**.

- The processor **keeps going down** the hierarchy...
- ...until it **finds** a sub-template to apply or **runs out** of nodes.

Creating and Applying Templates

- During the process described in the previous slide...
- ...if the processor comes across an attribute or text node...
- ...it **prints the text value** of that node to the output.

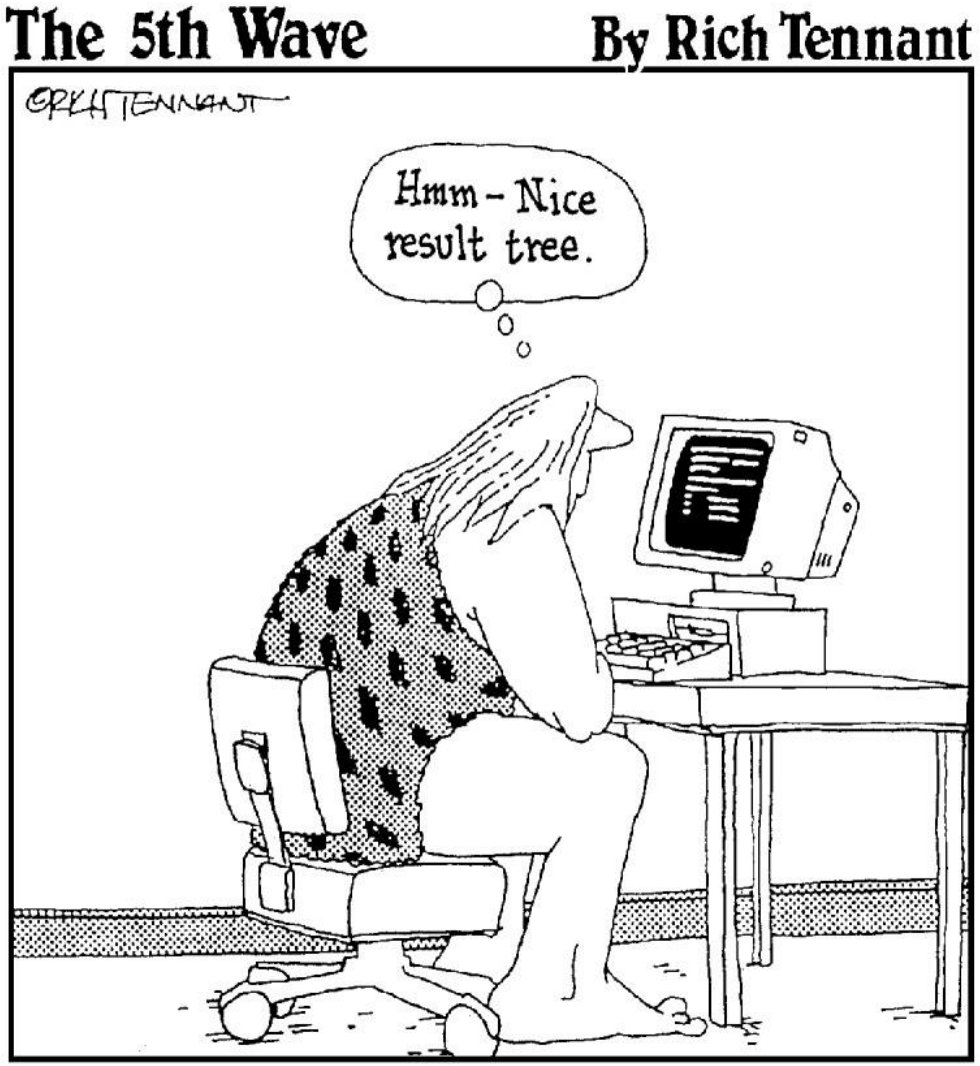
- (A **text node** is a child node of an element node...
- ...that stores that element’s text content (if it has any).)

Creating and Applying Templates

- You can also use **xsl:sort**...
- ...to sort the node set selected by **xsl:apply-templates**...
- ...before any sub-templates are applied to the nodes.

- To do so, create **separate** start and end tags...
- ...for **xsl:apply-templates**, and nest **xsl:sort** as its child.

Theme 4: XSLT


The 5th Wave
By Rich Tennant
©RICH TENNANT
Hmm - Nice result tree.