SQL Server 2005 - SQL Server Integration Services - Part 6 - Foreach Loop Container

In this article, we are continuing coverage of Foreach Loop container functionality, by discussing different types of loop enumerators. So far, we have presented methods for working with files in a folder as well as records in a recordset (using Foreach File and Foreach ADO enumerators, respectively). Our next enumerator is based on the SQL Server Management Objects technology (hence it is referred to as the SMO enumerator), which is the SQL Server 2005, .NET-based management framework providing replacement for COM-based DMO (Distributed Management Objects) that served an equivalent role in earlier versions of SQL Server. Programming capabilities available via SQL Server Management Objects are versatile and encompass practically every maintenance task on the server, database, and database object level (some examples can be found in the "Programming Specific Tasks" section of SQL Server 2005 Books Online). However, since our attention is focused on the interaction between SMO and SQL Server Integration Services, we will simply provide an example demonstrating such functionality (which should give you enough information to explore this topic in more in-depth fashion on your own).

A few preliminary steps are necessary in order to ensure that all required SMO features will be available within SSIS packages. This is partially due to the fact that the majority of them are implemented as part of .NET assemblies included in the June version of the SQL Server 2005 CTP and supported only in the version 2.0 of the Microsoft .NET Framework. An additional factor is the limitation inherent to the Visual Studio for Application, which requires that such assemblies reside in the current .NET Framework location. This means that you will most likely need to copy some of them to the %WinDir%\Microsoft.NET\Framework\v2.0.xxxxx subfolder (where the last five characters represent the number identifying the release). You can determine which files need to be copied by referencing a topic describing the object in question in the SQL Server 2005 Books Online. Each object type that you want to work with is equivalent to a .NET class, whose description includes (among other items) the name of its namespace (classes are grouped together into namespaces, which prevent naming clashes between classes and organizes them in a manageable manner) and an assembly containing its implementation. For example, when working with SMO, two primary namespaces you will deal with are Microsoft.SqlServer.Management.Common and Microsoft.SqlServer.Management.Smo. Availability of the first one depends on Microsoft.SqlServer.ConnectionInfo.dll and the second one requires Microsoft.SqlServer.Smo.dll. Assuming that you installed SQL Server in the default target location, you can find both of them in the Program Files\Microsoft SQL Server\90\SDK\Assemblies. Note that copying these files to the %WinDir%\Microsoft.NET\Framework\v2.0.xxxxx subfolder is not required for the functionality of Foreach Loop with SMO Enumerator (which works perfectly well without it), however the procedure is needed to allow its interaction with Script Task Editor, which manipulates SQL Server objects enumerated within the loop.

In addition to placing the SQL Server SMO assemblies in the proper location, it is also necessary to add references to them (when working within the Visual Studio for Applications interface), which makes their objects (classes) accessible to your code. Since an assembly can contain more than one namespace, you should also include an "Imports" directive (when writing programming in Visual Basic .NET - or "#using" directive when working with C#) followed by the name of the namespace, which further narrows down the scope of references (and eliminates the need for explicitly specifying the fully qualified names of referenced classes). We will describe these steps in more details in our example presented in this article.

There are several SMO methods, which are equivalent to DBCC-based integrity checks. For example, CheckCatalog method delivers the same functionality as the DBCC CHECKCATALOG T-SQL statement, performing various consistency checks across system metadata tables (for the list of all methods available within the Database SMO class, refer to the "Database Members" topic in the SQL Server Books Online). This method, similarly to its T-SQL equivalent, takes one parameter, which is either the name or ID of the database for which the checks should be performed. In our example, we will iterate through all databases on the target SQL Server (using Foreach Loop with SMO Enumerator) and execute this method against each of them.

