

## How To Retrieve Excel Schema Information Using SQL Server Integration Services (SSIS) 2005

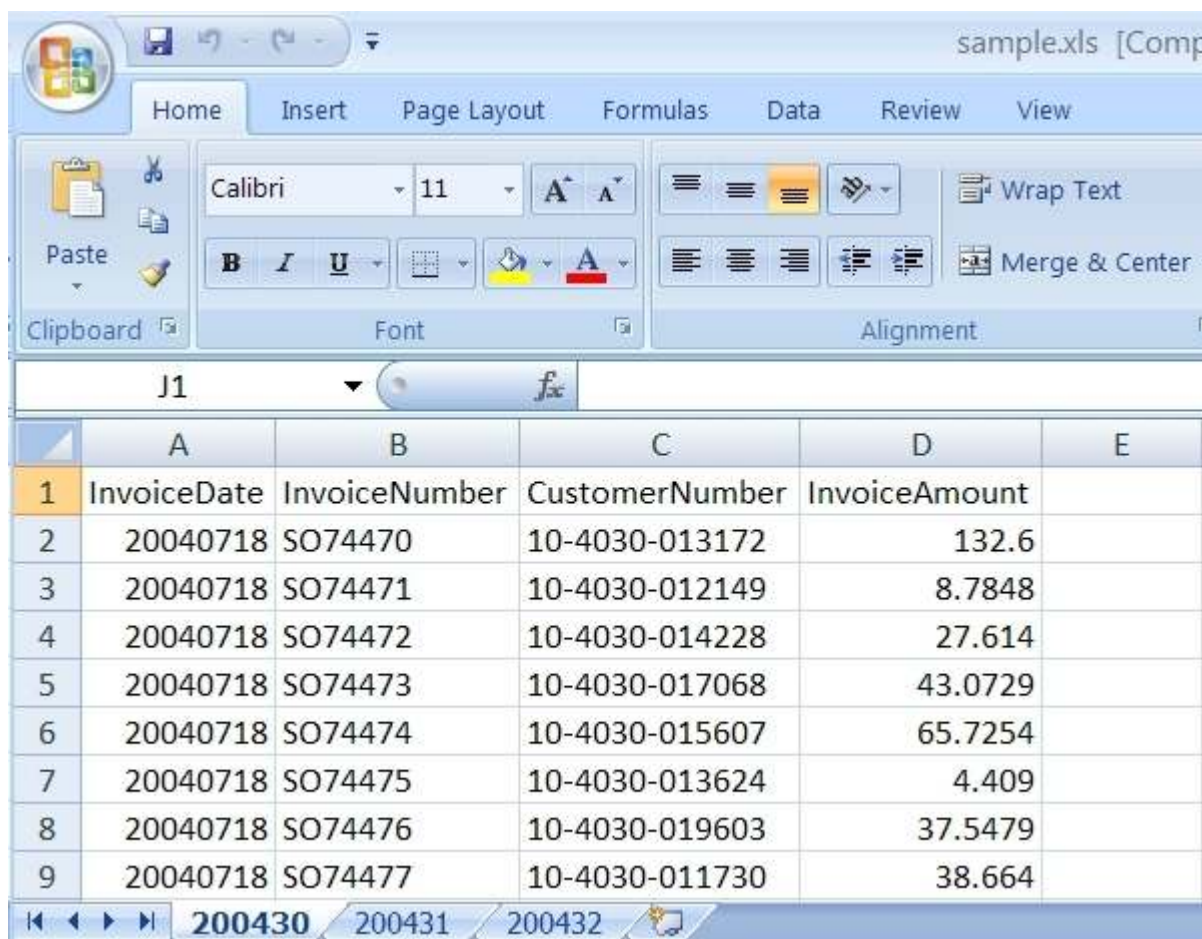
Written By: Ray Barley -- 1/27/2009

### Problem

We use SSIS to periodically load data into our data warehouse. While much of the data we process is in relational data stores, we do have some Excel spreadsheets that we need to process. In one particular case we load an Excel spreadsheet that is produced by an external application and every month the number of sheets varies and the sheet names are also different. How can we determine the sheet names in our SSIS package and process this variable number of sheets?

