Advanced XSL Concepts: Applying and Matching Templates

2016 OmniUpdate User Training Conference Activity Guide

OmniUpdate, Inc.

1320 Flynn Road, Suite 100

Camarillo, CA 93012

Content

Content **2**
Advanced XSL Concepts: Applying and Matching Templates **4**
Anatomy of XML Document 4
XML Tree Structure 5
Document Prolog 5
Declaration Attributes *5*
Document Type Declaration (DTD) 5
Elements 5
Attributes 6
Namespaces 6
Entities 6
Additional XML Definitions 6
Publish Control File (PCF) 7
PCF Stylesheet Declaration 7
Declaration Attributes *7*
XSL **9**
XSL Definition 9
Standard XSLT 2.0 Document Structure 9
XSL Transformations in OU Campus 10
XSL Elements **11**
xsl:copy-of *11*
xsl:value-of *11*
xsl:variable *12*
xsl:param *12*
Stylesheet Parameters *13*
Template Parameters *13*
Function Parameters *13*
xsl:if *14*
xsl:when *14*
xsl:otherwise *15*
xsl:comment *15*
xsl:import *16*
xsl:output *16*
xsl:template *17*
xsl:call-template *17*
xsl:apply-templates *18*
xsl:stylesheet *19*
xsl:copy *19*
xsl:text *20*
Element *20*
xsl:for-each *21*
XSL Statements 21
Copy-of *21*
Comment *21*
If statement *21*
Import *22*
Output *22*
When/Otherwise *22*
Template *23*
Call-template *23*
Apply-templates *23*
Stylesheet *23*
Processing Instruction *24*
Variable *24*
Param *24*
Attribute *24*
For-each *25*
Value-of *25*
Function *25*
Copy *25*
Text *26*
Attribute *26*
Element *26*
XPath Functions 26

Concat *26*
Tokenize *27*
Position *27*
Current *27*
Unparsed-text *27*
Document/Doc *27*
Introduction to XPath **28**
Axis selection in XPath 29
child *29*
descendant *29*
descendant-or-self *30*
preceding *30*
preceding-siblings *30*
following *30*
following-siblings *30*
ancestor *30*
ancestor-or-self *30*
System Defined XPath Functions **31**
concat() *31*
upper-case() *31*
lower-case() *31*
substring-after() *31*
substring-before() *32*
tokenize() *32*
OU Campus XSLT Variables **33**
ou:action 33
ou:root 33
ou:site 33
ou:path 33
ou:dirname 33
ou:filename 34
ou:httproot 34
ou:ftproot 34
ou:ftphome 34
ou:ftpdir 34
ou:username 34
ou:lastname 35
ou:firstname 35
ou:email 35
ou:feed 35
ou:email 35
Resources **36**

Advanced XSL Concepts: Applying and Matching Templates

XML (e**X**tensible **M**arkup **L**anguage) is a configurable vehicle for any kind of information that can be used to store and organize data. Although the name may suggest XML to be a markup language, XML is not itself a markup language; it is a set of rules for building markup languages.

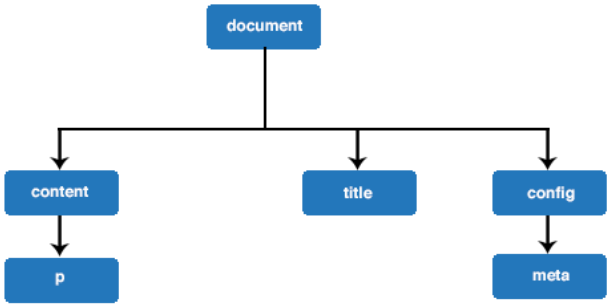
So, what is a markup language? It is information added to a document in order to enhance its meaning in certain ways. For example, when you are reading a webpage, you can differentiate the various segments of the page by their content, design and possibly by their placement. A markup language annotates the different segments of the data in a document.

Anatomy of XML Document

The fundamental building blocks of a XML document and all XML-derived languages are elements, attributes, entities, and processing instructions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "http://commons.omniupdate.com/dtd/standard.dtd">
<document>
	<config>
		<meta name="author" content="John Doe" />
	</config>
	<title>
		New Page
	</title>
	<content>
		Hello World
	</content>
</document>
```

XML Tree Structure



Document Prolog

In its simplest form, a document prolog merely denotes that the document is an XML document and declares the version of the XML being used with the encoding definition. All valid XML documents begin with an XML declaration. Always begin any XML document with the following XML declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Declaration Attributes

version – The latest version of XML is 1.0
encoding - This specifies the encoding of the document. When working with XML in OU Campus, please use UTF-8

Document Type Declaration (DTD)

The DTD is a set of rules or declaration, which is used to model an XML document. It is also reffered as document modeling. DTD allows developers to declare the set of allowed elements, attributes and the structure of the final XML document.