Start by copying Microsoft.SqlServer.ConnectionInfo.dll and Microsoft.SqlServer.Smo.dll from the Program Files\Microsoft SQL Server\90\SDK\Assemblies folder to the %WinDir%\Microsoft.NET\Framework\v2.0.xxxxx (for the reasons explained above). Next, launch SQL Server Business Intelligence Development Studio and create a new Integration Services project. Display the Variables window (you can activate it with the Variables entry from the Other Windows submenu of the View menu) and create oSMODB variable of Package scope and Object data type. Drag the Foreach Loop Container icon from the Toolbar to the Control Flow area of the Package Designer interface. Choose Edit item from its context-sensitive menu to bring up its Editor window. In the Collection section under Enumerator entry, specify "Foreach SMO Enumerator" and create a new Connection manager using the SMO Connection Manager Editor (specify your target server and the appropriate authentication method). Next, click on the Browse... button located to the right of the Enumerate text box. This will display the Select SMO Enumeration dialog box. From here, you can select the type of objects you intend to work with, including Linked Servers, Jobs, Logins, Databases, as well as for each database, its File Groups, Data Files, Log Files, Stored Procedures, User Defined Data Types, User Defined Functions, Users, Views, and Tables (and, for each table, its Columns, Foreign Keys, and Triggers). Note that you can alter enumeration type, by choosing objects, their names, or Uniform Resource Names (URNs uniquely identify each Microsoft SQL Server object). Since in our example, we want to obtain a listing of all SQL Server databases, select the Databases entry in the Enumerate box and leave the default Objects as the enumeration type. Click on OK to return to the Foreach Loop Editor. Switch to the Variable Mappings section and choose User::oSMODB from the dropdown list in the Variable column (this will automatically assign to it an Index value of 0). Close the editor window by clicking on the OK button.

While our loop does enumerate all SQL Server databases, at this point, it does not yet perform any useful work. To change this, we will execute (repetitively) a short piece of code within the scope of the newly created container. Since the code will be part of the Script Task, drag its icon from the toolbar and drop it into the area enclosed by the rectangle representing the Foreach Loop container. Display the Script Task Editor (using the context-sensitive menu entry), switch to the Script section, type User::oSMODB in the ReadOnlyVariables entry, and click on the Design Script button to bring up the Visual Studio for Applications interface. As we mentioned before, we need to include references to the assemblies that were copied to the %WinDir%\Microsoft.NET\Framework\v2.0.xxxxx folder. This can be done with the Add Reference entry from the Project menu (the same menu item is available from the context sensitive menu of the References node in the Project Explorer). From within the Add Reference dialog box, select the Microsoft.SqlServer.Smo.dll and click on the Add button. Repeat the same steps for Microsoft.SqlServer.ConnectionInfo.dll. Next, in the top portion of the code window, include the two Imports directives:

```
Imports Microsoft.SqlServer.Dts.Runtime
Imports Microsoft.SqlServer.Management.Smo
```

Finally, modify the Public Sub Main() by adding code that obtains the value of the SSIS User::oSMODB variable, makes its type compatible with the one required by the SMO assemblies, retrieves the current database name, displays an informative message about the pending operation, and invokes the CheckCatalog method. This produces the following outcome:

```
Public Sub Main()
Dim oDB As Database
Dim sDBName As String
oDB = CType(Dts.Variables("oSMODB").Value, Database)
sDBName = oDB.Name
MsgBox("Running the CheckCatalog against " & sDBName)
oDB.CheckCatalog()
Dts.TaskResult = Dts.Results.Success
End Sub
```

Close the Microsoft Visual Studio for Applications window and click on the OK button in the Script Task Editor. If you execute the task (using the appropriate option from the context sensitive menu of the Foreach Loop Container) or the package, you should see message boxes displayed in the sequence informing you about the CheckCatalog method being executed against each of the SQL Server databases.

Alternatively, we could select the Names entry when configuring the Enumeration type in the Select SMO Enumeration dialog box (accessible from Foreach Loop Editor interface once Foreach SMO Enumerator is chosen). This would allow us to obtain the database name directly as part of the loop enumeration, leading to slightly more compact code (note that we changed the variable to User:sSMODB of type String):

```
Public Sub Main()
Dim sDBName As String
sDBName = CType(Dts.Variables("sSMODB").Value, String)
MsgBox("Running the CheckCatalog against " & sDBName)
oDB.CheckCatalog()
Dts.TaskResult = Dts.Results.Success
End Sub
```

Another option would be to store the extracted values in variables and pass them as parameters to the Execute SQL Task (we will be discussing this task in more detail in the next article of this series). It is worth noting that SSIS offers a number of Maintenance Plan Tasks (allowing for example, checking database integrity, rebuilding its indexes, or updating statistics), which you could easily incorporate into the Foreach Loop container processing sequence. You should review all possible solutions before you decide which one fits your requirements the best.