DTS xChange Migrating

Last Modified on 30 September 2021

EOL: DTSxChange reached its end of life date on October 1, 2021. See the Solarwinds End of Life Policy for more information.

Task Migration

DTS xChange tries to migrate most of your tasks. If a task can't be migrated, DTS xChange leaves the task embedded in the DTS object in the converted SSIS Package.

The following table displays the tasks that can be fully converted and the tasks that may require manual work:

Task Type	Conversion Status
Active X Script Task	Converts to a native SSIS object, but may not work properly and requires manual work in some cases. **Note:** This task won't work in SSIS if you are using the DTS APIs Object Model (except DTSGlobalVariable). You have to replace ActiveX script with native SSIS objects.
Bulk Insert Task	Fully converts to native SSIS.
Custom Task	Cannot convert to a native SSIS object. Converts to an embedded DTS object that may not work. You might have to rewrite the task to native SSIS. **Note:** If you are using a custom task it may or may not work. Look for similar functionality in SSIS and you might be able to replace it with native SSIS.
Data Driven Query Task	Can not convert to a native SSIS object. Converts to an embedded DTS object that may not work. You might have to rewrite the task to native SSIS. **Note:** This task has been depreciated in SSIS. Rewrite the task, or keep using it in the embedded DTS object.
DataPump With ActiveX Script	Converts to a native SSIS object, but may not work properly and requires manual work in some cases. **Note:** There is no ActiveX script support for DataFlow in SSIS. The **Script Transform Placeholder** is created in the pipeline for you by DTS xChange. Change the commented VB Script to the equivalent VB.net syntax.
DataPump With Copy Column	Fully converts to native SSIS.
Dynamic Properties Task	Fully converts to native SSIS.

Task Type	Conversion Status
Execute Package Task	Fully converts to native SSIS.
ExecuteProcess Task	Fully converts to native SSIS.
ExecuteSQL Task	Fully converts to native SSIS.
FTP Task	Fully converts to native SSIS.
Mail Task	Fully converts to native SSIS.
MSMQ Task	Can not convert to a native SSIS object. Converts to an embedded DTS object that may not work. You might have to rewrite the task to native SSIS. **Note:** This task converts to embedded DTS object.
OLAP Task	Can not convert to a native SSIS object. Converts to an embedded DTS object that may not work. You might have to re-write the task to native SSIS. **Note:** This task converts to an embedded DTS object. There isn't a similar task in SSIS to process OLAP 2000 Cubes. You can only Process OLAP 2005/2008 Cubes using Native SSIS Analysis Services Task.
Transfer Database Task	Fully converts to native SSIS.
Transfer Errors Task	Fully converts to native SSIS.
Transfer Jobs Task	Fully converts to native SSIS.
Transfer Logins Task	Fully converts to native SSIS.
Transfer Master Procs Task	Fully converts to native SSIS.
Transfer SQL Server Objects Task	Fully converts to native SSIS.

ActiveX Script Task

How to convert ActiveX Script to native SSIS

DTS xChange converts the ActiveX Script task to the ActiveX Script Task provided in SSIS for legacy support. In certain cases ActiveX in SSIS does not work and you may have to modify the script or replace the ActiveX Script entirely with native SSIS Tasks.

Note: It's recommended that you replace the ActiveX Script to native Script Task (VB.net Script) or Other Native Tasks (For Each Loop, File System Task).

The following table lists the most common patterns found in the ActiveX script:

Common DTS ActiveX Script Patterns	SSIS Replacement
Task Looping using ActiveX Script	Replace with For Each Loop or For Loop Container.
ADODB Record set Looping and Other ADO Operations	Replace with combination of Execute SQL Task and For Each Loop.
Send SMTP email	Use the Native Send Mail Task (text format) or use the Script task to send Text/HTML formatted emails and get more control.
File System Operation using Scripting.FileSystem COM object	Use the native FileSystem Task to perform common files/folder tasks (copy, delete, create). In some cases you have to write code when the native task doesn't provide functionality. Write managed code in the Script Task using classes found under System.IO namespace.
DTS Object Model Access	Use Expressions or Configurations. Certain things can be also done in Script Task using VB.net/C# code.
Auditing related code (log or email on success/failure)	Use Auditing/Monitoring Framework provided by DTS xChange.

What doesn't work in SSIS ActiveX Script Task?

DTS Object Model Calls Examples:

- Set pkg = DTSGlobalVariables.Parent '// Will not work in SSIS
- Set conn = opkg.Connections("MyConnection") '// Will not work in SSIS
- Set stp = opkg.Steps("DTSExecuteSQL_1") '// Will not work in SSIS

DTS Variable Assignment to Int64 or Single DataType variable

You must perform proper casting or change the variable datatype to fix this error. You may receive the following error(s) in some cases when you try to execute ActiveX Task in SSIS:

- **[ActiveX Script Task] Error: Retrieving the file name for a component failed with error code 0x001868CC.**
- **[ActiveX Script Task] Variable uses an Automation type not supported in VBScript**

What works in SSIS ActiveX Script Task?

Any COM object created using CreateObject function

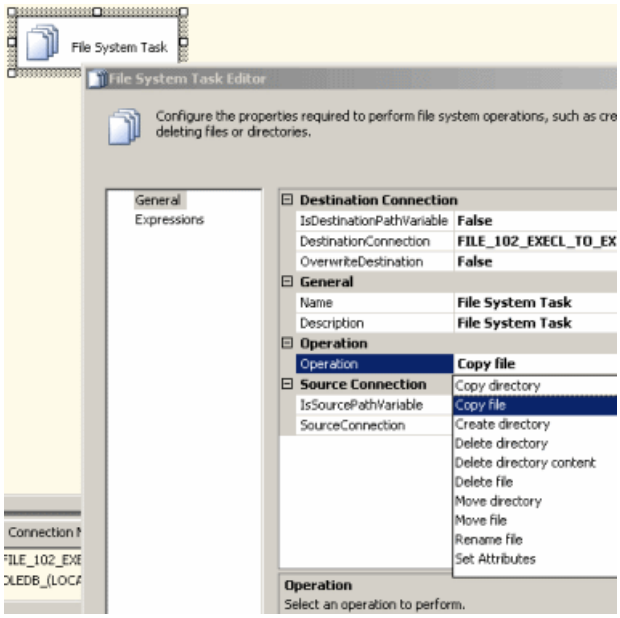
- Example : **CreateObject("Scripting.FileSystem")**

Variable read/write using DTSGlobalVariable (Old way to access global variable)

- Example : Msgbox "Customer ID is " & DTSGlobalVariables("gvCustID").Value

How to convert Scripting.FileSystem object of ActiveX Script to native SSIS Task

Developers use the Scripting.FileSystem object in the ActiveX script task to perform various file system related tasks (Copy file, Delete file etc). In SSIS you can perform common file system related tasks using the File System task. If any task you're performing using Scripting.FileSystem is not possible using the File System Task (Check File Exists), then you can use Script Task in SSIS and use System.IO namespace to perform File/Folder related tasks that aren't possible using the FileSystem Task.



You can perform any of the following operations using the FileSystem Task:

- Copy directory
- Create directory
- Delete directory content
- Move directory
- Rename file
- Copy file
- Delete directory
- Delete file
- Move file
- Set Attributes (Set file to Hidden, ReadOnly, Archive or System)

> **Note:** If your ActiveX script is using anything other than above listed operations, you might need to use SSIS Script Task and write the code using System.IO methods.