```
<!DOCTYPE document SYSTEM "http://commons.omniupdate.com/dtd/standard.dtd">
```

Elements

Elements are the building blocks of an XML document. Elements are nodes that include a tag set, along with any required attributes, attribute values, and content.

```
<content>
        Hello World
</content>
```

Attributes

An attribute is part of an element often used for data that identifies or describes the element. An attribute can be used to give an element a unique identity so that it can be easily located, or it can describe a property about the element.

```
<meta name="author" content="John Doe" />
```

Namespaces

XML namespaces allow developers to add vocabulary to an XML document. Namespaces provide a simple method of associating element and attribute names used in XML documents with namespaces identified. Following is an example of a name space declaration.

```
Namespace declaration syntax
ns-prefix:local-name

Example:
        <xsl:stylesheet></xsl:stylesheet>
```

Namespaces are useful to prevent name clashes and also allow processor to treat different groups of namespaces differently. An example of such a processing is the transformation language XSLT, which relies on the namespace to perform XSL specific instructions.

Entities

Elements Entities are placeholders for content, which you declare once and can use many times almost anywhere in the document. It doesn't add anything semantically to the markup. Rather, it's a convenience to make XML easier to write, maintain, and read.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "http://commons.omniupdate.com/dtd/standard.dtd" [
  <!ENTITY fullname "John Doe">
  <!ENTITY address "123 ABC Road, USA">
  <!ENTITY phone "<number>123-456-7890</number>">
]>
```

Additional XML Definitions

tag - A piece of text that describes the semantics or structure of a unit of data.
node - Smallest unit of structure in an XML document.
parent node - A node in a tree data structure that links to one or more child nodes.
child node - A node in a tree data structure that is linked to by a parent node.
root node - The first opened and last closed element in an XML document. This node contains all other nodes as children nodes.

Publish Control File (PCF)

pcf-stylesheet declaration - The processing instruction at the top of an XML document used to instruct OU Campus with which XSL file to use for a transformation.

Document Type Definition (DTD) - The formal definition of the elements, structures, and rules for marking up an XML document. You can store a DTD at the beginning of a document or externally in a separate file.

character entity- A reference to a named entity that has been predefined or explicitly declared in a DTD.

PCF Stylesheet Declaration

1. We must instruct OU Campus, using a pcf-stylesheet declaration, which XSL file to use to transform our PCF file. Following the XML stylesheet declaration, insert the following PCF stylesheet declaration:

```
<?pcf-stylesheet path="/_resources/xsl/default.xsl" title=”web” extension="html"?>
```

Declaration Attributes

path - Defines the location of the XSL file. Must be an absolute path from the root of the staging server. *This attribute is required.*

title - Defines the label on the OU Campus multi-output preview tab.

extension- Specifies the file extension to be used for the file output by the transformation. *This attribute is required.*

2. Since we'll be allowing users to enter in text through the XHTML-compliant WYSIWYG within OU Campus, we need to make sure they are able to use most, if not all, HTML character entities. We will use a DTD file as described in the example below. Link to the DTD in both headers of the PCF and XSL after all other declarations at the start of the file. Be aware that we add a doctype declaration to the XSL to utilize character entities within our XSL file.

```
<?xml version="1.0" encoding="utf-8"?>
<?pcf-stylesheet path="/_resources/xsl/default.xsl" extension="html" title="Web"?>
<?pcf-stylesheet path="/_resources/xsl/page2pdf.xsl" extension="pdf" title="PDF" alternate="yes"?>
<document>
	<config>
		<meta name="pagetype" content="3" />
		<!-- com.omniupdate.properties -->
		<title>Gallena Contacts</title>
		<meta name="keywords" content="Contacts" />
		<meta name="description" content="College Contact Information." />
		<meta name="show-pdf-link" content="false" />
	<!-- /com.omniupdate.properties -->
	</config>
	<content>
```

```
                    <!-- com.omniupdate.div label="content" group="Everyone" button="787" break="break" -
->
                    <!-- com.omniupdate.editor csspath="/_resources/ou/editor/controller.css"
cssmenu="/_resources/ou/editor/controller.txt" width="776" -->
                                <p><a name="top"> </a><br /><strong>Gallena University,
California</strong></p><p>2314 Running Springs Rd<br />Gallena, CA 91255-4901 USA<br />Tel: (818)
555-9401</p>
                                <p><a href="http://www.omniupdate.com">OmniUpdate</a></p>
                                <p><a href="/about/test9.html">test</a></p>
          <!-- /com.omniupdate.div -->
          </content>
</document>
```

XSL

XSL Definition

XSL (e**X**tensible **S**tylesheet **L**anguage) is a stylesheet language for XML documents. Just like CSS is used to tell the browser how to display an element in a special design, font or color, XSL is used to describe how the XML document should be displayed. The different components of XSL are:

XSL Transformations (XSLT) - a language used for transforming XML documents
XPath - a language used for navigating in XML documents
XSL-FO - a language for formatting XML documents, where FO refers to formatting objects.

