Import Excel unicode data with SQL Server Integration Services

Problem 1

One problem though that I have faced with importing data from Excel into a SQL Server table is the issue of having to convert data types from Unicode to non-Unicode. SSIS treats data in an Excel file as Unicode, but my database tables are defined as non-Unicode, because I don't have the need to store other code sets and therefore I don't want to waste additional storage space. Is there any simple way to do this in SSIS?

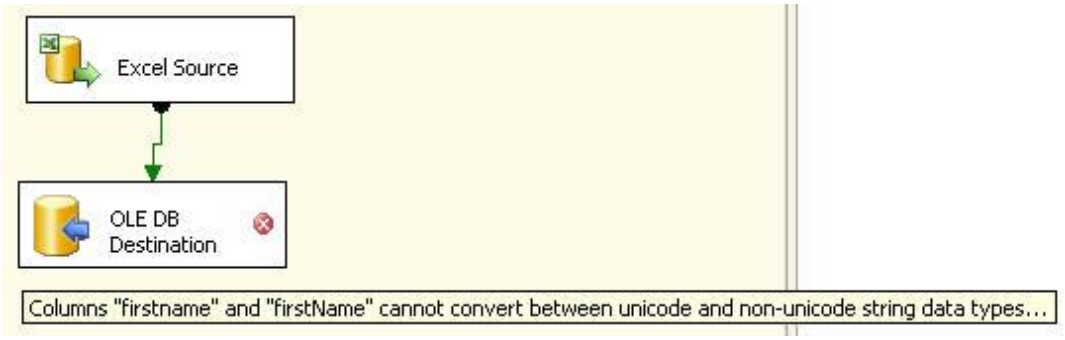
Solution 1

If you have used SSIS to import Excel data into SQL Server you may have run into the issue of having to convert data from Unicode to non-Unicode. By default Excel data is treated as Unicode and also by default when you create new tables SQL Server will make your character type columns Unicode as well (nchar, nvarchar,etc...) If you don't have the need to store Unicode data, you probably always use non-Unicode datatypes such as char and varchar when creating your tables, so what is the easiest way to import my Excel data into non-Unicode columns?

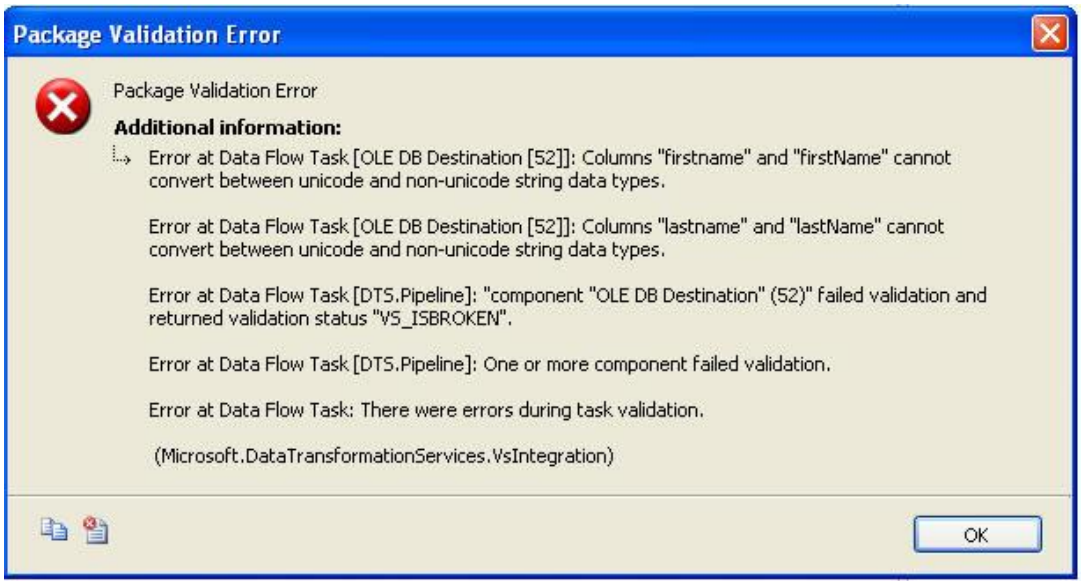
Using SSIS to import excel data into “Unicode” table in SQL Server is straightforward and error free.

Using SSIS to import excel data into “non Unicode” (varchar) table in SQL Server has data conversion errors. I.e.

Columns "firstname" and "firstname" cannot convert between unicode and non-unicode data types...



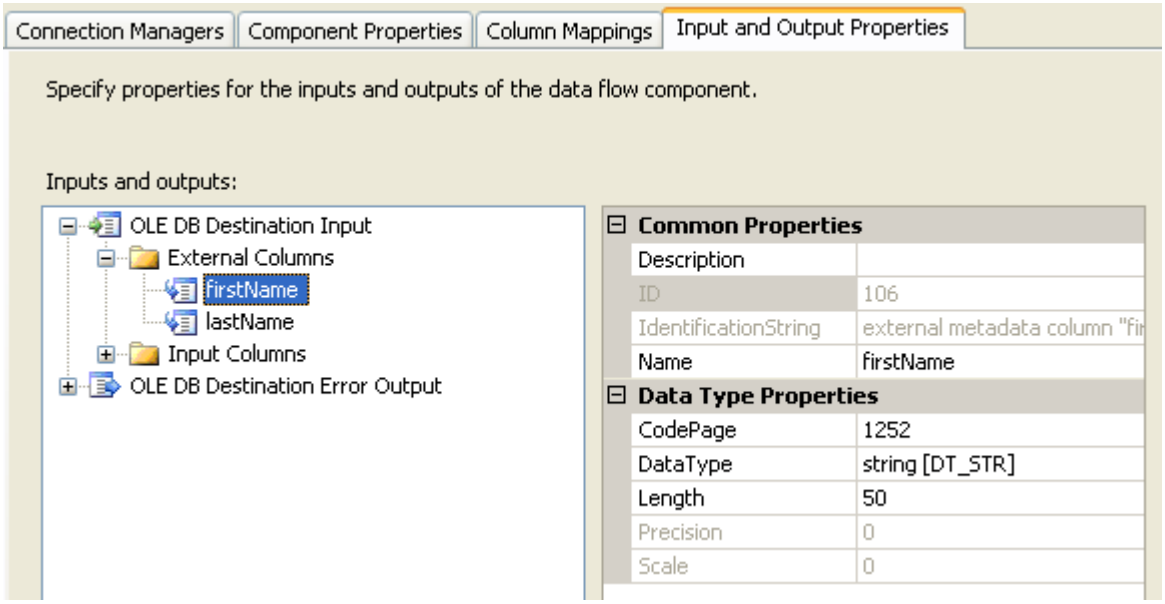
If we execute the task we get the following error dialog box which gives us additional information.



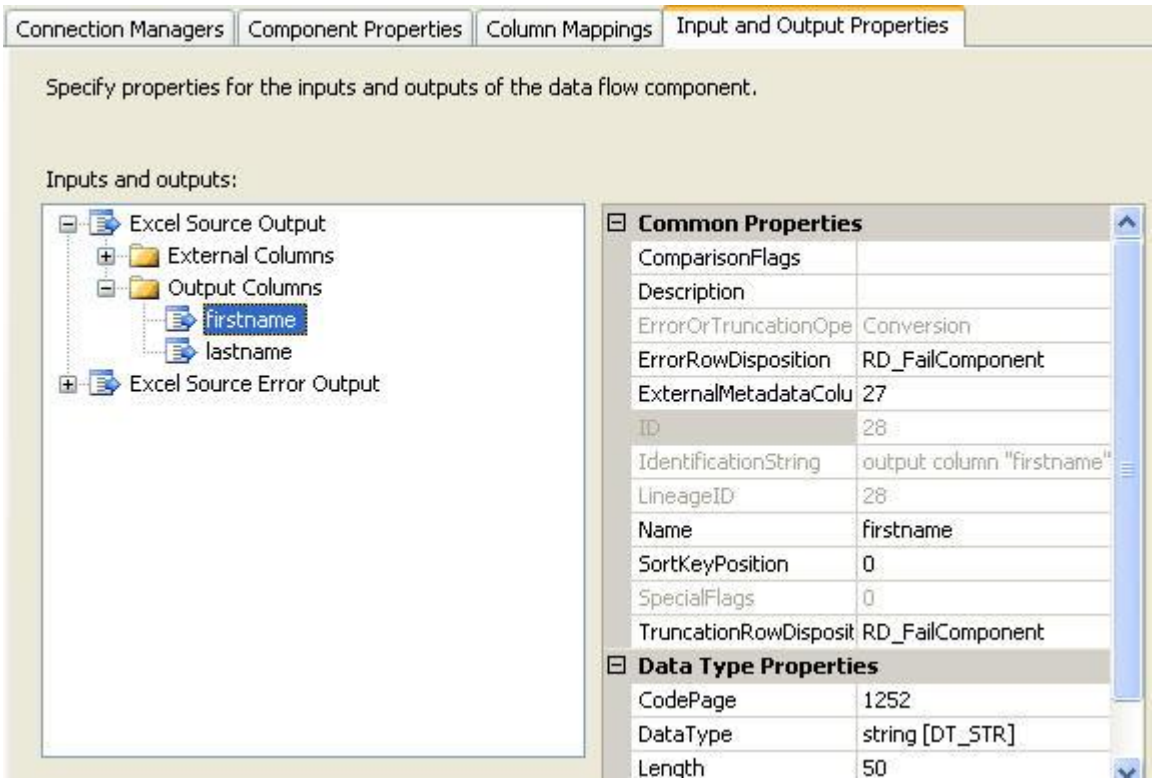
Solving the Problem

So based on the error we need to convert the data types so they are the same types.

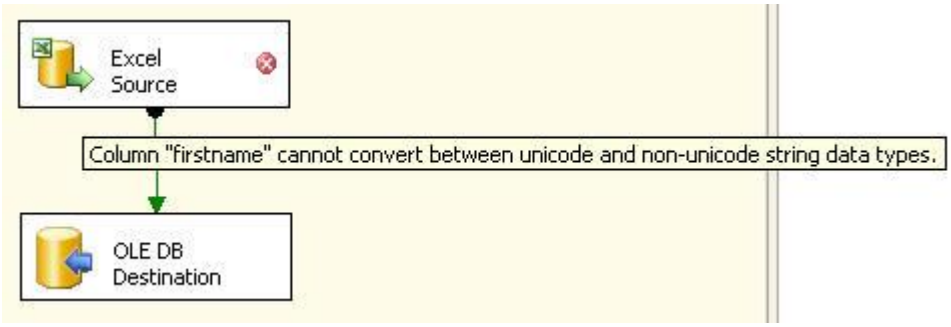
If you right click on the OLE Destination and select "Show Advanced Editor" you have the option of changing the Data Type from string [DT_STR] to Unicode string [DT_WSTR]. But once you click on OK it looks like the changed was saved, but if you open the editor again the change is gone and back to the original value. This makes sense since you cannot change the data type in the actual table.



If you right click on the Excel Source and select "Show Advanced Editor" you have the option of changing the Data Type from Unicode string [DT_WSTR] to string [DT_STR] and the change is saved.

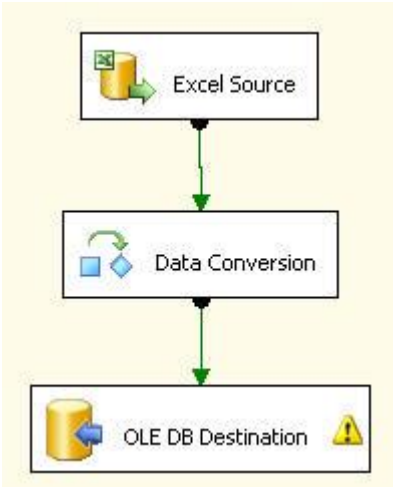


If you click OK the change is saved, but now you get the error in the Excel Source that you cannot convert between unicode and non-unicode as shown below. So this did not solve the problem either.



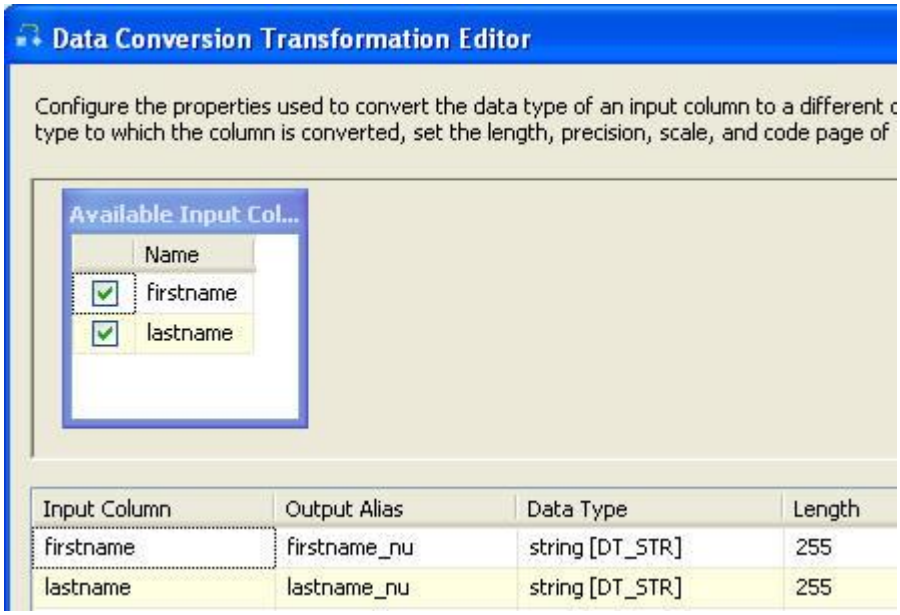
Using the Data Conversion Task

So to get around this problem we have to also use a Data Conversion task. This will allow us to convert data types so we can get the import completed. The following picture shows the "Data Conversion" task in between the Excel Source and the OLE DB Destination.

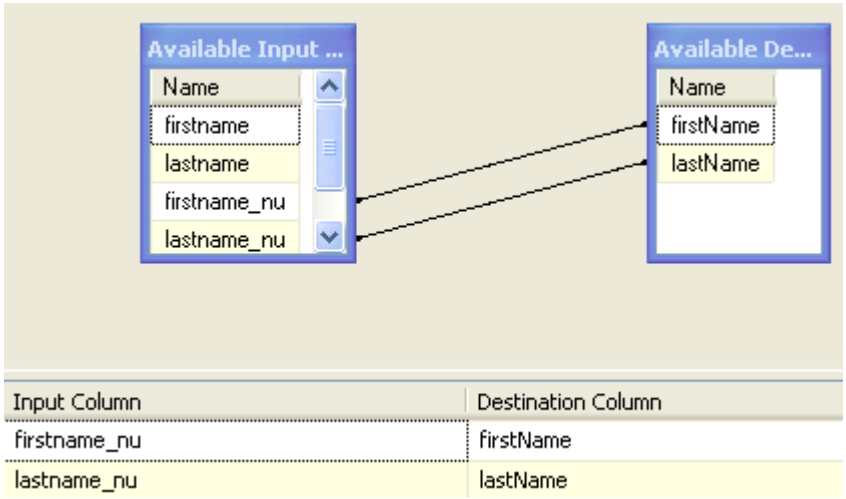


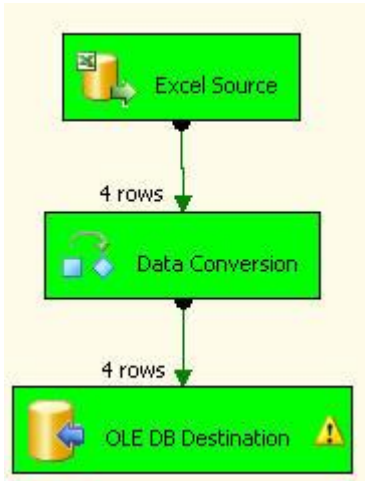
If you right click on "Data Conversion" and select properties you will get a dialog box such as the following. In here we created an Output Alias for each column.