### Solution

The first step is to retrieve the schema information from the Excel spreadsheet. The sheet name in the Excel spreadsheet becomes the table name in a SQL statement; the sheet columns are the columns in the SQL statement. Let's start out by looking at a sample Excel spreadsheet that we might need to process with an SSIS package:



	A	B	C	D	E
1	InvoiceDate	InvoiceNumber	CustomerNumber	InvoiceAmount	
2	20040718	SO74470	10-4030-013172	132.6	
3	20040718	SO74471	10-4030-012149	8.7848	
4	20040718	SO74472	10-4030-014228	27.614	
5	20040718	SO74473	10-4030-017068	43.0729	
6	20040718	SO74474	10-4030-015607	65.7254	
7	20040718	SO74475	10-4030-013624	4.409	
8	20040718	SO74476	10-4030-019603	37.5479	
9	20040718	SO74477	10-4030-011730	38.664	

The main points about the above Excel spreadsheet are:

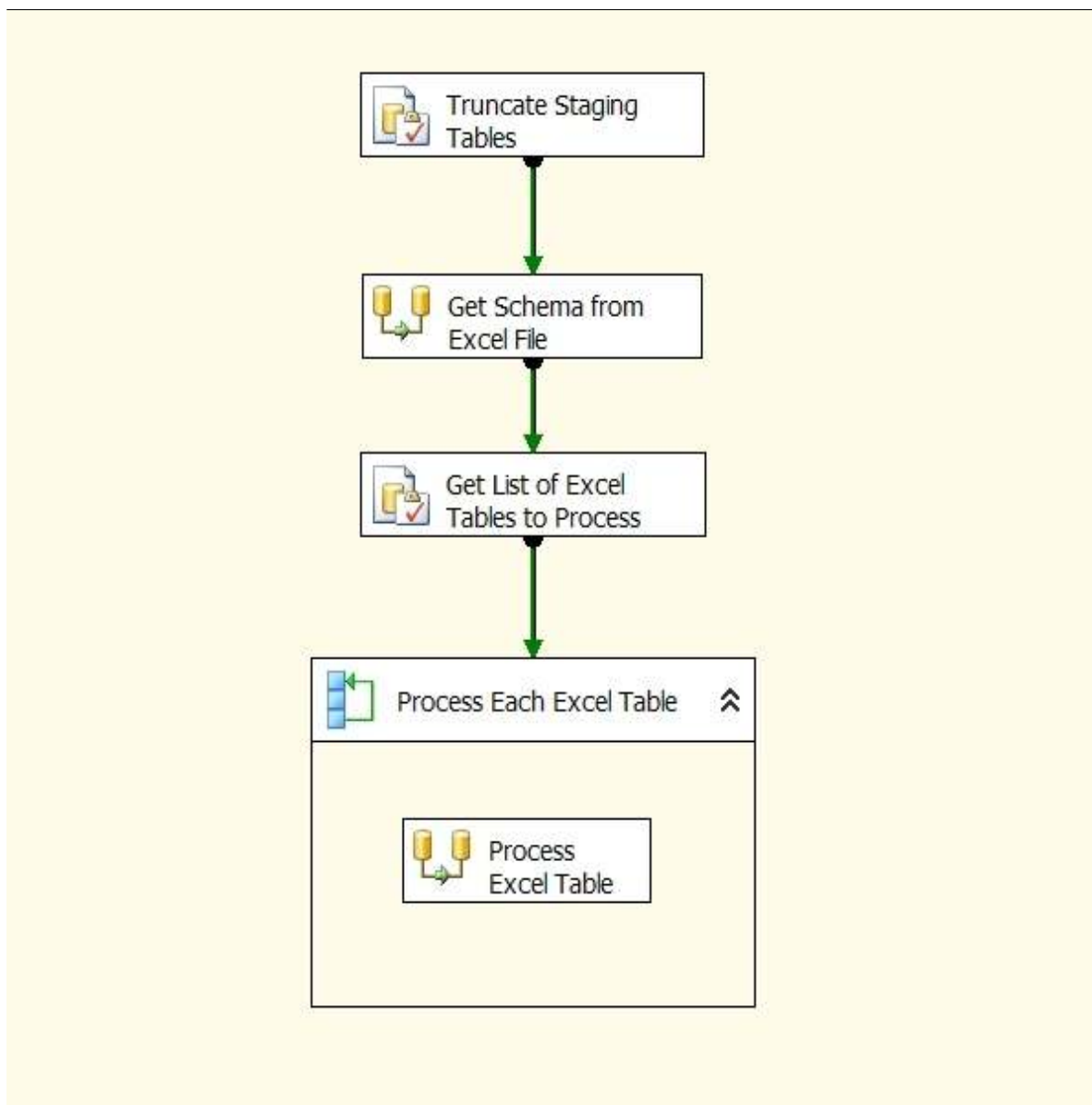
- We have an external system that generates an Excel spreadsheet with a list of invoices by week.
- The spreadsheet can be generated for any number of weeks; each week is in its own sheet.
- The sheet names represent the year and the week number in the year.

