XSL Concepts: Functions and Calling Templates

2016 OmniUpdate User Training Conference Activity Guide

OmniUpdate, Inc.
1320 Flynn Road, Suite 100
Camarillo, CA 93012

Contents

XSL Templates

XSL Templates are blocks/modules of XSL instruction that are applied based on either a matched pattern or a given name.

xsl:template (match)

The `xsl:template` match uses the pattern defined in the `match` attribute to identify which nodes apply the rule defined in the template. If more than one template rule meets the match criteria, they are first considered in order of import precedence. The templates that have the highest import precedence are applied to the resultant document. If there is still more than one template rule that meets the match criteria and shares the same import precedence, they are next considered in order of priority. The priority is either given by the value of the priority attribute, or is a default priority that depends on the match pattern.

```
<xsl:template
name="name"
match="pattern"
priority=number>
<!-- Content:(<xsl:param>*,template) -->
</xsl:template>
```

Example

```
      <!-- XSL Template with match -->
      <?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM
"http://commons.omniupdate.com/dtd/standard.dtd">
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ou="http://omniupdate.com/XSL/Variables">
      <!--  Main Template -->
      <xsl:template match="/">
            <!-- Template Content -->
      </xsl:template>
</xsl:stylesheet>
```

Attribute — `name` (optional): Specifies a name for the template. If this attribute is omitted, there must be a `match` attribute.

Attribute — `match` (optional): Contains the match pattern for the template. If this attribute is omitted, there must be a `name` attribute.

Attribute — `priority` (optional): A positive or negative integer or decimal number that denotes the priority of the template.

Once a match pattern is successful, the rules defined in the template are applied to every node satisfying the match pattern.

Example

```
<!-- Source XSL FILE -->
<root>
      <faculty>
            <first_name>John</first_name>
            <last_name>Doe</last_name>
            <department>Computer Science</department>
            <phone>123-234-1234</phone>
      </faculty>
      <faculty>
            <first_name>Peter</first_name>
            <last_name>Parker</last_name>
            <designation>Economics</designation>
            <phone>234-123-7890</phone>
      </faculty>
      <faculty>
            <first_name>Mary</first_name>
            <last_name>Jane</last_name>
            <designation></designation>
            <phone></phone>
      </faculty>

</root>

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM
"http://commons.omniupdate.com/dtd/standard.dtd">
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ou="http://omniupdate.com/XSL/Variables">
      <!--  Main Template -->
      <xsl:template match="/">
            <!-- Template Content -->
      </xsl:template>
</xsl:stylesheet>
```

```
<!-- Recursive XSL Template -->
<!DOCTYPE xsl:stylesheet SYSTEM
"http://commons.omniupdate.com/dtd/standard.dtd" >
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" encoding="UTF-8" omit-xml-
declaration="yes" />

      <xsl:template match="/">
```

```
      <html>
            <head>
                  <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"/>
                  <title>Faculty Table</title>
            </head>
            <body>
                  <table>
                        <thead>
                              <tr>
                                    <td>First Name</td>
                                    <td>Last Name</td>
                                    <td>Department</td>
                                    <td>Phone Number</td>
                              </tr>
                        </thead>
                        <tbody>
                              <xsl:apply-templates select="/root"/>
                        </tbody>
                  </table>
            </body>
      </html>
      </xsl:template>

      <xsl:template match="faculty">
            <tr>
                  <xsl:apply-templates />
            </tr>
      </xsl:template>

      <xsl:template match="first_name">
            <td><xsl:value-of select="." /></td>
      </xsl:template>

      <xsl:template match="last_name">
            <td><xsl:value-of select="." /></td>
      </xsl:template>

      <xsl:template match="department">
            <td><xsl:value-of select="." /></td>
      </xsl:template>

      <xsl:template match="phone">
            <td><xsl:value-of select="." /></td>
      </xsl:template>
</xsl:stylesheet>
```

XSL Elements:

xsl:copy-of:

Copies the sequence of nodes to the resultant sequence. Also known as “deep copy ”, when applied to a sequence, the node and all its decedents are copied. The `select` attribute is a required attribute. By default, xsl:copy-of will copy all the namespces.

```
<xsl:copy-of
  select = expression
  copy-namespaces? = "yes" | "no"
  type? = qname />
Example:
      <!--  Deep copy using XPath  -->
      1. <xsl:copy-of select=”//body”/>
      <!--  Deep copy using predefined variable  -->
      2. <xsl:copy-of select=”$variable_name”/>
```

Attribute – `select` (mandatory) : The sequence of values or nodes to be copied in the stylesheet.