Our firstname column becomes firstname_nu (this could be any name you want) and we are making the output be a non-unicode string. In addition we do the same thing for the lastname column.



If we save this and change the mapping as shown to use our new output columns and then execute the task we can see that the import was successful.





As you can see this is pretty simple to do once you know that you need to use the Data Conversion task to convert the data types.

Next Steps

- Next time you are importing data into SQL Server, don't forget about using the Data Conversion task if you are importing unicode data types into non-unicode columns
- If you encounter this error Columns "xx" and "xx" cannot convert between unicode and non-unicode data types...remember this tip

Problem 2

Sometimes an SSIS Package fails even though when there were no changes in the structure/schema of the Excel worksheet. I investigated it and I noticed that the SSIS Package succeeded for some set of files, but for others it failed. I found that the structure/schema of the worksheet from both these sets of Excel files were the same, the data was the only difference. How come just changing the data can make an SSIS Package fail? What actually causes this failure? What can we do to fix it?

Solution 2

This example should demonstrate the actual failure and solution for this problem. As you can see in the image below, I have 18 records in the Excel worksheet, when I ran my SSIS Package to load the data from this worksheet, it worked fine.

	A	B
1	**ProductDescriptionID**	**Description**
2	3	Chromoly steel.
3	630	Wide-link design.
4	912	Self-sealing tube.
5	909	High-density rubber.
6	618	Super rigid spindle.
7	914	General purpose tube.
8	907	Higher density rubber.
9	619	High-strength crank arm.
10	886	Designed to absorb shock.
11	850	Clipless pedals - aluminum.
12	889	Rubber bumpers absorb bumps.
13	613	Superior shifting performance.
14	913	Conventional all-purpose tube.
15	1203	Rugged weatherproof headlight.
16	888	Lightweight foam-padded saddle.
17	745	Sealed cartridge keeps dirt out.
18	1202	Rechargeable dual-beam headlight.
19	853	A stable pedal for all-day riding.
20		

In the next image, I made some changes to row number 7. The description for ProductDescriptionId 907 is much larger than the previous data load. When I ran my SSIS package again to load the data from this worksheet, it worked fine as well.

	A	B
1	ProductDescriptionID	Description
2	3	Chromoly steel.
3	630	Wide-link design.
4	912	Self-sealing tube.
5	909	High-density rubber.
6	618	Super rigid spindle.
7	914	General purpose tube.
8	907	This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has a extremely light heat-treated aluminum frame, and steering that allows precision control. Each frame is hand-crafted in our Bothell facility to the optimum diameter and wall-thickness required of a premium mountain frame. The heat-treated welded aluminum frame has a larger diameter tube that absorbs the bumps.
9	619	High-strength crank arm.
10	886	Designed to absorb shock.
11	850	Clipless pedals - aluminum.
12	889	Rubber bumpers absorb bumps.
13	613	Superior shifting performance.
14	913	Conventional all-purpose tube.
15	1203	Rugged weatherproof headlight.
16	888	Lightweight foam-padded saddle.
17	745	Sealed cartridge keeps dirt out.
18	1202	Rechargeable dual-beam headlight.
19	853	A stable pedal for all-day riding.
20		

In the next image, I reverted the previous change and made some changes to the row number 14. The description for ProductDescriptionId 1203 is much larger than the previous data load. When I ran my SSIS package again to load the data from this worksheet, it failed with the following exception:

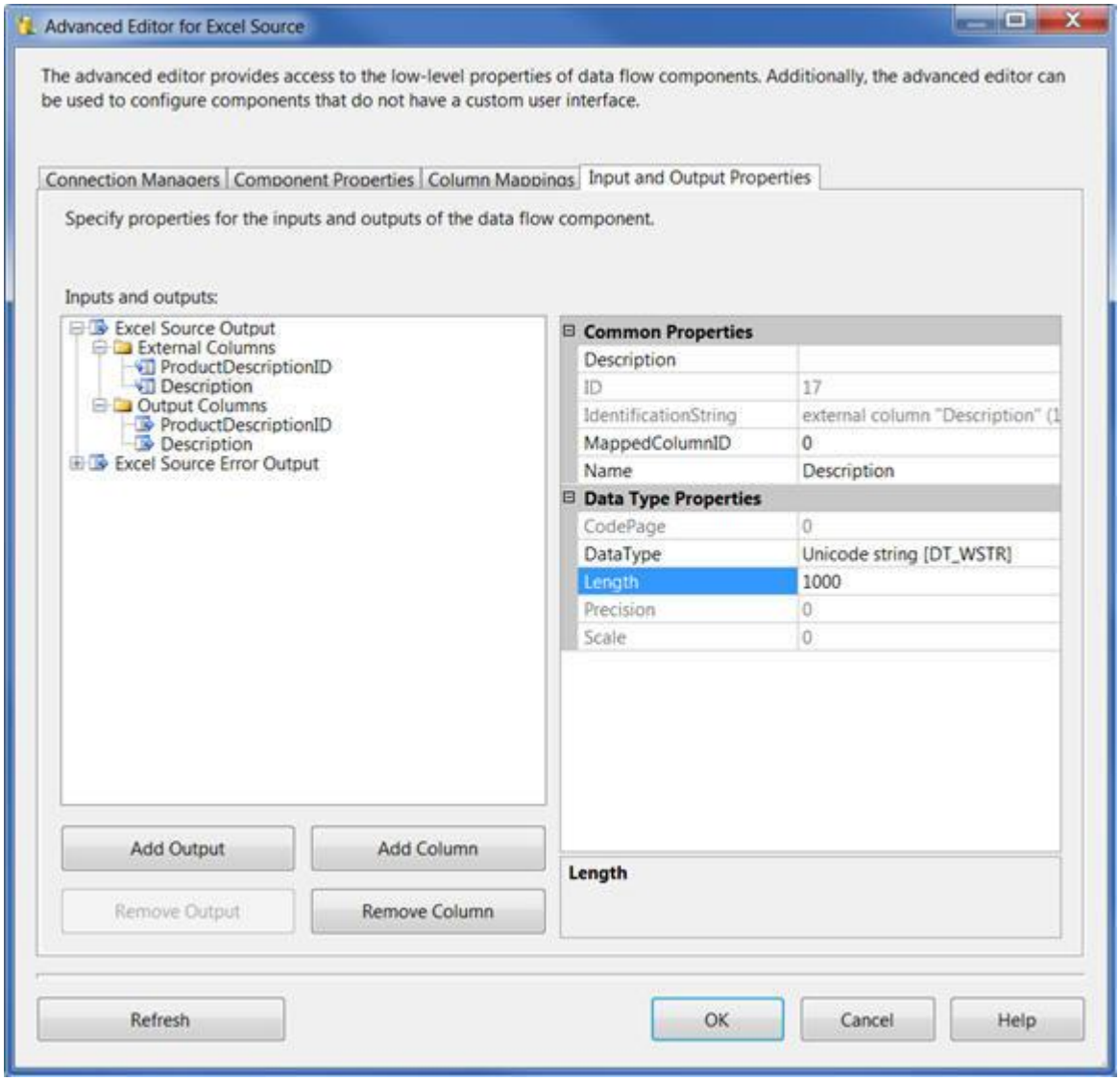
[Excel Source [1]] Error: There was an error with output column "Description" (18) on output "Excel Source Output" (9). The column status returned was: "Text was truncated or one or more characters had no match in the target code page.".
[Excel Source [1]] Error: The "output column "Description" (18)" failed because truncation occurred, and the truncation row disposition on "output column "Description" (18)" specifies failure on truncation. A truncation error occurred on the specified object of the specified component.
[SSIS.Pipeline] Error: SSIS Error Code DTS_E_PRIMEOUTPUTFAILED. The PrimeOutput method on component "Excel Source" (1) returned error code 0xC020902A. The component returned a failure code when the pipeline engine called PrimeOutput(). The meaning of the failure code is defined by the component, but the error is fatal and the pipeline stopped executing. There may be error messages posted before this with more information about the failure.

	A	B
1	ProductDescriptionID	Description
2	3	Chromoly steel.
3	630	Wide-link design.
4	912	Self-sealing tube.
5	909	High-density rubber.
6	618	Super rigid spindle.
7	914	General purpose tube.
8	907	Higher density rubber.
9	619	High-strength crank arm.
10	886	Designed to absorb shock.
11	850	Clipless pedals - aluminum.
12	889	Rubber bumpers absorb bumps.
13	613	Superior shifting performance.
14	913	Conventional all-purpose tube.
15	1203	This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has a extremely light heat-treated aluminum frame, and steering that allows precision control. Each frame is hand-crafted in our Bothell facility to the optimum diameter and wall-thickness required of a premium mountain frame. The heat-treated welded aluminum frame has a larger diameter tube that absorbs the bumps.
16	888	Lightweight foam-padded saddle.
17	745	Sealed cartridge keeps dirt out.
18	1202	Rechargeable dual-beam headlight.
19	853	A stable pedal for all-day riding.
20		

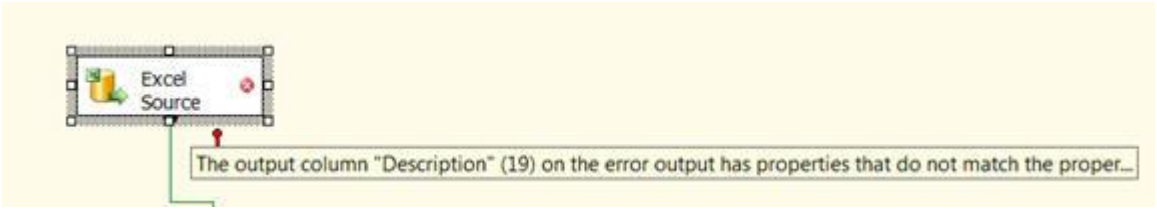
What caused the above SSIS package to fail?

SSIS Excel Connection Manager determines the data type of each column of the worksheet on the basis of the data of that particular column from the first 8 rows. This is the default behaviour and the SSIS connection manager uses a value in the registry to determine the number of rows for the data type determination.

In your SSIS Package, right click on the Excel source in the data flow task and click on "Advance Editor for Excel Source". Next change the data type and length of the column from its default value. For example, in my case I have a "Description" column which has text data and the length is up to 500 characters. I have put max length for this column as 1000. Now click on "OK" button.



But what is this, the validation of the Excel Source failed itself as you can see below:



When you double click on the Excel Source task, it will inform that the component is not in a valid state and asks for your confirmation to fix this issue. When you click on the "Yes" button it will reset the data type and length of the column to what it was earlier, before we made the changes as above.

Now to summarize the whole thing: SSIS Excel Connection manager determines the data type and length of columns from the worksheet on the basis of the first eight rows of data. Even though we can change the data type and length from the Advance Editor for Excel Source, it will not be valid and the information will be reset by SSIS Excel Connection manager automatically using the same determination process.

Fixing the problem now in the registry

So as I said before, the number of rows to consider when determining the data type and length is determined by a registry key called TypeGuessRows by the SSIS Excel Connection Manager. By default its value is 8 and hence 8 rows are considered when determining the data type and length.

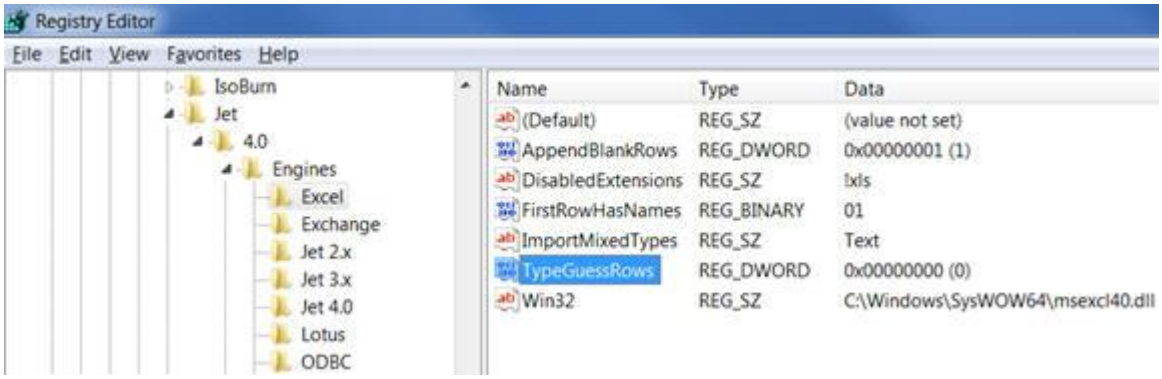
Now coming back to the solution, these are some of the options to address this problem:

- Configure the source system to provide your Excel file in which data is sorted on the basis of the length of the data in each column so that largest value of each column appears in the first row and alphanumeric data appears before numeric data.
- Configure the source system to provide a dummy record in your Excel file as first row with desired data type and size; then after data import you can remove/delete that dummy record from the database.
- Configure the source system to provide you a csv file instead of Excel file because with a csv file you have more control to determine data type and length of a column.
- Changing the TypeGuessRows registry key to 0 from its default value of 8. This will make the Excel connection manager consider all the rows when determining data type and length of each column.

Unfortunately, the first three options do not apply in my scenario as I do not have control on the source system providing data in worksheet. With this being said, I made the change in the TypeGuessRows registry key and updated its value from 8 to 0. After making this change, my same package worked like a charm for the same Excel worksheets for which it failed last time.

Registry Key Location - [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\4.0\Engines\Excel]

Please Note : On a 64-Bit Windows Server machine the registry key will be available here: HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Jet\4.0\Engines\Excel



Please be **cautious** when changing the value for TypeGuessRows registry key and keep these points in mind:

- TypeGuessRows registry key is global setting and its not only going to impact your SSIS Package, but it will impact every place where it is referenced.
- Changing the value for TypeGuessRows registry key to 0 from its default value of 8, makes the Excel connection manager consider all the rows when determining the data type and length of each column and hence it could have a severe impact on the performance of your SSIS package if the number of rows in your Excel worksheet is large.

Now, as we saw, to fix this issue the easiest solution is to make change in the registry, but in many scenarios you will not have control in making this change as the servers are managed by the Operations Team or there might be several other applications running on the same machine. Even if you have control in changing this setting, this change might cause your SSIS Package to perform poorly, based on the amount of data you have in your Excel worksheet, and it may impact other systems as well when this registry key is being referenced. So now the question is, is there any way we can avoid making changes in the registry, but still solve the problem?