Based on our knowledge of the Excel Source that we use in a Data Flow task, we need to know the sheet name in order to import the data. In our example, however, the sheet name varies and the number of sheets also varies. What we need then is a way to query the Excel spreadsheet and get the list of sheets.

You can get the schema information from an Excel spreadsheet by using the OleDbConnection .NET Framework class. The OleDbConnection class has a method called GetOleDbSchemaTable that will return the list of sheets (i.e. tables) in a spreadsheet and the list of columns in a particular sheet. Let's create a simple SSIS package to demonstrate how to query this information and process the sheets. For additional details on the GetOleDbSchemaTable method, refer to this [article](#) on the Microsoft web site.

### ***Sample SSIS Package***

We'll create an SSIS package that will process all of the sheets in a single Excel file. The Excel file to process will be specified on the command line and stored in the package variable ExcelFilePath. Our SSIS sample package will have the following control flow:



- **Truncate Staging Tables** is an Execute SQL task that truncates two staging tables used during processing.
- **Get Schema from Excel File** is a Data Flow task that retrieves the schema information from each sheet in the Excel spreadsheet and stores it in the stg\_ExcelMetadata staging table.
- **Get List of Excel Tables to Process** is an Execute SQL task that gets the distinct list of tables from the stg\_ExcelMetadata table and stores them in the package variable ExcelTableList.
- **Process Each Excel Table** is a Foreach Loop Container task that iterates through the list of Excel tables in the ExcelTableList package variable. Each time through the loop the Excel table to be processed is stored in the ExcelTable package variable.
- **Process Excel Table** is a Data Flow task that reads the data from the single Excel sheet per the ExcelTable package variable and inserts the data into the staging table stg\_Invoice.

The Get Schema from Excel File task is the most interesting part of our sample SSIS package and it looks like this:



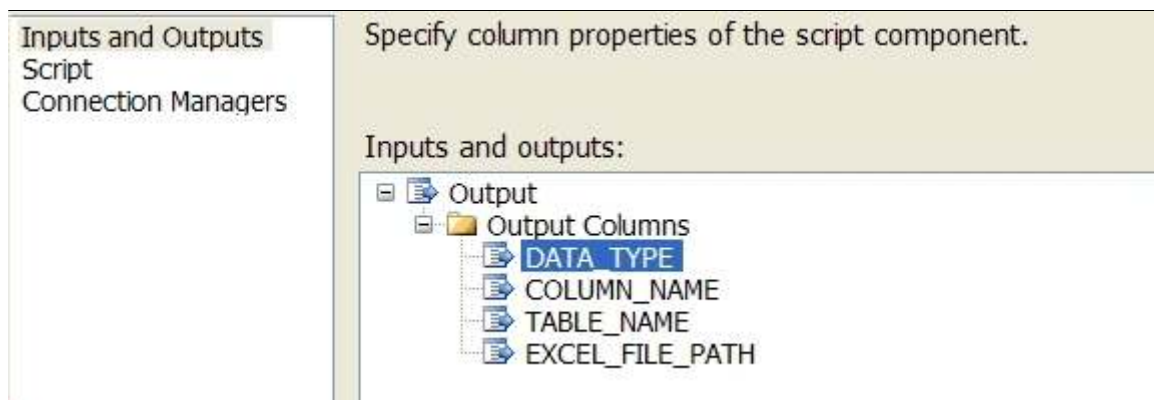
Get Excel Metadata is a Script Source. It contains the VB.NET code that retrieves the schema information from the Excel file. Write to Staging Table is an OLE DB Destination that inserts the schema information into the stg\_ExcelMetadata table which is defined as follows:

```

CREATE TABLE [dbo].[stg_ExcelMetadata] (
  [EXCEL_FILE_PATH] [nvarchar](256) NULL,
  [DATA_TYPE] [nvarchar](50) NULL,
  [COLUMN_NAME] [nvarchar](50) NULL,
  [TABLE_NAME] [nvarchar](50) NULL
)
  
