Element Transformation & Validation (ETV / XTT)

Posted Jan 24, 2014 • Updated Apr 24, 2019 • Read 10709 times

Tool Type	Tool / Template
Deployment Stage	Configure & Prototype
Product	Integrations
Audience	Integration Consultant
Services Product Lead Area	Integrations

Table Of Contents

Introduction

Additional References

Credits: Thanks to the original authors of the below content Doug Lee and James Pasley for putting this together.

Introduction

The Element Transformation and Validation (ETV) step transforms and validates the elements within an XML document as instructed by attributes attached to those elements. For example, the following XML document contains attributes specifying that the maximum length for City is 30 and that values longer than that should be truncated.

```
<?xml version="1.0" encoding="utf-8"?>
  <Sample xmlns:etv="urn:com.workday/etv">
  <City etv:truncate="true"
etv:maxLength="30">Llanfairpwllgwyngyllgogerychwyrndrobwyllllantysil
iogogogoch</City>
</Sample>
```

If this sample document is processed using the ETV step, it will be transformed to the following XML document:

```
<?xml version="1.0" encoding="utf-8"?>
  <Sample>
  <City>Llanfairpwllgwyngyllgogerychwy</City>
</Sample>
```

Subscriptions

Manage all My Subscriptions

Subscribe

Note that the value has been truncated to thirty characters and that the etv attributes are removed as part of the transformation.

The ETV step also supports a number of validation rules. For example, in this sample message the etv:required attribute indicates that the Social_Security_Number element is required to have a value.

```
<?xml version="1.0" encoding="utf-8"?>
<Sample xmlns:etv="urn:com.workday/etv">
  <Social_Security_Number etv:required="true">
</Social_Security_Number>
</Sample>
```

This sample doesn't contain any attributes that will transform the contents of the document, so if this sample document is processed using the ETV step, the resulting document will simply have the ETV attributes removed.

```
<?xml version="1.0" encoding="utf-8"?>
<Sample>
  <Social_Security_Number></Social_Security_Number>
</Sample>
```

In addition to this, the following error message will be generated. Such error messages can be reported on the integration event or in audit files.

```
No value is available for Social Security Number. Social Security
Number is a required field.
```

The ETV step provides a number of different ways to transform and validate elements in an XML document. It also provides number and date formatting rules and provides a mechanism to count elements or calculate the sum of their contents.

A second step called XML To Text (XTT) uses a similar approach to convert XML documents to text. It converts documents from XML to text formats such as comma separated values (CSV), fixed width or other text formats. For example, the following XML document includes attributes telling the XTT step that each row is separated by a newline, each item by a comma and each value is quoted according to the rules for CSV files:

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns:xtt="urn:com.workday/xtt" xtt:separator="&#xD;&#xA;"
xtt:quotes="csv">
  <Row xtt:separator=",">
    <Item>A1</Item>
    <Item>B1</Item>
    <Item>C1</Item>
  </Row>
  <Row xtt:separator=",">
    <Item>A2</Item>
    <Item>B2</Item>
    <Item>C2</Item>
  </Row>
  <Row xtt:separator=",">
    <Item>A3</Item>
    <Item>B3</Item>
    <Item>C3</Item>
  </Row>
</Document>
```

When the above document is processed using the XTT step, it will be transformed to the following text document:

```
A1,B1,C1
A2,B2,C2
A3,B3,C3
```

The XTT step also provides all the functionality of the ETV step. All attributes supported by the ETV step are also supported by the XTT step. This allows the XTT step be used standalone or in conjunction with the ETV step. Where the two steps are used together, the namespace is used to dictate which step will process which attribute. Attributes in the urn:com.workday/etv namespace will be processed by the ETV step. Attributes in the urn:com.workday/xtt will pass through the etv step and be processed by the XTT step.

The ETV and XTT steps are typically used in conjunction with XSLT. The XSLT style sheet is used to create the XML document that will be processed by these steps. A style sheet that generates the social security number sample message shown above would look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:etv="urn:com.workday/etv">
  <xsl:template match="/*">
    <Sample>
      <Social_Security_Number etv:required="true">
        <xsl:value-of select="Worker/PersonalData/SSN" />
      </Social_Security_Number>
    </Sample>
  </xsl:template>
</xsl:stylesheet>
```

Note how the etv:required attribute is added to the social security number element. This approach significantly simplifies the code written in XSLT. Much of the XSLT contains simple xsl:value-of statements that copy values from the input document to the output. Attributes are added indicating what further processing should be applied to the elements.

Overview

This section provides an overview of the attributes supported by the ETV and XTT step. The attributes are grouped together by functional area and a brief description of each of is provided. See the reference section for a complete description of each attribute.

Validation

The following attributes apply validation rules to the contents of the element they are attached to:

- **enumeration** - A comma separated list of values. The element must contain one of the values listed.
- **maxLength** – The maximum length of the value in the element.
- **minLength** - The minimum length of the value in the element.
- **required** - The element much contain at least one character.

These next attributes provide control over the message that is reported when an element fails a validation rule:

- **severity** - The severity of the message. One of info, warning, error or critical.
- **target** - The target of the validation message. For example, when processing employee data, this would typically be the employee name.
- **targetWID** - The Workday ID of the target, this is used to provide a hyperlink in the message.

The messages use the name of the element, with underscores replaced with space, to refer to the value that failed the validation check. This is a useful convention where the elements can be given names that are meaningful to an end user. The following two attributes provide alternatives to this approach which can be used where the element names are not appropriate. It is anticipated that in most cases the element names will be sufficient and it will be unnecessary to use either of these attributes.

- **name** - Use this attribute to provide a name to be used in validation messages instead of the element name.
- **nameAttribute** - This attribute contains the qualified name of another attribute that contains the name to be used in validation messages.

You might require other validation rules not supported by these existing set of attributes. Such validation rules could be implemented in XSLT a nd the message attribute can be used to add your message to the list of messages generated by the ETV or XTT step.

- **message** - The text of a message to be output in the same way as a message resulting from one of the validations.

Truncation

The maxLength attribute specifies the maximum number of characters that an element should have. Unless otherwise stated, an element that exceeds this length will be reported with an error message. The truncate attribute allows you to state that the value should be truncated.

- **truncate** - A Boolean value indicating if elements that exceed their maximum length should be truncated.
- **reportTruncation** - Controls whether the truncation of a value should be reported in a validation message or by adding an attribute to the XML docume nt itself indicating that the value is truncated.
- **fieldLengthCalculation** - Specifies whether the length should be measured in characters or bytes.

Date/Time Formatting

The following attributes allow a date or dateTime format to be specified. The element values provided as input to the steps should be formatted as defined by the XML Schema specification. The values will be converted to the format specified in the attributes.

- **dateFormat** - The date format pattern, for example “yyyyMMdd”.
- **dateTimeFormat** - The date time format pattern, for example “yyMMdd:HH:mm:ss”.
- **timezone** - The timezone to be used when creating the value for the output document, for example “PST”.

Number Formatting

The following attributes allow a number format to be specified. The element values provided as input to the steps should be formatted as defined by the XML Schema specification. The values will be converted to the format specified in the attributes.

- **numberFormat** - The number format pattern, for example “#,###.00”.
- **decimalSeparator** - The decimal separator character.
- **groupingSeparator** - The grouping separator character.
- **scale** – The position of an implied decimal place in the formatted value.

Workday Integration System

The following attributes replace the value of an element with a value from the Integration System associated with the integration that the step is used in.

