The Rational Guide To

Scripting with

SQL Server 2005 Integration Services

Beta Preview

Author:	Donald Farmer
ISBN:	1-932577-21-1
Audience:	Developer
Publication Date:	June 5, 2005
Description:	
Get an early look at how to use the scripting features of SQL Server 2005 Integration Services (SSIS) from the source – Donald Farmer. Donald is a Group Program Manager at Microsoft for SQL Server 2005 Integration Services. In less than 200 pages, this book covers the concepts, architecture, scripting tasks and components. In addition, Donald covers how to use .NET assemblies with your script tasks. This book contains numerous examples, code, and scripts that are available to download for free.	

Chapter 7

Your First Script Component

In Chapter 2, you learned about the three types of Data Flow components in SSIS: sources, transformations and destinations. This chapter will describe these different types and show you how to create your first transformation component using VB.NET in the Script Component. Once you have mastered the Script Component, you will find that you can readily integrate almost any functionality into the SSIS Data Flow. Custom scripted data sources can handle otherwise unsupported file formats. Custom transformations can call functions in managed assemblies, including the .NET Framework, and custom scripted destinations enable SSIS to output data in very flexible ways. In fact, it is quite possible to write sophisticated Data Flows for ETL and data integration using only Script Components!

Script Component Types

You will typically use script somewhat differently in each of the three component types:

Script Source Component

Source components have no input columns, but do have output columns. The purpose of the Script Component in this case is to deliver data to the output columns. For example, the component author could write a script using file and string handling functions to parse a complex text file whose format can not be handled by the Flat File Connection Manager. Text files like the one in Listing 7.1 are quite common:

```
RECORD _ START
REC _ ID: 17804-4
BATCH _ ID: O2 _ 2004 _ 5
CUST _ LNAME: MACLEOD
RECORD _ END

RECORD _ START
REC _ ID: 17815-2
ALT _ REC _ ID: Temp2 _ 31
BATCH _ ID: H1 _ 2004 _ 2
CUST _ LNAME: MACLEAN
RECORD _ END

RECORD _ START
REC _ ID: 17222-1
BATCH _ ID: O1 _ 2004 _ 2
CUST _ LNAME: MCLENNAN
RECORD _ END
```

Listing 7.1: A Good Candidate for a Script Source Component.

In this case, the text file has its data arranged in rows with labels rather than columns (as expected by the Flat File Connection Manager), and it has irregular information in each record (the second record contains an ALT_REC_ID item which is not in the first) which may need special handling. A script source component could readily handle this format, using .NET Framework string functions to convert the records and their items into rows and columns.

Script Transformation Component

Transformation components have input columns *and* output columns. In these components, the purpose of the script will typically be to transform the data in some way between inputs and outputs. What these transformations are is up to you—that is why scripts are so flexible. Chapter 2 discussed two different patterns of transformation: synchronous and asynchronous. Script Components can handle both patterns quite easily.

A synchronous component, you will remember, is particularly useful for row by row transformations. For example, I may have incoming data which includes customer last names. Perhaps for easy cross-referencing with other customer records, I would like to calculate a SOUNDEX value for each customer name. SOUNDEX is an algorithm invented by the US Census for codifying names to take account of different spellings. T-SQL has a SOUNDEX function, but SQL Server Integration Services does not. The component author can code their own SOUNDEX function in VB.NET script and transform every incoming customer name using that script, emitting the new SOUNDEX value at the output.

Note:

Soundex is an algorithm for matching strings—principally personal names—phonetically. Soundex has been used since the 1880 US Census to conform different spellings of surnames in census reports to a standard value. For example, Smith and Smyth have the same Soundex value. Soundex helped researchers using Census data to match and find surnames even with different spellings. The algorithm converts a name to a code where the first letter of the code is the first letter of the name, and a sequence of numbers represents the other syllables. Vowels are ignored, as are double consonants and the letters Y, H and W. So, in Soundex, Alan and Allen both become A45. Alonso becomes A452.

An asynchronous component is useful for performing operations which change the shape of the data significantly, or where incoming rows do not have related rows at the output. A good use of an asynchronous script would be to aggregate text. The SSIS Aggregate component is very powerful, but it can only perform **Min** and **Max** calculations against numeric columns. If you need to be able to calculate the **Min** and **Max** values of a string column, this can easily be achieved in VB.NET script, as shown in Chapter 14.

Script Destination Component

Listing 7.1 showed a text file which could not be parsed by the Flat File Connection Manager but which could be handled by a script source component. In an enterprise where text files like this are used by legacy applications, you may also need the ability to generate a file in this format for the legacy application to read. The script destination component is useful in these circumstances.

You will have guessed by now that a script destination component has input columns, but no output columns. Instead, it is the VB.NET script which handles the data, perhaps using file and string handling routines from the .NET Framework to output text files in the appropriate format.

Having discussed the various kinds of components which you can script, it is time to get started with our first Script Component.

Adding a Script Component to Your Package

The first Script Component we are going to look at it is a transformation component. In fact, we are going to build a simple SOUNDEX component to transform a column containing a name to a codified value representing the sound of the name, just as the US Census would do.

As this first component is a transformation, it requires some data to work with. To get us started, we can quickly build a package containing a Data Flow and a source component as follows.

Note:

This chapter assumes you have at least some familiarity with building a Data Flow with SSIS, even if just the simplest examples from the documentation or samples that ship with the product.

Preparing the package

Use the following steps to prepare the package:

1. Create a new SSIS Package.
2. In the designer, drag a **Data Flow Task** from the **Control Flow Items** tab of the Toolbox to the **Control Flow** design surface.
3. Double-click the **Data Flow Task** to open the **Data Flow** design surface.
4. Drag an **OLE DB Source** component from the **Data Flow Sources** tab of the Toolbox.

5. Double-click the **OLE DB Source** component shape on the design surface to open the **OLE DB Source Editor**.

6. Click the **New** button to create a new **OLE DB Connection Manager**.

7. Select an existing connection to your AdventureWorksDW database, or create a new one.

8. Click the **OK** button to return to the **OLE DB Source Editor**.

9. Select the **Table or View Data Access Mode**.

10. Select the **[dbo].[DimCustomer]** table. (I should say here that DimCustomer stands for Customer *Dimension* and is not intended as a reflection on the intelligence of AdventureWorks customers! DimEmployee, however, may be a different matter.)

11. Click **Preview** to see the data in this table. Note the **Last Name** field, which we shall be using later.

12. Click the **OK** button to close the **OLE DB Source Editor**.

Now we have some source data we can work with. At this stage, your package design should resemble Figure 7.1:

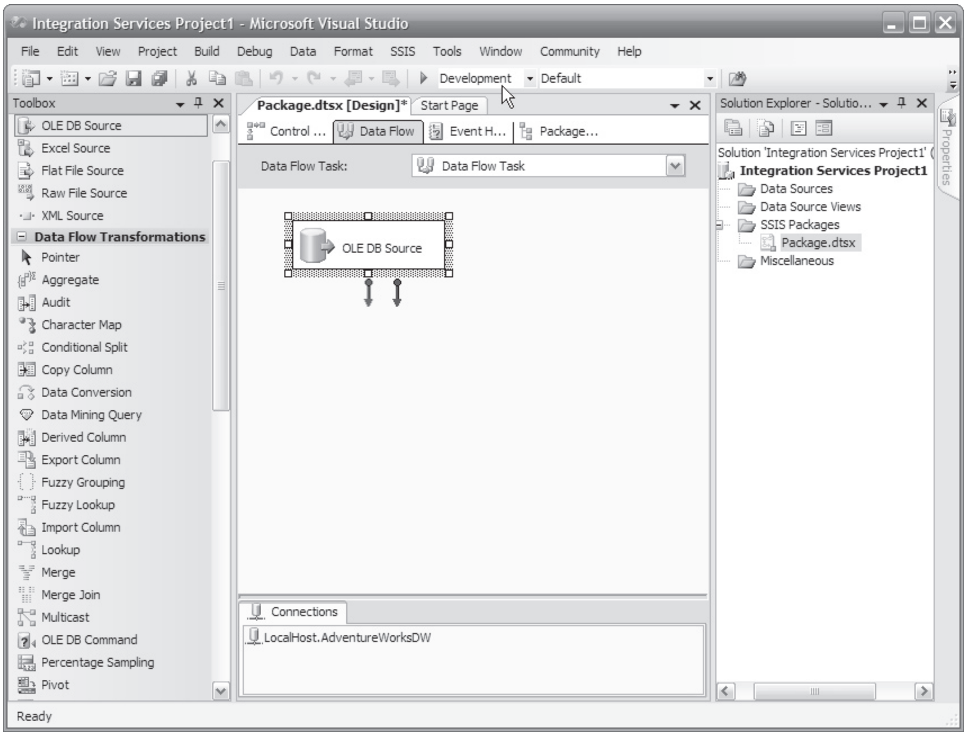


Figure 7.1: Package to Which We Will Add the First Script Component.