```

A Script Source allows you to write VB.NET code to retrieve data from just about any source and insert it into the data flow. The key point is that you have the entire .NET Framework at your disposal. There are two steps in the configuration of the Script Source component:

**Step 1:** Define the output columns; these are the columns that you want to insert into the data flow. In our case they are all strings:



**Step 2:** Write the VB.NET code:

```

Public Overrides Sub CreateNewOutputRows()
  Dim excelFilePath As String = Me.Variables.ExcelFilePath.ToString()
  Dim strCn As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=" + excelFilePath + ";Extended Properties=Excel 8.0"
  Dim dtTables As DataTable
  Dim dtColumns As DataTable
  Dim tableName As String
  
```

```
Dim cn As OleDbConnection = New OleDbConnection(strCn)
cn.Open()
dtTables = cn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, Nothing)
For Each row As DataRow In dtTables.Rows
    tableName = row("TABLE_NAME").ToString()
    dtColumns = cn.GetOleDbSchemaTable(OleDbSchemaGuid.Columns, _
        New Object() {Nothing, Nothing, tableName, Nothing})
    For Each columnRow As DataRow In dtColumns.Rows
        OutputBuffer.AddRow()
        OutputBuffer.EXCELFILEPATH = excelFilePath
        OutputBuffer.TABLENAME = tableName
        OutputBuffer.COLUMNNAME = columnRow("COLUMN_NAME").ToString()
        OutputBuffer.DATATYPE = columnRow("DATA_TYPE").ToString()
    Next
Next
cn.Close()
OutputBuffer.SetEndOfRowset()
End Sub
```

The main points about the above code snippet are:

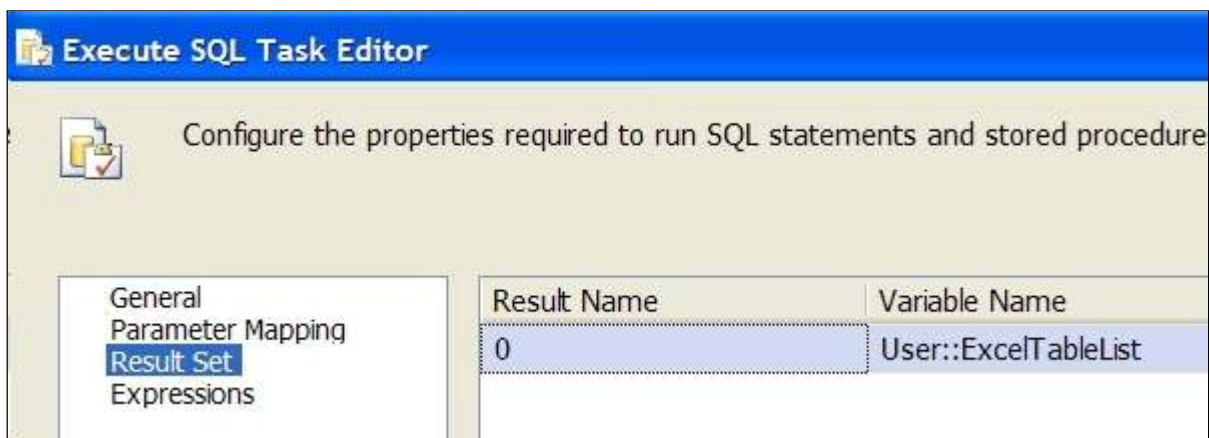
- In a Script Source component you add VB.NET code to the CreateNewOutputRows subroutine to retrieve data and insert it into the data flow.
- The ExcelFilePath package variable is passed in to the Script Source component and used in the connection string.
- Create an OleDbConnection object, open the connection and call the GetOleDbSchemaTable method to retrieve the list of sheets and the columns in each sheet.
- The GetOleDbSchemaTable method returns a DataTable; this is a standard ADO.NET class that has rows and columns.
- The OutputBuffer class is used to add rows to the data flow and also to assign values to the output columns. The OutputBuffer class gets its name based on the name you specify on the Inputs and Outputs page in Step 1 above.
- You should call the SetEndOfRowset method on the OutputBuffer to indicate that you are done adding rows.