The following code snippets display how to perform some of most common file/folder related tasks that aren't possible to implement using File System Task:

Path related functions

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        Dim sMyVar As String '//Get only file name from a specified path => Returns mydatafile_001.txt
        sMyVar = System.IO.Path.GetFileName("c:\temp\mydatafile_001.txt")  '//Get only directory path from a specified pat
h => Returns c:\temp
        sMyVar = System.IO.Path.GetDirectoryName("c:\temp\mydatafile_001.txt")  '//Combine two paths into one path =>
Returns c:\temp\mydatafile_001.txt
        sMyVar = System.IO.Path.Combine("c:\temp", "mydatafile_001.txt")  '//Get filename without extension => Returns
mydatafile_001
        sMyVar = System.IO.Path.GetFileNameWithoutExtension("c:\temp\mydatafile_001.txt")  '//Get extension of the file =
> Returns txt
        sMyVar = System.IO.Path.GetExtension("c:\temp\mydatafile_001.txt")        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

Check if file/folder exists

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        Dim sMyVar As String        If File.Exists("c:\temp\file_001.txt") = True Then
            '//Debug.Print "File Exists"
        End If        If Directory.Exists("c:\temp") = True Then
            '//Debug.Print "Folder Exists"
        End If        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

Read from text file

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        '//Read file content to string variable
        Dim sMyfileData As String
        Dim sReader As StreamReader = New StreamReader("c:\temp\file_001.txt")
        sMyfileData = sReader.ReadToEnd
        sReader.Close()        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

Read from text file (line by line)

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        '//Read file line by line
        Dim sReader As New StreamReader("c:\autoexec.bat")
        ' Display all the text lines in the file.
        Do Until sReader.Peek = -1
            ' The ReadLine methods reads whole lines.
            Console.WriteLine(sReader.ReadLine)
        Loop
        ' Always close a StreamReader when you've done with it.
        sReader.Close()        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

Write to text file

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()        '//Open existing file for append. If file doesn't exist then new file will be created
        Dim writer As StreamWriter = New StreamWriter("c:\write_test.txt", True)
        writer.WriteLine("Hello world - line1")
        writer.WriteLine("Hello world - line2")
        writer.Close()        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

Get file information

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()

        '//Get file properties
        Dim sInfo As String
        Dim FileProps As FileInfo = New FileInfo("c:\windows\notepad.exe")
        sInfo = sInfo & " File Name = " & FileProps.FullName
        sInfo = sInfo & " Creation Time = " & FileProps.CreationTime
        sInfo = sInfo & " Last Access Time = " & FileProps.LastAccessTime
        sInfo = sInfo & " Last Write Time = " & FileProps.LastWriteTime
        sInfo = sInfo & " Size = " & FileProps.Length        System.Diagnostics.Debug.Write(sInfo)        FileProps = Nothing
Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

List files (with wild card search pattern and recursive option)

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        Dim file As String
        '//Recursive listing (search for *.txt)
        '//Dim files() As String = Directory.GetFiles("c:\windows", "*.txt", SearchOption.AllDirectories)        '//Top level listing (
search for *.txt)
        Dim files() As String = Directory.GetFiles("c:\windows", "*.txt", SearchOption.TopDirectoryOnly)
        For Each file In files
            System.Diagnostics.Debug.WriteLine(file & "...found")
        Next
        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

List sub directories

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        Dim dir As String
        'Dim dirlist() As String = Directory.GetDirectories("c:\windows","*",SearchOption.AllDirectories)
        Dim dirlist() As String = Directory.GetDirectories("c:\windows")
        For Each dir In dirlist
            System.Diagnostics.Debug.WriteLine(dir)
        Next

        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

List disk drives

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        Dim dirInfo As Directory
        Dim drive As String
        Dim drives() As String = dirInfo.GetLogicalDrives()
        For Each drive In drives
            System.Diagnostics.Debug.WriteLine(drive)
        Next        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

File/Folder delete, copy move

```
Imports System.IOPublic Class ScriptMain
    Public Sub Main()
        If File.Exists("c:\dest\datafile.txt") Then
            File.Delete("c:\dest\datafile.txt")
        End If
        File.Copy("c:\src\datafile.txt", "c:\dest\datafile.txt")
        File.Move("c:\src\datafile.txt", "c:\dest\datafile.txt")        If Directory.Exists("c:\dest") Then
            Directory.Delete("c:\dest")
        End If
        Directory.CreateDirectory("c:\dest")
        Directory.Delete("c:\dest")
        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

How to convert ADODB object of ActiveX Script to native SSIS Task

Using ADO objects inside an ActiveX Script task is very common in DTS. The most common uses for ADO objects (ADODB.Connection, ADODB.RecordSet etc) are:

- Creating connections at runtime
- Executing Adhoc Queries at runtime
- Processing certain data row by row

Sample ActiveX Script using ADODB.Connection and ADODB.Recordset

The following example performs three actions:

- Establishes Connection to SQL Server using ADODB.Connection object
- Executes SQL Query and Load the recordset using ADODB.Recordset object
- Loops through each record to build HTML formatted string and emails it

```
'**********************************************************************
' Visual Basic ActiveX Script
'************************************************************************Function Main()
    Dim objRs
    Dim objConn
    Dim strSql
    Dim strHtml

    Set objConn = CreateObject("ADODB.Connection")
    objConn.Open "Provider=SQLOLEDB;Data Source=(local);Trusted_Connection=yes;Initial Catalog=Northwind"

    strSql = "Select top 10 b.ProductName,Sum(a.UnitPrice*a.Quantity) as Total from [Order Details] a "
    strSql = strSql & "join Products b on a.ProductID =b.ProductID "
    strSql = strSql & "group by b.ProductName order by Sum(a.UnitPrice*a.Quantity) desc"

    '--- //Method 1 : Using Execute Method of Connection Object
    '--- Set objRs=objConn.Execute(strSql)
    '--- //

    '*** /// OR ///****

    '//Method 2 : Using Execute Method of Connection Object

    Set objRs = CreateObject("ADODB.Recordset")
    objRs.Open strSql, objConn
    '//

    strHtml = "

# Top 10 Products

"
    strHtml = strHtml & ""
    Do While Not objRs.EOF
        strHtml = strHtml & ""
        objRs.MoveNext
    Loop
    strHtml = strHtml & "

Product Name                                                    Total
" & objRs(0) & "                                                " & objRs(1) & "

"