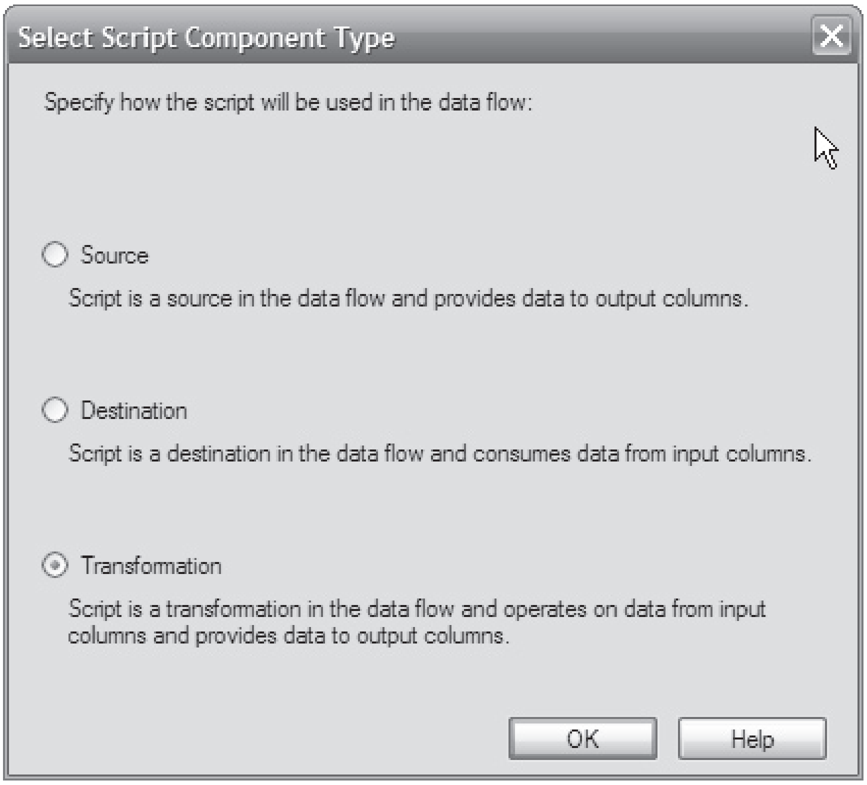
The next step is to add the Script Component.

Adding the Script Component

Do the following to add the Script Component:

1. In the designer, drag a **Script Component** from the **Data Flow Transformations** tab of the Toolbox to the **Data Flow** design surface.

2. When you drag the **Script Component**, the **Select Component Type** dialog box will appear, prompting you to select the kind of component you want to create. This is important, as the configuration of the Script Component is somewhat different for each type—this dialog sets up the component for you automatically, saving some work in creating this configuration yourself.


Select Script Component Type

Specify how the script will be used in the data flow:

Source
Script is a source in the data flow and provides data to output columns.

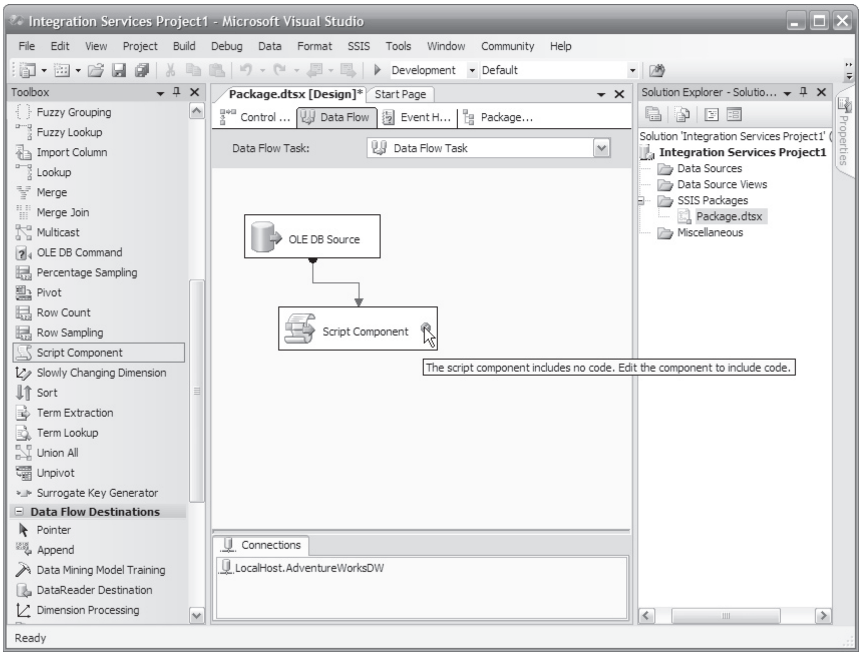
Destination
Script is a destination in the data flow and consumes data from input columns.

[✓ Transformation]
Script is a transformation in the data flow and operates on data from input columns and provides data to output columns.

Figure 7.2: Select Script Component Type Dialog Box.

3. In the **Select Script Component Type** dialog box, select **Transformation** (the default) and click the **OK** button.

4. Connect the output of the **OLE DB Source** to the **Script Component**. At this stage your package should resemble Figure 7.3:



The script component includes no code. Edit the component to include code.

Figure 7.3: Package with OLE DB Source and Script Component Connected.

Note the warning icon that appears in the **Script Component**. You can mouse over that to see the text of the warning. In this case, we have not yet added any script code, so the component at this point is in an invalid state.

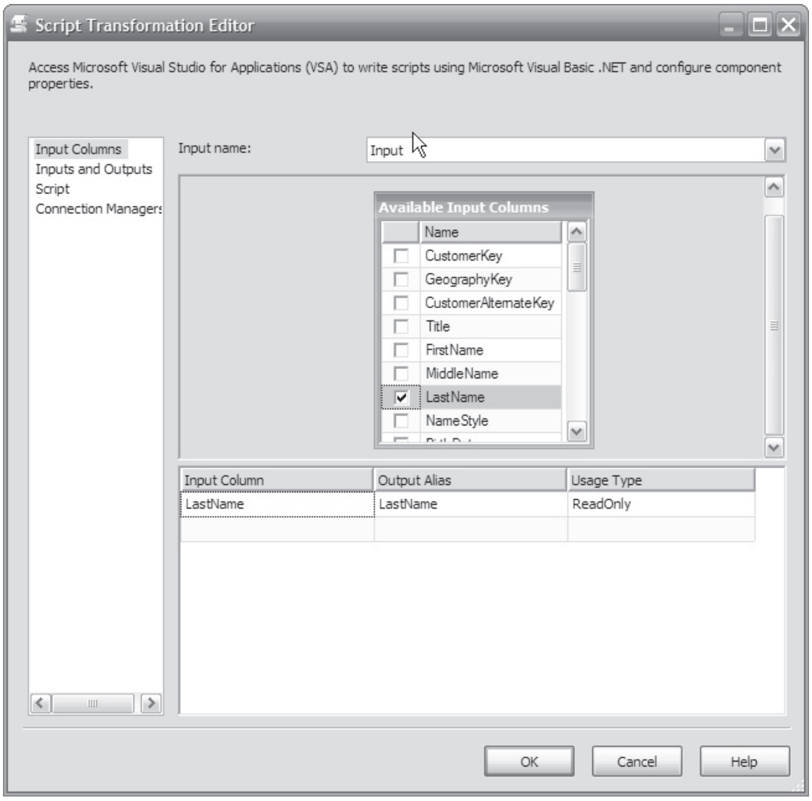
Adding Columns to Your Script Component

The next step is to tell the Script Component which input columns we want to work with. The Script Component requires you to explicitly say in advance which columns are to be used. This is because the component has to create a wrapper that exposes these columns to the scripting environment. It would be expensive and unnecessary to expose all the input columns by default when you may only be using one of them.