- **attribute** – The name of an integration attribute. The element value is replaced with the value from the integration attribute.
- **launchParameter** – The name of a launch parameter. The element value is replaced with the value from the launch parameter.
- **sequencedValue** - The name of a sequenced value. The element value is replaced with the value from the sequenced value.
- **map** – The name of an integration map. The element value is replaced with the equivalent value as specified in the integration map.
- **direction** – The direction that the integration map should be applied in. This attribute controls whether the map attribute uses an integration map to convert an internal Workday value to an external value or vice versa.
- **mapReferenceID** – If the integration map named by the map attribute is empty, then the value of the reference id named in this attribute will be used.

Note that integration attributes, launch parameters or integration maps that are defined using report fields may contain references to objects as opposed to simple types. If a

reference is found, the ETV step will place a reference id in the output file. See the details descriptions of these attributes for sample messages demonstrating this.

Counting or Summing Elements

These attributes provide a way to count the number of times an element occurs or to calculate the sum total of the values in a number of elements. The resulting number can then be placed into another element. This is a common requirement for text formats were a footer might be required to contain such totals. Each of these attributes contains the name of a variable or a comma separated list of variable names.

- **setNumber** – The variable is set to the value of the element.
- **addNumber** – The value of the element is added to the existing value of the variable.
- **incrementNumber** – The value of the variable is incremented by one.
- **number** – The value of the element is replaced by the value of the variable.

Scope Rules

Many attributes (such as dateFormat or maxLength) are intended to be attached to elements of simple types and only apply to the elements to which they are attached. Others (such as target or truncate) can be attached to any element and their value is inherit ed by child elements.

Reusing Attributes

The ETV and XTT steps allow groups of attributes to be defined as reusable classes. These are defined using an element called class and can be later referred to using the attribute class.

- **class** - The name of a class of attributes to apply this an element.

Classes are often used to define reusable number or date formats as shown by the following example:

```
  <etv:class etv:name="decimal" etv:numberFormat="##,##.00"
etv:decimalSeparator="," etv:groupingSeparator="." />

  <Salary etv:class="decimal">10000</Salary>
```

When the above document is processed using the XTT step, it will be transformed to the following:

```
  <Salary>10.000,00</Salary>
```

Note that the class element is not copied to the output document.

Using Multiple Attributes Together

Different attributes may be used together on the same elements. For example, the value of a launch parameters could be extract, mapped using an integration map and placed in the output file. Where multiple attributes are used together, they are applied in the following order:
1. Data extraction: For example *attribute*, *launchParameter*, *sequencedValue* and *number*.
2. Apply Integration Map
3. Number addition : For example, *addNumber*
4. Formatting: For example: *numberFormat*, *dateFormat*
5. Truncation
6. Validation

Note: Only one of the *attribute*, *launchParameter*, *sequencedValue* or *number* attributes should be present on a single element. Where more than one of these attributes is present, the behavior is undefined.

Format as Comma Separate Values

The following attributes provide support for formatting documents as comma separated values. The separator and quotes attributes are typically used together to indicate whether the separators should be added. The example shown in the introduction demonstrates how these attributes are used together. Both comma and pipe delimited formats can be created using a combination of the separator and quotes attributes. If other delimiters are required, the quotesWhenMatches attribute can be used to specify a regular expression that will be used to determine if a value should be quoted.

- separator - The characters to be added the output document between each of the child elements of the element that this attribute is attached to.
- quotes - The rules for when an element value should be quoted - possible values are “csv”, “pipe-delimited”, “always” and “never”.
- quoteWhenMatches - A regular expression that can be used to determine if the elements values should be included in quotes.

Format as Fixed Length

The following attributes allow fixed width format documents to be created. The fixed length of each element is specified using the fixedLength attribute. Values shorter than this will be padded using the paddingCharacter. Values that are longer will be truncated.

- fixedLength - The number of characters in this fixed length field.
- paddingCharacter - The padding character to use if the value is shorter than the fixed length.
- align - Indicates whether values should be aligned to the left or to the right when padding is added.

Other Formats

The following attributes allow additional text to be added before or after an element when it is converted to a text document. These attributes can be useful when creating documents that use more complex text file formats such as EDI.

- startTag - The contents of the attribute will be placed before the value of the element in the output document.
- endTag – The contents of the attribute will be placed after the value of the element in the output document.

Miscellaneous

The following attribute indicates that the contents of an element should not be copied to the output document. It may be useful to use this attribute with the XTT step where the XML document created by an XSLT is used for a number of different purposes before it’s converted to a text document. For example, the XML might be used with another style sheet to generate an audit file and so it may need to contain additional information not intended for the final text document. The omit attribute might also
be used setNumber. For example, an additional element might be introduced into the XML document containing a setNumber attribute purely for the purpose of resetting a variable to zero.

- omit – The contents of the element will not be copied to the output document.

Sample Use Case

The section contains a simple example showing how the XTT step can be used together with an XSLT step to generate the CSV file contain employee data. The CSV file should contain the employee’s name, social security number, address, date of birth and gender.

Below is an example XSLT style sheet that demonstrates how the XML retrieved from a web service might be reformatted for processing by the XTT step. The XSLT loops over the employees copying only the data required in the CSV file. The attributes that drive the XTT step are add ed by the XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:wd="urn:com.workday/bsvc" version="2.0"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    xmlns:xtt="urn:com.workday/xtt">
  <xsl:output indent="yes" method="xml" />
  <xsl:template match="/*">
      <Employees xtt:truncate="true" xtt:severity="error">
        <xtt:class xtt:name="date" xtt:dateFormat="dd/MM/yyyy" />
        <xsl:for-each select="wd:Worker">
        <!-- Variable declarations have been omitted. -->
        <Employee xtt:target="{$EmployeeName}"
            xtt:targetWID="{$EmployeeWID}"
            xtt:quotes="csv"
            xtt:endTag="&#xD;&#xA;">
          <Personal_Data xtt:separator=",">
            <First_Name xtt:required="true">
              <xsl:value-of select="$NameData/wd:First_Name" />
            </First_Name>
            <Last_Name xtt:required="true">
              <xsl:value-of select="$NameData/wd:Last_Name" />
            </Last_Name>
            <Social_Security_Number xtt:required="true">
              <xsl:value-of select="$SSN" />
            </Social_Security_Number>
            <Address_Line_1 xtt:required="true" xtt:maxLength="10">
              <xsl:value-of
select="$Address/wd:Address_Line_Data[1]" />
            </Address_Line_1>
            <Address_Line_2 xtt:maxLength="50">
              <xsl:value-of
select="$Address/wd:Address_Line_Data[2]" />
            </Address_Line_2>
            <City xtt:required="true" xtt:maxLength="20">
              <xsl:value-of select="$Address/wd:Municipality" />
            </City>
            <State>
              <xsl:value-of select="$State" />
            </State>
            <Zip>
              <xsl:value-of select="$Address/wd:Postal_Code"/>
            </Zip>
            <BirthDate xtt:class="date">
              <xsl:value-of select="$PersonalData/wd:Birth_Date" />
            </BirthDate>
            <Gender xtt:map="Gender">
              <xsl:copy-of
select="$PersonalData/wd:Gender_Reference" />
            </Gender>
          </Personal_Data>
        </Employee>
      </xsl:for-each>
     </Employees>
  </xsl:template>
</xsl:stylesheet>
```

There are a number of features to note about the above XSLT.

- The XML generated by the XSLT dictates the structure of the CSV file. So every element is created for each employee regardless of whether or not that employee has a value for that field. This approach ensures that each line has the correct number of commas.
- The target and targetWID attributes are set at the employee level so that any error messages will include the employee name.
- The XSLT itself is quite simple–most statements simple copy the value from the Web service response to the output format.
- Note that applying the Gender integration map to the simply requires that the reference ID is copied to the output using a copy-of statement and the map selected using the map attribute.
- The xtt:class element is used to define a date format pattern. This class is referred to by the class attribute on the BirthDate element.

The XML generated by such a style sheet will look as follows:

```
<Employees
        xmlns:xtt="urn:com.workday/xtt"
        xmlns:wd="urn:com.workday/bsvc"
        xtt:truncate="true"
        xtt:severity="error">
    <xtt:class xtt:name="date" xtt:dateFormat="yyyy/MM/dd" />
    <Employee
            xtt:target="Logan McNeil"
            xtt:targetWID="66a65677834f0a9b586dd5645e9004"
            xtt:quotes="csv"
            xtt:endTag="&#xD;&#xA;">
        <Personal_Data xtt:separator=",">
            <First_Name xtt:required="true">Logan</First_Name>
            <Last_Name xtt:required="true">McNeil</Last_Name>
            <Social_Security_Number xtt:required="true">
</Social_Security_Number>
            <Address_Line_1 xtt:required="true"
xtt:maxLength="10">3870 Pacific Avenue</Address_Line_1>
            <Address_Line_2 xtt:maxLength="50" />
            <City xtt:required="true" xtt:maxLength="20">San
Francisco</City>
            <State>CA</State>
            <Zip>94111</Zip>
            <BirthDate xtt:class="date">1972-05-25-08:00</BirthDate>
            <Gender xtt:map="Gender">
                <wd:Gender_Reference wd:Descriptor="Female">
                    <wd:ID
wd:type="WID">77da47525a434e98894dd95c641a68d4</wd:ID>
                    <wd:ID wd:type="Gender_Code">245.1</wd:ID>
                </wd:Gender_Reference>
            </Gender>
        </Personal_Data>
    </Employee>
</Employees>
```

When this is passed through the ETV step, the following XML is generated:

```
Logan,McNeil,,3870 Pacif,,San Francisco,CA,94111,1972/05/25,F
```

As the social security number is missing from the above example, the following error message will also be generated:

```
No Social Security Number available for Logan McNeil. Social
Security Number is a required field.
```

Reference

This section provides documentation of each individual attribute. An example of each is provided. The sample output provided is formatted as XML for each of the attributes supported by the ETV step. All attributes supported by the ETV step are also supported by the XTT
step. The attributes are handled in an equivalent way by the XTT step, but the output will be text.

addNumber:

Summary: The addNumber attribute adds the numeric value of the current element to a variable.
Valid Values: A comma separated list of variable names.
Scope: Element only.
The addNumber attribute adds the numeric value of the current element to the variable named in the attribute. Typically, a single variable will be updated. Multiple variables can be updated by providing a comma separated list of variable names.
The following example shows how the addNumber attribute is used to calculate the total of two record elements:

```
<Sample>
  <record etv:addNumber="aNumber">3.56</record>
  <record etv:addNumber="aNumber">1.01</record>
  <result etv:number="aNumber"/>
</Sample>
```

This XML will result in the following text:

```
<Sample>
  <record>3.56</record>
  <record>1.01</record>
  <result>4.57</result>
</Sample>
```

Note that the result element is populated with the value of the a Number variable.

align

Summary: The align attribute indicates whether fixed length values that require padding are aligned to the left or the right.
Valid Values: One of “left” or “right”. The default is “left”.
Scope: Element and all child elements.
The align attribute indicates whether fixed length values that require padding are aligned to the left or the right. The following example shows how the align attribute is used with the fixedLength attribute:

```
<Sample xtt:align="right">
  <item xtt:fixedLength="10">Short</item>
  <item xtt:fixedLength="10">Much Too Long</item>
  <item xtt:fixedLength="10">Just Right</item>
</Sample>
```

This XML will result in the following text:

```
ShortMuch Too LJust Right
```

attribute

Summary: The attribute called attribute the name of an Integration Attribute whose value is to be placed into the element.
Valid Values: The name of an integration attribute.
Scope: Element only.
The attribute attribute may be used to replace the value of an element with the value of an integration attribute. NOTE that xsl:param can also be used to get the value of an integration attribute. See further below.
An example is shown here:

```
<Cost_Center etv:attribute="Cost Center" />
```

The Element Transformation step will replace the value of this element with the value of the integration attribute.

```
<Cost_Center>CC46</Cost_Center>
```

The integration attribute may be defined on the integration system as a Reference Ids. If the integration attribute contains a Reference Id, then a wd:ID element will be created as shown in this example:

```
<Organization_Reference etv:attribute="Cost Center"/>
```

In this case, the following output will be created:

```
<wd:Organization_Reference wd:Descriptor="Cost Center 46">
<wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
</wd:Organization_Reference>
```

Note: If the attribute contains multiple values, then the ETV step will output the element multiple times, once for each value. The XTT step will output the values as text, concatenated together.

Using **xsl:param** to retrieve integration attribute value. The use case for this method would be when the attribute value is part of XSLT logic. Eg: Output file needs a Test flag "T" to be set and Integration attribute is of boolean type.

The name on the xsl:param is formed by prefixing the Integration Attribute name with "attr_" and replacing any spaces with an underscore. The param will contain the text value of any integration attribute value. Multi-valued attributes will be represented as a comma separated list. For example, if you had an integration attribute named "Root Organization" then the xsl:param is defined as below:

```
<xsl:param name="attr_Root_Organization"/>
```

> In order for the "attr_" prefix to work, the param element must not set as a local parameter within a template in the XSLT. It must be a global parameter.

class

Summary: Applies a number of ETV attributes to an element.
Valid Values: String – the name of a class defined using the etv:class element.
Scope: Element only.
The etv:class element allows one or more etv:attributes to be defined as part of a class. The etv:class attribute can then be used as a convenient way to apply all of those attributes. The attributes included in the etv:class element are only applied to the element that the etv:class attribute is attached to regardless of the scoping rules for the original attribute. The class attribute can be used in conjunction with other attributes. An attribute placed directly on an element will take precedence over the same within the class.
An example is shown here:

```
<etv:class etv:name="decimal" etv:numberFormat="##,##.00"
etv:decimalSeparator="," etv:groupingSeparator="."/>
<Salary etv:class="decimal">10000</Salary>
```

In this example, a class called decimal is defined using the etv:class element. This class contains three attributes: etv:numberFormat, etv:decimalSeparator and etv:groupingSeparator. This class is later used to format the contents of the Salary element by referring to it using the etc:class attribute.
A class defined using the etv:class element is visible on subsequent sibling elements and their children.
The Element Transformation step will remove the etv:class element from the document and replace the value of the Salary element with the reformatted value.

```
<Salary>10.000,00</Salary>
```

The definition of one class can refer to another class as shown below. The class will inherit all attributes from the parent class. These may be added to oroverridden by defining additional attributes.

```
<etv:class etv:name="requiredDecimal" etv:class="decimal"
etv:required="true"/>
```

dateFormat

Summary: The dateFormat attribute specified the date format pattern that dates will be converted to.
Valid Values: A date format pattern as used in Java.
Scope: Element only.
The dateFormat attribute may be used to specify a date format. The value of the element will be processed as a date and converted to the format specified in this attribute. An example is shown here:

```
<Birth_Date etv:dateFormat="dd/MM/yyyy">1960-03-24</Birth_Date>
```

The Element Transformation step will replace the value of this element the reformatted value.

```
<Birth_Date>24/03/1960</Birth_Date>
```

dateTimeFormat

Summary: The dateTimeFormat attribute specified the dateTime format pattern that dateTimes will be converted to.
Valid Values: A date format pattern as used in Java.
Scope: Element only.
The dateTimeFormat attribute may be used to specify a date format. The value of the element will be processed as a dateTime and converted to the format specified in this attribute. An example is shown here:

```
<Last_Update etv:dateFormat="dd/MM/yyyy:hh/mm/ss" >2010-10-
13T07:30:45</Last_Update>
```

The Element Transformation step will replace the value of this element the reformmatted value.