    'MsgBox strHtml    SendMail "dataservices@mycompany.com", "ceo@mycompany.com", "Top 10 Products", strHtml, T
rue
    Main = DTSTaskExecResult_Success
End FunctionSub SendMail(FromAddress, ToAddress, Subject, Body, IsHtml)
    '//Write Code to send email
End Sub
```

Note: The script listed above is the DTS way to perform some common operations. When you migrate to SSIS you can take a different approach without writing too much code. The best practice is to avoid hard-coded connection strings inside your script whenever possible.

The following table lists how to migrate various patterns you encounter in DTS:

DTS	SSIS steps
ADO Connection	1. Replace with Connection Manager
ADO Command (no result set returned INSERT/UPDATE/DELETE)	1. Add Execute SQL Task. Set Connection, Set SQL Statement 2. Set resultset property to "None" 3. Set Parameter mapping if required
ADO Recordset (which returns resultset)	1. Add SSIS variable of Object data type to hold Resultset 2. Add Execute SQL Task. Set Connection, Set SQL Statement 3. Set resultset property to "Full Resultset" 4. Set resultset mapping to variable of Object data type to hold Recordset. Make sure to rename Resultset Name="0"
ADO Recordset Looping	1. Add Foreach Loop Container 2. Use Foreach ADO Enumerator and specify variable which holds resultset (SSIS variable of Object Datatype) 3. In the Variable Mappings add recordset column index to SSIS variable map. Index starts from 0.

Convert Sample DTS ActiveX script to SSIS equivalent control flow

Install the Sample Package

Download the ADO_Object_Replace.zip sample package. After downloading the package, complete the following steps:

1. Extract the Sample zip file.
2. Open FAQTest.sln and analyze or run the Sample Package to test.

Step by Step

Step 1: Define Variables

1. The sample package uses four variables with the following settings:

Variable Name	Data Type	Value	Is Expresssion	Expression
strHTML	string		False	
objRs	object		False	
varProdName	object		False	
varTotal	object		False	

Note: The Is Expression column refers to the EvaluateAsExpression property of SSIS variable.

Step 2: Define Connections: The sample package requires one connection.

1. Create an oledb connection ((local).Northwind) to use Northwind database.

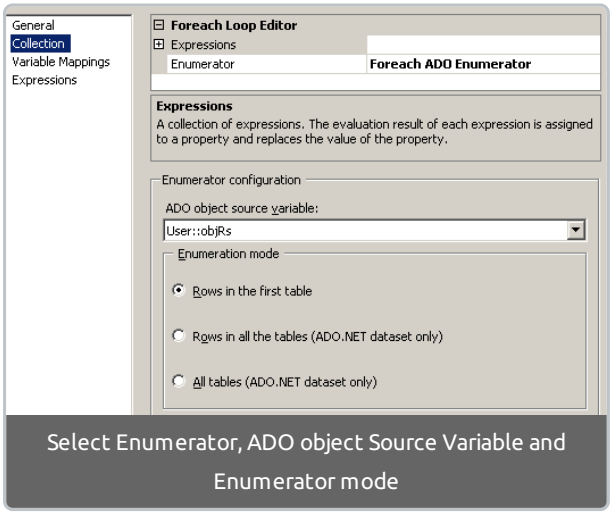
Step 3: Execute SQL Task (Get Recordset)

1. Create an execute SQL task to execute the following SQL Statement. Make sure you select the Northwind connection for this task.

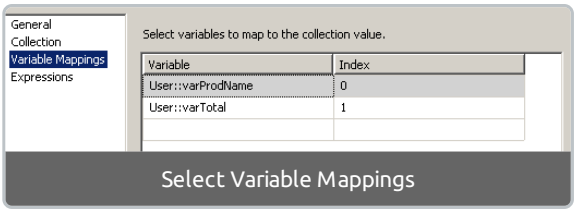
```
Select top 10 b.ProductName,Sum(a.UnitPrice*a.Quantity) as TotalSales
from [Order Details] a
join Products b on a.ProductID =b.ProductID
group by b.ProductName order by Sum(a.UnitPrice*a.Quantity) desc
```

Step 4: Foreach Loop Container (Loop Recordset)

1. Place the Foreach Loop Container and double click it to open the properties dialog box.
2. Apply the following settings displayed in the screenshots.



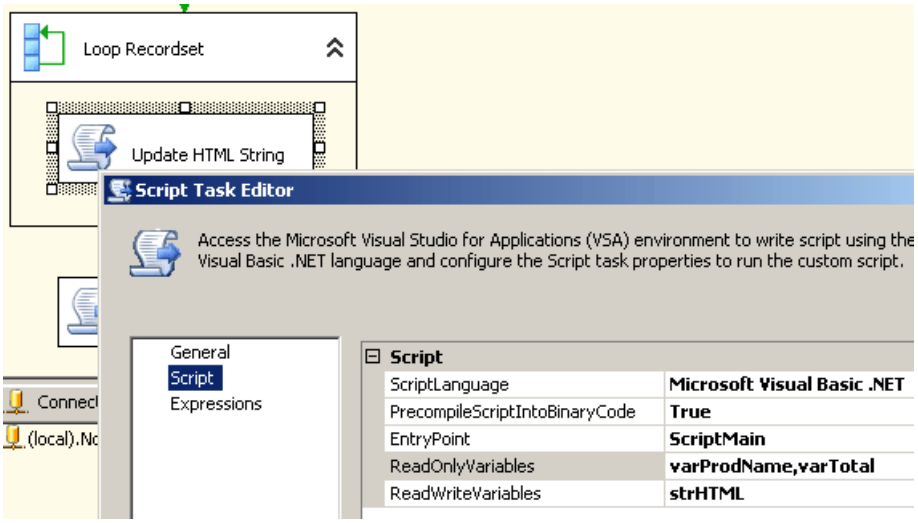
Select Enumerator, ADO object Source Variable and Enumerator mode



Select Variable Mappings

Step 5: Script Task (Update HTML String): Place a Script Task inside the Foreach Loop Container.

1. Double click the Script Task
2. Specify ReadOnlyVariables as displayed in the following screenshot (*varProdName,varTotal*)
3. Specify ReadWriteVariables as displayed in the following screenshot (*strHTML*)
4. Select the Design Script Button
5. Enter the script displayed below




```
Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.RuntimePublic Class ScriptMain
    Public Sub Main()
        Dim ProdName As Object
        Dim Total As Object
        Dim HTML As Object
        ProdName = Dts.Variables("varProdName").Value
        Total = Dts.Variables("varTotal").Value
        HTML = Dts.Variables("strHTML").Value        HTML = HTML.ToString & "" & ProdName.ToString & "" & Total.ToString
& ""        Dts.Variables("strHTML").Value = HTML        Dts.TaskResult = Dts.Results.Success
    End Sub
End Class
```

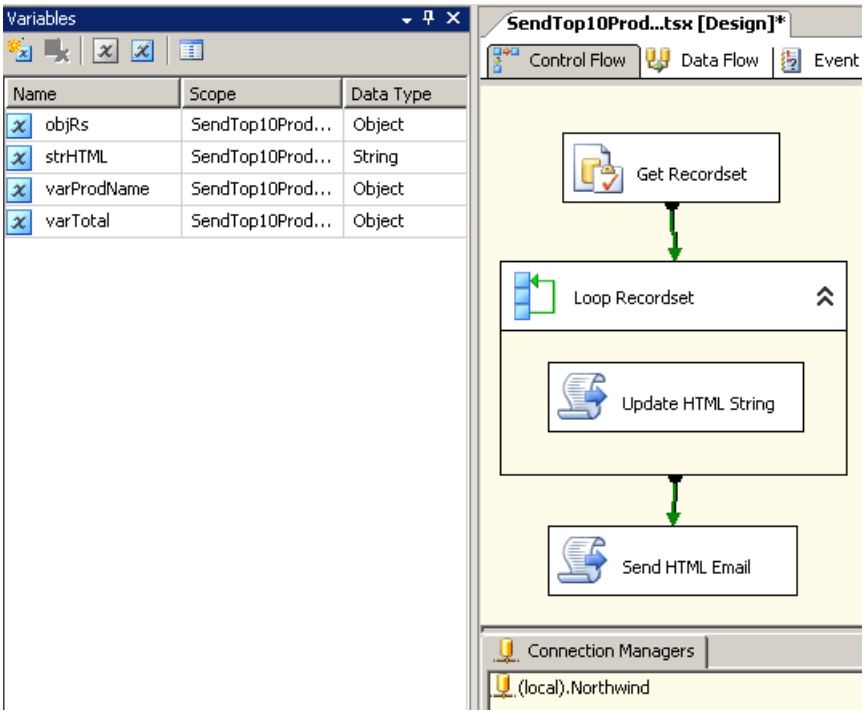
6. Script Task (Send HTML Email): Place the Script Task to execute SendHTMLEmail routine

1. Place a Script Task
2. Double click the Script Task
3. Specify strHTML in the ReadOnlyVariables
4. Select the Design Script Button
5. Enter the script displayed below