The Get List of Excel Tables to Process task executes a query to get the list of sheets in the Excel spreadsheet (from the staging table) then stores the list in the package variable ExcelTableList. The Process Each Excel Table task iterates over the list of tables in the Excel spreadsheet and executes the Process Each Excel Table task on each table. This technique was also used in our earlier tip [How To Implement Batch Processing in SQL Server Integration Services \(SSIS\)](#).

There are two steps required to configure the Get List of Excel Tables to Process task:

<b>General</b>	
Name	Get List of Excel Tables to Process
Description	Execute SQL Task
<b>Options</b>	
TimeOut	0
CodePage	1252
<b>Result Set</b>	
ResultSet	Full result set
<b>SQL Statement</b>	
ConnectionType	OLE DB
Connection	localhost.mssqltips
SQLSourceType	Direct input
SQLStatement	SELECT DISTINCT TABLE_NAME FROM dbo.stg_ExcelMetadata
IsQueryStoredPro	False
BypassPrepare	True

The General page (shown above) is where you specify the query; it's just selecting the list of tables from the staging table that we populated in the Get Schema from Excel File task. Setting the ResultSet to Full result set allows us to capture the query results into a package variable which we need to specify on the Result Set page:



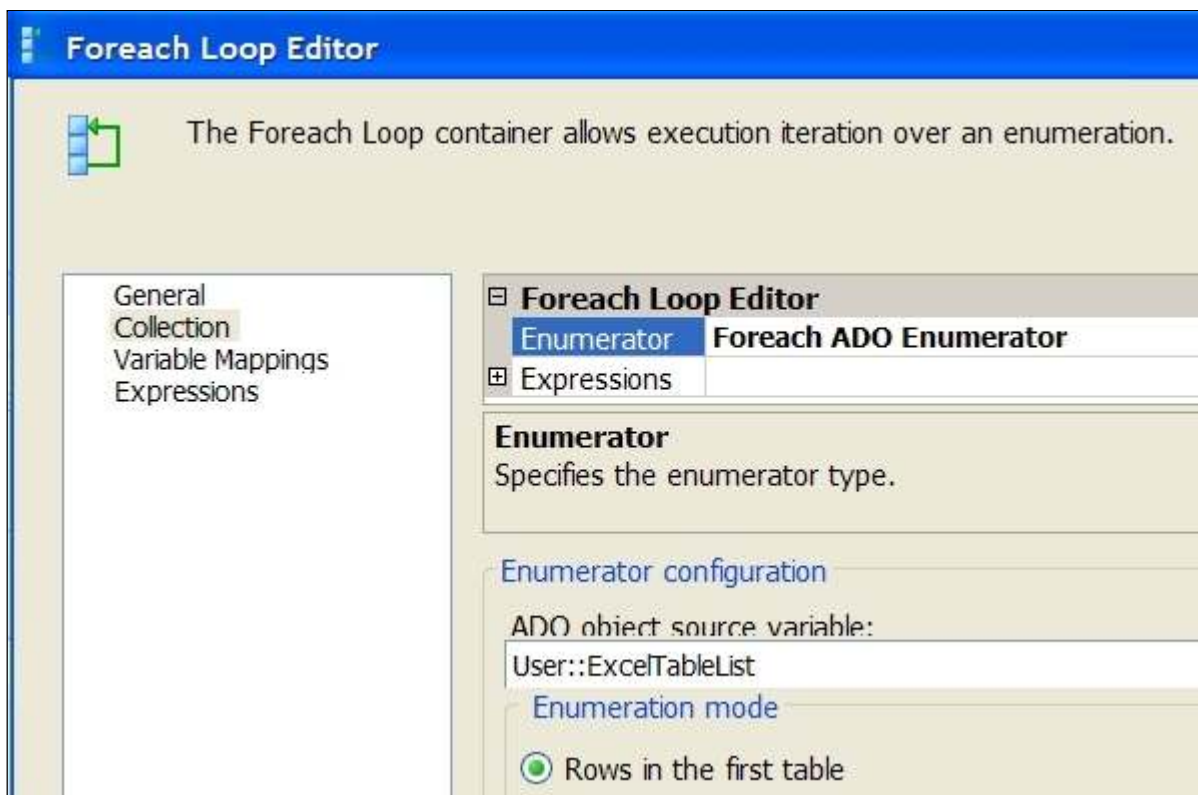
The image shows the 'Execute SQL Task Editor' dialog box. It has a blue title bar and a yellow background. On the left, there is a tree view with four items: 'General', 'Parameter Mapping', 'Result Set', and 'Expressions'. 'Result Set' is selected and highlighted in blue. To the right of the tree view, there is a text area with the instruction 'Configure the properties required to run SQL statements and stored procedure'. Below this, there is a table with two columns: 'Result Name' and 'Variable Name'. The 'Result Name' column has a value of '0' and the 'Variable Name' column has a value of 'User::ExcelTableList'.