```
<Last_Update>13/10/2010:07/30/45</Last_Update>
```

decimalSeparator

Summary: Specified the decimal separator to be used when reformatting numbers.
Valid Values: Any character.
Scope: Element and all child elements.
The decimalSeparator attribute is used to specify the symbol that will be used as the decimal separator when numbers are reformatted. An example is shown here:

```
<Salary etv:numberFormat="##,###.00" etv:decimalSeparator=","
etv:groupingSeparator=".">10000</Salary>
```

The Element Transformation step will replace the value of this element the reformmatted value.

```
<Salary>10.000,00</Salary>
```

direction

Summary: The direction attribute specifies the direction to be used when applying an integration maps
Valid Values: One of “out”, “in”. The default value used if this attributes is not set is “out”.
Scope: Element and all child elements.

Integration maps contain two sets of values: the internal values and the external values. By default, the map attribute will convert an internal value to an external value. The direction attribute allows an integration map to be used to convert external values to internal ones. The default value for direction is “out” and this means that internal values will be mapped to external ones. When the direction is set to “in”, external values will be mapped to internal values.
To use an direction attribute with map, add it to the element containing the map attribute or to a parent element of that element. An example is shown here:

```
<Bank_Account_Type etv:direction="in"
etv:map="AccountType">Checking</Bank_Account_Type>
```

The value of the element will be replaced with the internal value from the integration map, creating the output shown below.

```
<Bank_Account_Type>C</Bank_Account_Type>
```

The internal values in an integration map can also be Reference Ids. If the map contains Reference Ids, then a wd:ID element will be created as shown in this example:

```
<wd:Worker_Type_Reference etv:direction="in" etv:Map="Employee
Type">SEASONAL</wd:Worker_Type_Reference>
```

In this case, the following output will be created:

```
<wd:Worker_Type_Reference wd:Descriptor="Seasonal">
  <wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
</wd:Worker_Type_Reference>
```

Note: If the direction is “in” and the map contains multiple internal values contains multiple values for the mapped value, then the ETV step will output the element multiple times, once for each value. The XTT step will output the values as text, concatenated together.

endTag

Summary: The value of the endTag attribute will be placed in the output file after the value of the element.
Valid Values: Any text
Scope: Element only.
The value of the endTag attribute is placed in the output document after the value of the element it is attached to. For example:

```
<Sample xtt:endTag=":END">Hello World</Sample>
```

This XML will result in the following text:

```
Hello World:END
```

enumeration

Summary: The enumeration attribute provides a way to validate that an element contains one of a list of enumerated values.
Valid Values: A comma separated list of values.
Scope: Element only.
The list of values provided in the enumeration attribute are used to validate the contents of the element to which they it is attached. The following example demonstrates the use of this attribute.

```
<Sample>
  <A etv:enumeration="this,that">this</A>
```

```
  <B etv:enumeration="this,that">other</B>
</Sample>
```

The enumeration attribute doesn't alter the contents of the elements. The following output message is generated:

```
<Sample>
  <A>this</A>
  <B>other</B>
</Sample>
```

Element A will pass the validation check so no message is generated for it. Element B will fail and the following validation message will be created.

```
<wdext:Integration_Message>
  <wdext:Summary>The value of B is invalid</wdext:Summary>
</wdext:Integration_Message>
```

fieldLengthCalculation

Summary: The fieldLengthCalculation attribute controls how the length of a value is calculated for use with the maxLength attribute.
Valid Values: Either the value " characters " or "bytes:" followed by the character encoding. The default value is "characters".
Scope: Element only.
The fieldLengthCalculation attribute indicates whether the length of a field should be counted in characters or in bytes. If it is to be counted in bytes, then the character encoding should also be provided.
The following example shows how the fieldLengthCalculation attribute is used with the maxLength attribute:

```
<Sample etv:truncate="true" etv:fieldLengthCalculation="bytes:utf-8"
etv:maxLength="6">Straße</Sample>
```

This XML will result in the following text:

```
<Sample>Straß</Sample>
```

fixedLength

Summary: The fixedLength attribute indicates the length of a field width field.
Valid Values: A positive integer.
Scope: Element only.
The fixedLength attribute indicates that the values of the element should be placed in a fixed width field. Values shorter than this length will be padded out to the length specified. Values will be padded with the space character. An alternative character can be specified using the paddingCharacter attribute.
Values longer than the length will be truncated. Values are truncated without any error or warning messages. The maxLength attribute should also be used if error or warning messages are required.
The following example shows how the fixedLength attribute is used:

```
<Sample>
  <item xtt:fixedLength="10">Short</item>
  <item xtt:fixedLength="10">Much Too Long</item>
  <item xtt:fixedLength="10">Just Right</item>
</Sample>
```

This XML will result in the following text:

```
Short     Much Too LJust Right
```

groupingSeparator

Summary: Specified the grouping separator to be used when reformatting numbers.
Valid Values: Any character.
Scope: Element and all child elements.
The groupingSeparator attribute may be used to specify the symbol that will be used as the grouping separator when numbers are reformatted. An example is shown here:

```
<Salary etv:numberFormat="##,###.00" etv:decimalSeparator=","
etv:groupingSeparator=".">10000</Salary>
```

The Element Transformation step will replace the value of this element the reformmatted value.

```
<Salary>10.000,00</Salary>
```

incrementNumber

Summary: The incrementNumber attribute provides a way to count the number of occurrences of an element.
Valid Values: A comma separated list of property names.
Scope: Element only.
The incrementNumber attribute increments the value of a mediation context property named in the attribute. Typically, a single property will be updated. Multiple variables can be updated by providing a comma separated list of property names. The attribute should be attached to each occurrence of the element that is to be counted. The number attribute can then be used to place the value of the variable into the output document.
The following example shows how the incrementNumber attribute is used count the occurrence of elements:

```
<Sample>
  <record etv:incrementNumber="aNumber">3.56</record>
  <record etv:incrementNumber="aNumber">1.01</record>
  <count etv:number="aNumber"/>
</Sample>
```

This XML will result in the following text:

```
<Sample>
  <record>3.56</record>
  <record>1.01</record>
  <count>2</count>
</Sample>
```

Note that the count element is populated with the number of incrementNumber attributes that have been found in the document prior to that point.
Because the execution of the incrementNumber attribute updates the value of a mediation context property, it is possible to access/modify this property across ETV invocations. For example, when processing a large XML input set the assembly may split and apply ETV attributes to the XML in chunks. Using the same property with incrementNumber across all ETV invocations allows the developer to share a running count across the entire input.

setBoolean

Summary: The setBoolean attribute provides a way to set the value of a mediation context property to a boolean value.
Valid Values: A property name.
Scope: Element only.
The setBoolean attribute sets the value of a mediation context property named in the attribute to the boolean value indicated by the value of the attribute. The property can then be access ed by assembly, XSLT, or other programming constructs like any other property in the mediation context. The boolean value is deduced using Java rules for determining true/false values.

The following example shows how the setBoolean attribute is used to set the value of a property in the mediation context.

```
<Sample>
  <record etv:setBoolean="error.indicator">true</record>
</Sample>
```

This results in the mediation context property error.indicator to be populated with the Boolean true.

setProperty

Summary: The setProperty attribute provides a way to set the value of a mediation context property to a string value.
Valid Values: A property name.
Scope: Element only.
The setProperty attribute sets the value of a mediation context property named in the attribute to the string value indicated by the value of the attribute. The property can then be accessed by assembly, XSLT, or other programming constructs like any other property in the mediation context.
The following example shows how the setProperty attribute is used to set the value of a property in the mediation context.