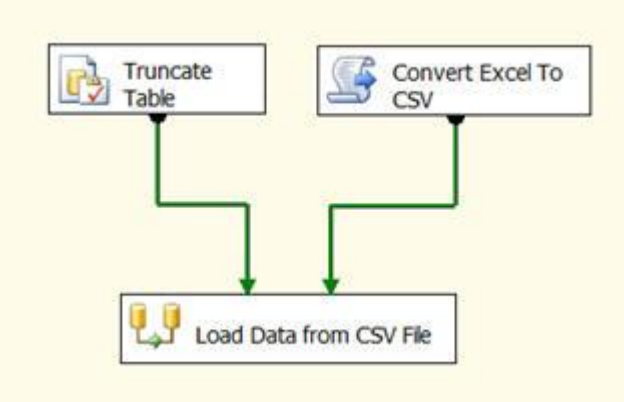
Solution

The basic idea of this solution is to convert the Excel worksheet to CSV file and use the Flat File Connection Manager/Source adapter to import data from the CSV file. With the Flat File Connection Manager/Source Adapter we have more control to define the data type and length of each column.

These are questions that come to mind when thinking about converting an Excel worksheet to a CSV file:

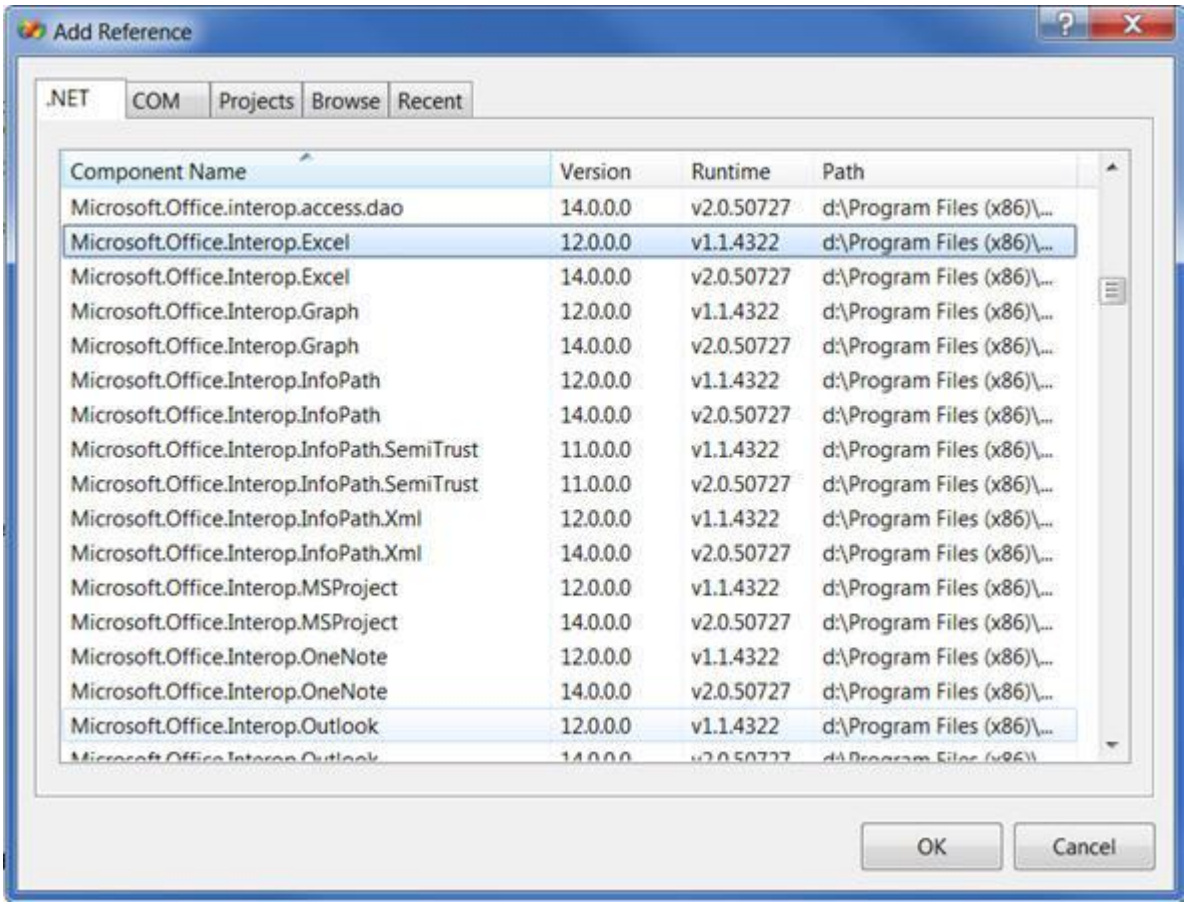
- When saving an Excel worksheet using a csv extension does this make it a CSV file?
- A CSV file uses a comma as column separator, but what if there are commas in the data itself?
- What is the impact of converting an Excel worksheet to CSV file?

Well, simply saving the Excel worksheet using a CSV extension will not make it CSV file as the storage format is different for both of these file types. Rather we can use the Excel Object Library to save the Excel worksheet as a CSV file using the Script Task in SSIS and then we can import the data directly from the CSV file as shown below.

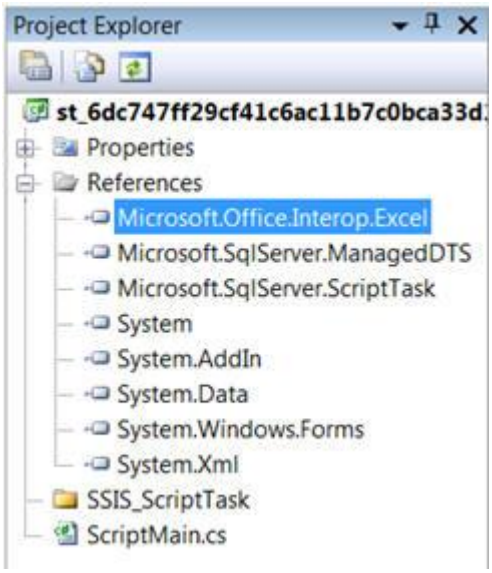


In the Script Task, when writing code for conversion, first of all you need to add a reference to Microsoft.Office.Interop.Excel under .NET components as shown below:

(Assuming Visual Studio 2010) In the solution explorer you can right click on the references tree and choose "add reference." Choose the .NET tab and look for "Microsoft.Office.[...]" Components. Add the ones you need.



After adding the required reference, the References node in the Solution Explorer will look like this:



Once you have added the required reference, you can add these lines of code. The complete list of code for converting Excel worksheet to CSV file is provided below. You need to provide the location and name of the Excel worksheet along with the name of the worksheet itself and then the location and name for the CSV file which will be created:

```
/*
   Microsoft SQL Server Integration Services Script Task
   Write scripts using Microsoft Visual C# 2008.
   The ScriptMain is the entry point class of the script.
*/
using System;
using System.Data;
using Microsoft.SqlServer.Dts.Runtime;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
using Microsoft.Office.Interop.Excel;
namespace ST_6dc747ff29cf41c6ac11b7c0bca33d19.csproj
{
    [System.AddIn.AddIn("ScriptMain", Version = "1.0", Publisher = "", Description
= "")]
    public partial class ScriptMain :
Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
    {
        #region VSTA generated code
        enum ScriptResults
        {
            Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
            Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
        };
        #endregion
        /*
  The execution engine calls this method when the task executes.
  To access the object model, use the Dts property. Connections, variables,
events,
  and logging features are available as members of the Dts property as shown in
the following examples.
  To reference a variable, call
Dts.Variables["MyCaseSensitiveVariableName"].Value;
  To post a log entry, call Dts.Log("This is my log text", 999, null);
  To fire an event, call Dts.Events.FireInformation(99, "test", "hit the help
message", "", 0, true);
  To use the connections collection use something like the following:
  ConnectionManager cm = Dts.Connections.Add("OLEDB");
  cm.ConnectionString = "Data Source=localhost;Initial
Catalog=AdventureWorks;Provider=SQLNCLI10;Integrated Security=SSPI;Auto
Translate=False;";
  Before returning from this method, set the value of Dts.TaskResult to indicate
success or failure.

  To open Help, press F1.
 */
        private static Workbook mWorkBook;
        private static Sheets mWorkSheets;
        private static Worksheet mWSheet1;
        private static Excel.Application oXL;
        private static string ErrorMessage = string.Empty;
        public void Main()
        {
            try
            {
```

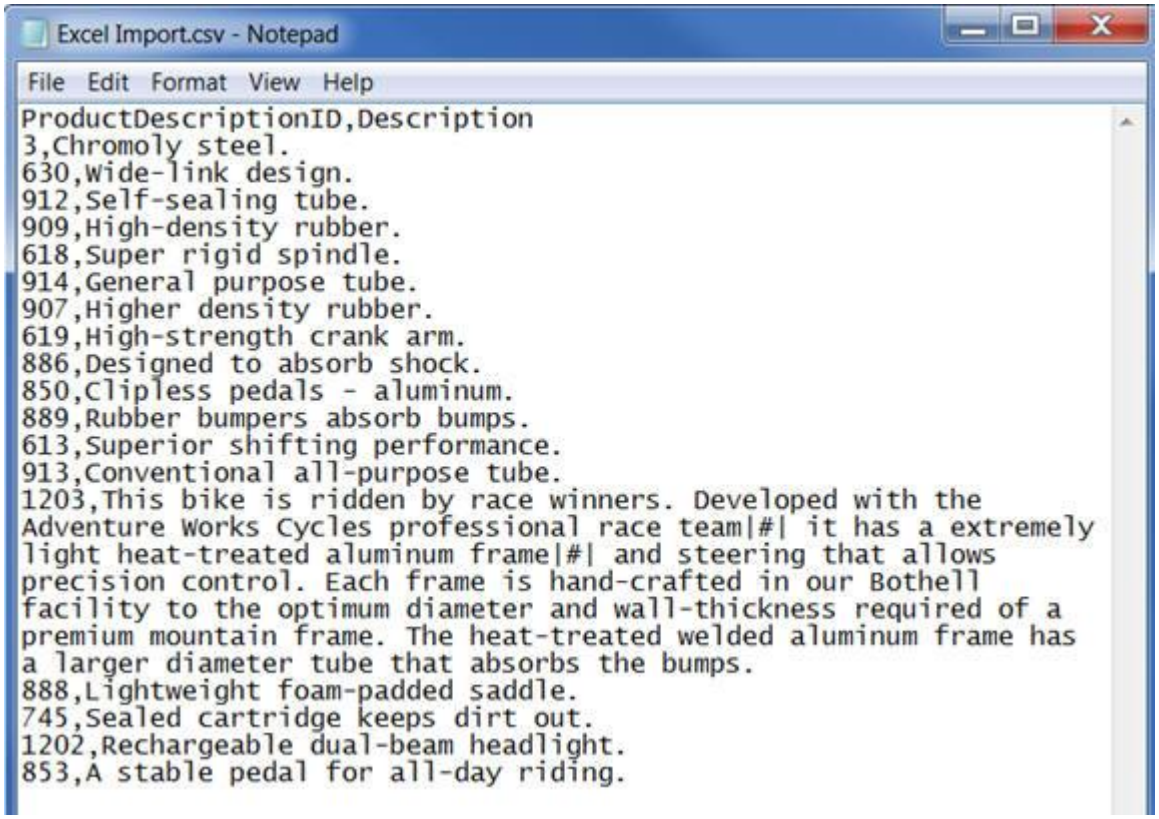
```
                string sourceExcelPathAndName = @"D:\Excel Import\Excel
Import.xls";
                string targetCSVPathAndName = @"D:\Excel Import\Excel Import.csv";
                string excelSheetName = @"Sheet1";
                string columnDelimeter = @"|#|";
                int headerRowsToSkip = 0;
                if (ConvertExcelToCSV(sourceExcelPathAndName,
targetCSVPathAndName, excelSheetName, columnDelimeter, headerRowsToSkip) == true)
                {
                    Dts.TaskResult = (int)ScriptResults.Success;
                }
                else
                {
                    Dts.TaskResult = (int)ScriptResults.Failure;
                }
            }
            catch (Exception ex)
            {
                Dts.TaskResult = (int)ScriptResults.Failure;
            }
        }
        public static bool ConvertExcelToCSV(string sourceExcelPathAndName, string
targetCSVPathAndName, string excelSheetName, string columnDelimeter, int
headerRowsToSkip)
        {
            try
            {
                oXL = new Excel.Application();
                oXL.Visible = false;
                oXL.DisplayAlerts = false;
                Excel.Workbooks workbooks = oXL.Workbooks;
                mWorkBook = workbooks.Open(sourceExcelPathAndName, 0, false, 5,
"", "", false, XlPlatform.xlWindows, "", true, false, 0, true, false, false);
                //Get all the sheets in the workbook
                mWorkSheets = mWorkBook.Worksheets;
                //Get the specified sheet
                mWSheet1 = (Worksheet)mWorkSheets.get_Item(excelSheetName);
                Excel.Range range = mWSheet1.UsedRange;
                //deleting the specified number of rows from the top
                Excel.Range rngCurrentRow;
                for (int i = 0; i < headerRowsToSkip; i++)
                {
                    rngCurrentRow = range.get_Range("A1", Type.Missing).EntireRow;
                    rngCurrentRow.Delete(XlDeleteShiftDirection.xlShiftUp);
                }
                //replacing ENTER with a space
                range.Replace("\n", " ", Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing);
                //replacing COMMA with the column delimeter
                range.Replace(",", columnDelimeter, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing);
                mWorkBook.SaveAs(targetCSVPathAndName, XlFileFormat.xlCSVMSDOS,
                Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Microsoft.Office.Interop.Excel.XlSaveAsAccessMode.xlExclusive,
                Type.Missing, Type.Missing, Type.Missing,
```

```
                Type.Missing, false);
                return true;
            }
            catch (Exception ex)
            {
                ErrorMessage = ex.ToString();
                return false;
            }
            finally
            {
                if (mWSheet1 != null) mWSheet1 = null;
                if (mWorkBook != null) mWorkBook.Close(Type.Missing, Type.Missing,
Type.Missing);
                if (mWorkBook != null) mWorkBook = null;
                if (oXL != null) oXL.Quit();
                System.Runtime.InteropServices.Marshal.ReleaseComObject(oXL);
                if (oXL != null) oXL = null;
                GC.WaitForPendingFinalizers();
                GC.Collect();
                GC.WaitForPendingFinalizers();
                GC.Collect();
            }
        }
    }
}
```

Below you can see the Excel worksheet which I am using as the source file for conversion. You will notice row number 15 has a long string and also the description of this row contains commas in it as well.