```
Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.RuntimePublic Class ScriptMain
    Public Sub Main()
        Dim HTML As String
        Dim Header, Footer As String
        Header = "

# Top 10 Products

"
        Footer = "

Product Name                                                Total

"
        HTML = Dts.Variables("strHTML").Value.ToString        HTML = Header & HTML & Footer        SendMail("dataservices
@mycompany.com", "ceo@mycompany.com", "Top 10 Products", HTML, True)        Dts.TaskResult = Dts.Results.Success
    End Sub    Sub SendMail(ByVal FromAddress As String, _
        ByVal ToAddress As String, _
        ByVal Subject As String, _
        ByVal Body As String, _
        ByVal IsHtml As Boolean)        MsgBox(Body)
        '//Write Code to send email
    End SubEnd Class
```

7. Connect all tasks and testing

1. Connect all tasks as shown and execute the package to test.



Data Pump Task

How to mDTS xChange migrates your DTS DataPump to a DataFlow task in SSIS. The following list displays the supported mapping types in DataPump that are migrated without any problems:igrate complex DataPump with ActiveX Script Transformations

- Copy Column Transformation
- Upper Case Transformation
- Lower Case Transformation
- DateTime Transformation
- Mid Transformation

The following mapping types can't be migrated properly with DTS xChange:

- ActiveX Script Transformation
- Import File Transformation
- Export File Transformation

Converting ActiveX Script Transformation

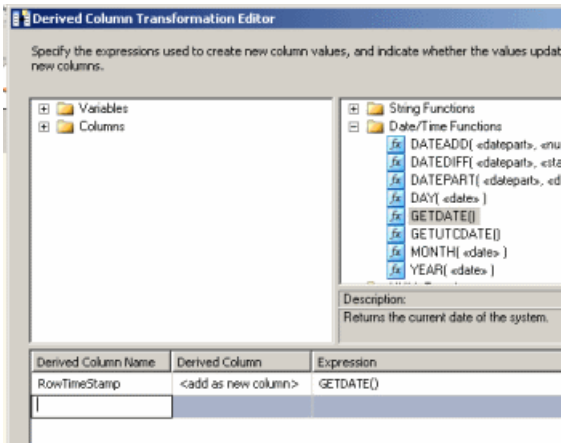
There are many approaches to convert ActiveX scripts to SSIS equivalent code. The most common approach is a combination of Derived Column and Script Component. Most of time you can use Derived Column if you have simple transformation (Upper, Trim, Substring) without any complex logic. If you have a complex transformation that's not possible with Derived Column then you can use script component and write VB.net/C# code.

Using Derived Column

Use the Derived column component to add new columns or modify existing columns in the pipeline. Derived column is faster than the Script Component. Avoid scripting unless it's required. The Derived Column component has many predefined functions that you can use for the transformation:

Function Type	Syntax	Example
String functions (Upper, Lower, Substring)		1. LOWER([ProductName]) 2. LTRIM([ProductName]) 3. SUBSTRING([ProductName],1,3)
DataType conversion functions		1. Convert String to DateTime (DT_DBTIMESTAMP) ("12" + "/31/" + "1900") 2. Convert non unicode string to unicode string (DT_WSTR, 255) [ProductName] Note: 255 is length of expression output. 3. Convert unicode string to nonunicode string (DT_STR, 255,1251)[ProductName]
IF... Else (Use Ternary Operator for If Else type expression)	? :	1. If ProductName is Blank then replace with "" TRIM([ProductName]) == "" ? "" : [ProductName

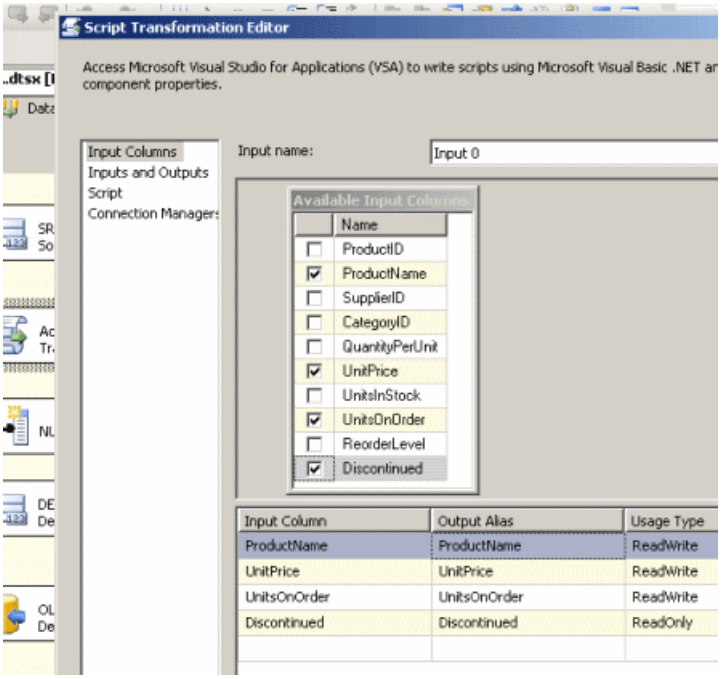
Add a new column that represents the current time. Under Derived Column you can select **Add New Column** or **Replace existing**.



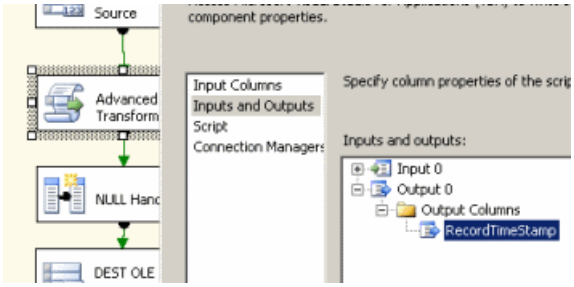
Using Script Component

To use the script component you must first select input columns and define output columns. In most of cases you don't have to define output columns unless you are adding new columns in the pipeline. In the following example, the new column **RecordTimeStamp** is added. Anything defined in the Input or Output column list can be accessed inside the Script component for READ or WRITE (this can be change by changing column usage property).

Select the input columns that will be used inside script component. Define Usage (READ/WRITE/READWRITE).



Add new column in the Output Columns.



After adding the column in the output column collection, edit the script.

The following table explains how to convert various patterns found in ActiveX Script to SSIS:

DataPump ActiveX Script Conversion Examples

DTS	SSIS
DTSDestination("ProductID") = DTSSource("ProductID")	There is no need to convert because there is no transformation involved in this example. Map ProductID <-> ProductID in the Destination Adapter. If you have a simple pattern like this, remove it from the script and remove it from the Script Component Input Columns to improve performance.