```
<Sample>
  <ssnetv:setProperty="employee.ssn">123-45-6789</record>
</Sample>
```

This results in the mediation context property employee.ssn to be populated with the value 123-45-6789.

launchParameter

Summary: The launchParameter attribute specifies the name of a sequenced value to be placed into the element.
Valid Values: The name of a Launch Parameter.
Scope: Element only.
The launchParameter attribute may be used to replace the value of an element with the value of a launch parameter.
An example is shown here:

```
<Cost_Center etv:launchParameter="Cost Center" />
```

The Element Transformation step will replace the value of this element with the value of the launch parameter.

```
<Cost_Center>CC46</Cost_Center>
```

The Launch Parameter may be defined on the integration system as a Reference Id. If the launch parameter contains a Reference Id, then a wd:ID element will be created as shown in this example:

```
<Organization_Reference etv:launchParameter=”Cost Center” />
```

In this case, the Element Transformation step will create the following output:

```
<wd:Organization_Reference wd:Descriptor="Cost Center 46">
  <wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
</wd:Organization_Reference>
```

Note: If the launch parameter contains multiple values, then the ETV step will output the element multiple times, once for each value. The XTT step will output the values as text,

concatenated together.

map

Summary: The map attribute applies an integration map to the contents of the element.
Valid Values: The name of an integration map on the integration system for the integration
Scope: Element only.
This attribute is used to apply integration maps to values. The values may be simple types or Reference IDs as dictated by the Integration Map. Unmapped values will be reported as messages. The severity of these messages is dictated by the severity attribute.
To apply an integration map to a simple text value, the map attribute is added to the element containing that value as shown below:

```
<Bank_Account_Type etv:map="Account Type">C</Bank_Account_Type>
```

The value of this element will be replaced with the value from the integration map, creating the output shown below.

```
<Bank_Account_Type etv:map="Account
Type">Checking</Bank_Account_Type>
```

Integration maps are often used with Workday Reference Ids. The following example shows the XML that should be used when mapping Reference Ids.

```
<Employee_Type etv:map="Employee Type">
  <wd:Worker_Type_Reference wd:Descriptor="Seasonal">
    <wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
    <wd:ID wd:type="Employee_Type_ID">EMPLOYEE_TYPE-10</wd:ID>
  </wd:Worker_Type_Reference>
</Employee_Type>
```

In this case, the following output is created:

```
<Employee_Type>SEASONAL</Employee_Type>
```

Note that in this case the name of the first child element (“Worker_Type_Reference” in the example above) is not important. The Workday Web Service return Reference IDs using a variety of names and this approach allows these elements to be copied directly from the Web service responses.
The following sample demonstrates the XSLT that is typically required to apply an integration map called Marital Status to a marital status Reference ID returned by a Workday Web service.

```
<Marital_Status etv:map="Marital Status">
  <xsl:copy-of
select="wd:Payee_Personal_Data/wd:Marital_Status_Reference"/>
</Marital_Status>
```

If the value of the element is not present in the Integration Map (and a default values has not been defined on the Integration Map), a message will be reported.

Note: when mapping simple types or actual values, use <xsl:value-of> to pass the particular value to the etv/ xtt map (see the "Bank_Account_Type" example above). When using a singe instance or reference ID type map, use <xsl:copy-of> to select the entire element for the single instance (see the "Employee_Type" example above).

Consider this example of integration system maps. One map uses the marital status single instance field, and the second uses a text mapping.

Integration Maps 4 items

Map Provider	Map	Description	Default Value	Map Values	
				Internal Value	External Value
INT Document Transformation Map Test	Marital Status One	Marital Status Single Instance Map		Married (United States of America)	M1-US
				Single (United States of America)	S1-US
	Marital Status Two	Marital Status Text Map		Married_USA	MUSA
				Single_USA	SUSA

In the example of **single instance mapping**, use <xsl:copy-of select="wd:Marital_Status_Reference"/> to identify the internal "Marital Status" single instance.

> **Note:** When you use "copy-of select" you must also change the following wd namespace reference in your XSLT to "xmlns:wd="urn:com.workday/bsvc"

Name Marital Status One

Description Marital Status Single Instance Map

Internal Value Type

Data Type Marital Status

External Value Type

Data Type Text

For the **text mapping example**, use <xsl:value-of select="wd:Marital_Status_Reference" /> to map an internal text value to an external text value.

Name Marital Status Two

Description Marital Status Text Map

Internal Value Type

Data Type Text

External Value Type

Data Type Text

mapAlternateValue

Summary: The map Alternate attribute identifies an alternate output value to use in the case were the integration map is empty.
Valid Values: The alternate value to output.
Scope : Element only.
This attribute is used to populate the output with an alternate value in cases where the integration map named in the map attribute is empty. This option allows integration maps to be provided on the integration which the user can choose not to populate. If the integration map is configured with atleast one value, then the mapAlternateValue attribute will be ignored.
The following example shows the XML that should be used when mapping to alternate values.

```
<Employee_Type etv:map="Employee Type" etv:mapAlternateValue="Full Time">Regular</Employee_Type>
```

If the integration map has not be configured with any values then the following output is created:

```
<Employee_Type>Full Time</Employee_Type>
```

The following example shows the XML that should be used when mapping to alternate values with reference IDs.

```
<Employee_Type etv:map="Employee Type"
etv:mapAlternaveValue="alternate"
etv:mapReferenceID="Employee_Type_ID">
  <wd:Worker_Type_Reference wd:Descriptor="Seasonal">
    <wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
  </wd:Worker_Type_Reference>
</Employee_Type>
```

If the integration map has not been configured with any values then the below output is created. Note the attempt to output the value from the reference ID Employee_Type_ID fails because that ID type is not available.

```
<Employee_Type>alternate</Employee_Type>
```

mapReferenceID

Summary: The mapReferenceID attribute identifies a reference id or reference descriptor value to use in the case were the integration map is empty.
Valid Values: The name of a reference id.
Scope : Element only.
This attribute is used to populate the output with the value of a reference id or reference descriptor in cases where the integration map named in the map attribute is empty. This option allows integration maps to be provided on the integration which the user can choose not to populate. If the integration map is configured with at least one value, then the mapReferenceID attribute will be ignored.
The following example shows the XML that should be used when mapping Reference Ids.

```
<Employee_Type etv:map="Employee Type"
etv:mapReferenceID="Employee_Type_ID">
  <wd:Worker_Type_Reference wd:Descriptor="Seasonal">
    <wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
    <wd:ID wd:type="Employee_Type_ID">EMPLOYEE_TYPE-10</wd:ID>
  </wd:Worker_Type_Reference>
</Employee_Type>
```

If the integration map has not be configured with any values then the following output is created:

```
<Employee_Type>EMPLOYEE_TYPE-10</Employee_Type>
```

If the mapReferenceID attribute is given a value of Descriptor, the output is set to the value of the reference descriptor. The following example shows the XML that should be used to output the reference descriptor

```
<Employee_Type etv:map="Employee Type"
etv:mapReferenceID="Descriptor">
  <wd:Worker_Type_Reference wd:Descriptor="Seasonal">
    <wd:ID wd:type="WID">cda8ef556ddf425ebed3bc40f4016ccb</wd:ID>
    <wd:ID wd:type="Employee_Type_ID">EMPLOYEE_TYPE-10</wd:ID>
  </wd:Worker_Type_Reference>
</Employee_Type>
```

If the integration map has not be configured with any values then the following output is created:

```
<Employee_Type>Seasonal</Employee_Type>
```

If the direction of the applied map is set to in via the direction attribute and the integration map is empty, the value is put into a Reference_ID element with the type attribute set to the value of the mapReferenceID attribute.
The following example shows the use of the mapReferenceID with a map direction of in.