General	Result Name	Variable Name
Parameter Mapping	0	User::ExcelTableList
Result Set		
Expressions		

Note that the data type of the ExcelTableList variable must be Object (i.e. the .NET Framework System.Object class) in order for the variable to hold the list of tables from our query.

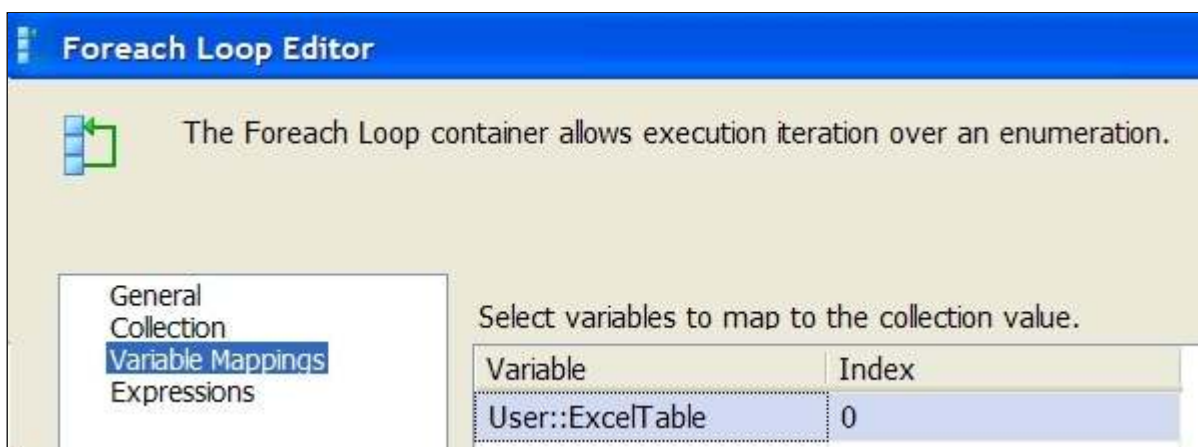
There are two steps required to configure the the Process Each Excel Table task:



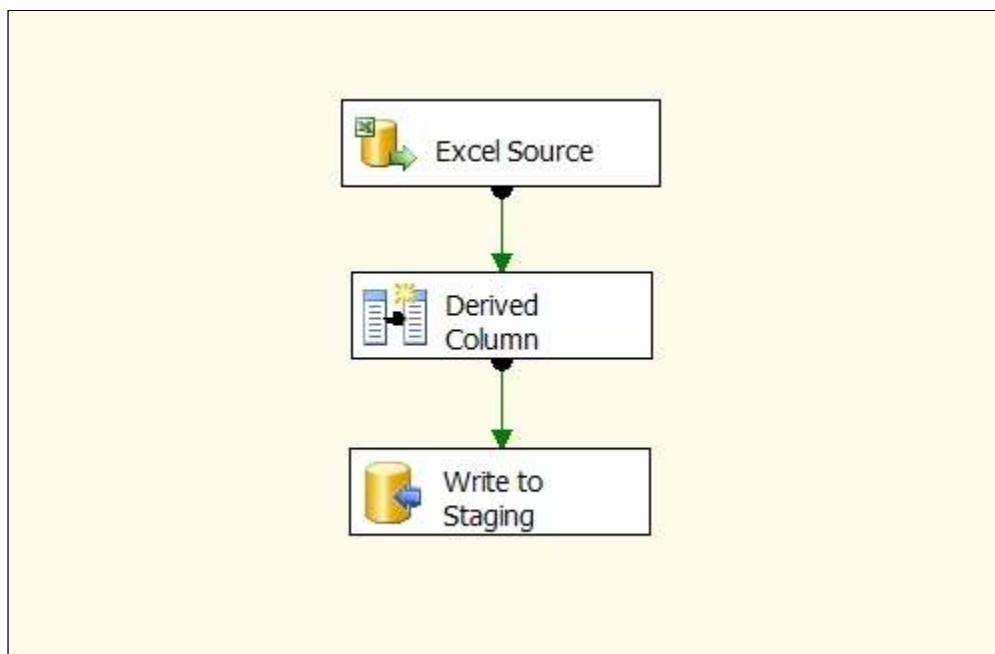


The Collection page (shown above) is where you specify the type of enumerator; in our case it must be Foreach ADO Enumerator. The ADO object source variable is ExcelTableList which is the variable we specified for the Result Set in the Get List of Excel Tables task. For Enumeration Mode we pick Rows in the first table (there is only one table in our result set).

The Variable Mappings page is used to assign value(s) from the result set to package variable(s) during each iteration. We'll use a package variable named ExcelTable:



Finally the last step in our SSIS package is the Process Excel Table task which looks like this:



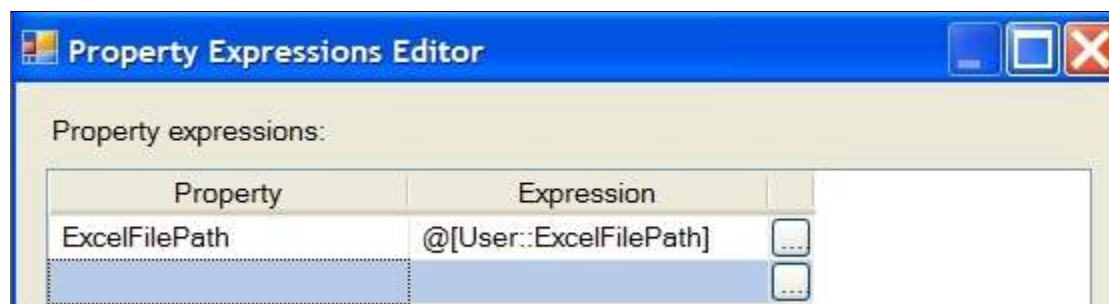
The main points about the above Data Flow task are:

- The Excel Source reads a single sheet from our Excel file.
- The Derived Column task adds the ExcelFilePath and ExcelTable package variables to the data flow so that we can save these in the staging table.
- Write to Staging is an OLD DB Destination that inserts rows into the staging table stg\_Invoice.

The staging table is defined as follows:

```
CREATE TABLE [dbo].[stg_Invoice] (  
    [ExcelFilePath] [nvarchar](255) NULL,  
    [ExcelTable] [nvarchar](255) NULL,  
    [InvoiceDate] [float] NULL,  
    [InvoiceNumber] [nvarchar](255) NULL,  
    [CustomerNumber] [nvarchar](255) NULL,  
    [InvoiceAmount] [float] NULL  
)
```

While the above Data Flow task is relatively straight forward, there are two subtle points that we need to take into consideration. First we need to configure the Excel Connection Manager that is used by the Excel Source. The ExcelFilePath property needs to be set to the ExcelFilePath package variable that is passed in on the command line. Click the button in the Expressions property of the Excel Connection Manager and assign the ExcelFilePath package variable to the ExcelFilePath property as shown below:



Second remember that the Data Flow task is being executed once for each sheet in our Excel



spreadsheet. The way to make this work is to configure the Excel Source Connection Manager page to specify the Data access mode as Table name or view name variable and select the ExcelTable package variable as the Variable name as shown below:

OLE DB connection manager:

Excel Connection Manager

Data access mode:

Table name or view name variable

Variable name:

User::ExcelTable

As the Process Each Excel Table task iterates through the list of sheets in the Excel file, it assigns each sheet to the ExcelTable package variable then executes the Process Excel Table task which operates on the sheet specified by the ExcelTable package variable.

