Advanced XSL Concepts: Transforming External Data Sources

2016 OmniUpdate User Training Conference Activity Guide

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

Contents

Advanced XSL Concepts: Transforming External Data Sources

XSL allows developers to fetch external documents to be parsed and processed to create new transformed content. There are two XSLT functions that allow developers to achieve this goal; `xsl:doc` and `xsl:unparsed-text`.

xsl:doc

The `xsl:doc` function finds an external XML document by resolving a URI reference, parses the XML into a tree structure, and returns its root node. The URI used as input to the doc() function should identify an XML document. If the URI is invalid, or the resource is unavailable, of if that the resource is not an XML document, then OU Campus will report the error.

```
Doc Declaration:
doc(href)

Example:
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM "http://commons.omniupdate.com/dtd/
standard.dtd">
<xsl:stylesheet version="3.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:ou="http://omniupdate.com/XSL/Variables">
<xsl:template match="/">
      <xsl:copy-of select="doc('http://example.com/sample.xml')"/>
</xsl:template>

</xsl:stylesheet>
```

Attribute – `href` (mandatory) : The URI used to locate the document to be loaded.

xsl:unparsed-text

The `xsl:unparsed-text` function returns the content of an external file in string format by resolving the provided URI. This function is quite similar to the `xsl:doc` except that unparsed-text returns text rather than XML.

```
Doc Declaration:
unparsed-text(href)

Example:
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM "http://commons.omniupdate.com/dtd/
standard.dtd">
<xsl:stylesheet version="3.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:ou="http://omniupdate.com/XSL/Variables">
<xsl:template match="/">
      <xsl:copy-of select="unparsed-text('http://example.com/sample.csv')"/>
</xsl:template>

</xsl:stylesheet>
```

Attribute – `href` (mandatory) : The URI used to locate the document to be loaded.

Example xsl:doc call:

```
 <!-- Call an external XML(RSS) file and transform it to HTML -->
Doc Declaration:
unparsed-text(href)

<!--?xml version="1.0" encoding="iso-8859-1"?-->
<!DOCTYPE xsl:stylesheet SYSTEM
"http://commons.omniupdate.com/dtd/standard.dtd" >

<xsl:stylesheet version="3.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:media="http://search.yahoo.com/mrss/">

<xsl:template match="/">

<html>
      <head>
            <title>News Listing</title>
      </head>
      <body>
            <xsl:for-each
select="doc('http://www.gallenauniversity.com/_resources/rss/news.xml')/rss/c
hannel/item">
                  <ul>
                        <li><a href="{./link/node()}"><xsl:copy-of
select="./title/node()"/></a></li>
                        <li><xsl:copy-of select="./description/node()"/></li>
                        <li><img src="{./media:content/media:thumbnail/@url}"
alt=""/></li>
                        <li><xsl:copy-of
select="./media:content/media:title/node()"/></li>
                  </ul>
            </xsl:for-each>
      </body>
</html>
</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 namespaces.

```
<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.

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/