	A	B
1	**ProductDescriptionID**	**Description**
2	3	Chromoly steel.
3	630	Wide-link design.
4	912	Self-sealing tube.
5	909	High-density rubber.
6	618	Super rigid spindle.
7	914	General purpose tube.
8	907	Higher density rubber.
9	619	High-strength crank arm.
10	886	Designed to absorb shock.
11	850	Clipless pedals - aluminum.
12	889	Rubber bumpers absorb bumps.
13	613	Superior shifting performance.
14	913	Conventional all-purpose tube.
15	1203	This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has a extremely light heat-treated aluminum frame, and steering that allows precision control. Each frame is hand-crafted in our Bothell facility to the optimum diameter and wall-thickness required of a premium mountain frame. The heat-treated welded aluminum frame has a larger diameter tube that absorbs the bumps.
16	888	Lightweight foam-padded saddle.
17	745	Sealed cartridge keeps dirt out.
18	1202	Rechargeable dual-beam headlight.
19	853	A stable pedal for all-day riding.

After the conversion, the CSV file will look like this. You will notice where there were commas we now have some special characters "|#|":



The idea behind this is, before conversion replace all the commas with some special characters and after importing the data from the CSV file update the special characters back to a comma. This can be done using code such as this.

```
SELECT * FROM [dbo].[ProductInformation]
UPDATE [dbo].[ProductInformation]
SET [Description] = REPLACE([Description], '|#|', ',')
SELECT * FROM [dbo].[ProductInformation]
```

In order to use this approach, we need to have extra storage space for having both a CSV file along with an Excel file. Apart from that, the Excel object library will take a few seconds to save the file as a CSV file. I haven't tried it on very large file, though I think this should not take much longer. Once you have the data loaded you can do anything else you need to at that point.

Use SSIS to import one cell of an Excel file into SQL Server

Problem

Recently I needed to find a method to import one cell of an Excel sheet into SQL Server 2005 using a scheduled job on a 64 bit clustered environment. For example, I have a monthly sales report listing sales per product category, such as Books, Magazines, DVDs, etc. I only want to import the Total Sales amount into SQL Server and this number is located in an Excel file "sales.xls.sheet1.cell.B5". I used to use OPENROWSET to solve the problem, however OPENROWSET requires Microsoft Jet OLE DB Provider which is not available in 64 bit. In this tip I am going to talk about how to use SSIS to accomplish this task easily.

Solution

The following steps show how to import one cell of an Excel sheet into SQL Server.

For example, this is what my Excel file looks like, the data I need to import is Total Sales (2300) in cell B5.

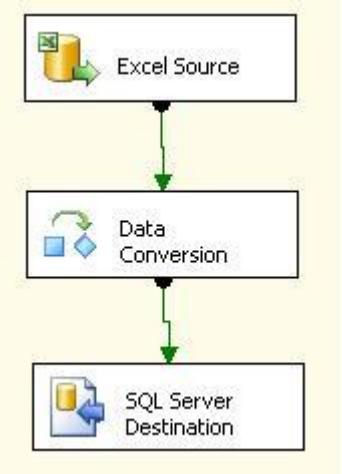
	A	B
1	**Product**	**Sales($)**
2	Books	800
3	Magazines	500
4	DVDs	1000
5	**Total**	**2300**

For simplicity, this is the structure of the table I am loading the data into.

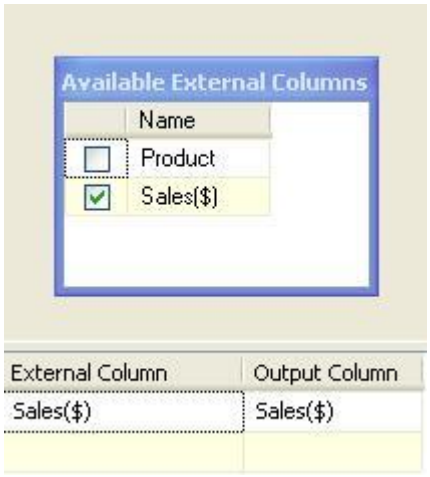
```
CREATE TABLE [dbo].[Sales](
[Product] [varchar](50) NULL,
[Sales] [numeric](18, 2) NULL
) ON [PRIMARY]
```

SSIS Package

1. Using SSIS, create a Data Flow Task as shown below. When adding the Data Flow Destinations, make sure to select "SQL Server Destination", not "OLE DB Destination". The difference between these two is: SQL Server Destination gives you ability to define which row or rows (row 5 in this case) you want to import, while an OLEDB Destination doesn't provide this option.

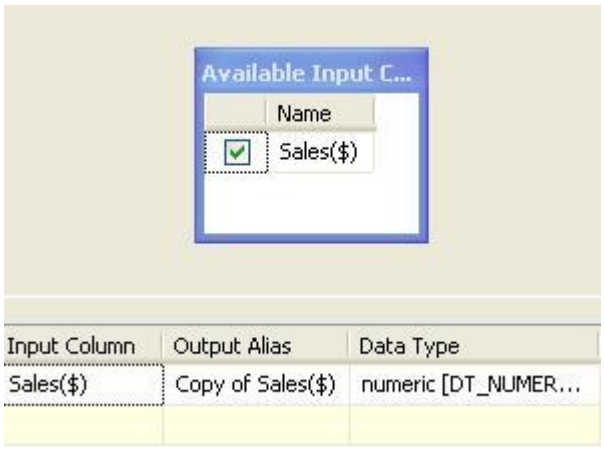


2. In the Excel Source Editor, select column "Sales($)" as shown below. In my example, I have 5 rows of data and when I selected my Excel file I specified that the first row does not have column names. In addition I gave column names instead of using the defaults F1, F2, etc... Otherwise if I said my Excel data had column names I would import data row 4.



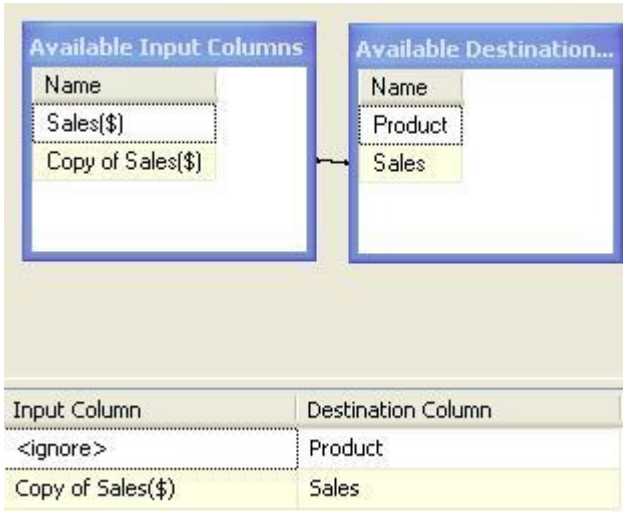
---

3. In the Data Conversion Transformation Editor, select Output Data Type as "numeric[DT_NUMERIC]". When setting this up I used a precision of 18 and scale of 2 to allow for decimal values.

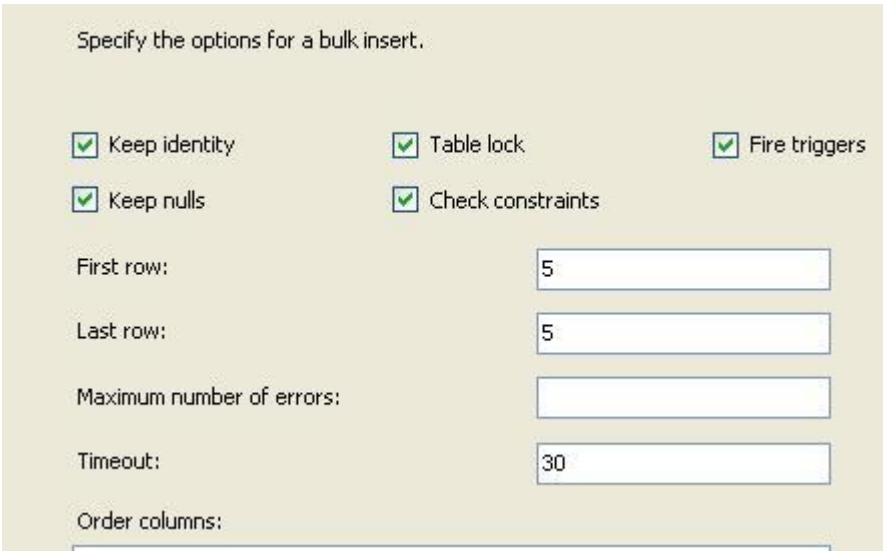


---

4. In SQL Destination Editor, map the converted number -- "Copy of Sales($)" in this example to the Destination Column.



5. Again in SQL Destination Editor, select Advanced, put 5 as both the "First row" and "Last row" as shown below. Row 5 contains the total sales value that we want to import. You can specify the other options shown below as needed.



That's all there is to it. In this example I am always going to pull in the value in cell B5, so if you have a static Excel sheet this is a nice simple approach to pull in just one value.

Next Steps

- Please notice that in order to use a SQL Server Destination, the SSIS package has to be run on local server, in this case it will be the data import destination server. The package file can be located remotely though. Therefore you need set up the schedule job on the destination server to run this package, not on a remote server.
- Since there is no 64 bit driver for Excel, when setting up a scheduled job on a 64 bit server, you have to create a CmdExec step to manually call the 32 bit DTExec.exe to execute the package, such as: C:\SQL 2005 Tools (x86)\90\DTS\Binn\DTExec.exe /FILE "D:\Package.dtsx" /MAXCONCURRENT " -1 " /CHECKPOINTING OFF /REPORTING E
- File DTExec.exe is installed with the SQL Server client component, not the SSIS component. So in a clustered environment, make sure the SQL Server 2005 client is installed on all nodes and DTExec.exe is located on same folder structure, such as C:\SQL 2005 Tools(x86) on all nodes, otherwise the job may fail when a SQL instance fails over to a different node and the file DTExec.exe is not available.

Dynamically find where table data starts in Excel using SSIS

Problem

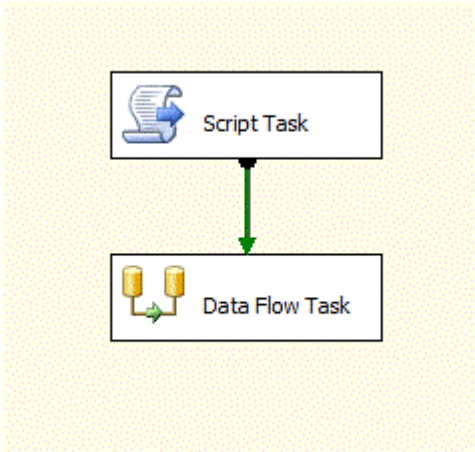
Recently I worked on a project to import Excel spreadsheets from various vendors into our database and not all spreadsheets had the same row number for the header record. This is because most of the vendors had additional information on the top of the header row. In this tip I am going to show how the SSIS script task can be used to solve this issue.

Solution

In order to know where the header record starts we need to open the spreadsheet and search for the headers cell by cell, in this tip I am going to show how we could accomplish this with SSIS script task. Lets look at a sample spreadsheet, the structure is changed for simplicity purposes.

	A	B	C	D
1	Date	11/15/2010		
2	Description Line1			
3	Description Line2			
4				
5	**HdrCol1**	**HdrCol2**	**HdrCol3**	**HdrCol4**
6	1	$ 100.00	1/1/2010	Some Text
7	2	$ 150.00	2/5/2010	Some Text 2
8	3	$ 175.00	2/22/2010	Some Text 3
9	4	$ 200.00	3/1/2010	Some Text 4
10	5	$ 210.00	4/10/2010	Some Text 5
11	6	$ 220.00	4/15/2010	Some Text 6

In the example the header row starts at row 5, to get this information during run time of the SSIS package we used a script task before the data flow task. Below is how the SSIS package Control Flow looks like.

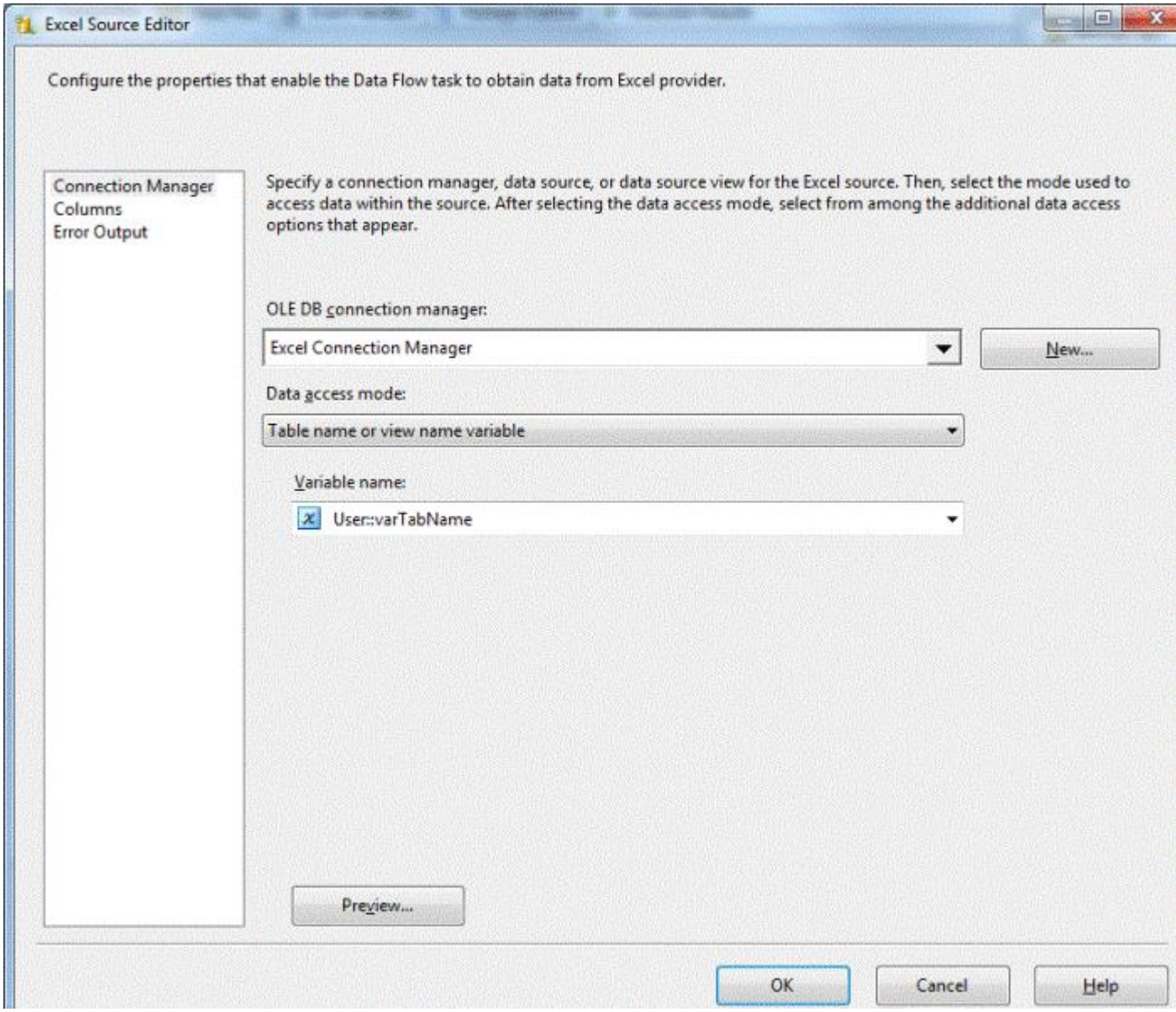


Add a new string variable called "varTabName", and include this variable as a ReadWriteVariable for the Script Task. Add the following code to Script Task