Selecting the input columns is easy. Simply perform the following steps:

1. Double-click the **Script Component** shape on the design surface to open the **Script Transformation Editor**.

2. The default view is of the **Input Columns** tab. In this case, check the box next to the **Last Name** column. The component editor should now look like the one in Figure 7.4:



Input Column	Output Alias	Usage Type
LastName	LastName	ReadOnly

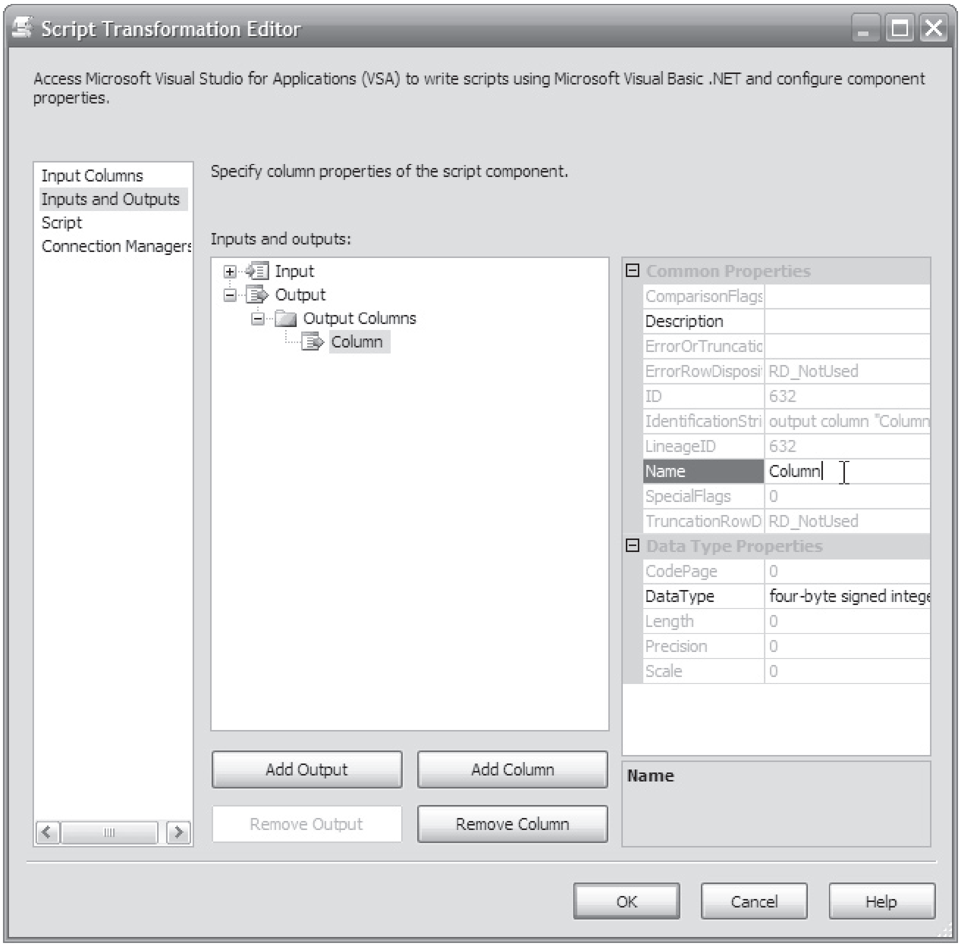
Figure 7.4: Component Editor with Last Name Column Selected.

Now we can use the **Last Name** column in our component. Note that we have left the **Usage Type** of the column as **Read Only**. This is because we do not want to *alter* the last name. We want to calculate a new column from the values in this column. But where will these new values go?

We need to create a new column to hold these new values. This new column will not appear at the input; it is being calculated by the script, so it can only appear at the output. So now, we should add an output column, as follows:

1. Select the **Inputs and Outputs** tab of the **Script Transformation Editor**.
2. Expand the **Output** node of the **Inputs and Outputs** tree view.
3. Select the **Output Columns** folder under the **Output** node.
4. Click the **Add Column** button.

This will add a new output column, named **Column**. The data type will be a four-byte signed integer. You can see this in Figure 7.5:


Script Transformation Editor

Access Microsoft Visual Studio for Applications (VSA) to write scripts using Microsoft Visual Basic .NET and configure component properties.

Specify column properties of the script component.

Inputs and outputs:

Common Properties	
ComparisonFlags	
Description	
ErrorOrTruncatic	
ErrorRowDisposi	RD_NotUsed
ID	632
IdentificationStri	output column "Column
LineageID	632
Name	Column
SpecialFlags	0
TruncationRowD	RD_NotUsed
Data Type Properties	
CodePage	0
DataType	four-byte signed intege
Length	0
Precision	0
Scale	0

Name

Figure 7.5: New Output Column.

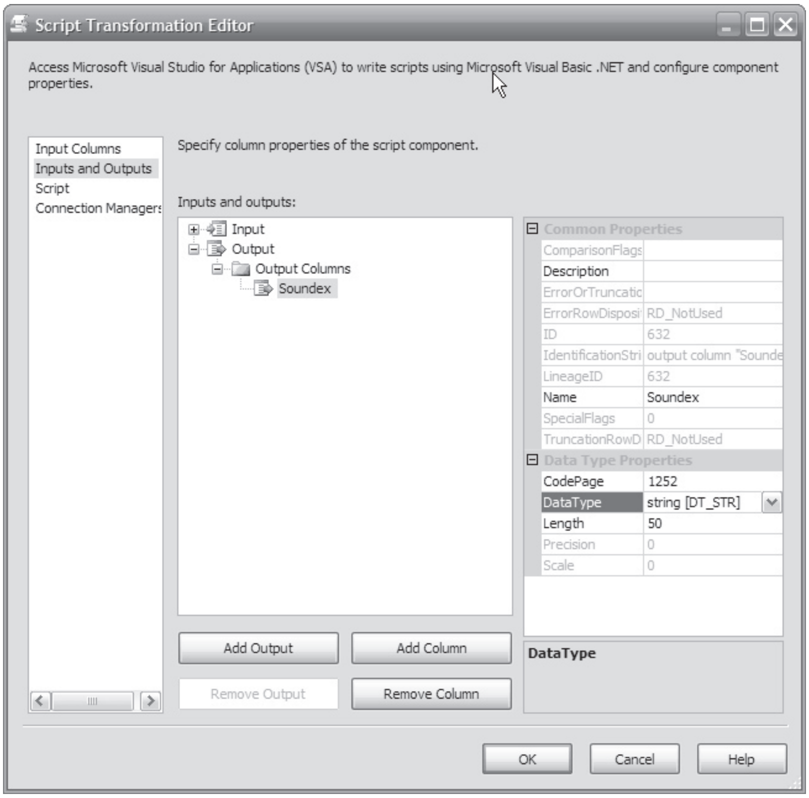
This default column is not quite what we need. We can, however, edit its properties, as follows:

1. Edit the **Name** property of the column to be **Soundex**. The column name should change in the tree view, too.

2. Soundex codes are strings, so select the **Data Type** property of the column and use the drop-down list to select the type **String [DT_STR]**.

Note that [**DT_STR**] is the SSIS data type. SSIS data types are sometimes slightly different from SQL, Visual Basic, or .NET Framework types because SSIS uses very deterministic types in order to optimize memory usage.

At this stage, your component should look like the one in Figure 7.6:



Script Transformation Editor

Access Microsoft Visual Studio for Applications (VSA) to write scripts using Microsoft Visual Basic .NET and configure component properties.

Specify column properties of the script component.

Inputs and outputs:

```
⊞ Input
⊟ Output
   ⊟ Output Columns
        Soundex
```

Common Properties	
ComparisonFlags	
Description	
ErrorOrTruncatic	
ErrorRowDisposi	RD_NotUsed
ID	632
IdentificationStri	output column "Sounde
LineageID	632
Name	Soundex
SpecialFlags	0
TruncationRowD	RD_NotUsed
Data Type Properties	
CodePage	1252
DataType	string [DT_STR]
Length	50
Precision	0
Scale	0

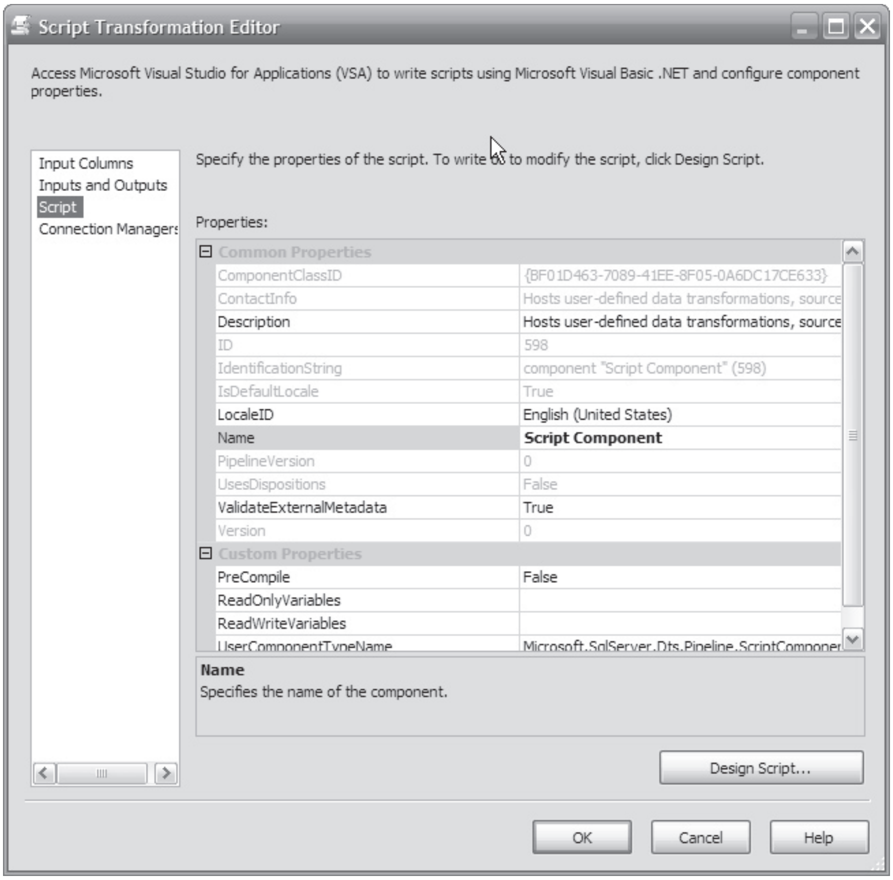
DataType

Figure 7.6: Renamed Output Column.