Attribute – `copy-namespace` (optional) : Indicates whether the namespace of an element should be copied. The default value is “yes”

Attribute -- `validation` (optional) : Specifies whether the copied nodes should be subjected to schema validation, or whether existing type annotations should be retained or removed

Attribute -- `type` (optional) : Identifies the type declaration against copied nodes.

xsl:value-of:

Creates a text node for a given expression. If the resultant sequence contains a single value, this is converted to a string otherwise it returns a zero-length string.

```
<xsl:value-of
   select = expression
   disable-output-escaping = "yes" | "no"/>

Example:
      <!--  Single value from a XPath  -->
      1. <xsl:copy-of select=”//body”/>
      <!--  Single value from a predefined variable  -->
      2. <xsl:copy-of select=”$variable_name”/>
```

Attribute – `select` (optional) : Defines the name of the function.

Attribute – `disable-output-escaping` (optional) : Indicates whether or not the special characters of the output should be XML escaped. The default value is “no.”

xsl:variable:

Used to create a variable. If the variable is declared at the top level (immediately within `xsl:stylesheet`), it declares a global variable. Otherwise, it declares a local variable that can be visible only within the style sheet element. The `name` attribute is required.

```
<xsl:variable
name="name"
select="expression"
as="sequence-type"
>
      <!-- Content: sequence-constructor -->
</xsl:variable>
```

Attribute – `name` (mandatory) : Defines the name of the variable.

Attribute – `select` (optional) : The expression that is evaluated as the value of the variable . If omitted, the value is determined by the value of the sequence constructor of the `xsl:valriable` element.

Attribute – `as` (optional) : The return type of the variable.

xsl:param:

Used to create a parameter, which if declared at the top level defines as global parameter. If the parameter is defined immediately within an `xsl:template` element, creates a localized parameter to a template. The `name` attribute is required. `xsl:param` may be created with a local scope, if an `xsl:param` definition appears as an child element of `xsl:template` or `xsl:function` or `xsl:iterate` element and before any non-`xsl:param` children of that element.

```
<xsl:param
name="name"
select="expression"
as="sequence-type"
required= “yes” | “no”
>
      <!-- Content: sequence-constructor -->
</xsl:param>
```

Attribute – `name` (mandatory) : Defines the name of the variable.

Attribute – `select` (optional) : The expression that is evaluated as the value of the variable . If omitted, the value is determined by the value of the sequence constructor of the `xsl:param` element.

Attribute – `as` (optional) : The return type of the variable.

Attribute – `required` (optional) : Specifies whether the parameter is optional or mandatory.

Stylesheet Parameters:

Stylesheet parameters are useful in creating global parameters that are accessible throughout the entire stylesheet declaration. A stylesheet parameter may be declared as mandatory, or may have a default value specified for use when no value is supplied.

Template Parameters:

As a child of the template, `xsl:param` is used to define the parameter of the template, which may be supplied when the template is invoked.

Function Parameters:

As a child of the function, `xsl:param` is used to define the parameter of the function, which will be supplied when the function is invoked. Functional parameters are mandatory when called within an XPath.

xsl:for-each:

`xsl:for-each` selects a sequence of items using an XPath expression, and performs the same processing for each item in the sequence.

```
<xsl:for-each
select="sequence-expression"
>
      <!-- Content: sequence-constructor -->
</xsl:for-each>
```

Argument – `select` (mandatory): Expression returning a sequence of nodes/values that needs to be processed.

XSL Functions

XSL functions allow developers to create stylesheet functions that can be called from any XPath expression used in the stylesheet. The default syntax for an xsl:function is as follows

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:function
      name= qname
      as=sequence-type
      override = “yes” | “no”
>
<!-- Content: (xsl:param, sequence-constructor) -->
</xsl:function>

Example:
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM "http://commons.omniupdate.com/dtd/
standard.dtd">
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:ou="http://omniupdate.com/XSL/Variables">
<xsl:template match="/">
      <xsl:copy-of select=”ou:sample_function()”/>
</xsl:template>

<!-- function Definition -->
<xsl:function
      name= “ou:sample_function”
      as=”xs:string”>
<!--
Sequence processing for the function.
-->
</xsl:function>
</xsl:stylesheet>
```

Attribute – `name` (mandatory): Defines the name of the function. The name must have a namespace prefix. This will ensure that the name does not clash with the names of functions in the standard function library. XSLT specification has reserved pre-defined namespace that cannot be used for the names of user-defined functions. Please refer to http://w3.org.

Attribute -- `as` (optional): The type of the value returned when this function is evaluated. A sequence type holds the definition of the type of the data contained.

Attribute -- `override` (optional): Specifies whether the function overrides any vendor-supplied function of the same name.

Sample Functions

1. Capitalize the first letter of every word.