```
<Employee_Type etv:map="Employee Type"
etv:mapReferenceID="Employee_Type_ID"
etv:direction="in">EMPLOYEE_TYPE-10</Employee_Type>
```

If the integration map has not be configured with any values then the following output is created:

```
<wd:ID wd:type="Employee_Type_ID">EMPLOYEE_TYPE-10</wd:ID>
```

maxLength

Summary: The maxLength attribute specifies the maximum length for the value of an element.
Valid Values: A numeric value indicating the maximum length.
Scope: Element only.
The behavior of this attribute is dictated by the truncate attribute. If truncate is set to true, then a value that exceeds the max length will be truncated.
If truncate is set to false, then a message will be reported indicating that the value exceeds the maximum length.

message

Summary: The message attribute is used to provide the text of a message.
Valid Values: Any text value.
Scope: Element only.
This attribute allows additional validation rules to be implemented within the XSL Transformation that creates the document. In the event that the data fails the custom validation rule, some appropriate text can be provided in the message attribute. The presence of this attribute will always result in a validation message being created.
Below is an example of the message attribute in use:

```
<Social_Security_Number etv:message="The Social Security Number is
invalid" etv:severity="error">1234123412345</Social_Security_Number>
```

The following message would then be created:

```
<wdext:Integration_Message>
  <wdext:Severity>ERROR</wdext:Severity>
  <wdext:Summary>The Social Security Number is invalid.
</wdext:Summary>
</wdext:Integration_Message>
```

minLength

Summary: The minLength attribute specifies the minimum length for the value of an element.
Valid Values: A numeric value indicating the minimum length.
Scope: Element only.
If the value of the element is shorter than the length specified in this attribute, then a message will be reported indicating that the value does not satisfy the minimum length requirement.

name

Summary: The name attribute provides the name to be used in validation messages.
Valid Values: Any text.
Scope: Element only.
The name attribute provides the name of the field to be used in validation messages. The default behavior is to use the element name in the validation message. This is illustrated by the following example of the use of the required attribute:

```
<SSN etv:required="true"></SSN>
```

The following message is created:

```
<wdext:Integration_Message>
  <wdext:Summary>No SSN was found. This is a required field for the
output document.</wdext:Summary>
</wdext:Integration_Message>
```

The following example shows the name attribute in use:

```
<SSN etv:name="Social Security Number" etv:required="true"></SSN>
```

In this case, the following message is created:

```
<wdext:Integration_Message>
  <wdext:Summary>No Social Security Number was found. This is a
required field for the output document.</wdext:Summary>
</wdext:Integration_Message>
```

nameAttribute

Summary: The nameAttribute attribute provides a reference to an attribute containing the name of the field to be used in validation messages.
Valid Values: A QName identifying another attribute
Scope: Element and all child elements.
The nameAttribute attribute refers to another attribute that contains the name of the field. The following example shows how the nameAttribute attribute can be used in conjunction with the required attribute.

```
<Sample xtt:nameAttribute="ID">
  <item ID="Z453" etv:required="true">One</item>
  <item ID="Z454" etv:required="true"> </item>
  <item ID="Z455" etv:required="true">Three</item>
</Sample>
```

The following message is created:

```
<wdext:Integration_Message>
  <wdext:Summary>No Z454 was found. This is a required field for the
output document.</wdext:Summary>
</wdext:Integration_Message>
```

number

Summary: The number attribute specifies the name of a variable. The value of the element will be replaced with the value of the variable, or 0 if the variable has not been initialized.
Valid Values: A variable name.
Scope: Element only.
The number attribute is used to place the value of a variable into an element, or 0 if the variable has not been initialized. It is used in conjunction with setNumber, incrementNumber and addNumber to provide counts and totals in the footers of documents. All number variables are assumed to have a value of zero when first used or retrieved. They are not scoped and can be re-assigned with the setNumber attribute.
The following example shows how the number attribute is used to populate the value of the result element:

```
<Sample>
  <record etv:addNumber="aNumber">3.56</record>
  <record etv:addNumber="aNumber">1.01</record>
```

```
  <result etv:number="aNumber"/>
</Sample>
```

This XML will result in the following text:

```
<Sample>
  <record>3.56</record>
  <record>1.01</record>
  <result>4.57</result>
</Sample>
```

Note that the result element is populated with the total of the two record elements.

numberFormat

Summary: The numberFormat attribute specifies the number format pattern that numbers will be converted to.
Valid Values: String, a number format pattern as used in Java.
Scope: Element only.
The numberFormat attribute may be used to specify a number format. The value of the element will be processed as a number and converted to the format specified in this attribute. An example is shown here:

```
<Salary etv:numberFormat="##,###.00">10000</Salary>
```

The value will be replaced with the reformatted value.

```
<Salary>10,000.00</Salary>
```

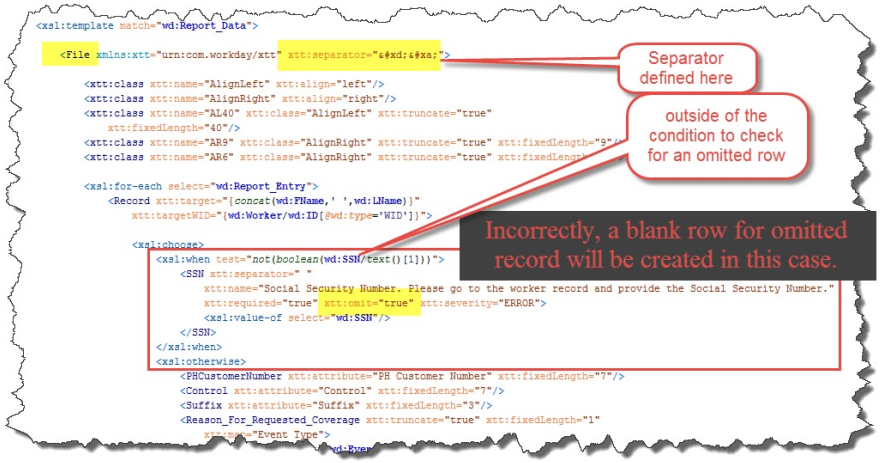
omit

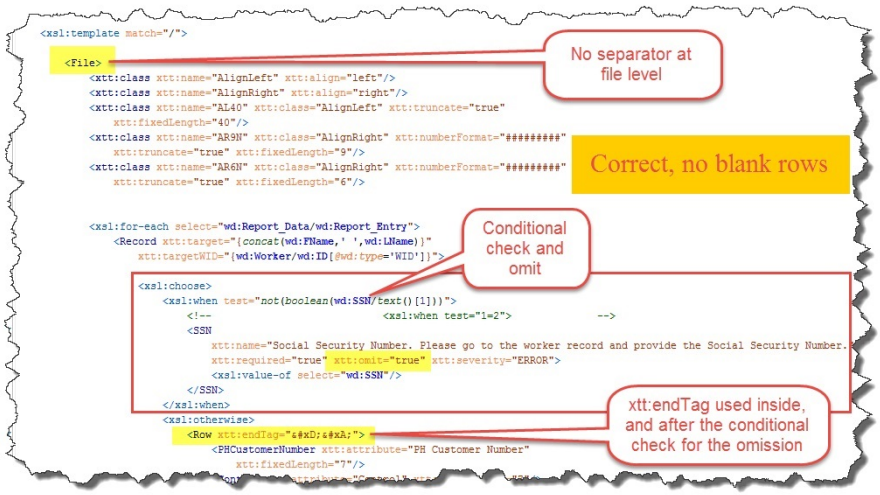
Summary: The omit attribute indicates that the element it is attached to should not be copied to the output message.
Valid Values: 'true' or 'false'
Scope: Element and all child elements.
When the omit attribute contains the value "true" on an element then the element and it's contents are not sent to the output although the values are processed. This might typically be used when summing or incrementing number values which are not directly used but from which derived information is required. The attribute value is inherited. It can be set false in a descendent element if output from that is required although this is not expected to be a common use case.
The following example demonstrates the use of the omit attribute.