We have selected an input column (**LastName**) and created an output column (**Soundex**) to hold the transformed value. Now we are ready to write some code!

Elements of the Script

Navigate to the **Script** tab of the component. The component editor will now look like the one in Figure 7.7:

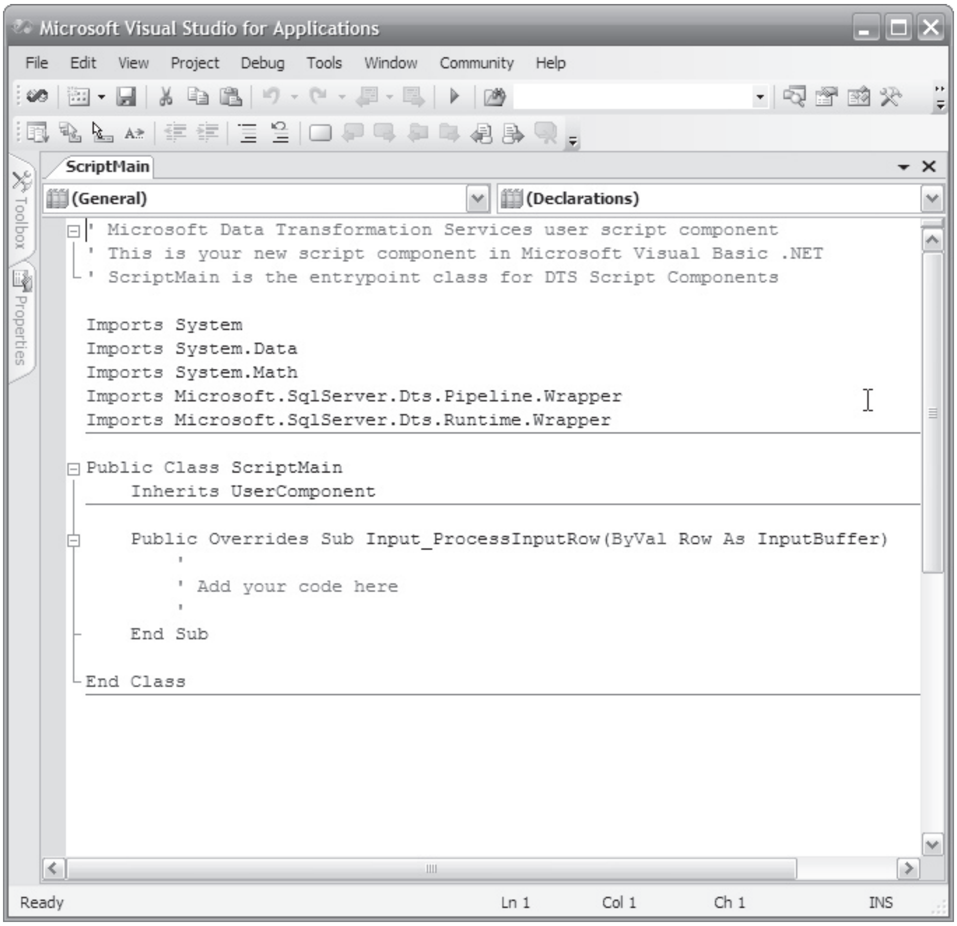

Script Transformation Editor
Access Microsoft Visual Studio for Applications (VSA) to write scripts using Microsoft Visual Basic .NET and configure component properties.
Input Columns
Inputs and Outputs
Script
Connection Managers
Specify the properties of the script. To write or to modify the script, click Design Script.
Properties:
Design Script...
OK
Cancel
Help

Common Properties	
ComponentClassID	{BF01D463-7089-41EE-8F05-0A6DC17CE633}
ContactInfo	Hosts user-defined data transformations, source
Description	Hosts user-defined data transformations, source
ID	598
IdentificationString	component "Script Component" (598)
IsDefaultLocale	True
LocaleID	English (United States)
Name	**Script Component**
PipelineVersion	0
UsesDispositions	False
ValidateExternalMetadata	True
Version	0
Custom Properties	
PreCompile	False
ReadOnlyVariables	
ReadWriteVariables	
UserComponentTypeName	Microsoft.SqlServer.Dts.Pipeline.ScriptComponer

Name
Specifies the name of the component.

Figure 7.7: Script Tab.

There are some useful properties available in the property grid, but they are, in fact, optional. For now, to get started, just click the **Design Script** button. The **Visual Studio for Applications** editor will open, as shown in Figure 7.8:



Microsoft Visual Studio for Applications
File Edit View Project Debug Tools Window Community Help
Toolbox
Properties
ScriptMain
(General)
(Declarations)
Ready
Ln 1
Col 1
Ch 1
INS

```
' Microsoft Data Transformation Services user script component
' This is your new script component in Microsoft Visual Basic .NET
' ScriptMain is the entrypoint class for DTS Script Components

Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper

Public Class ScriptMain
    Inherits UserComponent

    Public Overrides Sub Input_ProcessInputRow(ByVal Row As InputBuffer)
        '
        ' Add your code here
        '
    End Sub

End Class
```

Figure 7.8: Script Editing Environment.

Chapter 3 discussed the VSA environment and its various elements. These should be familiar to you now, but some aspects of the Script Component will certainly be new, compared to the script task we looked at earlier.

In the Script Task, the **Main()** subroutine was where most of the work was done, but here in the Script Component, we see the following code:

```
Public Class ScriptMain
    Inherits UserComponent
    Public Overrides Sub Input_ProcessInputRow(ByVal Row As InputBuffer)
        `
        ` Add your code here
        `
    End Sub
End Class
```

Listing 7.3: ScriptMain Class in the Script Component.

The **ScriptMain** class performs those duties mentioned earlier—wrapping and exposing the Data Flow columns and objects to the script. In Listing 7.3, within `ScriptMain`, you can see the following subroutine:

```
Input_ProcessInputRow(ByVal Row As InputBuffer)
```

What does this mean? We'll cover this in more detail later, but at this stage all we need to know is that when the input to the Script Component is processed, this routine is called, passing in the `InputBuffer` (the set of input rows being processed) as the object **Row**. In practice, this means that we can immediately start writing useful script code.

Editing the Script Component

The example we are going to implement is a function to convert the Customer's `LastName` to a Soundex code.

Listing 7.4 shows the function we which we will use for Soundex, written in VB.NET code:

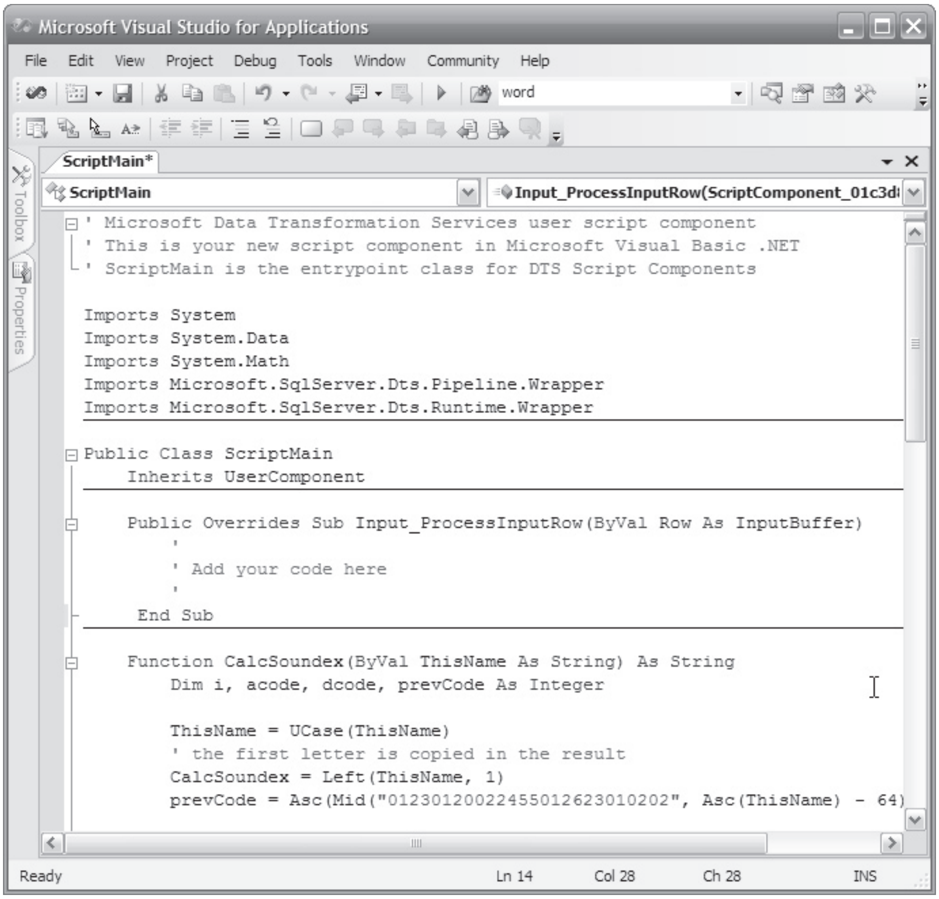
```
Function CalcSoundex(ByVal sName As String)
   As String   Dim i, acode, dcode, prevCode As Integer

   SName = UCase(SName)
   ‘ the first letter is copied in the result
   CalcSoundex = Left(SName, 1)
   prevCode = Asc(Mid(“01230120022455012623010202”, Asc(SName) - 64))

   For i = 2 To Len(SName)
      acode = Asc(Mid(SName, i, 1)) - 64
      ‘ discard non-alphabetic chars
      If acode >= 1 And acode <= 26 Then
         ‘ convert to a digit
         dcode = Asc(Mid(“01230120022455012623010202”, acode, 1))
         ‘ don’t insert repeated digits
         If dcode <> 48 And dcode <> prevCode Then
            CalcSoundex = CalcSoundex & Chr(dcode)
            If Len(CalcSoundex) = 4 Then Exit For
            End If
            prevCode = dcode
         End If
   Next
End Function
```

Listing 7.4: Simple Soundex Function in VB.NET Script.

You don’t really need to understand much about the internals of this function for this example, except to know that it returns a code based on the consonants in a string according to the Soundex algorithm described earlier.

For now, let’s paste this function into our script code, immediately after the `Input _ ProcessInputRow` function, but still within `ScriptMain();`

Your script editor will now resemble Figure 7.9:

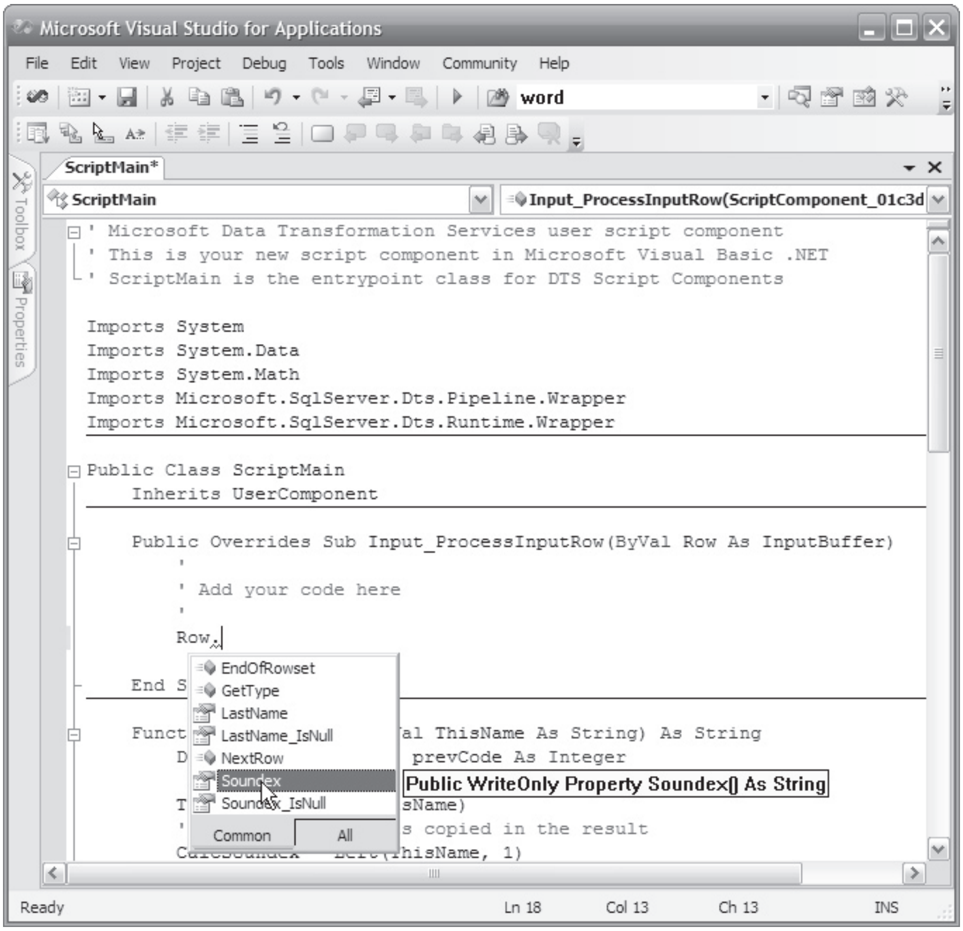

Microsoft Visual Studio for Applications
File Edit View Project Debug Tools Window Community Help
word
ScriptMain*
ScriptMain
Input_ProcessInputRow(ScriptComponent_01c3d
Toolbox
Properties
Ready
Ln 14
Col 28
Ch 28
INS

```
' Microsoft Data Transformation Services user script component
' This is your new script component in Microsoft Visual Basic .NET
' ScriptMain is the entrypoint class for DTS Script Components

Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper

Public Class ScriptMain
    Inherits UserComponent

    Public Overrides Sub Input_ProcessInputRow(ByVal Row As InputBuffer)
        '
        ' Add your code here
        '
     End Sub

    Function CalcSoundex(ByVal ThisName As String) As String
        Dim i, acode, dcode, prevCode As Integer

        ThisName = UCase(ThisName)
        ' the first letter is copied in the result
        CalcSoundex = Left(ThisName, 1)
        prevCode = Asc(Mid("01230120022455012623010202", Asc(ThisName) - 64)
```

Figure 7.9: CalcSoundex Function Pasted into the Editor.

The next step is reference the function in our script. For each row coming in to the component (in the input buffer) we wish to set the value of the `Soundex` column to the value returned by `CalcSoundex` when we pass in the value of the `LastName` column. This is remarkably easy.

Just below the line which reads "`Add your code here`", type `Row`. That is, Row followed by a period. As shown in Figure 7.10, **Intellisense** offers a list of options to complete this object, including the available columns—in this case **LastName** and **Soundex**.


Microsoft Visual Studio for Applications
File Edit View Project Debug Tools Window Community Help
word
ScriptMain*
ScriptMain
Input_ProcessInputRow(ScriptComponent_01c3d
Toolbox
Properties

```
' Microsoft Data Transformation Services user script component
' This is your new script component in Microsoft Visual Basic .NET
' ScriptMain is the entrypoint class for DTS Script Components

Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper

Public Class ScriptMain
    Inherits UserComponent

    Public Overrides Sub Input_ProcessInputRow(ByVal Row As InputBuffer)
        '
        ' Add your code here
        '
        Row.
```

Figure 7.10: Using Intellisense in the Scripting Environment.

It is worth noting that there are also specific options for handling **NULL** values in those columns. In this case, select **Soundex** and complete the rest of the statement as follows. (If you like, use **Intellisense** to complete the other references in the statement).

```
Row.Soundex = CalcSoundex(Row.LastName)
```

That's all—your script is ready to run! You can close the scripting environment. The script is automatically saved for you. When you return to the **Script Transformation Editor**, it is a good idea to set the **PreCompile** property to **True**; this option compiles the script into an assembly which is embedded in the SSIS package. This saves some time at runtime, especially if the script is complex or has many references to other assemblies. Compiled script runs with excellent performance, but do be aware that in SSIS all script components are compiled—this option just determines whether they are compiled in advance or just-in-time.

Now click the **OK** button in the **Script Transformation Editor** and return to the **Data Flow** design surface. The validation warning icon has now gone—your script is valid and you are ready to complete the Data Flow for testing.