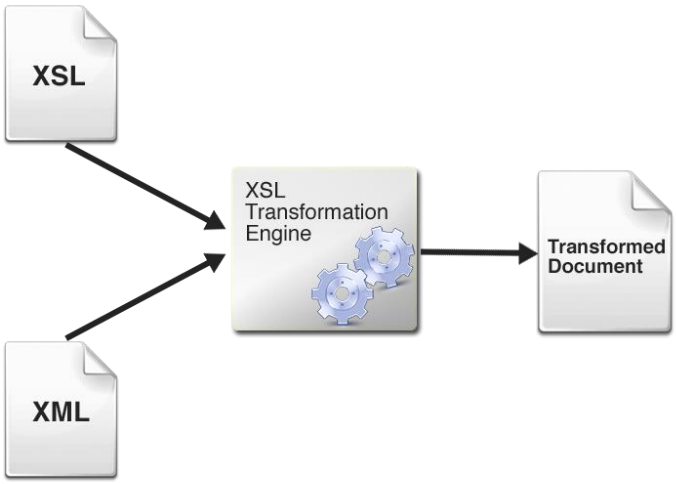
An important concept to understand is that XSLT is the most important part of XSL. Using XPath to navigate the structure of the XML document, XSLT transforms an XML document into another XML document.

Standard XSLT 2.0 Document Structure

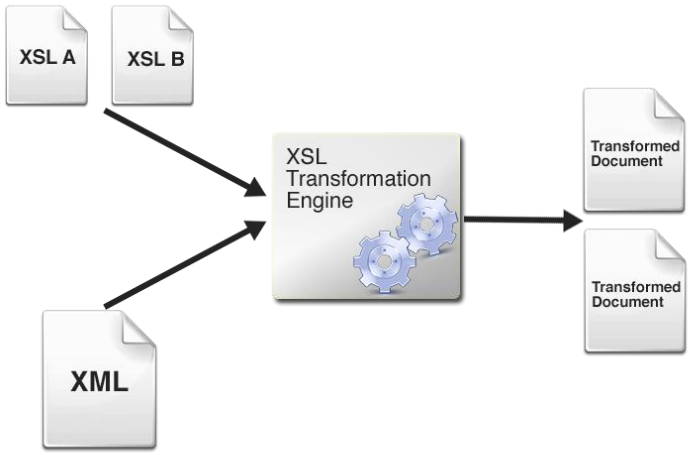
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsl:stylesheet SYSTEM "http://commons.omniupdate.com/dtd/
standard.dtd">
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform" xmlns:ou="http://omniupdate.com/XSL/Variables">
<xsl:output omit-xml-declaration="yes" />
<xsl:template match="/">
<!--
XSL Statements
And/Or
HTML Content
Goes Here
-->
</xsl:template>
</xsl:stylesheet>
```

XML is a simple and human readable way of interchanging structured document between computer systems. An XML document is a great way of storing data and one of its fundamental qualities is to separate data from presentation. Since there is a clear separation of data and presentation of the document, it becomes much easy to transform a document from one form to another with ease. This is where XSL comes into play, since XSL comprises of XPath, which can be used to navigate through an XML document and use XSLT, to transform a document to a virtual document (for other software systems) or use XSL-FO, to create a standard documents.

XSL Transformations in OU Campus



In OU Campus, every PCF(XML) document goes through a transformation call, where the XSL file based on the pcf-stylesheet declaration and the XML file is passed over to the transformation engine. Upon a successful transformation, OU Campus system takes the transformed document and creates a suitable preview. If the listed XSL file in the pcf-stylesheet declaration is missing or if the XSL/XML file is invalid, the user is prompted with the appropriate error message.



A single PCF file can contain multiple pcf-stylesheet declaration, in such a case, each pcf-stylesheet declaration goes through individual transformation call and upon successful transformation, each pcf-stylesheet declaration will create individual file.

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): The sequence of values to be copied in the stylesheet.

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>

Example:
        <!--  variable without type definition   -->
                <xsl:variable name=”test1” select=”document/content/p”/>
                <xsl:variable name=”test2”>
              <xsl:value-of select=”document/content/p”/>
            </xsl:variable>
                <xsl:variable name=”test3” select=”’text’”/>

        <!--  variable with type definition   -->
                <xsl:variable name="test4" as="xs:string" select="document/content/p"/>
                <xsl:variable name="test5" as="xs:string">
                <xsl:value-of select="document/content/p"/>
        </xsl:variable>
                <xsl:variable name="test6" as="xs:string" select="'text'"/>
```

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:variable 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. 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. The name attribute is required.

```
<xsl:param
name="name"
```

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

Example:
        <!--  parameter without type definition   -->
                <xsl:param name="test1" select="document/content/p"/>
                <xsl:param name="test2">
                <xsl:value-of select="document/content/p"/>
        </xsl:param>
                <xsl:param name="test3" select="'text'"/>

        <!--  parameter with type definition   -->
                <xsl:param name="test4" as="xs:string" select="document/content/p"/>
                <xsl:param name="test5" as="xs:string">
                <xsl:value-of select="document/content/p"/>
        </xsl:param>
                <xsl:param name="test6" as="xs:string" select="'text'"/>
```

Attribute – name (mandatory): Defines the name of the parameter.

Attribute – select (optional): The expression that is evaluated as the value of the parameter. 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 parameter.

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:if

xsl:if is used for conditional processing. It takes a mandatory test attribute, with a boolean expression.