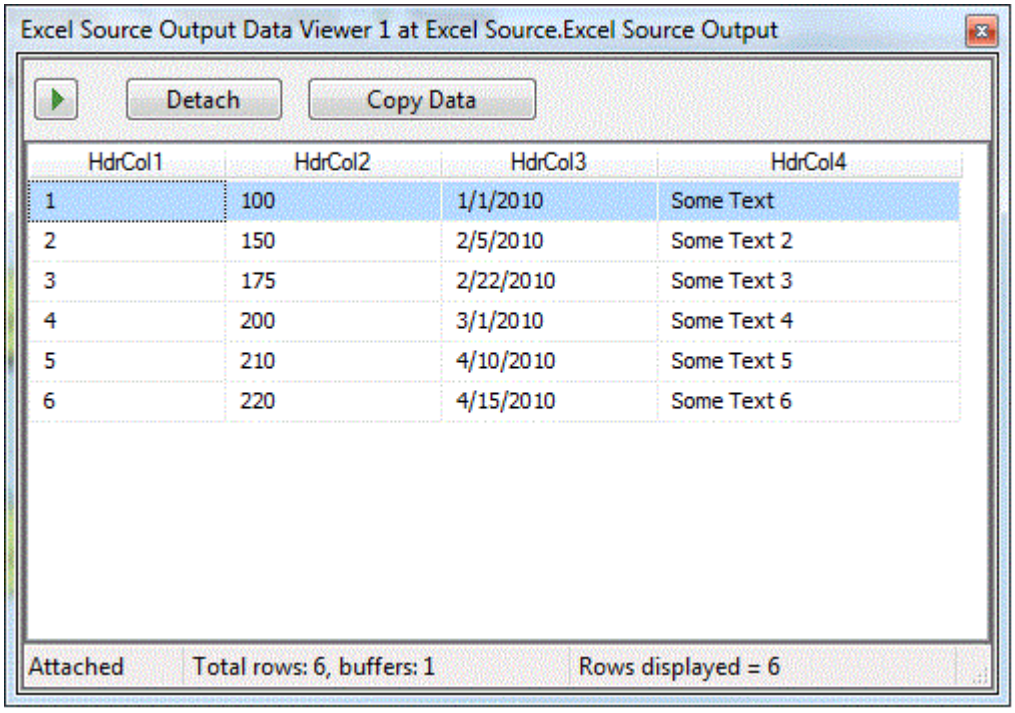
```
public void Main()
{
 string filePath = "C:\\MSSQLTips\\MSSQLTip1.xlsx";
 string tabName = "Sheet1$";
 String strCn = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source="
 + filePath + ";Extended Properties=\"Excel 12.0;HDR=NO;IMEX=1\";";
 OleDbConnection cn = new OleDbConnection(strCn);
 string strSQL = "Select * From [" + tabName + "A1:D100]";
 int iCnt = 0;
 OleDbDataAdapter objAdapter = new OleDbDataAdapter(strSQL, cn);
 DataSet ds = new DataSet();
 objAdapter.Fill(ds, "dSheet1");
 DataTable dt = ds.Tables["dSheet1"];
 foreach (DataRow row in dt.Rows)
 {
  iCnt = iCnt + 1;
  if ((row[0].ToString() == "HdrCol1")
  & (row[1].ToString() == "HdrCol2")
  & (row[2].ToString() == "HdrCol3")
  & (row[3].ToString() == "HdrCol4"))
  {
   Dts.Variables["varTabName"].Value = tabName + "A"
   + iCnt.ToString() + ":D1048576";
   break;
  }
 }
 cn.Close();
 Dts.TaskResult = (int)ScriptResults.Success;
}
```

For simplicity reasons the file path and the tab name of the excel spreadsheet have been hardcoded. We use an OLEDB Connection to connect and query the spreadsheet on a range of cells. Since our sample spreadsheet has 4 columns of data and since we know that the header is always in the top 100 rows we can use the following query "select * from Sheet1$A1:D100" to pull this data into a data set. We then loop through the rows of the data set until we find the Header row, then we append the counter to the tab name and set the varTabName variable. For this example the value of the varTabName variable would be Sheet1$A5:D1048576, where 1048576 is the maximum number of rows for excel 2007.

In the Data flow task, choose Excel as the source and for Data access mode use "Table name or view name variable" option and in the Variable Name drop down choose User::varTabName



Below is the output from the data viewer grid for the data flow task.



We are now able to import the data from the spreadsheet by dynamically setting the data range in the SSIS script task.

Next Steps

- Add exception handling for the script task.
- We can use for Each loop container to loop though multiple files, the file path needs to be a variable which has to be passed to the Script Task.
- We can use GetOleDbSchemaTable to find all the tabs and loop through those.

Configure the Flat File Source in SQL Server Integration Services 2012 to read CSV files

Problem

I am new to SSIS and need to know how to read a comma-separated value (CSV) file into an SQL Server Integration Services (SSIS) 2012 package? What are the steps that you need to follow? Check out this tip to learn more.

Solution

For this solution, we will use the CSV file shown below. The file is named tip.csv and it has five columns and a header row.

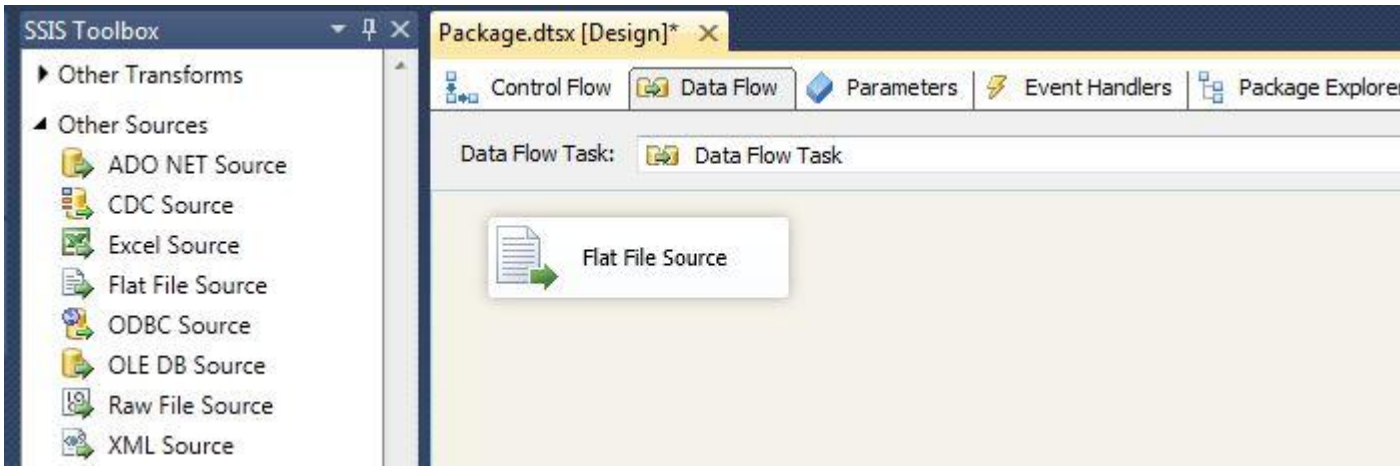
tip.csv

```
ID,Amount,Category,EntryDate,Description
1,123.45,A,2012-11-30,"Aluminum widget"
2,246.80,B,2012-12-31,"Copper widget"
3,368.12,A,2013-01-31,"Silver widget"
4,481.20,B,2013-02-01,"Gold object"
5,510.15,B,2013-03-01,"Platinum widget"
```

The first step is to drag a Data Flow Task onto the package palette as shown below.

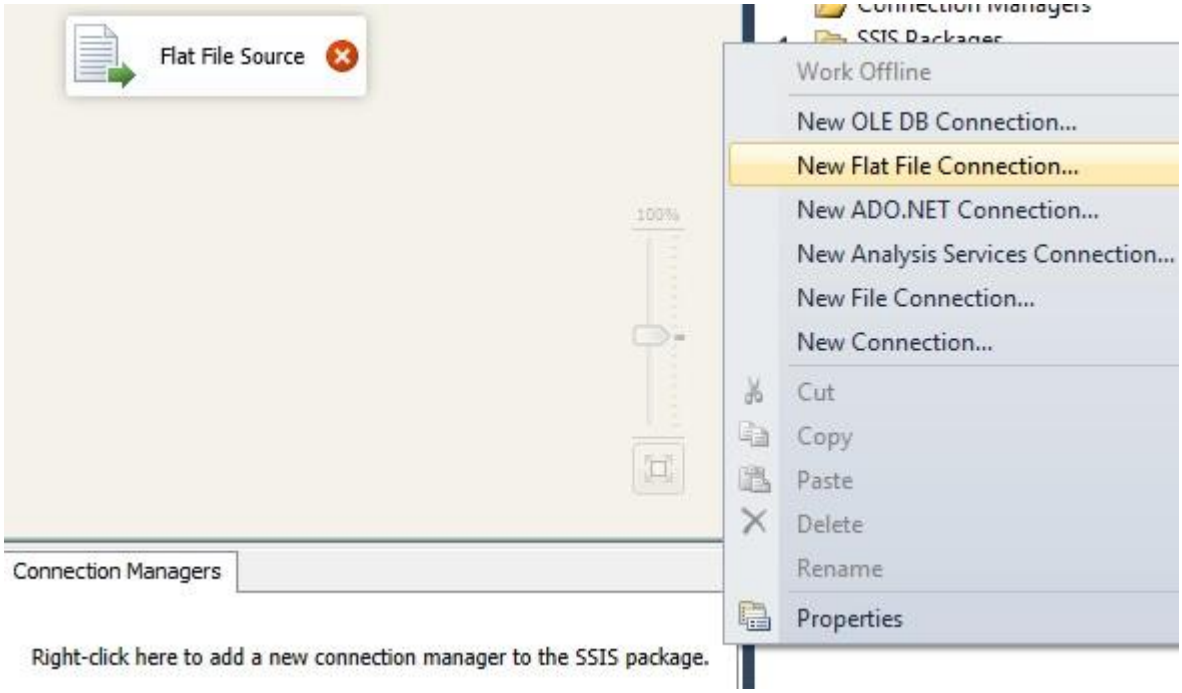
Double click on the Data Flow Task or click on the Data Flow tab.

Next, drag a Flat File Source from the SSIS Toolbox onto the Data Flow palette as shown below.

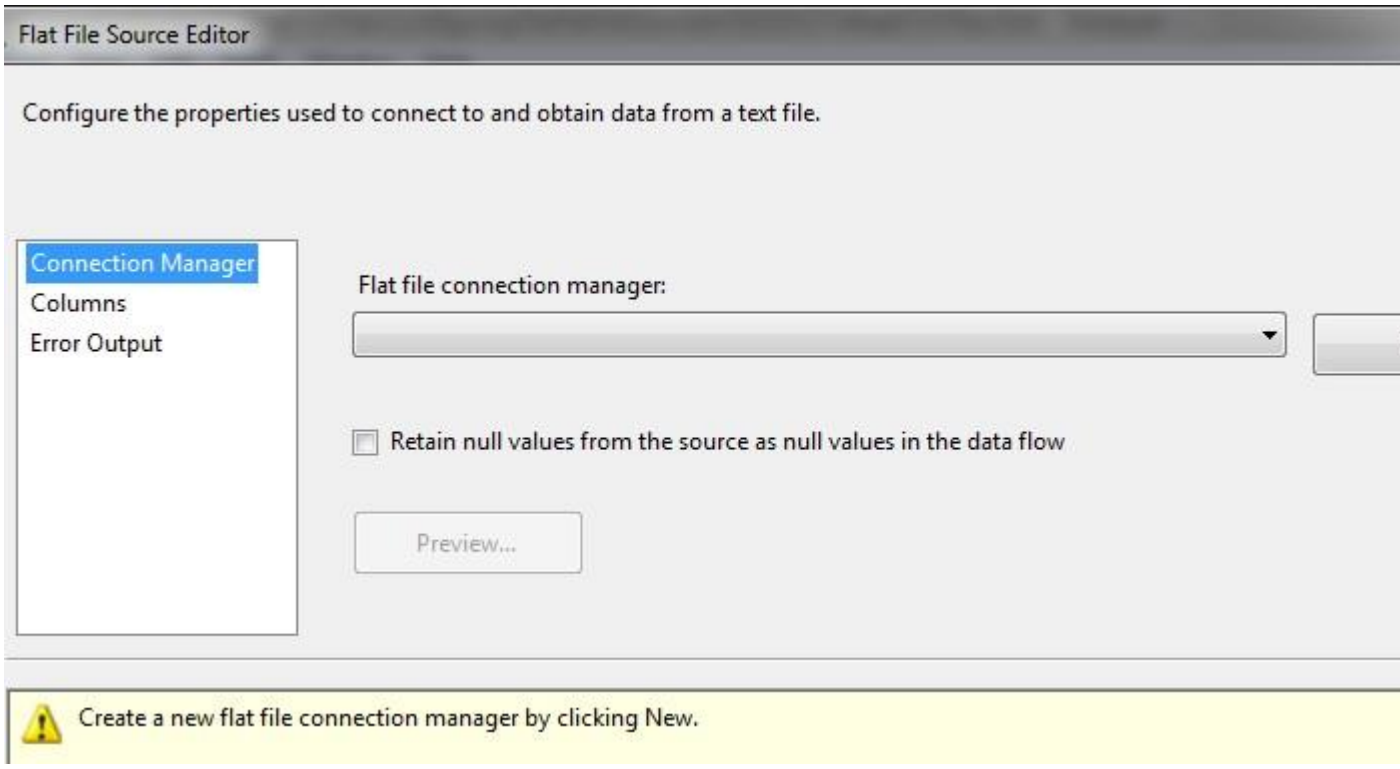


If you see a white X in the red circle, hover over the object to show the help message. The message below states that we need to set up a connection to the flat file.