If DTSSource("TransCode") = "ACT" Then DTSDestination("TransDescription") = "Acct Trans" Else DTSDestination("TransDescription") = "UNKNOWN" End If	In this example TransDescription column is derived from the value of TransCode. When you don't have the exact mapping of the source column to the target column, or the Target column value is derived based on one or more source column values then use the derived column. You can also use the script component but first consider the Derived Column if possible because its faster compared to the script component. Some times you have to use the Script component because of the complexity of code. The Derived column is good for simple expressions. **Using Derived Column** Complete the following steps: 1. Add Derived Column Component in the pipeline. 2. Add new column (MyTransDescription). 3. Enter the following expression: **[TransCode]="ACT" ? "Acct Transaction" : "UNKNOWN"** 4. On the Destination Adapter Mappings screen map MyTransDescription to Target column TransDescription. **Using Script Component** You can use the Script component but it's slower than Derived column. Give the first preference to Derived Column if it's possible. Complete the following steps: 1. Select Input column TransCode (ReadOnly). 2. Add New Output Column "MyTransDescription" under Input Output Columns tab and specify data type and length of the column. 3. Enter the following script: **If Row."TransCode" = "ACT" Then Row.MyTransDescription = "Acct Trans" Else Row.MyTransDescription = "UNKNOWN" End If** 4. On the Destination Adapter Mappings screen map MyTransDescription to Target column TransDescription.
If DTSSource("Discontinued") = 0 Or _ DTSSource("Discontinued") Is Null Then DTSDestination("UnitPrice") = DTSSource("UnitPrice")+50 Else	If Row.Discontinued = 0 Or _ Row.Discontinued_IsNull Then Row.UnitPrice = Row.UnitPrice + 50 Else

DTS	SSIS
DTSDestination("UnitPrice") = Null End If	Row.UnitPrice_IsNull = True End If
DTSDestination("ProductName") = UCase(DTSSource("ProductName"))	Row.ProductName = UCase(Row.ProductName)
DTSDestination("RecordTimeStamp") = Now()	In this example source table doesn't have RecordLoadTimeStamp column and value for target field is generated at run time. For patterns like this you have to either use Derived Column component or use Script Component.

Data Driven Query Task

How to migrate Data Driven Query Task to native SSIS

Purpose

The Data Driven Query task allows you to perform flexible, Transact-SQL based operations on data, including stored procedures and INSERT, UPDATE or DELETE statements. For each row in a source row set, the Data Driven Query task selects, customizes, and executes one of several SQL statements. Select statements that you want to execute through a constant return value set in a Microsoft ActiveX script transformation. Based on the return constant used in the script, one of four different parameterized SQL statements that you create can be executed for each source row.

Summary

> **Note:** DTS xChange does not convert a Data Driven Query to native SSIS.

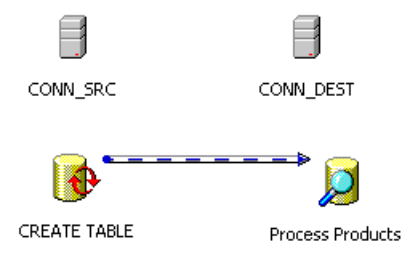
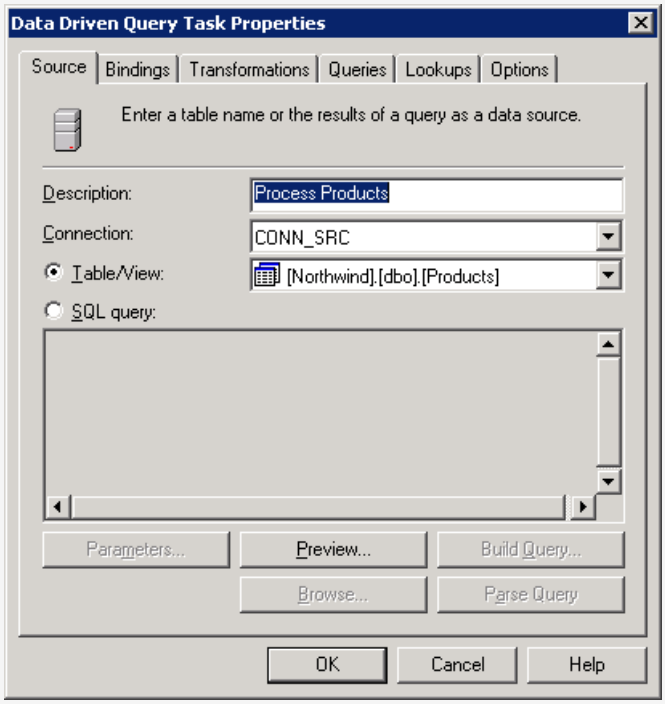
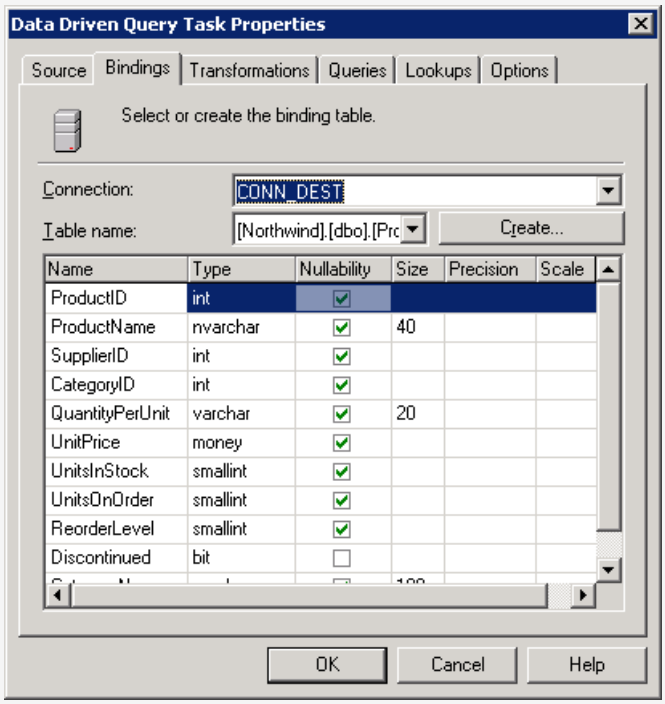
DTS xChange converts a Data Driven Query to an embedded DTS object. The Data Driven Query Task has been discontinued in SSIS. You may want to rewrite the task using native SSIS tasks and components. The Step-by-Step example section covers how to use native SSIS components to convert a Data Driven Query Task manually.

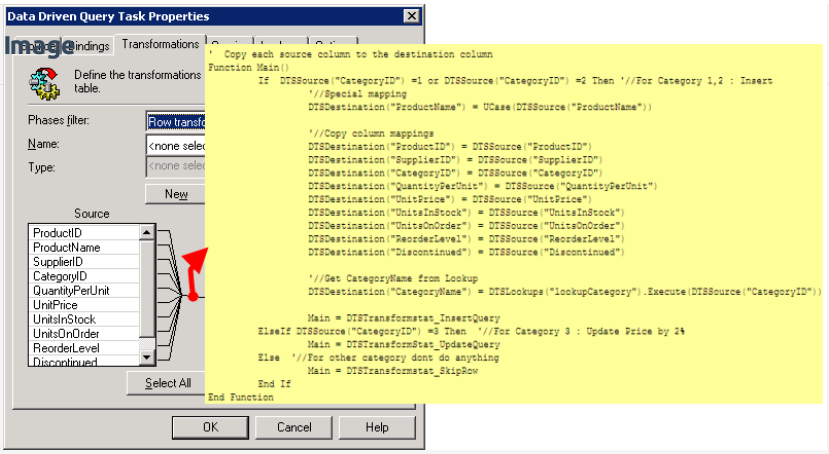
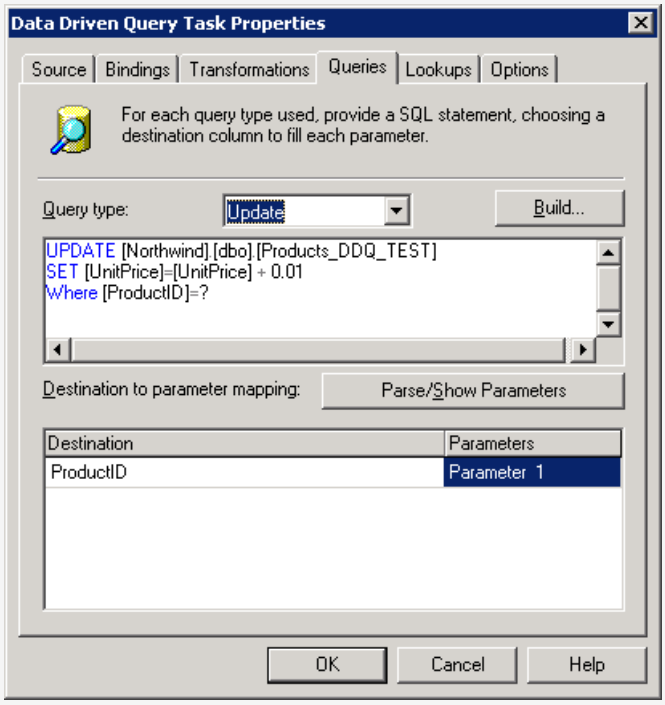
Step-by-Step Example

This section displays how to convert a Data Driven Query task to a Data Flow, and apply various transformations. The following items are covered in this example:

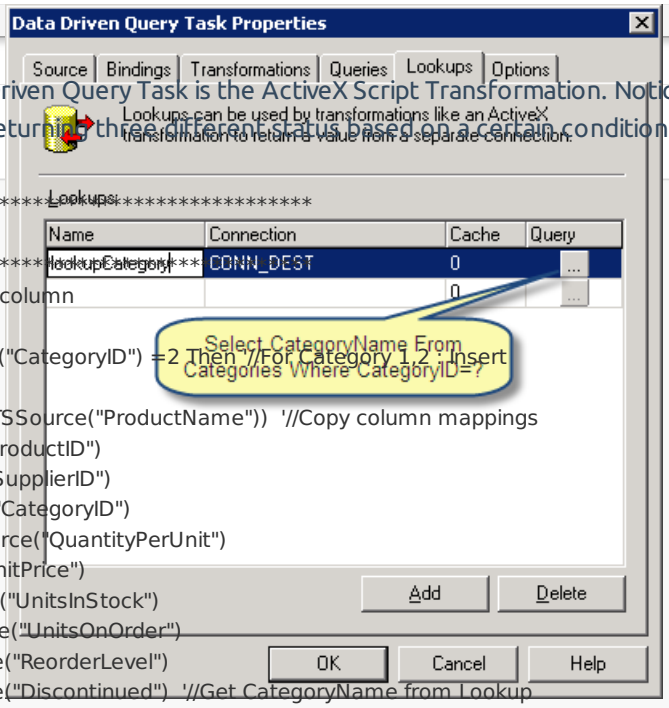
- Converting Lookups
- Perform Insert/Update Operations based on certain condition for each row
- Skip rows if no condition match
- How to do copy column mappings
- How to do special mappings (Upper Case)

Description	Image

Description	Image
Original sample DTS Package	
Data Driven Query Task Source tab	
Data Driven Query Task Binding tab	

Description	Image
Data Driven Query Task ActiveX script transformation tab	
Data Driven Query Task Queries tab	
Data Driven Query Task Lookup Queries tab	

Description	Image



The challenging part of converting a Data Driven Query Task is the ActiveX Script Transformation. Notice that in this source code that the Main function is returning three different status based on a certain condition:

```
'**********************************************************************
'  Visual Basic Transformation Script
'**********************************************************************
'  Copy each source column to the destination column
Function Main()
 If  DTSSource("CategoryID") =1 or DTSSource("CategoryID") =2 Then '//For Category 1,2 : Insert
  '//Special mapping
  DTSDestination("ProductName") = UCase(DTSSource("ProductName"))  '//Copy column mappings
  DTSDestination("ProductID") = DTSSource("ProductID")
  DTSDestination("SupplierID") = DTSSource("SupplierID")
  DTSDestination("CategoryID") = DTSSource("CategoryID")
  DTSDestination("QuantityPerUnit") = DTSSource("QuantityPerUnit")
  DTSDestination("UnitPrice") = DTSSource("UnitPrice")
  DTSDestination("UnitsInStock") = DTSSource("UnitsInStock")
  DTSDestination("UnitsOnOrder") = DTSSource("UnitsOnOrder")
  DTSDestination("ReorderLevel") = DTSSource("ReorderLevel")
  DTSDestination("Discontinued") = DTSSource("Discontinued")  '//Get CategoryName from Lookup
  DTSDestination("CategoryName") = DTSLookups("lookupCategory").Execute(DTSSource("CategoryID"))  Main = DTSTra
nsformstat_InsertQuery
 ElseIf DTSSource("CategoryID") =3 Then  '//For Category 3 : Update Price by 2%
  Main = DTSTransformStat_UpdateQuery
 Else  '//For other category dont do anything
  Main = DTSTransformstat_SkipRow
 End If
End Function
```

In our example:

- OLEDB Destination is used to handle DTSTransformstat_InsertQuery
- OLEDB Command is used to handle DTSTransformstat_UpdateQuery , you can also use the same component to handle DTSTransformstat_DeleteQuery
- Conditional Split is used to divert records to certain flow based on condition. Default output of conditional flow is DTSTransformstat_SkipRow in our example
- Derived Column is used to handle special transformation (Upper, Mid, Lower). If you can't handle transformations using inbuilt functions of derived column then you may want to consider Script Transformation

DTS xChange can convert everything except the Data Driven Query Task. Complete the following steps to replace the Data Driven Query task with a DataFlow:

Step 1: Set up the Test Database

> **Note:** This sample uses northwind database.

1. Download Create_Northwind_Schema_And_Data.zip and run it in a new SQL Server Management Studio query window to your local instance (or select different server). This script creates the northwind database and

populates sample data.

Step 2: Migrate the DTS Package using DTS xChange

1. Migrate the DDQ_Sample_Packages.zip using DTS xChange.

> **Note:** DTS xChange will migrate all tasks except the Data Driven Query.

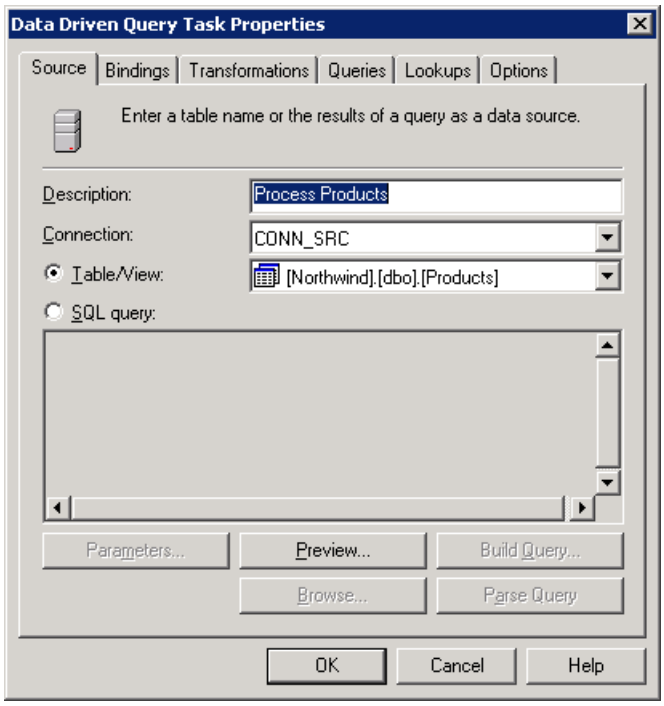
Step 3: Delete the Data Driven Query Task and add the DataFlow task

1. Delete the Data Driven Query Task container, and add a new DataFlow task in the package surface. Rename the DataFlow to **Process Products**.