```
<xsl:if
test=”expression”>
        <!-- Content: sequence-constructor -->
</xsl:if>

Example:
        <!-- Test if it is raining -->
        <xsl:variable name=”raining” select=”'yes'”/>
        <xsl:if test=”$raining='yes'”>
          Yes
        </xsl:if>
Output:
        Yes
```

Attribute – test (mandatory): The boolean condition to be tested.

xsl:when

xsl:when is always used as a child element of xsl:choose. It defines a condition to be tested and if the condition is true, the action to be performed.

```
<xsl:choose>
        <xsl:when
        test=”expression”>
                <!-- Content: sequence-constructor -->
        </xsl:when>
<xsl:choose>

Example:
        <!-- Test if it is raining -->
        <xsl:variable name="raining" select="'yes'"/>
        <xsl:choose>
                <xsl:when test="$raining='yes'">
                        Yes
                </xsl:when>
                <xsl:when test="$raining='maybe'">
                        May be
                </xsl:when>
        </xsl:choose>
Output:
        Yes
```

Attribute – test (mandatory): The boolean condition to be tested.

xsl:otherwise

xsl:otherwise is always used as a child element of xsl:choose. It the action to be performed when none of the xsl:when conditions is satisfied.

```
<xsl:choose>
	<xsl:when
	test=”expression”>
		<!-- Content: sequence-constructor -->
	</xsl:when>
	<xsl:otherwise>
		<!-- Content: sequence-constructor -->
	</xsl:otherwise>
<xsl:choose>

Example:
	<!-- Test if it is raining -->
	<xsl:variable name="raining" select="'yes'"/>
	<xsl:choose>
		<xsl:when test="$raining='yes'">
			Yes
		</xsl:when>
		<xsl:when test="$raining='maybe'">
			May be
		</xsl:when>
		<xsl:when test="$raining='no'">
			No
		</xsl:when>
		<xsl:otherwise>
			Don&apos;t know
		</xsl:otherwise>
	</xsl:choose>
Output:
	Yes
```

xsl:comment

xsl:comment is used to create comment node.

```
<xsl:comment
select=”expression”>
	<!-- Content: sequence-constructor -->
</xsl:comment>

Example:
	<xsl:comment select="/document/content/node()" />
Output:
	<!-- Content of the content node -->
Example:
```

```
        <xsl:comment select="'abc'" />
Output:
        <!-- abc -->
```

Attribute – select (optional): The sequence of values or nodes to be commented.

xsl:import

xsl:import Imports an external stylesheet with the required href attribute. The xsl:import element is always used at the top level of the stylesheet. When a stylesheet is imported into another stylesheet, the importing stylesheet has a higher precedence than the imported stylesheet.

```
<xsl:import href="URI"/>

Example:
        <xsl:import href="commons.xsl" />
```

Attribute – href (mandatory): The URI of the stylesheet to be imported.

xsl:output

The xsl:output element is a top level declaration used to control the output serialization of the resultant document.

```
<xsl:output
  method = "xml" | "html" | "xhtml" | "text"
  doctype-public = string
  doctype-system = string
  encoding = string
  indent = "yes" | "no"
  omit-xml-declaration = "yes" | "no"/>

Example:
        <xsl:output method="xml"/>
```

Attribute – method (optional): Defines the required output format.
Attribute – doctype-public (optional): Indicates the public identifier to be used in the DOCTYPE declaration.
Attribute – doctype-system (optional): Indicates the system identifier to be used in the DOCTYPE declaration.
Attribute – encoding (optional): Defines the character encoding of output file.
Attribute – indent (optional): indicates the source indentation of the output file.
Attribute – omit-xml-declaration (optional): Indicates whether an XML declaration should be included in the output file.

xsl:template

The xsl:template defines the processing instruction for the source elements or nodes of a particular type. When the optional name attribute is present, the template may be invoked directly using xsl:call-template.

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

        <!-- XSL Template with name -->
        <?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="/">
                <xsl:call-template name="test_template" />
        </xsl:template>
        <xsl:template name="test_template">
                <!-- Template Content -->
        </xsl:template>

</xsl:stylesheet>
```

Attribute – name(optional): It is an optional attribute that specifies a name for the template. If this attribute is omitted, there must be a match attribute.

