XML Functions

XML functions are typically used in transformation mapping scripts that require the manipulation and access of XML data.

Search in This Topic

Related Topics

- Cloud Studio
- Functions
- Scripts

Last updated: Sep 03, 2019

Attribute

Declaration

```
type Attribute(string attributeName, string attributeValue)
```

Syntax

```
Attribute(<attributeName>, <attributeValue>)
```

Required Parameters

- **`attributeName:`** Attribute name
- **`attributeValue:`** Attribute value

Description

Creates an attribute for an XML node. See also the `CreateNode()` function.

Examples

```
// Adds an attribute "name" with a value of "value"
Attribute("name","value");
```

[Back to Top]

CreateNode

Declaration

```
string CreateNode(string namespace, string nodeName, type
attributeSubelement[,...])
```

Syntax

```
CreateNode(<namespace>, <nodeName>, <attributeSubelement>[,...])
```

Required Parameters

- **`namespace:`** The namespace of the node
- **`nodeName:`** Name of the node
- **`attributeSubelement:`** A subelement: a value, attribute, or sub-node

Optional Parameters

- **`attributeSubelement:`** Additional subelements (a value, attribute, or sub-node), as required

Description

Creates a string representing an XML node. If the target node is the value node of an XML Any node, an XML element corresponding to the `nodeName` and `nodeValue` will be created.

Starting from the third argument, a series of values, attributes, and sub-nodes can be specified. Values are specified directly. Attributes and sub-nodes can be created with the `Attribute()` and `CreateNode()` functions respectively.

Examples

```
// Creates a node with a sub-node
CreateNode("http://example.com/xml",
    "Contact", "Bill  G.", Attribute("Type", "VIP"),
    CreateNode("http://example.com/xml",
        "Title", "Manager")
    );
```

[Back to Top]

GetNodeName

Declaration

```
string GetNodeName(type path)
```

Syntax

```
GetNodeName(<path>)
```

Required Parameters

- **`path:`** A path expression that selects a single node

Description

Retrieves the name of a node. This method is typically used to retrieve the name of a node returned by either of the `SelectNodeFromXMLAny()` or `SelectSingleNode()` functions.

Examples

```
// Retrieves the name of a node
GetNodeName(SelectNodeFromXMLAny("Account", Root$Any.));
```

[Back to Top]

GetNodeValue

Declaration

```
string GetNodeValue(type path)
```

Syntax

```
GetNodeValue(<path>)
```

Required Parameters

- **`path:`** A path expression that selects a single node

Description

Retrieves the value of a node. This method is typically used to retrieve the value of a node returned by either of the `SelectNodeFromXMLAny()` or `SelectSingleNode()` functions.

Examples

```
// Retrieves the value of a node
GetNodeValue(SelectNodeFromXMLAny("Account", Root$Any.));
```

[Back to Top]

GetXMLString

Declaration

```
string GetXMLString(type path[, bool qualified])
```

Syntax

```
GetXMLString(<path>[, <qualified>])
```

Required Parameters

- **`path:`** A path expression that selects a single node

Optional Parameters

- **`qualified:`** A boolean value (default `false`); when qualified (`true`), the XML string returned is easier to read but may not by itself be valid XML

Description

Returns (when used in a transformation mapping) the corresponding XML string found in the source XML document at the specified path.

To enter a path into the function, manually enter the reference path of the desired XML node folder. For more information, see the instructions on inserting source objects under the Source Objects section in Jitterbit Script. For information on constructing node reference paths, see Nodes and Fields.

Examples

```
// Retrieves the corresponding XML string at the specified path
GetXMLString([Header$DETAIL.]);
```

[Back to Top]

IsNil

Declaration

```
bool IsNil(string path)
```

Syntax

```
IsNil(<path>)
```

Required Parameters

- **`path:`** A path expression that selects a single node

Description

Returns (when used in a Formula Builder mapping) if the corresponding XML node has the attribute "`xsi:nil`" with the value of `true` (or 1).

To enter a path into the function, manually enter the reference path of the desired XML node folder. For more information, see the instructions on inserting source objects under the Source Objects section in Jitterbit Script. For information on constructing node reference paths, see Nodes and Fields.

As described in XML Schema, an element may be valid without content if it has the attribute `xsi:nil` with the value `true`.