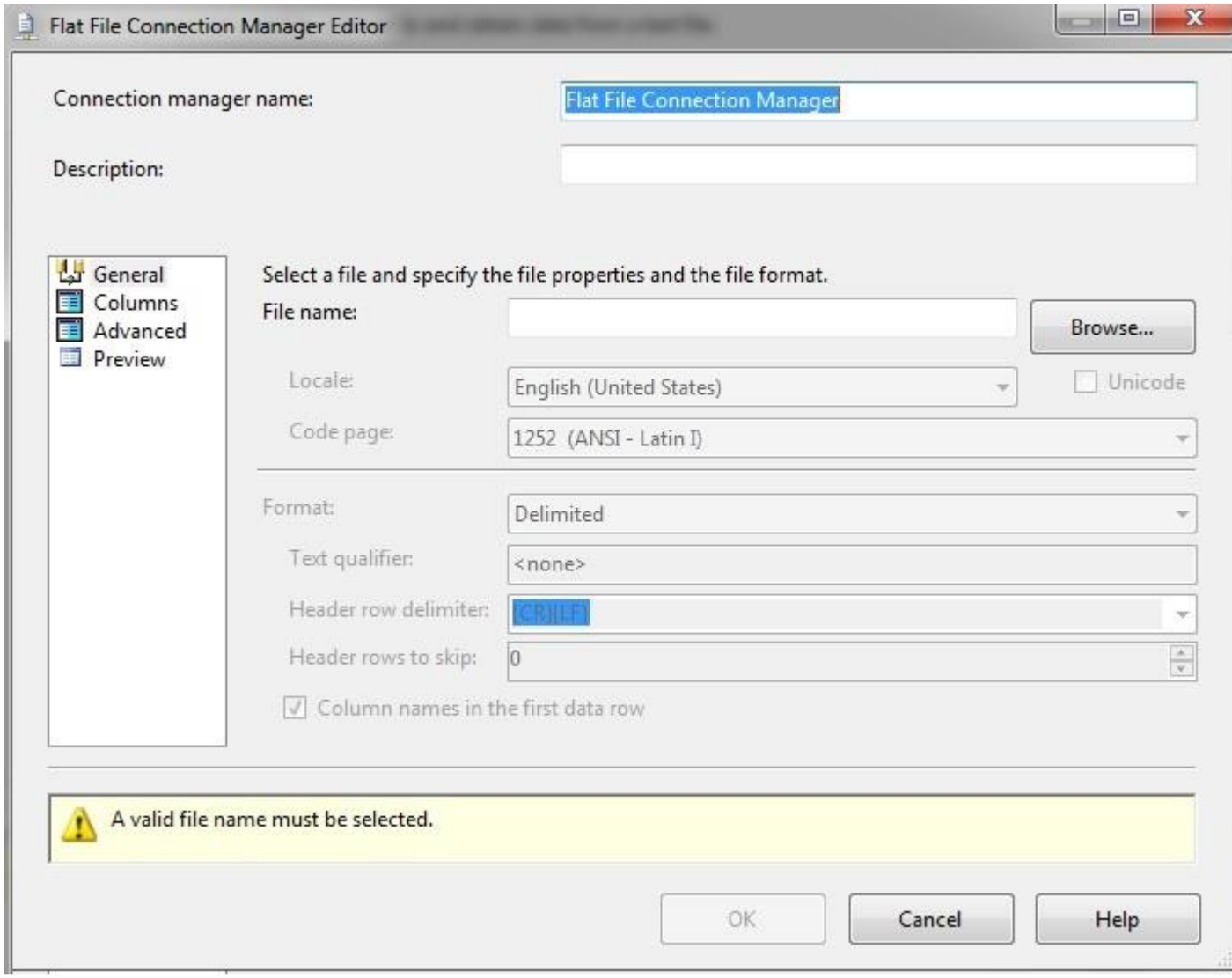
At this point there are two ways to initiate the setup of the flat file connection. The first way is to right click in the Connection Manager window and select "New Flat File Connection..." as shown below.



The second way is to double click on the Flat File Source to bring up the Flat File Source Editor and then click on "New...".



Either of the above ways will initiate the Flat File Connection Manager Editor.



Complete the Flat File Connection Manager Editor:

For the "Connection manager name" on the "General" tab, I typically will use the actual file name or the type of file if there are multiple files of the same layout.

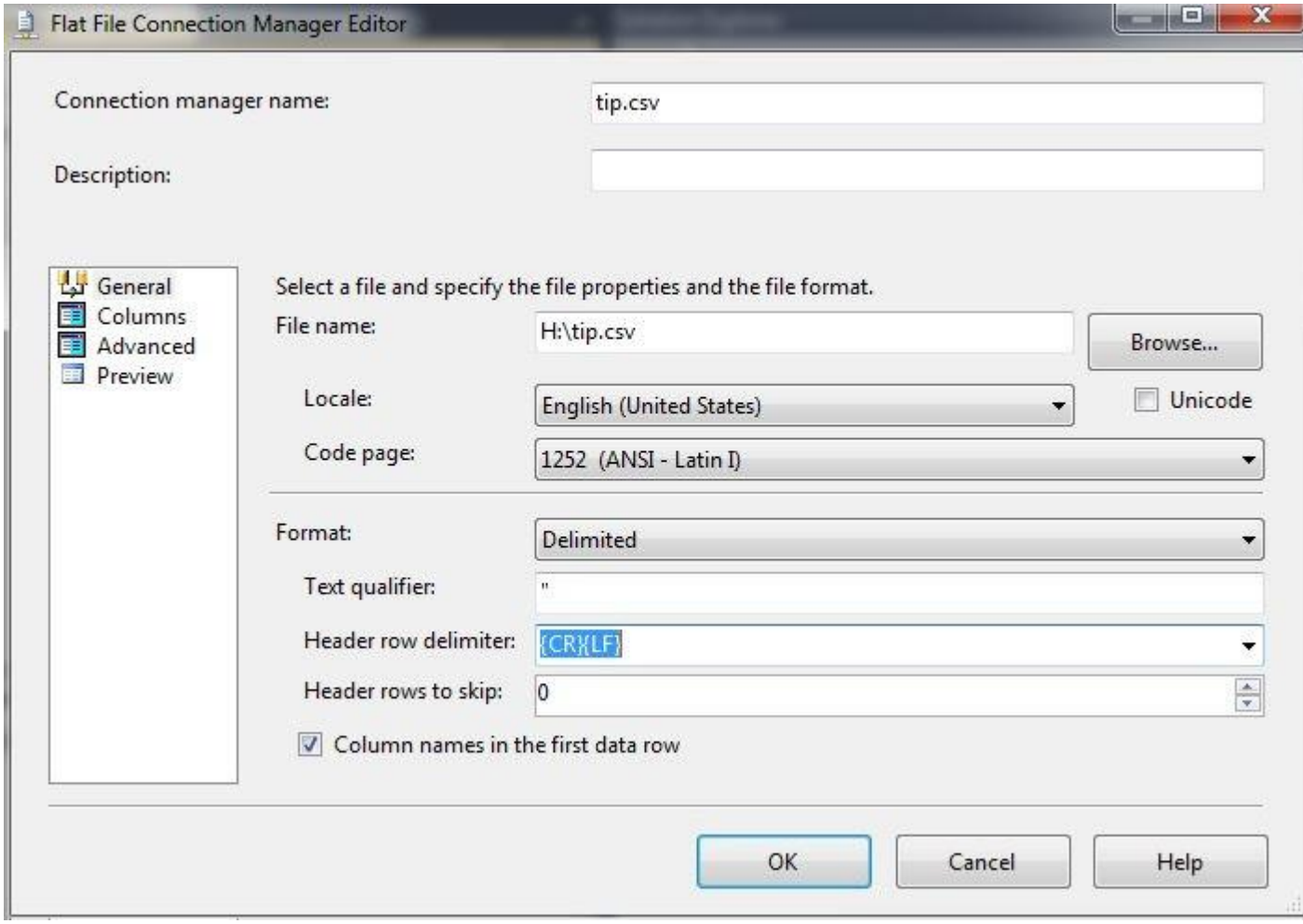
Just remember that what is entered for the name will be displayed in the Connection Manager window.

For the "File name", click on "Browse..." to navigate to the file or enter the file name manually. When using the browse feature, there are predefined filters for *.txt and *.csv files.

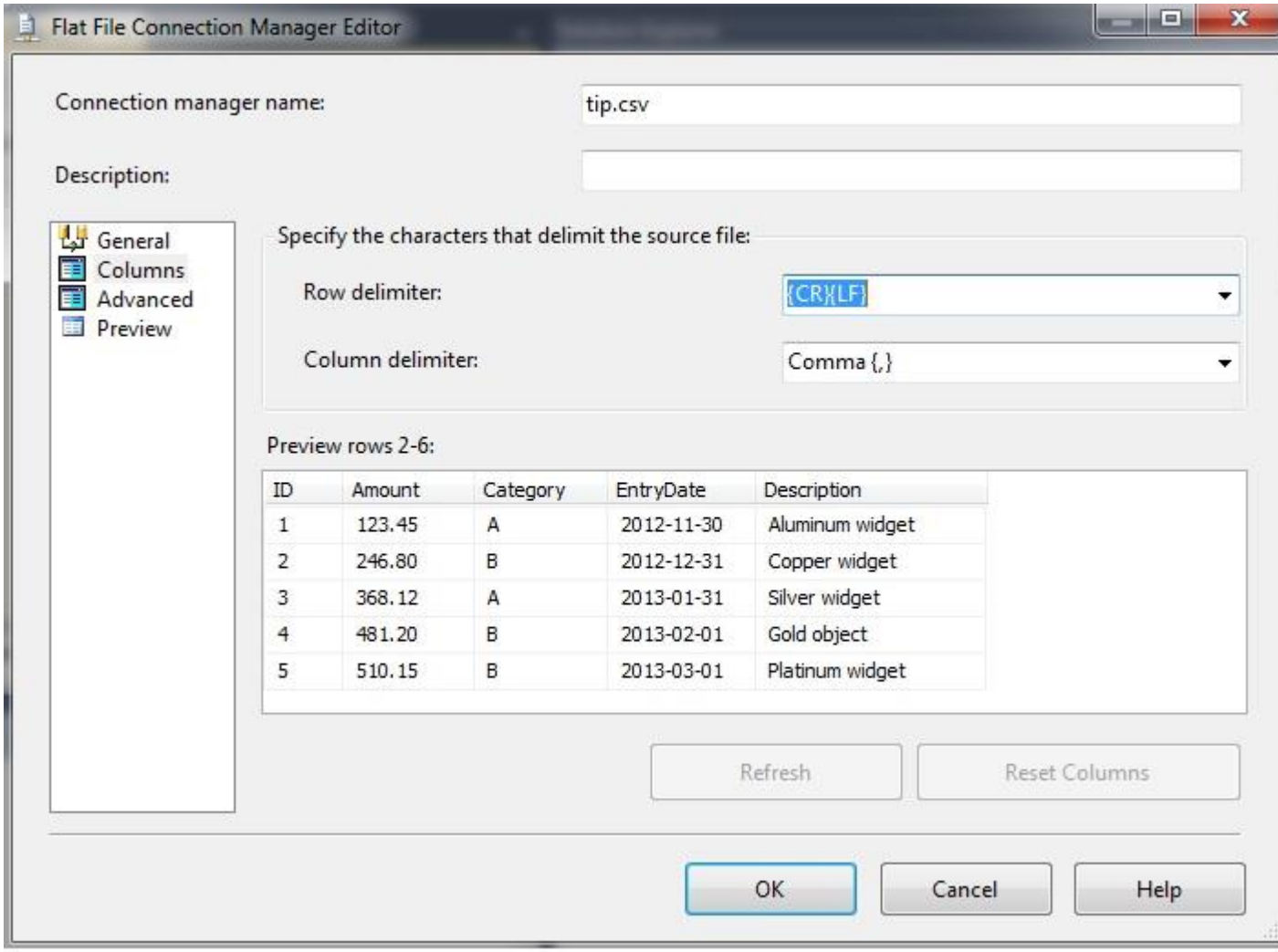
Because the Description column in our CSV file uses double quotes to qualify text strings, we must place a double quote in the "Text qualifier:" box.

For our file, the "Header row delimiter:" is the default of {CR}{LF} (carriage return/line feed).

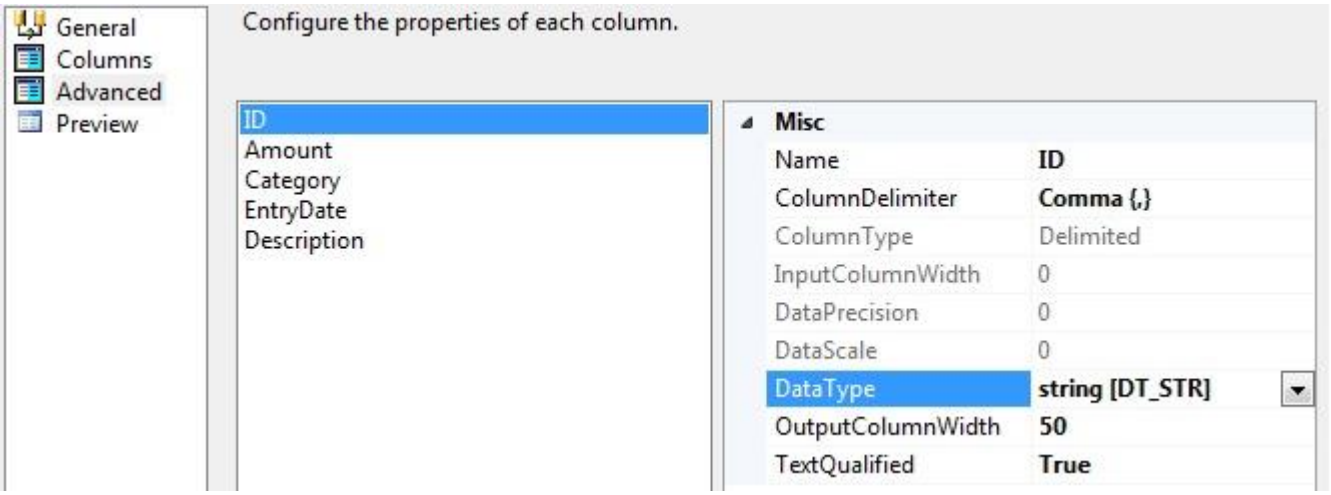
"Header rows to skip:" also remains at the default of 0 because we only have one header row in this example and we make sure that "Column names in the first data row" is checked.



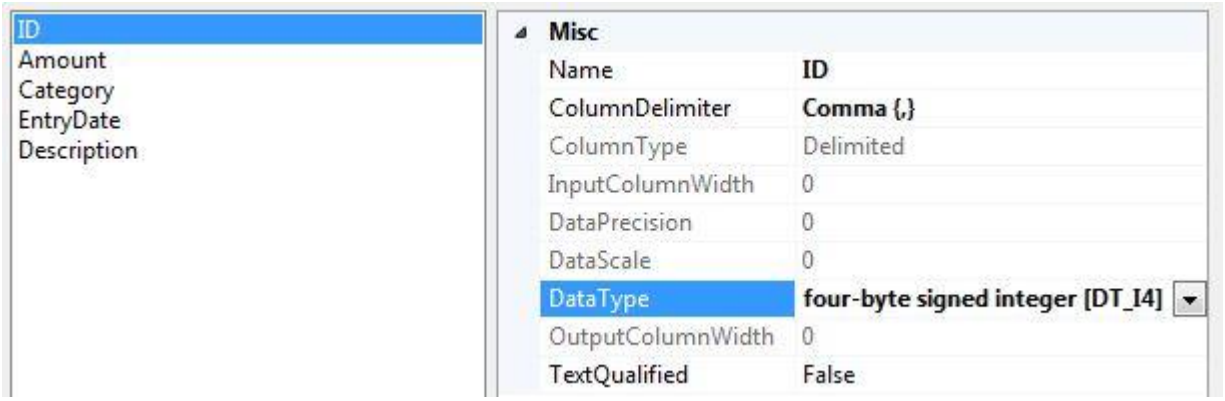
Now, we click on "Columns" tab to review the Columns page. We leave the "Row delimiter" set to the default of {CR}{LF} (carriage return/line feed). We leave the "Column delimiter" set to the default of Comma {,}. The preview section on this page allows us to see that SSIS is reading the file properly according to the format. This preview page will also allow us to make changes as necessary to accommodate the file's format.