Attribute - match(optional): It is an optional attribute that contains the match pattern for the template. If this attribute is omitted, there must be a name attribute.

xsl:call-template

The xsl:call-template element is used to invoke a named template. The name attribute is a required attribute.

```
<xsl:call-template name="template_name"/>

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">
        <!--  Main Template -->
        <xsl:template match="/">
                <xsl:call-template name="test_template" />
        </xsl:template>
        <xsl:template name="test_template">
                <!-- Template Content -->
        </xsl:template>
</xsl:stylesheet>
```

Attribute - name (mandatory): The name of the template to be called.

xsl:apply-templates

the xsl:apply-templates element defines the set of nodes to be processed.

```
<xsl:apply-templates
   select = expression
   mode = token>
   <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>

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">
        <!--  Main Template -->
        <xsl:template match="/">
                <xsl:apply-templates select=”document/content” />
        </xsl:template>
        <xsl:template match=”table”>
                <!-- Template Content -->
        </xsl:template>
</xsl:stylesheet>
```

Attribute – select (optional): Defines the sequence of nodes to be processed.
Attribute – mode (optional): Defines the processing mode. The default is an unnamed mode, however when a mode is defined, selected nodes must have the same mode.

xsl:stylesheet

Used to define the root element of the stylesheet. It is the outermost element of a stylesheet. The xsl:stylesheet should contain at least one namespace declaration, xmlns:xsl="http://www.w3.org/1999/XSL/Transform" is typically defined as it allows to identify the document as an XSLT stylesheet.

```
<xsl:stylesheet
version="version"

exclude-result-prefixes=tokens >
<!-- Content:(<xsl:import>*,top-level-elements) -->
</xsl:stylesheet>

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
        exclude-result-prefixes="ou">
        <!--  Main Template -->
        <xsl:template match="/">
                <!-- Template Content -->
        </xsl:template>
</xsl:stylesheet>
```

Attribute – version (mandatory): Defines the XSLT version required by the stylesheet.
Attribute – xmlns:[namespace] (mandatory): Defines the XSLT namespace declaration used in the stylesheet.
Attribute – exclude-result-prefixes (optional): Indicates any namespaces used in the stylesheet that should not be copied to the output destination.

xsl:copy

Used to create an attribute node and adds it to the result sequence. The <xsl:copy> element copies the context node to the resultant sequence. It is also referred as "shallow copy", which means that it does not copy the children, descendants, or attributes of the context node, only the context node itself with its namespaces (if its an element).

```
<xsl:copy
  copy-namespaces? = "yes" | "no"
  type? = qname>
  <!-- Content: sequence-constructor -->
</xsl:copy>

Example:
        <xsl:template match=" table | thead | tbody | tr | th | td ">
```

```
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

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

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

xsl:text

Used to output literal text to the resultant sequence.

```
<xsl:text
disable-output-escaping=”yes|no”
>
  <!-- Content: Character data -->
</xsl:text>

Example:
        <xsl:text disable-output-escaping="yes">
                &lt;div&gt; test &lt;/div&gt;
        </xsl:text>

Output:
        <div>test</div>
```

Attribute – name (mandatory): Defines the name of the function.

Attribute – disable-output-escaping (optional): Defines whether the output should be returned as is (“yes”) or XML escaped (“no”). The default is “no”.

Element

Constructs an element node

```
<xsl:element
  name = qname>
  <!-- Content: (sequence-constructor) -->
</xsl:element>

Example:
        <xsl:element name="test">
                <xsl:text>This is a test document</xsl:text>
        </xsl:element>
Output:
        <test>This is a test document</test>
```

Attribute – name (mandatory): Defines the name of the element.

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>

Example:
        <!-- Get all table body content -->
                <xsl:for-each select="//tbody/tr">
                        <xsl:copy-of select="current()"/>
                <xsl:for-each>
```

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

XSL Statements

Copy-of

Copies the value of the expression. The select attribute is a required attribute.

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

Comment

Creates a comment statement, which is useful for creating OU Campus tags.

```
<xsl:comment>This is a comment!</xsl:comment>
```

If statement

It is used for conditional processing. It takes a mandatory test attribute, with a boolean expression.

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

Import

Imports an external stylesheet with the required href attribute. The xsl:import element is always used at the top level of the stylesheet.

```
<xsl:import href="URI"/>
```

Output

The xsl:output element is a top level declaration used to control the output serialization of the resultant document.

```
<xsl:output
  method = "xml" | "html" | "xhtml" | "text"
  doctype-public = string
  doctype-system = string
  encoding = string
  indent = "yes" | "no"
  omit-xml-declaration = "yes" | "no"/>
```

Attribute – method (optional): Defines the required output format.
Attribute – doctype-public (optional): Indicates the public identifier to be used in the DOCTYPE declaration.
Attribute – doctype-system (optional): Indicates the system identifier to be used in the DOCTYPE declaration.
Attribute – encoding (optional): Defines the character encoding of output file.
Attribute – indent (optional): indicates the source indentation of the output file.
Attribute – omit-xml-declaration (optional): Indicates whether an XML declaration should be included in the output file.