```
<Sample>
  <item>One</item>
  <item etv:omit="true">Two</item>
  <item>Three</item>
</Sample>

The following output is created:
<Sample>
  <item>One</item>
  <item>Three</item>
</Sample>
```

> Note: If you use the omit attribute, you will have a blank row created if a separator is defined before you test your condition for the ommission. You many need to change from using xtt:separator to xtt:endTag or xtt:startTag and add this inside your conditional test. See examples below:





paddingCharacter

Summary: The paddingCharacter attribute indicates the character to be used for fixed length values that need to be padded.
Valid Values: Any character. The default is the space character.
Scope: Element and all child elements.
The paddingCharacter attribute indicates the padding character that should be used when fixed length values require padding.
The following example shows how the paddingCharacter attribute is used with the fixedLength attribute:

```
<Sample xtt:paddingCharacter="-">
  <item xtt:fixedLength="10">Short</item>
  <item xtt:fixedLength="10">Much Too Long</item>
  <item xtt:fixedLength="10">Just Right</item>
</Sample>
```

This XML will result in the following text:

```
Short-----Much Too LJust Right
```

quotes

Summary: The quotes attribute indicates whether the values from elements should be included in quotes when placed in the output file.
Valid Values: One of “always”, “never”, “csv”, “pipe-delimited”. The default is “never”.
Scope: Element and all child elements.
The quotes attribute indicates whether the values from elements should be included in quotes. If the value is “always”, then quotes will always be used. If the value is “csv” then the rules commonly associated with the use of quotes in CSV files will be applied. In this

case, a value will be quoted if it contains a comma, newline or quote character. It will also be quoted if it starts or ends with white space. A quote character in the value will be replaced with two quote characters.
The following example shows how the csv rules for quotes are applied:

```
<Sample xtt:separator="," xtt:quotes="csv">
  <item>Simple</item>
  <item>Includes a ,</item>
  <item> Starts with space</item>
  <item>Ends with space </item>
  <item>Contains a " character</item>
  <item>Contains a newline character</item>
</Sample>
```

This XML will result in the following text:

```
Simple,"Includes a ","," Starts with space","Ends with space
","Contains a "" character","Contains a newline character"
```

quoteStyle

Summary: The quoteStyle attribute indicates whether to use single or double quotes when the output is quoted via the quotes attribute.
Valid Values: One of single or double. The default is double.
Scope: Element and all child elements.
The following example shows how the quoteStyle attribute is used:

```
<Sample xtt:separator="," xtt:quotes="csv" xtt:quoteStyle="single">
  <item>Simple</item>
  <item>Includes a ,</item>
  <item> Starts with space</item>
  <item>Ends with space </item>
</Sample>
```

This XML will result in the following text:

```
Simple,'Includes a,','Startswith space','Ends with space '
```

quoteWhenMatches

Summary: The quotesWhenMatches attribute provides a way to quote values when unusual separator characters are used in CSV style files.
Valid Values: A regular expression
Scope: Element and all child elements.
The quotesWhenMatches attribute is provided for unusual quotation cases. It allows a regular expression to be specified. The value will be quoted when the regular expression matches.
The following example shows how the csv rules for quotes are applied:

```
<Sample xtt:separator="X" xtt:quoteWhenMatches=".*X.*">
  <item>Simple</item>
  <item>Includes an X character</item>
  <item>Simple</item>
</Sample>
```

This XML will result in the following text:

```
SimpleX"Includes an X character"XSimple
```

reportTruncation

Summary: The reportTruncation attribute specifies how the truncation of a value will be reported
Data Type: A comma separated list containing one or more of the following values: none, attribute, critical, error, warning, info. The default value used if this attributes is not set is none.
Scope: Element and all child elements.
Values that exceed the maximum length specified for them using the maxLength attribute will be truncated if the truncate attribute is set to 'true'. The reportTruncation attribute controls how the truncation of a value is reported if at all. It may be reported as a validation message or an attribute may be added to the message itself. When the reportTruncation attribute is set to "none", the truncation of values will be performed without reporting them. When set to "attribute", an etv:truncated="true" attribute will be added to any elements whose values have been truncated. This may be used to subsequent processing steps to identify truncated values.
For example, an audit file created from the resulting XML document might use this attribute to indicate that a value has been truncated. When set one of "critical", "error", "warning" or "info" a validation message will be created with that severity reporting the error.
"attribute" may be combined with one of "critical", "error", "warning" or "info" in a comma separated list so that the truncation is reported using both the etv:truncated attribute and a message.
The following XML shows an example of an etv:reportTruncation attribute with the value 'attribute':

```
<City etv:truncate="true" etv:reportTruncation="attribute"
etv:maxLength="30">Llanfairpwllgwyngyllgogerychwyrndrobwyllllantysil
iogogogoch</City>
```

The value of this element is truncated creating the output shown below.

```
<City etv:truncated="true">Llanfairpwllgwyngyllgogerychwy</City>
```

The etv:truncated attribute is added to indicate that the value has been truncated.
The following XML shows an example of an etv:reportTruncation attribute with the value 'warning':

```
<City etv:truncate="true" etv:reportTruncation="warning"
etv:maxLength="30">Llanfairpwllgwyngyllgogerychwyrndrobwyllllantysil
iogogogoch</City>
```

The value of the element is truncated, creating the output shown below.

```
<City>Llanfairpwllgwyngyllgogerychwy</City>
```

The following validation message will also be created:

```
<wdext:Integration_Message>
  <wdext:Severity>WARNING</wdext:Severity>
  <wdext:Summary> The value of City exceeds the maximum length of
30. The value has been truncated.</wdext:Summary>
</wdext:Integration_Message>
```

required

Summary: The required attribute indicates that the element must contain a value.
Valid Values: 'true' or 'false'
Scope: Element only.
To indicate that a value is required for an element add the etv:required attribute as shown here:

```
<Social_Security_Number
etv:required="true">1234123412345</Social_Security_Number>
```

If the Social Security Number was not present, then the following XML would be created:

```
<Social_Security_Number etv:required="true">
</Social_Security_Number>
```

The following message is created:

```
<wdext:Integration_Message>
  <wdext:Summary>No Social Security Number was found for John Smith.
This is a required field for the output document.</wdext:Summary>
</wdext:Integration_Message>
```

scale

Summary: Specified the position of an implied decimal place to be used when reformatting numbers.
Valid Values: A positive integer
Scope: Element and all child elements.
The scale attribute is used to specify the position of an implied decimal place that will be used when numbers are reformatted. An example is shown here:

```
<Salary etv:numberFormat="0000000000"
etv:scale="2">12345.67</Salary>
```

The Element Transformation step will replace the value of this element the reformmatted value.