Examples

```
// Returns if the node is nil
IsNil("Header$DETAIL.");
```

[Back to Top]

RunXSLT

Declaration

```
array RunXSLT(string xslt, string xml1[, string xml2,... string xmlN)
```

Syntax

```
RunXSLT(<xslt>, <xml1>[, <xml2>,... <xmlN>])
```

Required Parameters

- **`xslt:`** A string reference path to an activity associated with a file-type endpoint in the current project
  that returns an XSLT stylesheet
- **`xml1...xmlN:`** A string reference path to an activity associated with a file-type endpoint in the current
  project that returns one or more XML documents

Description

Supports running XSLT (v1-3) on a number of input XML documents. Takes as input an XSLT stylesheet and one or more XML documents and returns an array of XML documents.

The `xslt` parameter used in this function call must be defined as an activity associated with a file-type endpoint in the current project that returns an XSLT stylesheet. The `xml1...xmlN` parameters used in this function call must be defined as one or more activities associated with file-type endpoints in the current project that return one or more XML documents. These include configured file share, FTP, HTTP, local storage, and temporary storage activities. For more information, see the instructions on inserting endpoints.

Referring to Files from within an XSLT Stylesheet

While Jitterbit has broad support for XSLT, projects that use XSLT must follow rules to pass the security standards. These rules mean that access to files or URIs must always be through the use of activities defined in the same project.

To reference these files or URIs in a stylesheet, use the XLST `fn:doc` function and specify `'param1'` for the first XML input, `'param2'` for the second XML input, and so on for each additional XML file.

Given the example of processing three XML files (`customers, salesPeople, salesOrders`), you could refer to them in an XSLT stylesheet using:

```
<xsl:for-each select="doc('param1')/*:Customers/*:SalesOrder
[customer_external_id!=''][sales_person_external_id!='']">
...
<xsl:for-each select="doc('param2')/*:SalesPeople/*:Account[*:customer_id
/string(number(text()))=$CustomerExtId][1]">
...
<xsl:for-each select="doc('param3')/*:SalesOrders/*:SalesOrder
[customer_external_id!=''][sales_person_external_id!='']">
```

If you have a single input XML file, it is not required to use the `doc('param1')` syntax as Jitterbit will set the initial context to that single file.

Specifying a Starting Template in the Stylesheet

In some situations, Jitterbit needs to know where in a stylesheet to begin processing. This is achieved by including in the stylesheet a template with the specific name of `start_here`:

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

Examples

```
// Running XSLT on XML Files

// Read in a stylesheet
xslt = ReadFile("<TAG>activity:ftp/FTP Endpoint/ftp_read/XSLT</TAG>");

// Read in two XML files for processing
xml1 = ReadFile("<TAG>activity:ftp/FTP Endpoint/ftp_read/XML1</TAG>");
xml2 = ReadFile("<TAG>activity:ftp/FTP Endpoint/ftp_read/XML2</TAG>");

// RunXSLT on the stylesheet and the two XML files
output = RunXSLT(xslt, xml1, xml2);

// As RunXSLT() returns an array,
// retrieve the output from each element
i = 0;
While(i < Length(output),
  // Write output to a file
  WriteFile("<TAG>activity:ftp/FTP Endpoint/ftp_write/XML Output FTP<
/TAG>",
    output[i], "output" + (i+1) + ".xml");
  i++;
);
```

[Back to Top]

SelectNodeFromXMLAny

Declaration

```
type SelectNodeFromXMLAny(string nodeName, type anyNodes)
```

Syntax

```
SelectNodeFromXMLAny(<nodeName>, <anyNodes>)
```

Required Parameters

- **`nodeName`:** The desired node name

- **anyNodes:** The data element path of a value node of an XML Any element or an array of XML nodes

Description

Returns the first XML node from a list of XML Any nodes that match the node name.

Examples

```
// Looks for the first node that matches "Account"
SelectNodeFromXMLAny("Account", Root$Any#.);
```

[Back to Top]

SelectNodes

Declaration

```
array SelectNodes(type node, string xPathQuery[, string xPathArg1,...
string xPathArgN])
```

Syntax

```
SelectNodes(<node>, <xPathQuery>[, <xPathArg1>,... <xPathArgN>])
```

Required Parameters

- **node:** The desired node or XML fragment to run the query on
- **xPathQuery:** The data element path of a value node of an XML Any element

Optional Parameters

- **xPathArg1...xPathArgN:** Prefixes specifying the namespaces of nodes in the XPath query

Description

Runs an XPath query (see the XPath standard, v1-v3) on either an XML fragment or an XML node returned from another function, and returns the results of the query.