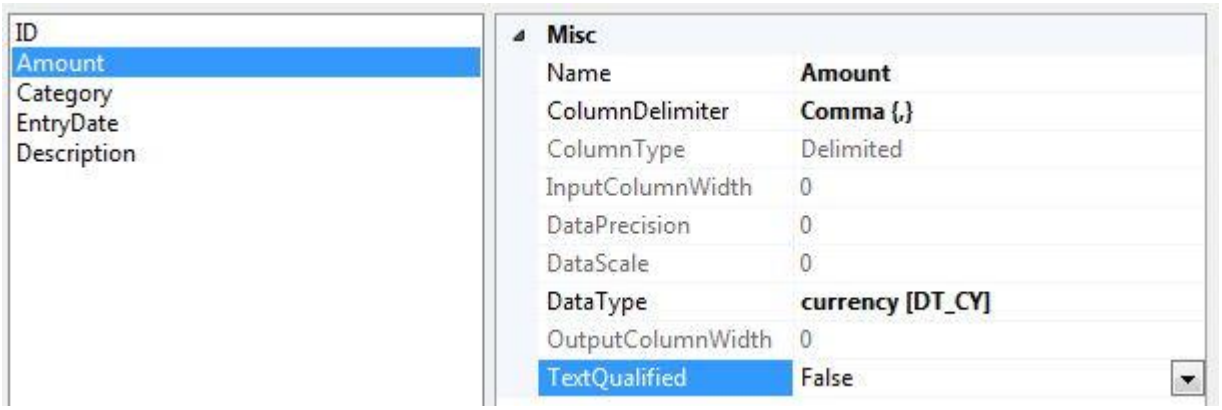
Next, we click on "Advanced" tab to review the Advanced page. By default, SSIS sets the "DataType" for each column to string [DT_STR], "OutputColumnWidth" to 50, and "TextQualified" to True. Setting each column to the proper data type as it is read in from its source eliminates the need to convert the data type downstream.



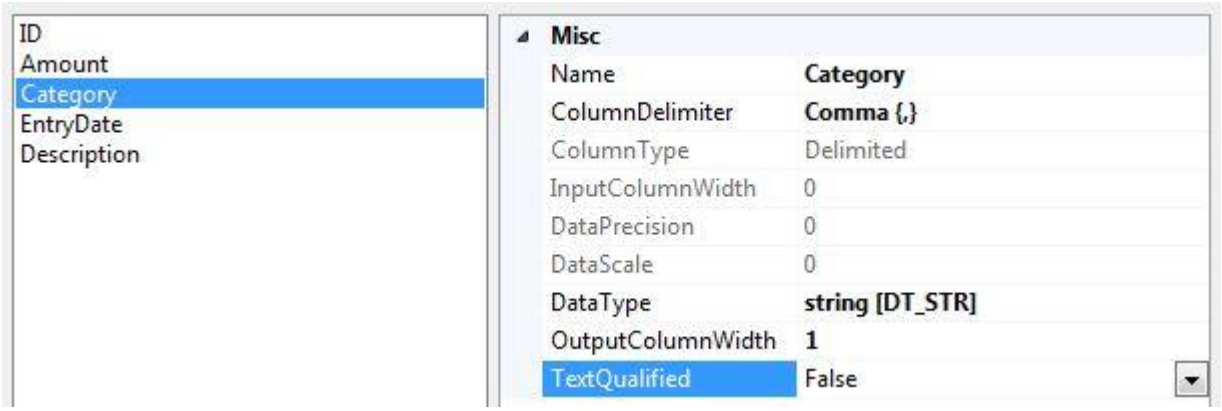
We change the data type of the ID column to a four-byte signed integer [DT_I4] and "TextQualified" to False.



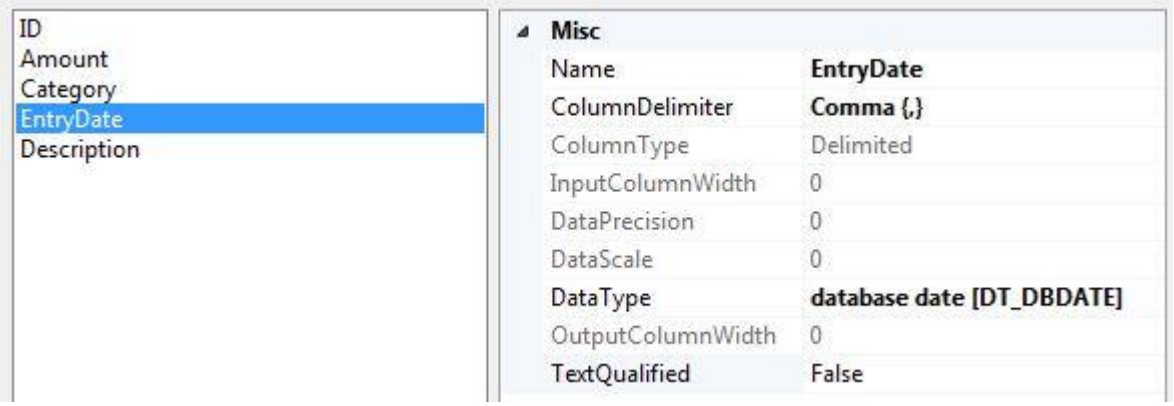
We change the data type of the Amount column to currency [DT_CY] and "TextQualified" to False.



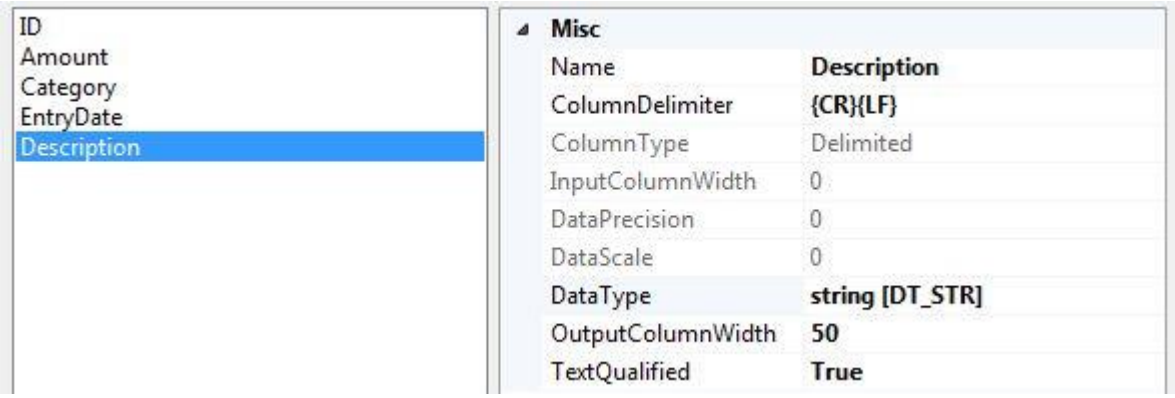
We change the length of the Category column to 1 and "TextQualified" to False.



We change the data type of the EntryDate column to database date [DT_DBDATE] and "TextQualified" to False.

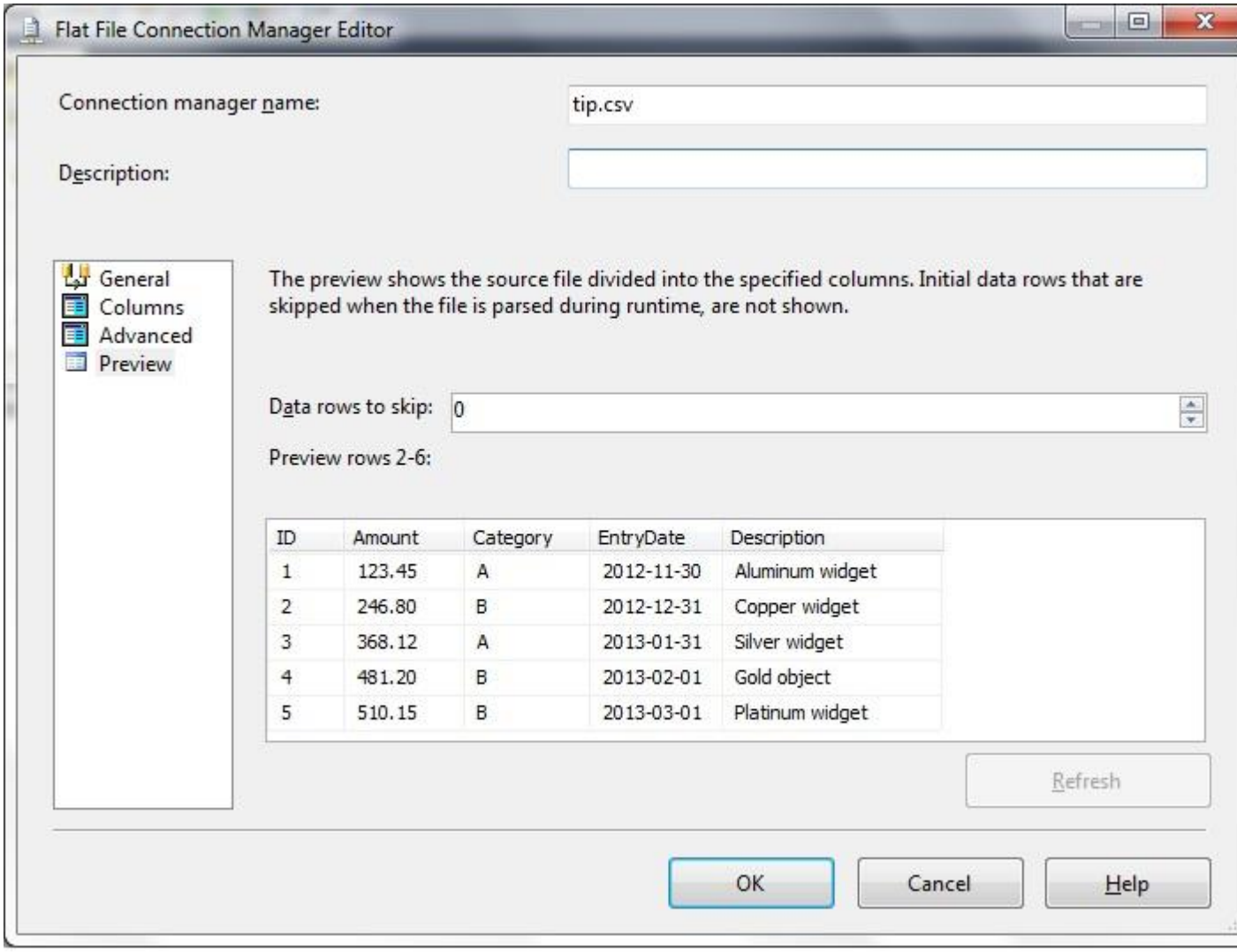


We leave the attributes of the Description column unchanged because we want the DataType to be string [DT_STR], the OutputColumnWidth to be 50, and TextQualified to be True.

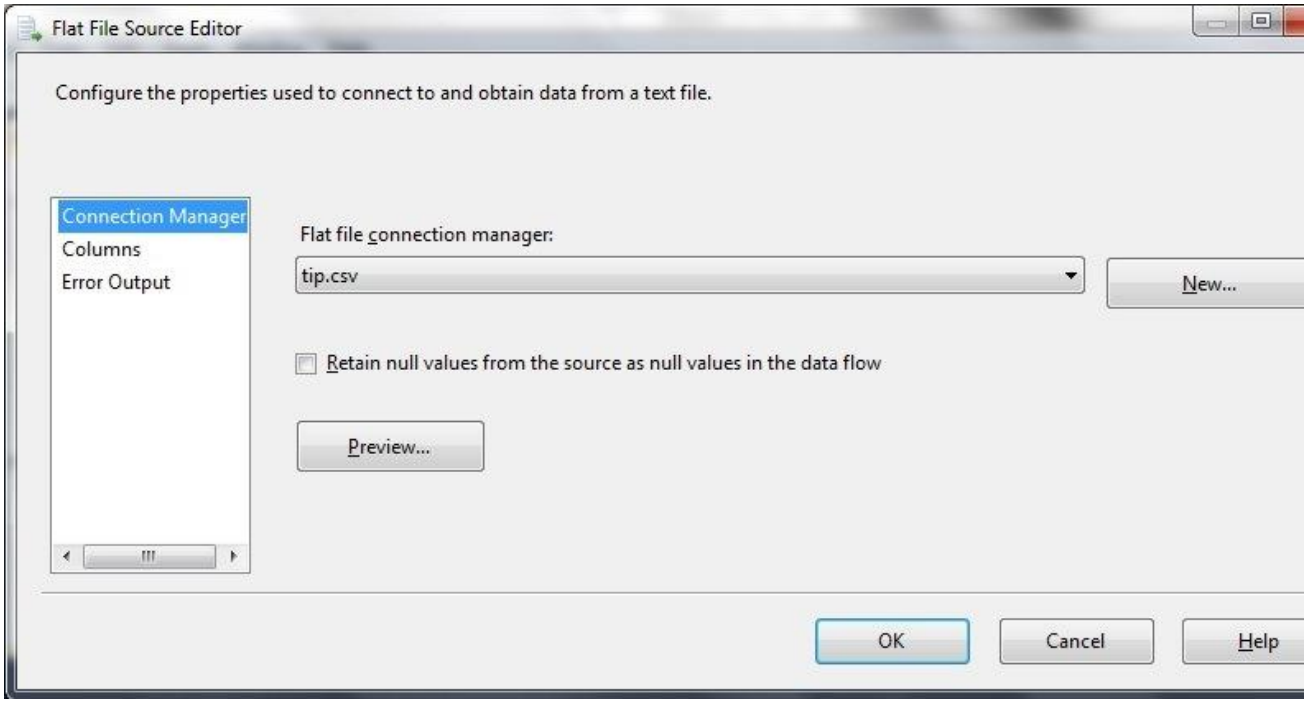


Click on the "Preview" tab to review another preview page. This preview page differs from the one shown previously because the user has the capability to skip a specified number of rows to look into their file the way the SSIS Flat File Source views it.