```
<Salary>0001234567</Salary>
```

separator

Summary: The value of the separator attribute will be placed between the text contained in each of the child elements of the element that the attribute is attached to.
Valid Values: Any text.
Scope: Element only.
The value from this attribute will be placed between the text contained in each child element as shown in the example below:

```
<Sample xtt:separator=",">
  <item>One</item>
  <item>Two</item>
  <item>Three</item>
</Sample>
```

This XML will result in the following text:

```
One,Two,Three
```

sequencedValue

Summary: The sequencedValue attribute specifies the name of a sequenced value to be placed into the element
Valid Values: String, the name of a sequenced value.
Scope: Element only.
Sequence Generators defined on an integration system generate unique values for use within integrations. These values can be placed into a element using the sequencedValue

attribute.
An example is shown here:

```
<Document_Number etv:sequencedValue="Document Number" />
```

The contents of the element, if any, will be replaced with the value of the sequenced value. For example:

```
<Document_Number>57</Document_Number>
```

setNumber

Summary: The setNumber attribute sets the value for a variable to the value in the element.
Valid Values: A variable name.
Scope: Element only.
The setNumber attribute is used to place the value of an element into a variable. It can be used in conjunction with incrementNumber, addNumber and number to provide counts and totals in the footers of documents. All number variables are assumed to have a value of zero when first used or retrieved. They are not scoped and can be re-assigned with the setNumber attribute.
The following example shows how the setNumber can be used to initialize a variable which is subsequently incremented and then used to populate the count element:

```
<Sample>
  <initial etv:setNumber="aNumber">5</initial>
  <records>
    <record etv:incrementNumber="aNumber">3.56</record>
    <record etv:incrementNumber="aNumber">1.01</record>
  </records>
  <count etv:number="aNumber"/>
</Sample>
```

This XML will result in the following text:

```
<Sample>
  <initial>5</initial>
  <records>
    <record>3.56</record>
    <record>1.01</record>
  </records>
  <count>7</count>
</Sample>
```

severity

Summary: The severity attribute is used to set the severity of the messages.
Valid Values: One of info, warning, error, critical. The default value used if this at tributes is not set is error.
Scope: Element and all child elements.
This attribute is used to control the severity of the messages. This applies to messages created by the validation attributes as well as the message attribute. When used in conjunction with the required attribute as shown below the Severity element is set on the validation message.

```
<Social_Security_Number etv:required="true" etv:severity="ERROR">
</Social_Security_Number>
```

The following message is created:

```
<wdext:Integration_Message>
  <wdext:Severity>ERROR</wdext:Severity>
  <wdext:Summary> No value was found for Social Security Number.
```

```
This is a required field for the output document.</wdext:Summary>
</wdext:Integration_Message>
```

startTag

Summary: The value of the startTag attribute will be placed in the output file prior to the value of the element.
Valid Values: Any text.
Scope: Element only.
The value of the startTag attribute is placed in the output document prior to the value of the element it is attached to. For example:

```
<Sample xtt:startTag="START:">Hello World</Sample>
```

This XML will result in the following text:

```
START:Hello World
```

target

Summary: The target attribute is used to provide the name of the target which is typically an employee.
Valid Values: Any text value
Scope: Element and all child elements.
This attribute is used to create messages contain more information about the target of the validation. When used in conjunction with the required attribute as shown below additional text is added to the error message that is generated.

```
<Social_Security_Number etv:required="true" etv:target="John Smith">
</Social_Security_Number>
```

The following message is created:

```
<wdext:Integration_Message>
  <wdext:Summary>No Social Security Number was found for John Smith.
This is a required field for the output document.</wdext:Summary>
</wdext:Integration_Message>
```

targetWID

Summary: This targetWID attribute is used to provide the Workday ID of the target.
Valid Values: A Workday ID. Multiple Workday IDs can be provided as a comma separated list.
Scope: Element and all child elements.
This attribute is used to create messages contain the WID of the target object. When a message is attached to the Integration Event it will include a hyperlink to the target object. When used in conjunction with the etv:required attribute as shown below additional text is added to the error message that is generated.

```
<Social_Security_Number etv:required="true" etv:target="John Smith"
etv:targetWID="2af9579a2bde48d49bfc16ce5bcda138">
</Social_Security_Number>
```

The validation step would then create the following message:

```
<wdext:Integration_Message>
  <wdext:Target>2af9579a2bde48d49bfc16ce5bcda138</wdext:Target>
  <wdext:Summary>No Social Security Number was found for John Smith.
This is a required field for the output document.</wdext:Summary>
</wdext:Integration_Message>
```

timezone

Summary: Specifies the timezone to be used for dateTimes in the output file.
Valid Values: The name of a Timezone as used by the Java class java.util.TimeZone.
Scope: Element and all child elements.
The timezone attribute specifies the timezone to be used for a ny values formatted using the dateTimeFormat attribute. Values formatted using the dateFormat attribute are unaffected by the timezone attribute. A dateTime that includes a timezone offset will be converted to timezone specified in the etv:timezone attribute. A dateTime that does not include a timezone offset will not be converted.
An example is shown here:

```
<Last_Update etv:timezone="EST"
etv:dateTimeFormat="dd/MM/yyyy:hh|mm|ss">2010-11-18T02:57:22.019-
08:00</Last_Update>
```

The value of this element is reformmatted value as specified by the dateTime attribute. Note that the hour has been adjusted from 02 to 05

```
<Last_Update>18/11/2010:05|57|22</Last_Update>
```

In example this dateTime doesn't include a timezone offset:

```
<Last_Update etv:timezone="EST" etv:dateTime
Format="dd/MM/yyyy:hh|mm|ss">2010-11-18T02:57:22.019</Last_Update>
```

The value of this element is reformmatted as shown below. Note that in this case the hour has not been adjusted.

```
<Last_Update>18/11/2010:02|57|22</Last_Update>
```

truncate

Summary: The truncate attribute controls whether or not a value will be truncated if it exceeds the length specified in either the maxLength attribute.
Data Type: Boolean (When true, values will be truncated).
Scope: Element and all child elements.
To truncate a value, the maxLength attribute must be added to the element containing the value. A truncate attribute with the value ‘true’ should be present on that element or one of its parent elements:

```
<City etv:truncate="true"
etv:maxLength="30">Llanfairpwllgwyngyllgogerychwyrndrobwyllllantysil
iogogogoch</City>
```

The value of this element will be truncated creating the output shown below.

```
<City>Llanfairpwllgwyngyllgogerychwy</City>
```

charConversion

Summary: Converts an encoding format in input file to the output encoding format. Characters are converted to their close equivalent (in output encoding). Characters which can't be represented in the target character set will be handled according to the normal Java character conversion rules i.e replaced with relevant character with a question mark ("?").

Data Type: Boolean value.
Scope: Element and Child elements.

NOTE: Viewing the output, would depend on the Operating system default encoding / editor used to view the data.

Eg: Assuming output encoding = ASCII, any characters in the output text which have a close ASCII equivalent are converted to that character.

```
<Value xtt:charConversion="true">José</Value>
```

or

```
<File xtt:charConversion="true">

  <Value>José</Value>

</File>
```

would result in

```
Jose
```

Any characters which can't be represented in the target character set will be handled according to the normal Java character conversion rules which replace the relevant character with a question mark ("?"). So, if you'd set the output encoding to be "ascii", then without character conversion

```
<Value>José</Value>
```

would result in the output of

```
Jos?
```

repeat

Summary: Provides the ability to repeat an element, once or more, with identical value. A common use case for proprietary formats such as fixed width or CSV format files.
Data Type: A Positive Integer.
Scope: Element Only.

When set to an integer value > 1, will repeat the content *value* times. For example, if the *repeat* attribute is set to 3, the content will be output a total of 3 times.

```
<Sample xtt:separator=',>
    <A>a</A>
    <B xtt:repeat='3'>b</B>
    <C>c</C>
</Sample>
```

Output for the above would be:

```
a,b,b,b,c
```

For ETV the entire element is repeated (start tag, attributes, value, and end tag). If the parent has a separator defined then this is included in the output as interelement text.

For XTT the value is repeated *value* times. If the parent separator is defined the values are separted by the separator value.

Additional References :

Altitude 2015 Presentation : How to take advantage of all that Document Transformation offers

Workday Documentation : Document Transformation Connector

Ask a related question

3 Related Questions

Question	Answers
The etv:severity="warning" displays error when the validation fails? snath4 • A year ago	1
Using Multiple Attributes Together references using xtt:maxLength but I can't find a reference to this attribute. Is this a typo? dmartin10 • A year ago	2
Is there any xtt/etv function for converting text to Uppercase/Lowercase? achugh • A year ago	1