2. Attach a Create Table task to the Data Flow task.

Step 4: Add OLEDB source

1. Select the DataFlow and add the **OLEDB Source Adapter** from the toolbar.

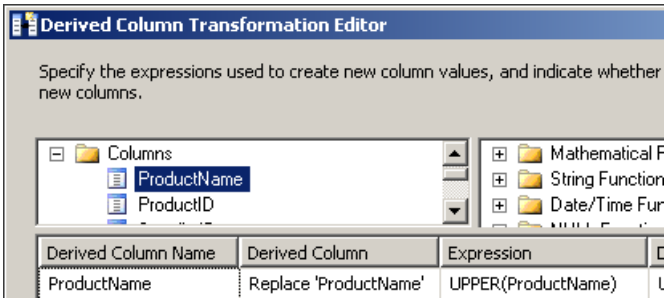


2. Double click on the source and select **CONN_SRC** in the connection dropdown. Select the Products table in the Table or View Dropdown.

Step 5: Add Derived column

1. Add the derived column component and name it Special Transformation. Add a new Replace ProductName expression, this changes the ProductName to upper case.

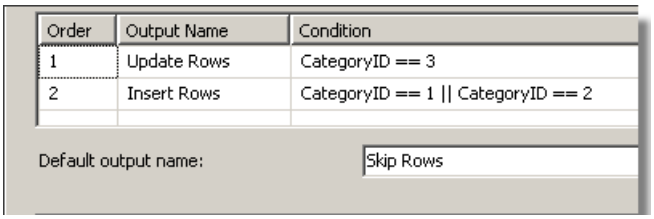
2. Connect OLEDB Source with derived column.



Step 6: Add Conditional Split

1. Add a conditional split component. Attach the Derived column with the conditional split0. Double click on the conditional split to configure various outputs based on conditions. **Note:** The ActiveX script above has two IF conditions and one default (ELSE block). We will do the same thing here in the conditional split and give it meaningful name for each output.

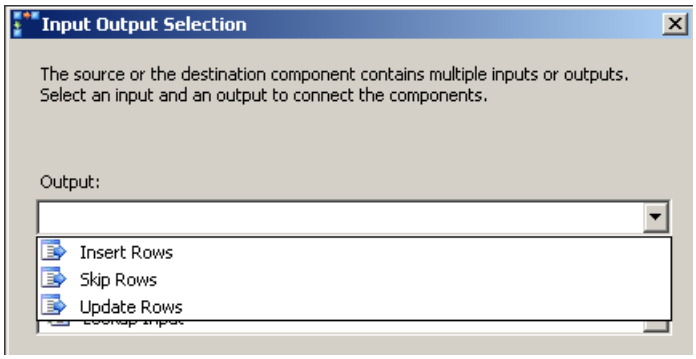
2. Create two outputs as diplayedbelow. The SSIS uses expression language very similar to C++ or C# language syntax. Enter Skip Rows in the default output and select OK to the save changes.



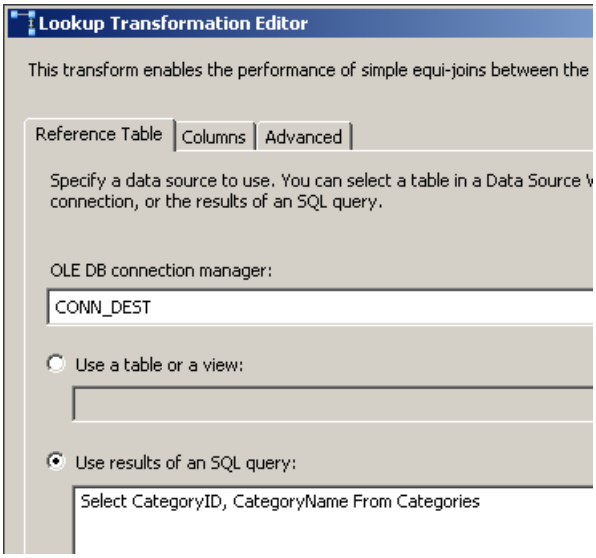
Step 7: Add Lookup

The Data Driven Query Task ActiveX Script Transformation uses Lookup to find CategoryName by sourcing CategoryID. Use the Lookup Component found in SSIS but without passing a parameter.

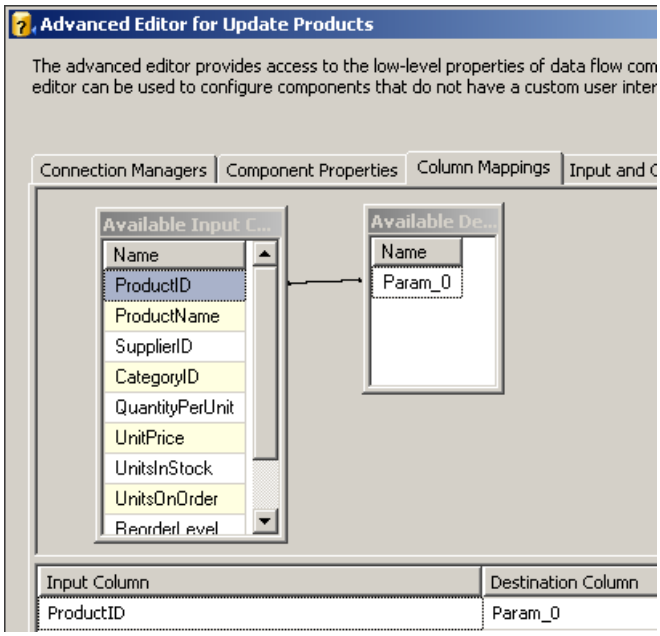
1. Drop the Lookup component. Attach the Conditional Split component to the Lookup component, and select the Input. Select the **Insert Rows** output and select **OK**.



2. Double click Lookup component to open the configuration options. Enter the query displayed below:



3. Select the columns tab and drag the **CategoryID** column from Input columns to Lookup columns. Select the **CategoryName** column, and then select **OK** to save your changes.



Step 8: Add OLEDB Destination

1. Add the OLEDB Destination adapter and rename it to Insert Products.

2. Double click the component to open the property configuration options.

3. Select CONN_DEST in the connection manager dropdown.

4. Select PRODUCTS_DDQ_TEST from the Table/View drop-down menu.

> **Note:** If the table isn't found, you have to run the Execute SQL task to create the missing table

5. Attach the Lookup component to the OLEDB Destination.

Step 9: Add OLEDB Command

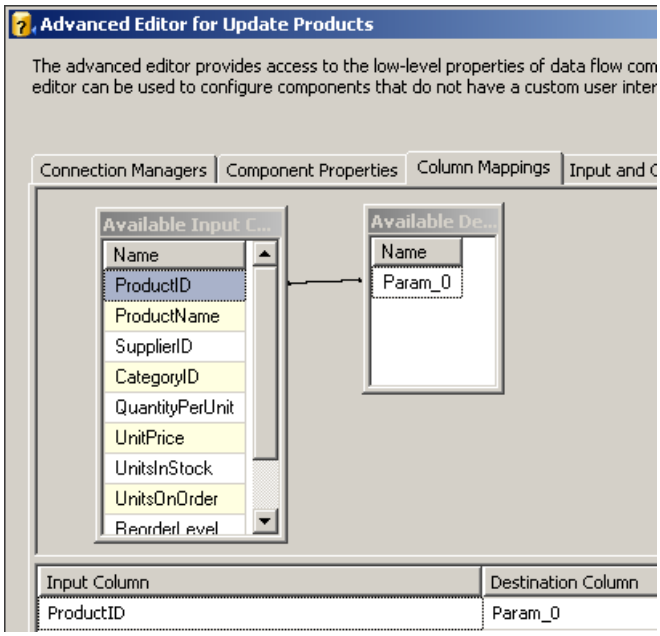
The OLEDB command can be used to execute SQL Statements for each row in the input pipeline.

1. Add the OLEDB Command component and rename it to Update Products. Double click the component to open the property configuration options.

2. Select CONN_DEST in the connection manager dropdown. Select the Component properties tab and enter the following SQL Statement in the SQLCommand Property:

```
UPDATE [Northwind].[dbo].[Products_DDQ_TEST]
SET [UnitPrice]=[UnitPrice] + 0.01
Where [ProductID]=?
```

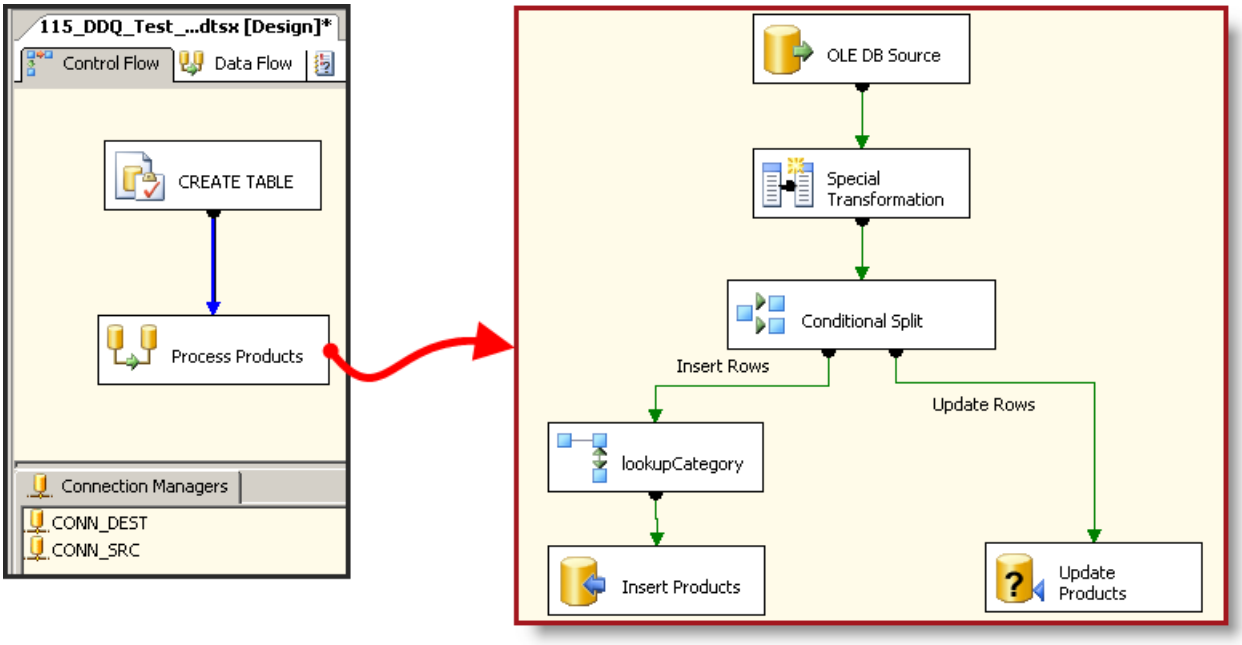
3. Attach ProductID column with Param_0 on the column mappings tab, and then select **Ok** to save your changes.



4. Attach the Conditional Split to the OLEDB command. Select Update Rows from the Input list once prompted.

Step 10: Final Control Flow

1. Select **Debug** > **Start Debugging** to run the package.



Connection Migration

DTS xChange migrates all of your DTS connections to the equivalent SSIS connection.

Flat File Connection

The Flat File connection has been dramatically changed in SSIS. You have to create columns for the Flat File connection before you can use the component. Creating columns was not required in DTS but SSIS made some architectural changes to get better performance. DTS xChange automatically generates all of the columns for the flat file based on several factors. DTS xChange determines the columns for the flat file by doing the following:

- Checks DTS Metadata to get column names, length, and number of columns
- Checks mappings and validates metadata
- Reads Sample rows from the file if the metadata that's retrieved from the DTS package is invalid

Note: You may be prompted to browse the file during migration. Browsing is optional, but it is highly recommended that you supply sample files so DTS xChange can make better decisions about flat file columns and data types.

UDL Connection

The UDL Connection is similar to the SSIS Config file where you can store connection strings and reuse them across multiple packages. The UDL Connection has been depreciated in SSIS. DTS xChange converts UDL Connections to the OLEDB Connection Manager.

Note: You may be prompted to browse the package if the UDL file is missing during migration.

ODBC Connection

The ODBC Connection has limited support in SSIS. It's highly recommended to use OLEDB Providers whenever possible in SSIS. Using an ODBC connection as a destination to load data will not work in SSIS. ODBC is only supported as a Source in SSIS by using ADO.NET Connection for ODBC Connection Wrapper.

DTS xChange creates an ADO.NET Connection if an ODBC Connection is detected in DataPump Source. ADO.Net DataReader is used to load data using ODBC Driver.

Note: SQL 2008 has added the ADO.net Destination that uses the ADO.net OdbcConnection Wrapper in the Destination Adapter of a DataFlow.

OLEDB Connection

SSIS supports most of the OLEDB drivers without any problems. It's recommended that you check the Supported Data Provider Support List For SSIS.

Variable Migration

DTS xChange migrates all of your DTS variables. The following actions are performed during variable migration:

- Com Object Data Type is converted to Object Data Type
- Invalid characters are removed from variable names. SSIS only allows AlphaNumeric characters and underscore (_) in the variable name.
- Empty variables without any data type are changed to System.String data type

The following table displays DTS to SSIS Data Type Mapping performed by DTS xChange:

DTS Data Type	SSIS Data Type
Bool	Boolean
Currency	Double
Date	DateTime
Decimal	Double
Int	Int32

DTS Data Type	SSIS Data Type
Integer	Int32
Integer (1 Byte)	SByte
Integer (Small)	Int16
Real (4 Bytes)	Single
Real (8 Bytes)	Double
String	String
Unsigned Int	UInt32
Unsigned Int (1 Byte)	Byte
Unsigned Int (2 Byte)	Char
Unsigned Int (4 Byte)	UInt32
Empty	String
_ComObject	Object