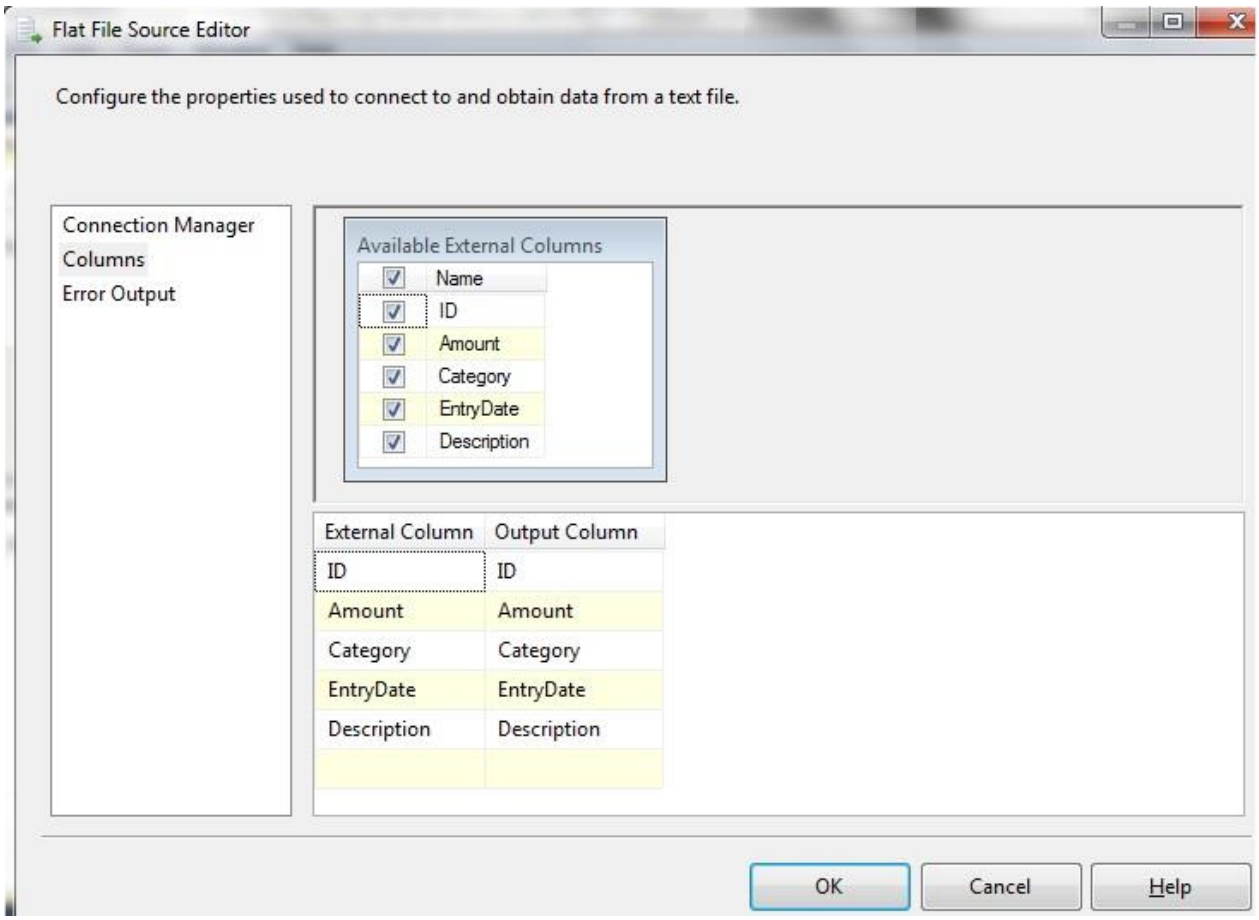
Click on OK when you are done.



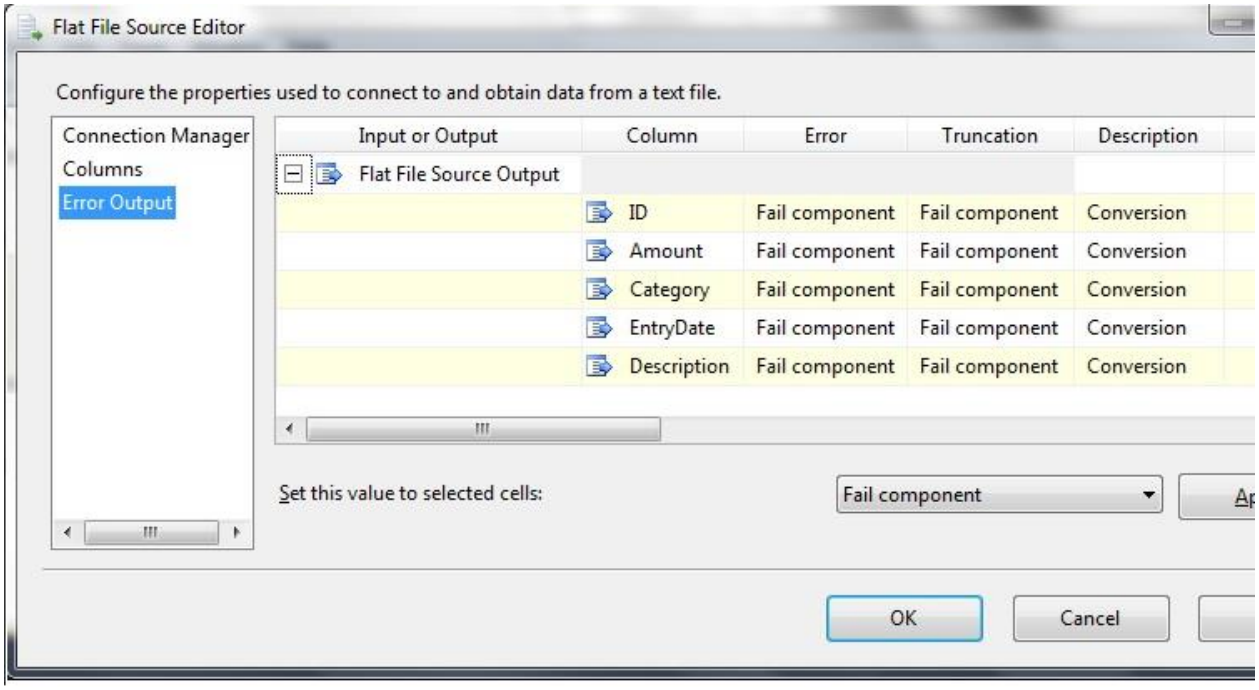
On the "Connection Manager" page of the Flat File Source Editor, make sure the newly created Flat File connection is selected.



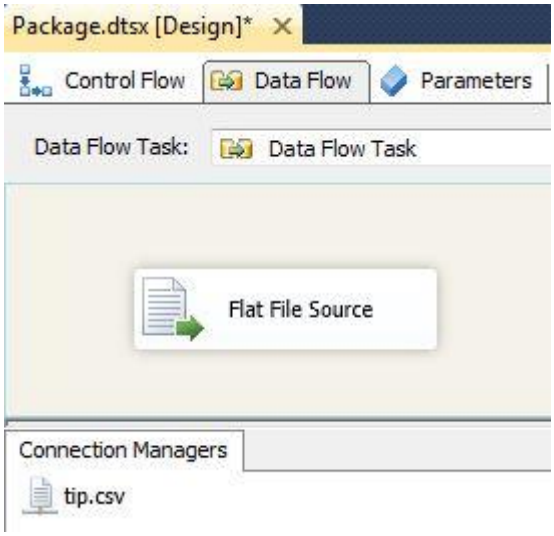
On the "Columns" page of the Flat File Source Editor, the columns to be output from the Flat File Source are displayed.



On the "Error Output" page of the Flat File Source Editor, we will leave the default values to have SSIS fail the Flat File Source component on truncation or an error. Click the OK button when you are finished.



When setup correctly, the SSIS window Data Flow window should appear as shown below.



Next Steps

- After creating the Flat File Source, try connecting it to a Flat File or OLE DB Destination.

Problem
When I save a **Webi** report with data as a CSV file, all of the leading zeros are cut off any numbers when I open the file in **Excel**. How can I keep the zeros?

This is actually an **Excel** issue. The program automatically truncates all leading zeros from numbers in CSV files. **The key is to change at least the columns where the leading zeros occur (i.e. ORG or Fund numbers) to "text."** There are several options to do this. Start with the report open:

Option A (preferred option, most user control)

1. Click on **Save to my computer as**
2. Select either **CSV** or **CSV (with options)**
3. **Click Save -- DO NOT OPEN THE CSV FILE DIRECTLY WITH EXCEL!**
4. Open a new worksheet in **Excel** (*see below for Excel screenshots.*)
5. Open the **Data tab**
6. Click on the **From text** button in the **Get External Data** section
7. Select your CSV file to import
8. Select the **"Delimited"** radio button -- **Text Import Wizard, Step 1** determines that your data is delimited
9. Click **Next**
10. Check **"Comma"** as a delimiter (column dividers will appear in preview)-- **Step 2** lets you set delimiters
11. Click **Next**
12. Highlight the column(s) with leading zeros in **Step 3**
13. Mark those columns format as **"text"** by clicking the radio button in the **Column Data Format** section. NOTE: You will need to do this for each column where the data contains leading zeros.
14. Click **Finish**
15. The leading zeros will still be there in the new worksheet with the imported data. The columns with real numbers will still be able to be used with calculations.

==========

Option B

1. Click on **Save to my computer as**
2. Select either **CSV** or **CSV (with options)**
3. Click **Save -- DO NOT OPEN THE CSV FILE DIRECTLY WITH EXCEL!**
4. Change the file extension from *.csv to *.txt.
5. Follow the process #4 through #14 described above.

Actually, there are two things that need to be checked here. First, is Excel putting the leading zeros in the CSV file it initially creates? Second, is it maintaining the zeros in the CSV file when you reload it and then resave it? These are two separate issues.

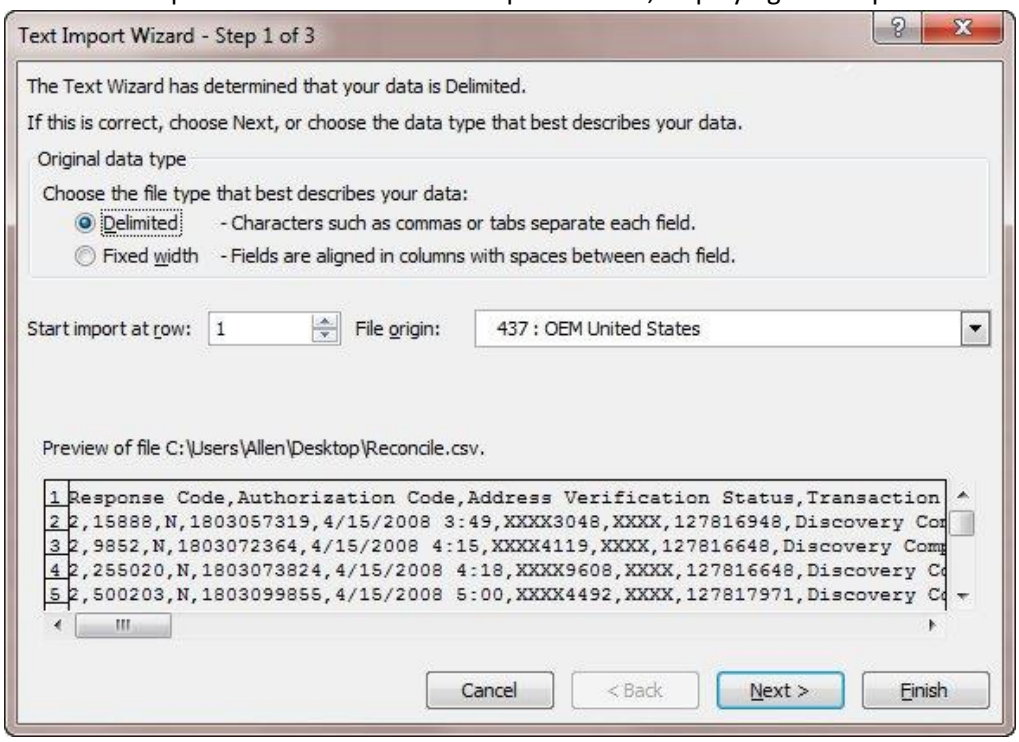
You can check the first issue easily enough. All you need to do is rename the CSV file so it has a TXT extension, then you can load it into a text editor, such as Notepad. There you can examine the actual CSV file, as created by Excel, to make sure that everything is in the format you expect. If it is not—for instance, there are no leading zeros where you need them—then you need to be concerned with how Excel is creating the CSV file in the first place.

You need to check whether there are leading zeros in the original Excel information. If there are, and they are displayed, then you need to make sure that the column in which the data is contained is formatted as Text in the Number tab of the Format Cells dialog box. If they are not, then you need to format the cells using a Custom number format that displays the zeros. In both of these cases, the leading zeros will be included in the CSV file created by Excel.

This brings us to the second issue. When you load a CSV file into Excel, it tries to determine the format of the data being loaded. You probably noticed when you loaded your CSV file in Notepad that even though Excel includes leading zeros in the output file, there are no quotes around the field itself. This means that Excel automatically recognizes the field as a number when importing it. By default, then, the number is displayed using one of the number fields, thereby expunging any leading zeros in what Excel displays.

The way around this problem should be fairly obvious based on information earlier in this tip—somehow you need to get Excel to recognize the incoming information as text so that it treats the leading zeros as significant. The quickest way to do this is to follow these steps, prior to loading the CSV file:

1. Make sure the CSV file is renamed so it has a TXT extension. You must perform this step, or the rest of the steps will not work because Excel won't start the Text Import Wizard in step 5.
2. Display the Open dialog box. (In Excel 2007, click the Office button and then click Open. In Excel 2010, click the File tab of the ribbon and then click Open.)
3. Using the Files of Type drop-down list at the bottom of the dialog box, indicate that you want to open Text Files (*.prn; *.txt; *.csv).
4. Select the file you renamed in step 1.
5. Click on Open. Excel starts the Text Import Wizard, displaying the Step 1 of 3 dialog box.



6. Make sure the Delimited choice is selected, then click on Next. Excel displays the Step 2 of 3 dialog box.
7. Make sure Comma is selected as a delimiter, then click on Next. Excel displays the Step 3 of 3 dialog box. The interesting thing is that the data in your TXT file should be displayed at the bottom of the dialog box, including any leading zeros in your fields.
8. At the bottom of the dialog box, click on the field that has leading zeros. The entire column should now be selected.
9. In the Column Data Format area, make sure the Text radio button is selected.
10. Repeat steps 8 and 9 for any other fields that have leading zeros.
11. Click on Finish. Your file is imported, with leading zeros still intact.

Now you can do your work in Excel, as desired, and again save your data in CSV format. (You will, however, need to use Save As rather than simply using Save.) The leading zeros will be included in the data that is saved.

Why does Excel treat long numeric strings as scientific notation even after changing cell format to text

1. Take a long number like 1240800388917 and paste it in a cell in a new worksheet.
2. Excel's default cell format is general, so the string is presented in scientific notation as 1.2408E+12

This occurs at point of entry, so once it's in, any additional detail is lost, and so even if you tell Excel that you really meant that was text, and not a number, you're kind of out of luck, and hence why your first sequence doesn't work.

A single apostrophe ' before a number will force Excel to treat a number as text (including a default left align). And if you have errors flagged, it will show as a number stored as text error on the cell. In Office 2010, if you format the column first, and then paste in the data, it will show the large numbers correctly.

This worked for me in Excel 2010. Just select the column, right click, Format cells, Custom and choose the option that says 0 (second option below General).