When/Otherwise

Used within an xsl:choose element to indicate one of many choices. It takes test as a mandatory parameter with a match pattern. If the match pattern is successful, then the xsl:when element is expanded; otherwise, it is ignored.

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

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

Template

Defines the processing instruction for the source elements or nodes of a particular type. When the optional name attribute is present, the template may be invoked directly using xsl:call-template.

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

Attribute - name: It is an optional attribute that specifies a name for the template. If this attribute is omitted, there must be a match attribute.
Attribute - match: It is an optional attribute that contains the match pattern for the template. If this attribute is omitted, there must be a name attribute.

Call-template

This element is used to invoke a named template. The name attribute is a required attribute.

```
<xsl:call-template name="template_name"/>
```

Apply-templates

This element defines the set of nodes to be processed.

```
<xsl:apply-templates
   select = expression
   mode = token>
   <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

Attribute – select (optional): Defines the sequence of nodes to be processed.
Attribute – mode (optional): Defines the processing mode. The default is an unnamed mode, however when a mode is defined, selected nodes must have the same mode.

Stylesheet

Used to define the root element of the stylesheet. It is always the top-level element of an XSLT stylesheet. The version attribute is required, while the id attribute is an optional attribute used to uniquely identify a stylesheet. Additionally, it also specifies the included namespace prefixes for the extension elements.

```
<xsl:stylesheet
version="version"
exclude-result-prefixes=”namespace” >
<!-- Content:(<xsl:import>*,top-level-elements) -->
</xsl:stylesheet>
```

Attribute – version (optional): Defines the XSLT version required by the stylesheet.
Attribute – exclude-result-prefixes (optional): Indicates any namespaces used in the stylesheet that should not be copied to the output destination.

Processing Instruction

Writes a processing instruction to the output (<? processing language specific instructions ?>). It can be used to insert PHP code into a stylesheet.

```
<xsl:processing-instruction
name="process-name">
</xsl:processing-instruction>
```

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 stylesheet element. The name attribute is required.

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

Attribute – name (mandatory): Defines the name of the variable.
Attribute – select (optional): The value of the variable.
Attribute – as (optional): The return type of the variable.

Param

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

```
<xsl:param
name="name"
select="expression"
as="sequence-type"
>
</xsl:param>
```

Attribute – name (mandatory): Defines the name of the parameter.
Attribute – select (optional): The value of the parameter.
Attribute – as (optional): The return type of the parameter.

Attribute

Used to create an attribute node and adds it to the result sequence.

```
<xsl:attribute
  name = {qname}
  select = expression
  <!-- Content: sequence-constructor -->
</xsl:attribute>
```

Attribute – name (mandatory): Defines the name of the attribute node to be created.
Attribute – select (optional): Defines the value of the attribute.

For-each

Selects a sequence of items based upon an XPath expression and perform a set of operation on every item in the sequence.

```
<xsl:for-each
  select = sequence-expression>
</xsl:for-each>
```

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"/>
```

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.

Function

Used to create a stylesheet function that can be invoked by using a function call from an XPath expression.

```
<xsl:function
  name = qname
  as = sequence-type
  >
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:function>
```

Attribute – name (mandatory): Defines the name of the function.
Attribute – as (optional): Defines the return type of the function.

Copy

The xsl:copy instruction is a shallow copy, which is, it does not copy the children, descendent or attributes of the context node.

```
<xsl:copy>
  <!-- Content: (sequence-constructor) -->
</xsl:copy>
```

Text

Used to output literal text to the resultant sequence.

```
<xsl:text
disable-output-escaping=”yes|no”
>
  <!-- Content: Character data -->
</xsl:text>
```

Attribute – name (mandatory): Defines the name of the function.
Attribute – disable-output-escaping (optional): Defines whether the output should be returned as is (“yes”) or XML escaped (“no”). The default is “no”.

Attribute

Constructs an attribute node

```
<xsl:attribute
  name = qname
  as = sequence-type
  >
  <!-- Content: (sequence-constructor) -->
</xsl:attribute>
```

Attribute – name (mandatory): Defines the name of the attribute.

Element

Constructs an element node

```
<xsl:element
  name = qname>
  <!-- Content: (sequence-constructor) -->
</xsl:element>
```

Attribute – name (mandatory): Defines the name of the element.

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
```

Tokenize

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

```
tokenize(string,deliminator)
Example: <xsl:value-of select=”tokenize(‘one, two, three’)”/>
Output:
```

Position

Returns the value of the context position.

```
position()
Example: <xsl:value-of select=”position”/>
Output: 1
```

Current

Returns the context item at the point in the stylesheet where the XPath expression is being called.