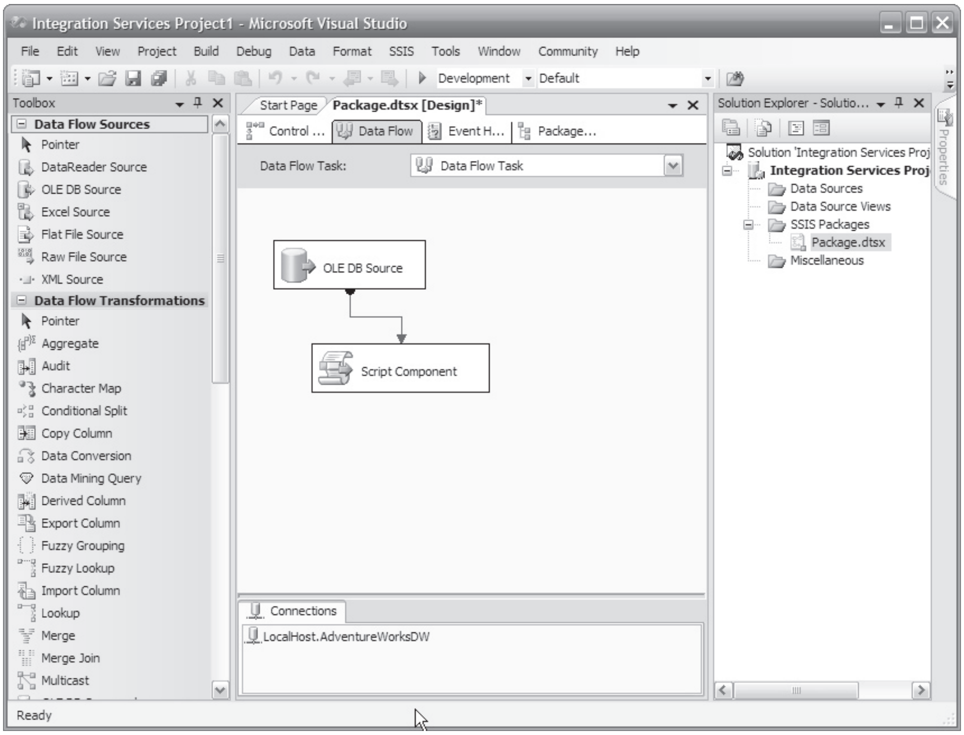


Figure 7.11: Completed Script Component in the Data Flow. (Note that the validation warning icon has been cleared).

Debugging the Script Component

Chapter 4 showed how to debug the Script Task by setting breakpoints within the script itself. The VSA design environment does enable you to set breakpoints in a Script Component, but the SSIS designer and debugger ignores them when executing a package. There are a number of reasons for this, but it is mainly due to the nature of the Script Component, which (unlike the Script Task) does not call a script only once, but many times within an execution: for each incoming row, or each incoming buffer.

However, SSIS does provide some very elegant techniques for debugging Data Flows in general, and you can use these techniques to understand if a Script is performing as expected. Later, we will look at more fine-grained, row-by-row debugging. For now, we will see how to debug the script working against the Data Flow as many rows pass through it.

Debugging a Data Flow with a RowCount Component

You might think that you could execute the SQL Server Integration Services package right now, with just an **OLE DB Source** and a **Script Component**. In fact, this is possible, but not much will happen. If you were to try it, you would see no activity. The explanation would be found on the **Progress** tab of the designer where you would meet the following message: **Warning: Source "OLE DB Source Output" () will not be read because none of its data ever becomes visible outside the Data Flow Task**.

SQL Server Integration Services has recognized that your data is not going anywhere – and optimized out the components! And of course, the data from your Script Component, including your **Soundex** column, is *not* going anywhere, so you have no opportunity to see this data.

In most Data Integration applications, you would need to send the data to a temporary destination in order to examine it and see if it had been transformed correctly. SQL Server Integration Services provides a great way to debug your data *without* having to create temporary tables or files.

The **Row Count** component simply counts the passing rows wherever you add it to the Data Flow. This component can be very useful for debugging, because it gives you a count of the rows at a particular point, and also because SQL Server Integration Services will not optimize out a Row Count component. This behavior enables you to execute the Data

Flow with a Row Count component at the end of the flow, even though no destination has been defined. In other words, you can execute your SQL Server Integration Services and debug it, even though the data is not going anywhere (it is just being counted)! This is a very convenient feature indeed.

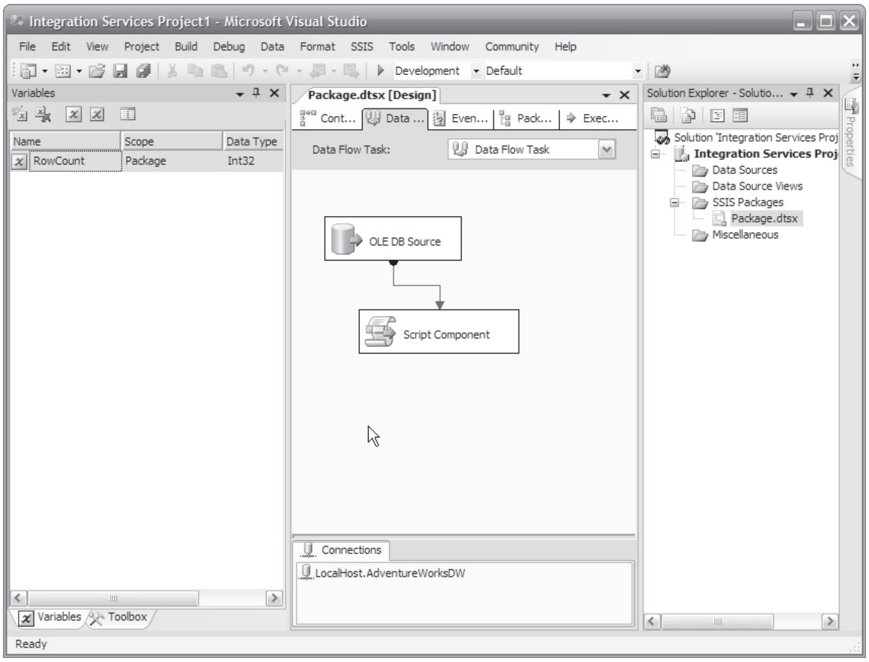
Chapter 1 showed how to create SQL Server Integration Services variables. Create a variable now, named **RowCount** (of **Int32** type), as follows:

1. Select **SSIS** from the top level menu.

2. Select **Variables** to show the **Variables** window.

3. In the **Variables** window, click the **New Variable** button to create a new variable. This variable will be of type **Int32** by default.

4. Rename the variable to `RowCount`. Your package should resemble the one shown in Figure 7.12:


Integration Services Project1 - Microsoft Visual Studio
File Edit View Project Build Debug Data Format SSIS Tools Window Community Help
Development
Default
Variables
Package.dtsx [Design]
Cont... Data ... Even... Pack... Exec...
Data Flow Task:
Data Flow Task
OLE DB Source
Script Component
Connections
LocalHost.AdventureWorksDW
Solution Explorer - Solutio...
Solution 'Integration Services Proj
Integration Services Proj
Data Sources
Data Source Views
SSIS Packages
Package.dtsx
Miscellaneous
Variables
Toolbox
Properties
Ready

Name	Scope	Data Type
RowCount	Package	Int32

Figure 7.12: Data Flow Designer and Variables Window, with RowCount Variable Added.

You can now close the **Variables** window if you like.

5. Drag a **Row Count Component** from the **Data Flow Transformations** tab of the Toolbox to the **Data Flow** design surface.

6. Connect the output of the **Script Component** to the **Row Count Component**.

7. Edit the **Row Count** component by double clicking the component shape in the designer.

8. In the editor, set the **VariableName** property to the name of the variable you created: `RowCount`.

If you were to execute the package now, the number of rows passing through the Row Count component would be written to the named variable.

Note:

The variable value does not change until the Data Flow has completed. This is the same for all SSIS package variables referenced in the Data Flow, even when using the Script Component. The values are locked when execution of the Data Flow starts and they are only updated at the end. See Chapter 9 for more information on variables in Script Components.

Now that you can execute the package and the Data Flow within it, you can start to debug the output of the script, and see whether it has calculated the Soundex value correctly.