```
<!--  Main Template -->
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM "http://commons.omniupdate.com/dtd/
standard.dtd">
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:ou="http://omniupdate.com/XSL/Variables">
<xsl:template match="/">
      <xsl:copy-of select=”ou:capital(‘camel case this sentence’)”/>
</xsl:template>

<!-- Function Definition -->
<!--  CAPITALIZE THE FIRST LETTER OF EVERY WORD  -->
<xsl:function name="ou:capital">
      <xsl:param name="string"/>
      <xsl:variable name="chars" select="tokenize($string, ' ')"/>
            <xsl:for-each select="$chars">
                  <xsl:variable name="key">
                        <xsl:value-of select="."/>
                  </xsl:variable>
                  <xsl:variable name="firstletter1">
                        <xsl:value-of select="upper-
case(substring($key,1,1))"/>
                  </xsl:variable>
                  <xsl:variable name="rest1">
                        <xsl:value-of select="lower-
case(substring($key,2))"/>
                  </xsl:variable>
                  <xsl:variable name="result">
                        <xsl:value-of select="concat($firstletter1,$rest1,'
')"/>
                  </xsl:variable>
                  <xsl:value-of select="$result"/>
            </xsl:for-each>
</xsl:function>
</xsl:stylesheet>
```

2. Clean content by replacing content with space.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM "http://commons.omniupdate.com/dtd/
standard.dtd">
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:ou="http://omniupdate.com/XSL/Variables">
<xsl:template match="/">
      <xsl:copy-of select=”ou:clean(‘sample_text_with_underscore’)”/>
</xsl:template>

<xsl:function name="ou:clean">
    <xsl:param name="string"/>
    <xsl:param name="delimiter"/>
```

```
      <xsl:variable name="chars" select="tokenize($string,$delimiter)"/>
       <xsl:for-each select="$chars">
            <xsl:variable name="key"><xsl:value-of
select="."/></xsl:variable>
            <xsl:variable name="firstletter1"><xsl:value-of select="upper-
case(substring($key,1,1))" /></xsl:variable>
                  <xsl:variable name="rest1"><xsl:value-of select="lower-
case(substring($key,2))" /></xsl:variable>
                  <xsl:variable name="result"><xsl:value-of
select="concat($firstletter1,$rest1,' ')" /> </xsl:variable>
                  <xsl:value-of select="$result" />
     </xsl:for-each>
</xsl:function>
</xsl:stylesheet>
```

System Defined XPath Functions:

concat():

Concat takes two or more arguments and return a string. Please note that each argument is converted into string.

```
concat(arg1, arg2,…)
Example: <xsl:value-of select="concat('Hello','World')"/>
Output: Hello World
```

Argument – `arg(n)` (mandatory): More than one argument of string to be combined.

upper-case():

The upper-case function converts lower case characters in a string to upper case.

```
Upper-case(string)
Example: <xsl:value-of select="upper-case('Hello World')"/>
Output: HELLO WORLD
```

Argument – `arg` (mandatory): String argument to be converted into upper case.

substring-after():

The substring-after function returns the part of the string, after a delimiter. Please note that the function will return the first occurrence of the specified substring.

```
substring-after(string, delimiter)
Example: <xsl:value-of select="substring-after('index.html', '.')"/>
Output: html
```

Argument – `string` (mandatory): The containing string

Argument – `delimiter` (mandatory): The delimiter substring

substring-before():

The substring-after function returns the part of the string, after a delimiter. Please note that the function will return the first occurrence of the specified substring.

```
substring-before(string, delimiter)
Example: <xsl:value-of select="substring-after('index.html', '.')"/>
Output: index
```

Argument – `string` (mandatory) : The containing string

Argument – `delimiter` (mandatory) : The delimiter substring

tokenize:

Tokenize function splits a string in a sequence of substrings, based on a provided delimiter.

```
tokenize(string, delimiter)
Example: <xsl:value-of select="tokenize('one, two, three', ',')"/>
Output: one two three
```

Resources

W3C XSLT 2.0 Reference: http://www.w3.org/TR/xslt20/

W3C XSLT 3.0 Reference: http://www.w3.org/TR/xslt-30/

W3C XPath Reference: http://www.w3.org/TR/xpath-functions/