```
current()
Example: <xsl:copy-of select=”current()”/>
Output:
```

Unparsed-text

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

```
Unparsed-text(string,deliminator)
Example: <xsl:value-of select=”tokenize(‘one, two, three’)”/>
Output:
```

Document/Doc

Document or doc (XSLT 2.0) function locates an XML document with the provided URI and the resulting XML document is parsed and a tree is constructed. Upon completion the resultant output is the document node of the new document.

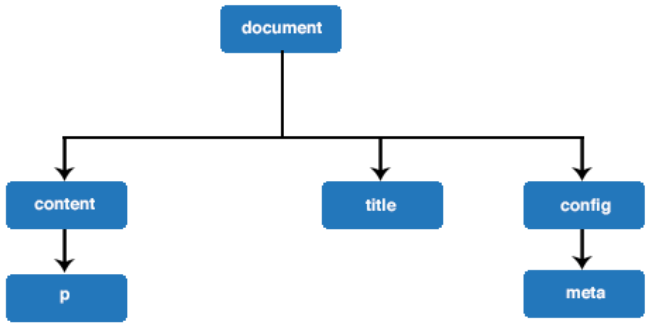
```
<xsl:value-of select=”document(URI)”/>
```

Introduction to XPath

XPath (XML Path Language) is a language used to address parts of an XML document. XPath models an XML document as a tree and XPath locator syntax uses core functions based on the node hierarchy of a document, and evaluates expressions to determine a location object.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "http://commons.omniupdate.com/dtd/standard.dtd">
<document>
	<config>
		<meta name="author" content="John Doe" />
	</config>
	<title>
		New Page
	</title>
	<content>
		<p>Hello World</p>
	</content>
</document>
```

Every XML document can be represented in a structural form called tree. A XML tree originates from a single point, called root. All the child elements that follow the root can be referred as branches or nodes. A XML document tree can be divided into smaller tree in such a way that any node can be considered the root of its own subtree. Following is the tree representation of the XML document above.



Based on the XML document and tree structure, let's evaluate some XPath expressions:

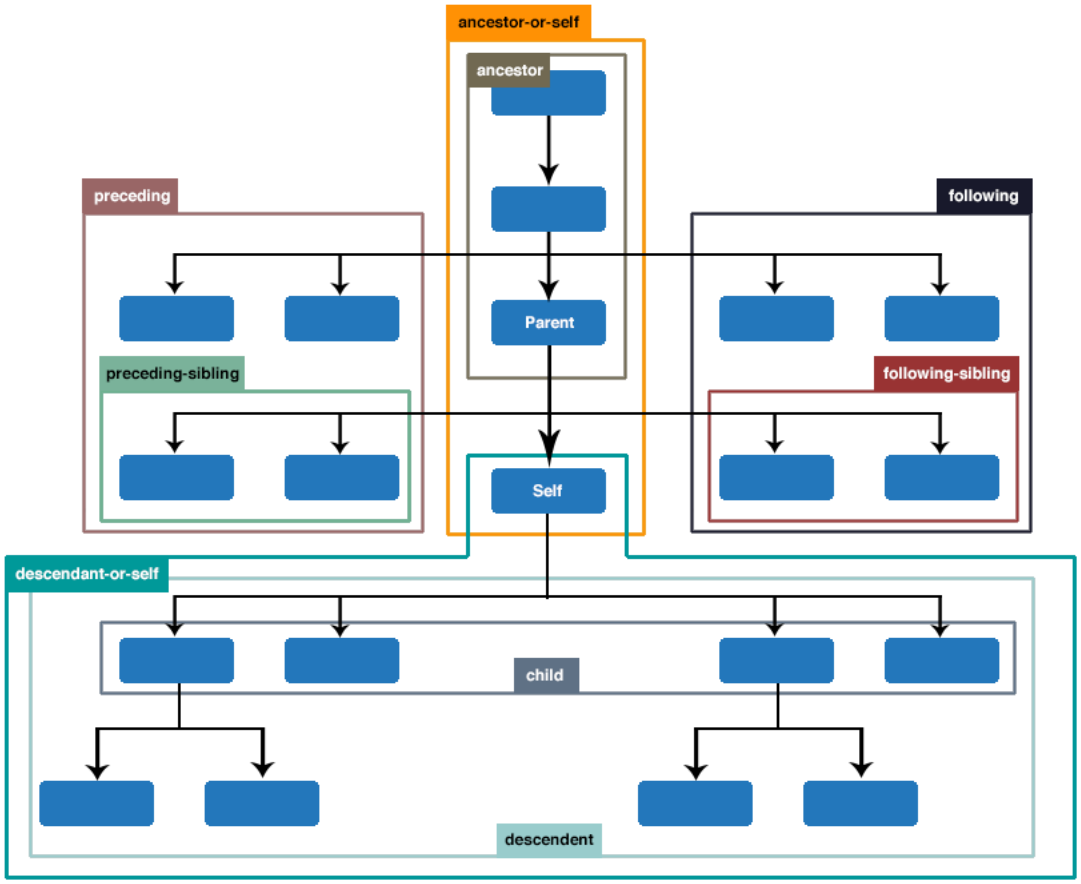
```
XPath expression:
/document/title

Result:
title node
```