The best way to see the output from the Script Component is to add a Data Viewer to the Data Flow on the path between the output of the Script Component and the input of the Row Count Component, as follows:

1. Right-click the line between the output of the **Script Component** and the input of the **Row Count Component** and select **Data Viewers …**

2. The **Data Flow Path Editor** will appear (see Figure 7.13), with the **Data Viewers** tab selected by default.

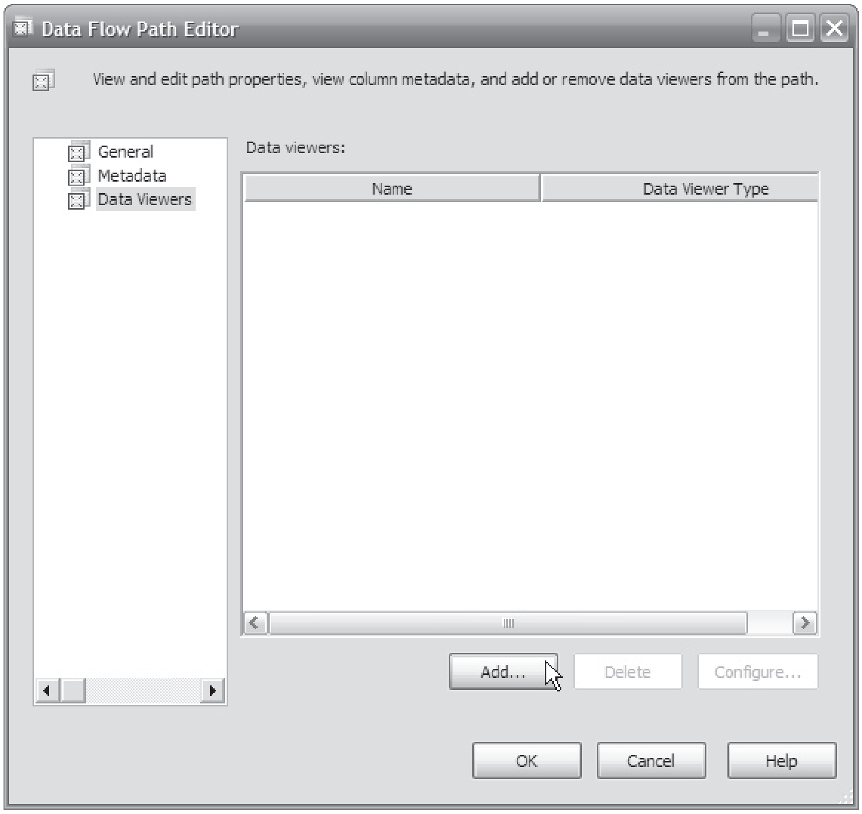
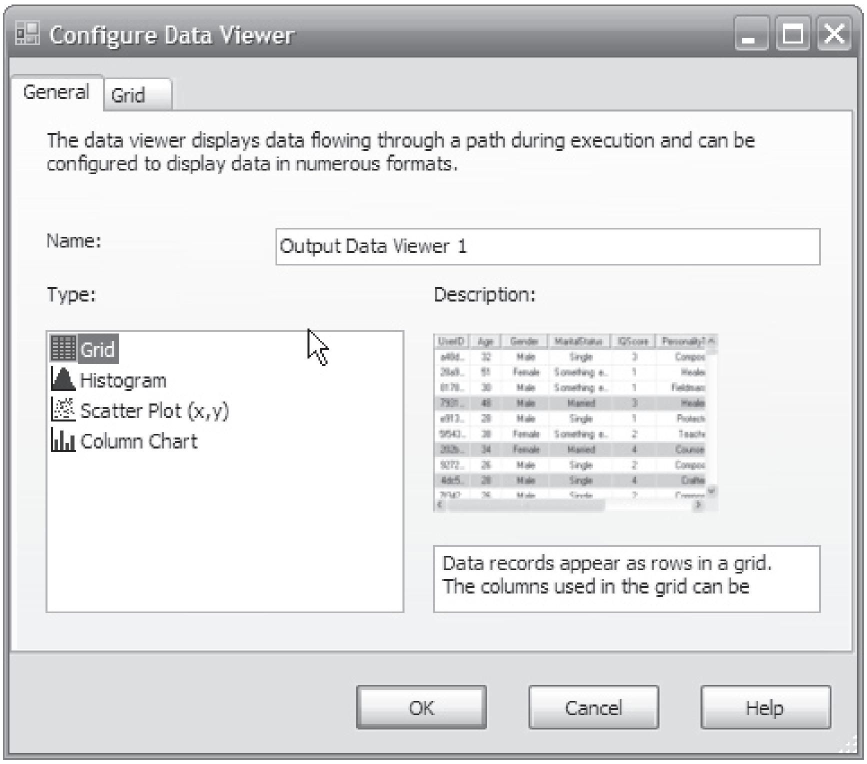


Figure 7.13: Data Flow Path Editor.

3. Click the **Add** … button in the **Data Flow Path Editor** to show the **Configure Data Viewer** dialog box, as shown in Figure 7.14.



The data viewer displays data flowing through a path during execution and can be configured to display data in numerous formats.

Name: Output Data Viewer 1

Type:
- Grid
- Histogram
- Scatter Plot (x,y)
- Column Chart

Description:

Data records appear as rows in a grid. The columns used in the grid can be

Figure 7.14: Configure Data Viewer Dialog Box.

4. Select the **Grid** type of Data Viewer. Then click the **Grid** tab.

5. The **Grid** tab shows all the columns currently available selected on the right hand side. In fact, we only need to see the **LastName** and **Soundex** columns in order to debug our script, so use the arrow buttons to move all the other rows to the **Unused Columns** list on the left-hand side. In fact, it may be easier to move all the columns over to the left-hand side and reselect **LastName** and **Soundex**. Also note that these columns—because they are actively used by the Script Component—will appear at the bottom of the list of all columns (see Figure 7.15).

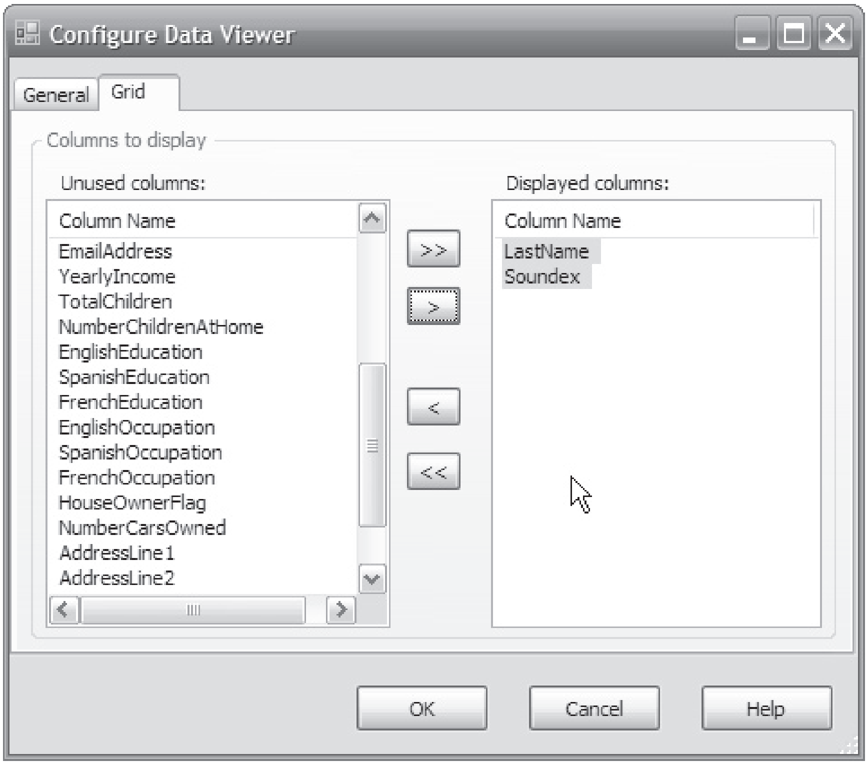


Figure 7.15: Grid Viewer Configured to Show Only the Required Columns.

6. Click the **OK** button to close the **Configure Data Viewer** dialog box and click the **OK** button again to close the **Data Flow Path Editor**.

Now the Data Flow is ready for debugging. The Data Flow designer shows the Data Viewer on the path between the Script Component and the Row Count Component. Note the "spectacles" icon, which shows that the Data Viewer has been added to the path (see Figure 7.16).

