www.databasejournal.com/features/mssql/article.php/3919016

Back to Article

Accessing Excel Via ADO.NET Using SSIS Script Task
January 10, 2011

SQL Server 2008 Integration Services offers a variety of ways to access `Excel`-based data. The most popular ones (such as Export and Import Wizard, Data Conversion Transformation, or Derived Column Transformation, which we have already presented on this forum) are quite straightforward to implement, however their simplicity comes at the cost of limited flexibility. If you find their feature set too restrictive, you might want to consider using `Script Task`-based code to deliver functionality that meets your needs more elaborately. One way to accomplish such goal is to leverage functionality built into the `Microsoft OLE DB` provider as described in this article.

From the programming perspective, we will be leveraging the `System.Data.OleDb` namespace of `ADO.NET`, which facilitates access to properties and methods of the `OleDbConection` class implemented by the `Microsoft OLE DB` data source provider (more specifically, `Microsoft.ACE.OLEDB.12.0`) and available via downloadable `2007 Office System Driver: Data Connectivity Components`.

In our example, we will demonstrate a generic scenario involving enumerating `Excel` workbooks residing in an arbitrary location, identifying their individual worksheets and named ranges, and reading their content. Obviously it is up to you to determine whether such a `Script Task`-based approach is best suitable in your specific circumstances (in addition to the techniques mentioned in the beginning of this article, you can also use `Excel Source Data Flow` component for this purpose, which we will be covering in more detail in the near future).

Before we start designing our sample code, we need to point out a couple of caveats. Firstly, when creating packages on the `x64` platform, you are likely to encounter "*The Excel Connection Manager is not supported in the 64-bit version of SSIS, as no OLE DB provider is available*" error message once you attempt their compilation. This is the result of lack of `64`-bit version of `Excel Provider` in `Microsoft Office 2007` (`64`-bit support has been introduced in `Office 2010`). If that is the case,

select the Properties item from the context sensitive menu of your project displayed in the Solution Explorer window of Business Intelligence Developement Studio. In the resulting dialog box, switch to the Debugging section, and change the value of the Run64BitRuntime property to False (effectively, forcing package execution inside the development environment to be carried out in the 32-bit mode).

Secondly, when running the package outside of the development environment on x64 computers, make sure to use the 32-bit versions of DTExec.exe and DTExecUI.exe utilities (one way to establish whether this is the case involves verifying that they are located within the Program Files (x86)Microsoft SQL Server folder structure). In addition, when scheduling to run such packages as SQL Server Agent jobs, enable Use 32 bit runtime checkbox on the Execution options tab of the New Job Step dialog box (in the job's Properties dialog box of the SQL Server Management Studio interface).

Having covered our prerequisites, let's focus on the task at hand. Launch Business Intelligence Development Studio and create a new project based on the Integration Services template. Drag the Script Task icon from the Toolbox and drop it on the Designer interface. Display its Editor dialog box, designate Visual Basic .NET 2008 as the ScriptLanguage, and confirm your choice by clicking on the OK command button. Use the Variables window to define a new variable (we will call it FolderName) of String type and set its value to the name of a folder where Excel spreadsheets reside.

In our code, we need to dynamically define a connection object allowing us to interact with Excel files. In order to identify the connection string that should be assigned to it, we will create a temporary Excel Connection Manager entry. To accomplish this, select New Connection... in the context sensitive menu of the Connection Managers tab of the Designer interface. In the resulting Add SSIS Connection Manager dialog box, choose the EXCEL entry, specify the Excel file path, Excel version (we will be using Microsoft Excel 2007), and clear the First row has column names checkbox. Once the Excel Connection Manager appears under the Connection Managers tab, take note of the value of its ConnectionString property by referencing the Properties window (you can delete the connection manager afterwards).

Next, activate the Script Task Editor dialog box (accessible via Edit... option in its context sensitive menu), add the User::FolderName to the ReadOnlyVariables entry of the Script section, and click on the Edit Script... button to display Visual Studio Tools for Applications. Add the Imports System.Data.OleDb and Imports System.IO (used for file system operations) lines to the (General)(Declarations) section and copy the following code as the content of Public Sub Main():

```
Public Sub Main()

Dim strFile As String
Dim strConnection As String
Dim objConnection As OleDbConnection
Dim colTables As DataTable
Dim objTable As DataRow
Dim strOleDbCmd As OleDbCommand
Dim colReader As OleDbDataReader
Dim intCol As Integer

Dts.TaskResult = ScriptResults.Success

Try

    Dim strFolder = Dts.Variables("FolderName").Value.ToString
    Dim colFiles As String() = Directory.GetFiles(strFolder, "*.xlsx")

    For Each strFile In colFiles
        MessageBox.Show(strFile, "Files", MessageBoxButtons.OK)
        strConnection = "Provider=Microsoft.ACE.OLEDB.12.0;" & "Data Source=" & _
                        strFile & ";Extended Properties=""Excel 12.0 XML;HDR=NO"""
        objConnection = New OleDbConnection(strConnection)
        objConnection.Open()
        colTables = objConnection.GetSchema("Tables")

        For Each objTable In colTables.Rows
            MessageBox.Show(objTable.Item("TABLE_NAME").ToString)
            strOleDbCmd = New System.Data.OleDb.OleDbCommand("SELECT * FROM [" _
                + objTable.Item("TABLE_NAME").ToString + "]", objConnection)
            colReader = strOleDbCmd.ExecuteReader()
            If colReader.HasRows Then
                Do While colReader.Read()
                    For intCol = 0 To colReader.FieldCount - 1
                        If IsDBNull(colReader(intCol)) Then
                            MessageBox.Show("Null")
                        Else
                            MessageBox.Show(colReader.GetString(intCol))
                        End If
                    Next
                Loop
            End If
            colReader.Close()
```

```
        Next
        objConnection.Close()
    Next
Catch ex As Exception
    Dts.TaskResult = ScriptResults.Failure
    MessageBox.Show(ex.Message.ToString(), "Exception", MessageBoxButtons.OK)
End Try
End Sub
```

As you can see, we start our script by enumerating all files with extension `.xlsx` residing in the folder, which name is obtained by referencing the `FolderName SSIS` variable. For each of them, we establish an `OLE DB`-based connection, which allows us to identify their worksheets and named ranges (which are represented as tables in the `ADO.NET` object model). We retrieve their content by leveraging `DataReader` functionality, allowing us to traverse them row by row, displaying the content of individual cells.

» See All Articles by Columnist **Marcin Policht**