If the optional prefixes are used to specify the namespaces of the node in the XPath query, the prefixes must be specified as one or more string arguments after the XPath (see the second example).

Examples

Example 1

```
// Connects to an LDAP server, runs a search,
// and then runs an XPath Query on the results
LDAPConnect("directory.company.example.com","ghvwright","1LikesPe@ches",0);
searchResults = LDAPSearch("CN=Users,DC=company,DC=example,DC=com",
    "(&(objectCategory=person)(objectClass=user))", 1,
    "name", "whenCreated", "description", "telephoneNumber");
resultNodes = SelectNodes(searchResults,
    "/DirectoryResults/Entry[name='Administrator']/whenCreated" );
```

Example 2

```
SelectNodes(Root$Any$,
    "ns1:E2/ns2:E3",
    "ns1=http://xyz1.example.com/",
    "ns2=http://xyz2.example.com/");
```

NOTE: In this second example, the reference node is an XML node for an XML Any node. In the XPath query, `ns1:E2` is not the reference node itself.

Since the prefixes `ns1` and `ns2` are used in the XPath query, they are defined as additional arguments after the XPath.

[Back to Top]

SelectNodesFromXMLAny

Declaration

```
array SelectNodesFromXMLAny(string xPathQuery, type anyNodes[, string
xPathArg1,... string xPathArgN])
```

Syntax

```
SelectNodesFromXMLAny(<xPathQuery>, <anyNodes>[, <xPathArg1>,...
<xPathArgN>])
```

Required Parameters

- **`xPathQuery:`** An XPath query
- **`anyNodes:`** The data element path of a value node of an XML Any element or an array of XML nodes

Optional Parameters

- **`xPathArg1...xPathArgN:`** Prefixes specifying the namespaces of nodes in the XPath query

Description

Returns an array of all the XML nodes that are matched by an XPath query (see the XPath standard, v1-v3) run against either a path of a value node of an XML Any element or an array of XML nodes.

If the optional prefixes are used to specify the namespaces of the node in the XPath query, the prefixes must be specified as one or more string arguments after the XPath (see the second example).

Examples

Example 1

```
// Select all the nodes with the given names
SelectNodesFromXMLAny("Account|Customer|Name", Root$Any#.);
```

Example 2

```
// Select email addresses and phone numbers only
$nodes = SelectNodesFromXMLAny("cust:EmailAddress|cust:PhoneNumber",
    Customer$Any#., "cust=urn:xmlns:25hoursaday-com:customer");
```

[Back to Top]

SelectSingleNode

Declaration

```
type SelectSingleNode(type node, string xPath,...)
```

Syntax

```
SelectSingleNode(<node>, <xPath>,...)
```

Required Parameters

- **`node:`** The desired node or XML fragment to run the query on
- **`xPathQuery:`** The data element path of a value node of an XML Any element

Optional Parameters

- **`xPathArg1...xPathArgN:`** Prefixes specifying the namespaces of nodes in the XPath query

Description

Runs an XPath query (see the XPath standard, v1-v3) on either an XML fragment or an XML node returned from another function, and returns the first node in the results of the query.

If the optional prefixes are used to specify the namespaces of the node in the XPath query, the prefixes must be specified as one or more string arguments after the XPath (see the second example).

Examples

Example 1

```
// Connects to an LDAP server, runs a search,
// runs an XPath Query on the results, and
// selects the first node in the query results
LDAPConnect("directory.company.example.com", "ghvwright", "1LikesPe@ches",
0);
searchResults = LDAPSearch("CN=Users,DC=company,DC=example,DC=com",
    "(&(objectCategory=person)(objectClass=user))", 1, "name",
    "whenCreated", "description", "telephoneNumber");
resultNode = SelectSingleNode(searchResults,
    "/DirectoryResults/Entry[name='Administrator']/whenCreated");
```

Example 2

```
// Selects the first node in an XPath query result.
// The reference node is an XML node for an XML Any node.
// In the XPath query, "ns1:E2" is not the reference node itself.
// Since the prefixes ns1 and ns2 are used in the XPath query,
// they are defined as additional arguments after the XPath.
SelectSingleNode(Root$Any$,
    "ns1:E2/ns2:E3",
    "ns1=http://xyz1.example.com",
    "ns2=http://xyz2.example.com/");
```

[Back to Top]