XPath expression:
/document/content

Result:
“content” and “p” node

Axis selection in XPath

With XPath you can select nodes based on relative paths. Axis specifies the tree relationship between the nodes selected by the location step and the context node. The following are the available axes:



child

Contains the child elements of the context node.

descendant

Contains the descendants of the context node, which is a child or a child of a child and so on.

descendant-or-self

Contains the context node and the descendants of the context node.

preceding

Contains all nodes in the same document as the context node that precedes the context node in document order, excluding any ancestors.

preceding-siblings

Contains all the preceding siblings of the context node; if the context node is an attribute node or namespace node.

following

Contains all nodes in the same document as the context node that follows the context node in document order, excluding any descendants.

following-siblings

Contains all the following siblings of the context node; if the context node is an attribute node or namespace node.

ancestor

Contains the ancestors of the context node and all its parent nodes.

ancestor-or-self

Contains the context node along with the ancestors of the context node and all its parent nodes.

System Defined XPath Functions

concat()

Concat takes two or more arguments and returns 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.

lower-case()

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

```
lower-case(string)

Example: <xsl:value-of select=”lower-case(‘Hello World’)”/>
Output: hello world
```

Argument – arg (mandatory): String argument to be converted into lower 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, before 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-before('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
```

OU Campus XSLT Variables

OU Campus provides a number of XSLT variables that allow developers to create powerful and robust XSL templates.

ou:action

The ou:action variable allows developers to determine the current state of the document in OU Campus. There are four states in the OU Campus system, preview (prv), publish (pub), edit (edt) and compare (cmp).

```
<xsl:param name="ou:action" />
<xsl:if test="$ou:action='prv'">
        This is preview mode
</xsl:if>
```

ou:root

The ou:root variable allows users to determine the path from the system root in OU Campus.

```
<xsl:param name="ou:root" />
<xsl:value-of select="$ou:root"/>

<!--Output: /usr/local/oucampus/main_site/ -->
```

ou:site

The ou:site variable outputs the name of the existing site.

```
<xsl:param name="ou:site" />
<xsl:value-of select="$ou:site"/>

<!--Output: main_site -->
```

ou:path

The ou:path variable outputs the location of the working PCF file.

```
<xsl:param name="ou:path" />
<xsl:value-of select="$ou:path"/>

<!--Output: /level1/index.html -->
```

ou:dirname

The ou:dirname variable outputs the location of the existing working folder.

```
<xsl:param name="ou:dirname" />
<xsl:value-of select="$ou:dirname"/>

<!--Output: /level1 -->
```

ou:filename

The ou:filename variable outputs the name of the output file.

```
<xsl:param name="ou:filename" />
<xsl:value-of select="$ou:filename"/>

<!--Output: index.html (from index.pcf)-->
```

ou:httproot

The ou:httproot variable outputs the HTTP root defined in the site setup.

```
<xsl:param name="ou:httproot" />
<xsl:value-of select="$ou:httproot"/>

<!--Output: http://www.gallenauniversity.com/ -->
```

ou:ftproot

The ou:ftproot variable outputs the FTP root defined in the site setup.

```
<xsl:param name="ou:ftproot" />
<xsl:value-of select="$ou:ftproot"/>

<!--Output: /public_html -->
```

ou:ftphome

The ou:ftphome variable outputs the FTP root defined in the site setup.

```
<xsl:param name="ou:ftphome" />
<xsl:value-of select="$ou:ftphome"/>

<!--Output: /public_html -->
```

ou:ftpdir

The ou:ftpdir variable outputs the FTP home defined in the site setup.

```
<xsl:param name="ou:ftpdir" />
<xsl:value-of select="$ou:ftpdir"/>

<!--Output: /public_html -->
```

ou:username

The ou:username variable outputs the user's username.

```
<xsl:param name="ou:username" />
```

```
<xsl:value-of select=”$ou:username”/>

<!--Output: jdoe -->
```

ou:lastname

The ou:lastname variable outputs the user’s last name.

```
<xsl:param name="ou:lastname" />
<xsl:value-of select=”$ou:lastname”/>

<!--Output: Doe -->
```

ou:firstname

The ou:firstname variable outputs the user’s first name.

```
<xsl:param name="ou:ftphome" />
<xsl:value-of select=”$ou:ftphome”/>

<!--Output: John -->
```

ou:email

The ou:email variable outputs the user’s email address.

```
<xsl:param name="ou:email" />
<xsl:value-of select=”$ou:email”/>

<!--Output: jdoe@gallenauniversity.com -->
```

ou:feed

The ou:feed variable outputs the RSS feed assigned to a document (if any).

```
<xsl:param name="ou:feed" />
<xsl:value-of select=”$ou:feed”/>

<!--Output: /rss/feed.xml -->
```

ou:email

The ou:modified variable outputs the most recent modified date of a document.

```
<xsl:param name="ou:modified" />
<xsl:value-of select=”$ou:modified”/>

<!--Output: 2011-11-21T22:08:00Z -->
```

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/