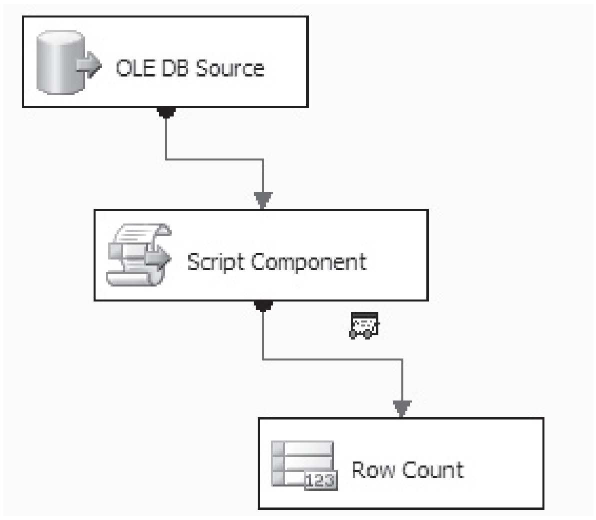


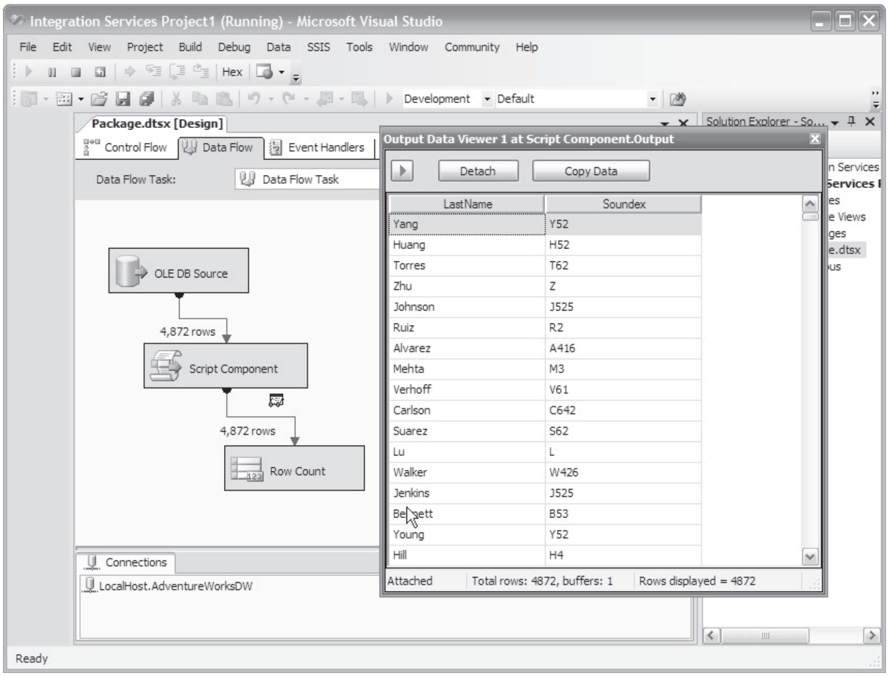
Figure 7.16: Completed Data Flow.

Running a Data Flow with a Script Component

To execute the package, and thus this Data Flow, you can use the **F5** key or any of the standard ways of executing a package described in Chapter 1. When the package executes, you will notice that the **Data Viewer** window appears. You can drag and dock this Data Viewer to any size or convenient location in your designer or on your screen.

On execution, the Data Flow task reads a buffer of data from the **OLE DB Source** component. The Script Component acts on this buffer, transforming each row using our function. The Row Count component should next count these rows, but before it can do so, the Data Viewer pauses execution of the Data Flow and the package.

At this point, all the components in the flow will be colored yellow, to indicate that they are in progress and have not yet completed. When the Data Viewer has received a buffer of data, it displays the columns we selected in the viewer. As illustrated in Figure 7.17, the grid now shows the results of our Soundex calculation for the first buffer of data (4872 rows in this example).


Integration Services Project1 (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Data SSIS Tools Window Community Help
Hex
Development
Default
Package.dtsx [Design]
Control Flow
Data Flow
Event Handlers
Data Flow Task:
Data Flow Task
OLE DB Source
4,872 rows
Script Component
4,872 rows
Row Count
Connections
LocalHost.AdventureWorksDW
Solution Explorer - So...
Ready
Output Data Viewer 1 at Script Component.Output
Detach
Copy Data

LastName	Soundex
Yang	Y52
Huang	H52
Torres	T62
Zhu	Z
Johnson	J525
Ruiz	R2
Alvarez	A416
Mehta	M3
Verhoff	V61
Carlson	C642
Suarez	S62
Lu	L
Walker	W426
Jenkins	J525
Bennett	B53
Young	Y52
Hill	H4

Attached   Total rows: 4872, buffers: 1   Rows displayed = 4872

Figure 7.17: Data Flow Executed.

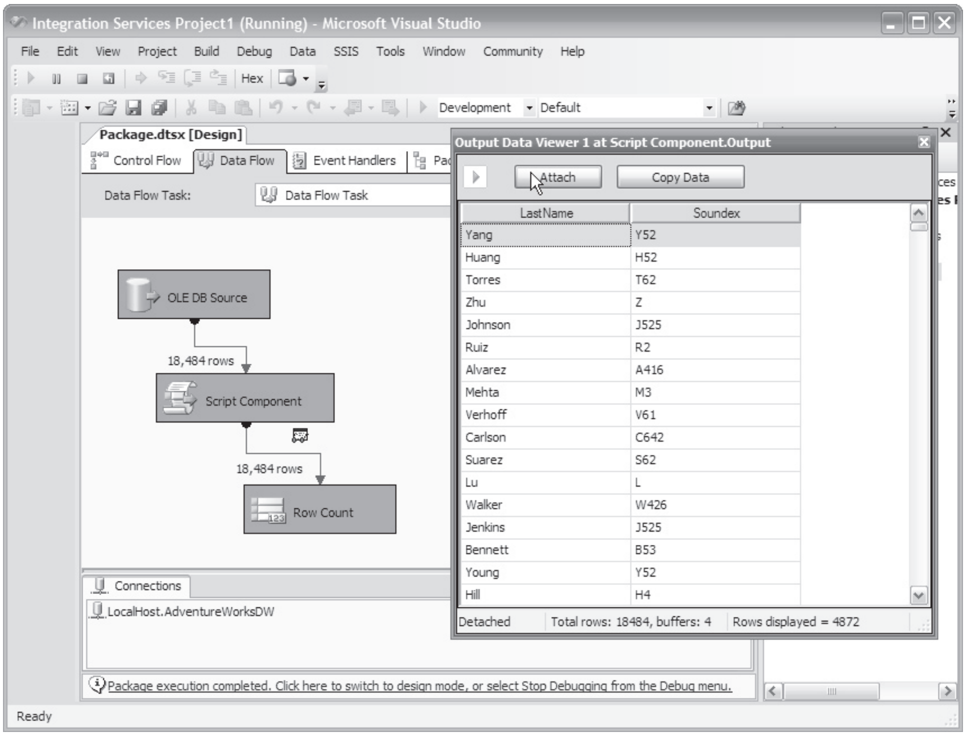
You can examine all the values to ensure that the function is working correctly. In this example they certainly appear to be correct. However, if necessary you can even copy the data to the clipboard and paste it into other applications, such as Excel, for further analysis.

To see the next buffer of data, click the green **Continue** button in the top left hand corner of the Data Viewer. The Data Viewer will show the next buffer of data and pause again. Alternatively, if you have finished examining the data, you could click the **Detach** button and the flow will continue without pausing, or you could just close the Data Viewer.

Note:

If you Detach the Data Viewer or close it, you will see the package execute without pausing. This will give you a good idea of the performance of the script component, which you will find to be excellent. On a Tablet PC, this Script Component will calculate the Soundex function for these 18484 rows in about 1-2 seconds!

When the Data Flow is complete, all the components in the Data Flow should be colored green to show that they have all completed successfully. The final view will look like Figure 7.18:


Integration Services Project1 (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Data SSIS Tools Window Community Help
Hex
Development
Default
Package.dtsx [Design]
Control Flow
Data Flow
Event Handlers
Data Flow Task:
Data Flow Task
OLE DB Source
18,484 rows
Script Component
18,484 rows
Row Count
Connections
LocalHost.AdventureWorksDW
Output Data Viewer 1 at Script Component.Output
Attach
Copy Data
Ready

LastName	Soundex
Yang	Y52
Huang	H52
Torres	T62
Zhu	Z
Johnson	J525
Ruiz	R2
Alvarez	A416
Mehta	M3
Verhoff	V61
Carlson	C642
Suarez	S62
Lu	L
Walker	W426
Jenkins	J525
Bennett	B53
Young	Y52
Hill	H4

Detached | Total rows: 18484, buffers: 4 | Rows displayed = 4872

Package execution completed. Click here to switch to design mode, or select Stop Debugging from the Debug menu.

Figure 7.18: Completed Data Flow.

You can now stop debugging, using **F5** if you like, just like any other SQL Server Integration Services package.

Summary

As you can see, Script Components are extremely versatile. They are easy to add to your Data Flow, quite simple to program, and give excellent performance. Add to that the power of visual debugging and you will realize why they are such an exciting feature of SQL Server Integration Services. Mastering the Script Component can be your key to a vast range of data integration functionality.