### Running the SSIS Package

Run the sample SSIS package using the DTEXEC command line utility and set the ExcelFilePath package variable to the full path of the Excel file to process; e.g.:

```
DTEXEC /FILE ExcelMetadata.dtsx /SET "\Package.Variables[User::ExcelFilePath].Value";"c:\drop\sample.xls"
```

After running the package you can query the stg\_ExcelMetadata table to see the schema information:

	EXCEL_FILE_PATH	DATA_TYPE	COLUMN_NAME	TABLE_NAME
1	c:\drop\sample.xls	130	CustomerNumber	'200430\$'
2	c:\drop\sample.xls	5	InvoiceAmount	'200430\$'
3	c:\drop\sample.xls	5	InvoiceDate	'200430\$'
4	c:\drop\sample.xls	130	InvoiceNumber	'200430\$'
5	c:\drop\sample.xls	130	CustomerNumber	'200431\$'
6	c:\drop\sample.xls	5	InvoiceAmount	'200431\$'
7	c:\drop\sample.xls	5	InvoiceDate	'200431\$'
8	c:\drop\sample.xls	130	InvoiceNumber	'200431\$'
9	c:\drop\sample.xls	130	CustomerNumber	'200432\$'
10	c:\drop\sample.xls	5	InvoiceAmount	'200432\$'
11	c:\drop\sample.xls	5	InvoiceDate	'200432\$'
12	c:\drop\sample.xls	130	InvoiceNumber	'200432\$'

Note that the TABLE\_NAME column has a '\$' character at the end of it and is enclosed in single quotes. The sheet name does not show the '\$' character or the single quotes in Excel. The DATA\_TYPE column values are: 5=double precision, 130=text. You can find the details on additional types [here](#). Excel supports a very small set of column types.

You can also query the stg\_Invoice table to see the data that was loaded from the Excel sheets:

Results		Messages			
	ExcelFilePath	ExcelTable	InvoiceDate	InvoiceNumber	CustomerNumber
1	c:\drop\sample.xls	'200432\$'	20040801	SO74913	10-4030-011501
2	c:\drop\sample.xls	'200432\$'	20040801	SO74914	10-4030-016492
3	c:\drop\sample.xls	'200432\$'	20040801	SO74915	10-4030-021991
4	c:\drop\sample.xls	'200432\$'	20040801	SO74916	10-4030-024363
5	c:\drop\sample.xls	'200432\$'	20040801	SO74917	10-4030-028782
6	c:\drop\sample.xls	'200432\$'	20040801	SO74918	10-4030-012093
7	c:\drop\sample.xls	'200432\$'	20040801	SO74919	10-4030-011262
8	c:\drop\sample.xls	'200432\$'	20040801	SO74920	10-4030-011506
9	c:\drop\sample.xls	'200432\$'	20040801	SO74921	10-4030-029024
10	c:\drop\sample.xls	'200432\$'	20040801	SO74922	10-4030-011330

The results above only show the first ten rows; if you scroll through all of the results you will see rows from each of the three Excel sheets in our sample Excel spreadsheet.

### **Next Steps**

- Take a look at the sample code [here](#) and experiment with retrieving schema information from an Excel file.
- Although the column information from the Excel sheets was retrieved, it wasn't actually used in the example. However, you could certainly use the column list in a variety of ways; e.g. you could validate that all columns you expect are available and abort with an appropriate error if necessary.
- The Script Source component in the data flow is a great way to retrieve data using VB.NET code and get that data into the data flow. SQL Server 2008 SSIS supports C# code in Script components in addition to VB.NET.
- The Foreach Loop Container task is a very useful technique for implementing a batch or repetitive type of process.
- Each sheet in the example Excel spreadsheet has the same schema; i.e. column list. To process sheets with different column lists you would need a separate Data Flow task for each distinct list of columns. You could also use a Script task to write ADO.NET code to insert the Excel data into a table. The Data Flow doesn't support a dynamic column list.

Copyright (c) 2006-2009 [Edgewood Solutions, LLC](#) All rights reserved  
[privacy statement](#) | [disclaimer](#) | [copyright](#)

Some names and products listed are the registered trademarks of their respective owners.