SQL Server 2005 - SQL Server Integration Services - Part 7 - Foreach Loop Container

In this article, we are continuing the overview of various enumerator types within the Foreach Loop Container of SQL Server 2005 Integration Services. So far we have covered such enumerators as Foreach File (iterating over files selected according to arbitrarily assigned criteria and residing in a folder of your choice), Foreach ADO (enumerating records within an ADO recordset), and Foreach SMO (intended for working with a wide range of SQL Server Management Objects). The next one in our series is the Foreach Nodelist enumerator, which deals with result sets generated by XML Path Language (XPath) expressions.

With the introduction of SQL Server 2005, XML (which stands for eXtensible Markup Language - platform-independent data representation format, which has been hailed as a panacea for resolving data interchange issues) has become an integral part of the product. While important XML-related capabilities were present in the previous version (for their review, you can refer to a series of articles published on the DatabaseJournal Web site), a number of new features and enhancements in this area is significant (such as, native XML data store and native XML queries, support for Extensible Schema Definition, or extending bulk loading, replication, triggers, and indexing functionality to cover XML-based data structures). These changes include an enumerator implemented specifically for the purpose of processing XML data.

In order to understand the way the Foreach NodeList enumerator functions, you need to be familiar with the principles of XML data structure and its manipulation. Such data typically takes the form of an XML document. The document conforms to specific formatting rules and consists of two main parts - a prolog and a body. Within the prolog, you can find an XML declaration, which determines the version of XML standard that is followed, as well as encoding method (such as default 8-bit UTF-8 or 16-bit Unicode, commonly used in multi-language scenarios). <? and ?> mark the beginning and end of the declaration. In the simplest case, the prolog could contain solely `<? xml version="1.0" ?>` entry, however, frequently it also includes a Document Type Declaration, specifying the root element (which appears in the body) and identifying a document type definition (describing additional rules that the XML document needs to comply with). For more information on this and other XML-related topics, you can refer to XML Beginner's Guide on the www.xml.org Web site.

The body consists of a single root and any number of non-root nodes, which most commonly belong to the attribute, element, or text category. Root, text, and element nodes are marked with a start and an end tag, consisting of a tag-identifying label enclosed between "less than" (<) and "greater than" (>) symbols. The closing tag of each pair also includes a forward slash ("/") preceding the tag identifier. Tags mark the beginning and end of the data associated with the identifier (note that such data might contain other XML elements, allowing for element nesting). An element node, unlike a text node, can contain one or more attribute nodes, which are represented by their identifiers, followed by the equal sign and associated value enclosed in the single or double quotes. The space is used as a separator in case multiple attribute nodes appear in the same element node.

It might be easier to follow the above definitions by referring to a specific example. Here is a sample XML document representing part of the inventory of a widget factory warehouse. Within it, there are several element nodes (such as Product or

Price), text nodes (such as Name or InStock), as well as attribute nodes (such as ID within the Product node or Category within the Price node). Note that you can change the structure of the document by turning attributes into elements or vice versa (in either case, they represent properties of a parent element):

```
<? xml version="1.0" ?>
<Inventory>
  <Product ID="00001">
    <Name>"Very Small Widget"</Name>
    <InStock>20</InStock>
    <Price Category="Wholesale">0.49</Price>
    <Price Category="Retail">0.99</Price>
  </Product>
  <Product ID="00002">
    <Name>"Not So Small Widget"</Name>
    <InStock>200</InStock>
    <Price Category="Wholesale">0.99</Price>
    <Price Category="Retail">1.49</Price>
  </Product>
</Inventory>
```

Now that you have a general idea about the structure of XML data, let's review the way we can reference its components. One of the methods providing this functionality is XPath (an abbreviation derived from the term XML Path Language). XPath views an XML document as a tree-like structure consisting of multiple nodes (similar to the way file system is organized, with a root folder, subfolders, and files being equivalent to the XML root element, its sub-elements, and their attributes). By following specific syntax rules, you can define a path between any two nodes of such tree, and, at the same time, uniquely identify a target node or set of nodes. Path expressions can be relative (outlining a sequence from one XML node to another - with the assumption that the starting node is implicitly known) or absolute (always starting at the root element, designated by a single forward slash, followed by a relative path from the root to a target node). Either type is constructed by combining individual location steps (separated by the forward slash), which, in turn, consist of the following elements:

- axis - one of several keywords, including the parent, child, self, and attribute, which describe the relationship between the current and next node in the path. According to syntactical rules, axis precede the name of an XML element that they refer to by the double semicolon. For example, in case of our sample XML document, child::Inventory references child nodes of the Inventory root node (similarly, child:Inventory/child:Product points to all child nodes of the Product node or child::Inventory/child::Product/attribute::ID identifies the ID attribute of the Product node). This notation can be abbreviated - in particular, since child is the default axis, it can be omitted (as long as you intend to use it), attribute:: can be replaced with @ symbol, and single and double periods are permitted instead of self::node() and parent::node(), respectively.
- node test - determines the type of a node specified by the axis. Use of the child, parent, or self results in the selection of element children matching the criteria provided, while the attribute axis points by default to attribute nodes.
- selection predicate (optional) - provides the ability to further narrow down the scope of target elements pointed to by XPath expressions. It takes the form of a condition that evaluates to a Boolean value (i.e. true or false), and the outcome of this evaluation is used to determine whether the node to which it is applied is included in the result set. For example, if we wanted to

access only the first product in our sample Inventory, we could use the notation child::Inventory/child::Product [attribute::ID="00001"] (or Inventory/Product[@ID="00001"] in abbreviated format).

Equipped with this information, let's find out how we can apply it to configure a Foreach Loop container with NodeList Enumerator. Start by launching SQL Server Business Intelligence Development Studio and creating a new Integration Services Project. Drag a Foreach Loop Container icon from the Toolbox onto the Control Flow area of the Designer interface. Choose Edit from the container's context sensitive menu to display the Foreach Loop Editor. Click on the Collection entry in the left part of the Editor's window. Select Foreach NodeList Enumerator from the Enumerator listbox. Enumerator configuration section below provides a number of options, which we will explore in more details in the next article of this series. For now, we will use the most straightforward ones, with both DocumentSourceType and OuterXPathStringSourceType set to the value of Direct Input, and with our sample XML document as the DocumentSource (without the prolog). Select NodeText as the XPath EnumerationType and assign child::Inventory value to OuterXPathString. Switch to the Variable Mappings section of the Foreach Loop Editor and create a new variable sResult of the Package scope and String data type (the variable will automatically be assigned Index 0). Confirm your choices and return to the Designer interface.

To display the results of the package execution, drag the Script Task from the Toolbox and drop it inside the boundaries of the Foreach Loop Container. Activate the Script Task Editor by selecting Edit from its context sensitive menu, switch to the Script section, type in User::sResult in the ReadOnlyVariables entry, and click on the Design Script command button. In the Microsoft Visual Studio for Applications, modify the Public Sub Main() code as follows:

```
Public Sub Main()
 '
 ' Add your code here
 '
 MsgBox(Dts.Variables("sResult").Value.ToString)
 Dts.TaskResult = Dts.Results.Success
End Sub
```

Close the Visual Basic for Applications window, click on OK to return to the Designer interface and execute the package. Since we selected the NodeText as the XPath EnumerationType and assigned child::Inventory to the OuterXPathString, the result displayed in a message box contains the combined value of all text nodes within our Inventory. Note that we could obtain the same result using /Inventory or simply / as the value of the OuterXPathString. If our intention was to display the content of text nodes separately for each product, we could accomplish this by setting OuterXPathString to child::Inventory/child::Product (or simply Inventory/Product). Setting OuterXPath String to child::Inventory/child::Product/attribute::ID would display the sequence of Product ID values for all of the Products in the inventory (the same result could be obtained by setting OuterXPathString to //@ID). As mentioned before, applying a selection predicate would allow you to further limit the result set (e.g. the value of child::Inventory/child::Product [attribute::ID="00001"] assigned to OuterXPathString yields the product with ID of 00001).

There are additional configuration options available within the Collection section of Foreach Loop Editor. We will review them (as well as continue the coverage of remaining Foreach Enumerators) in the next article of our series.