UNIT - 3 :Creating ETL Solutions with SSIS, Implementing Control Flow in SSIS

Introduction to ETL with SSIS

- **ETL** is short for ***e**xtract*, ***t**ransform*, ***and load***, three database functions that are combined into one tool to pull data out of one database and place it into another database.
- **Extract** is the process of *reading data* from a database. In this stage, the data is collected, often from multiple and different types of sources.
- **Transform** is the process of *converting the extracted data* from its previous form into the form it needs to be in so that it can be placed into another database. Transformation occurs by using rules or lookup tables or by combining the data with other data.
- **Load** is the process of *writing the data* into the target database.

How it Works

Data from one or more sources is extracted and then copied to the data warehouse. When dealing with large volumes of data and multiple source systems, the data is consolidated. ETL is used to migrate data from one database to another, and is often the specific process required to load data to and from data marts and data warehouses, but is a process that is also used to to large convert (transform) databases from one format or type to another.

Microsoft SSIS (SQL Server Integration Services) is an enterprise data integration, data transformation and data migration tool built into Microsoft's SQL Server database. It can be used for a variety of integration-related tasks, such as analyzing and cleansing data and running extract, transform and load, or ETL, processes to update data warehouses.

SSIS can extract, transform and consolidate data from multiple relational databases, as well as sources such as XML data files and flat files, then load the processed information into an enterprise data warehouse or other target systems. It includes a set of tools for developing and testing integration programs, called SQL Server Data Tools, plus a server component for deploying and running the programs. In addition, Microsoft's SQL Server Management Studio software is incorporated into SSIS for managing and monitoring integration routines.

SSIS history

Microsoft SSIS was introduced with SQL Server 2005 as a replacement for an earlier integration tool called Data

Transformation Services (DTS). Before the release of DTS with SQL Server 7.0 in 1998, database administrators either wrote custom data transformation tools or used third-party tools to transfer data.

Microsoft has added a number of features to SSIS over the years, including graphical tools and wizards, which allow users to build and debug packages; workflow functionality, such as file transfer protocol operations; the ability to execute SQL statements; the ability to email messages; data sources and destinations for ETL; transformations for collecting, cleansing, merging, and copying data; a management service; the ability to administer package execution and storage; and application programming interfaces (APIs) for SSIS object models. Among the most popular features are the data import/export wizard and packaged data source connectors.

Exploring Data Sources

You can use the **Explore Data** dialog box in Data Source View Designer in SQL Server Data Tools (SSDT) to browse data for a table, view, or named query in a data source view (DSV). When you explore the data in Data Source View Designer, you can view the contents of each column of data in a selected table, view, or named query. Viewing the actual contents assists you in determining whether all columns are needed, if named calculations are required to increase user friendliness and usability, and whether existing named calculations or named queries return the anticipated values.

To view data, you must have an active connection to the data source or sources for the selected object in the DSV. Any named calculations in a table are also sent in the query.

The data returned in a tabular format that you can sort and copy. Click on the column headers to re-sort the rows by that column. You can also highlight data in the grid and press Ctrl-C to copy the selection to the clipboard.

You can also control the sample method and the sample count. By default, the top 5000 rows are returned.

To browse data or change sampling options

1. In SQL Server Data Tools (SSDT), open the project or connect to the database that contains the data source view in which you want to browse data.
2. In Solution Explorer, expand the **Data Source Views** folder, and then double-click the data source view.
3. Right-click the table, view, or named query that contains the data you want to view, and then click **Explore Data**.

The data source underlying the table, view, or named query in the data source view are queries, and the results appear in the **Explore <object name> Table** tab.

4. On the **Explore <object name> Table** toolbar, click the **Sampling options** icon.

   The **Data Exploration Options** dialog box opens. In this dialog box you can specify the sampling method (fewer or more records than the default sampling size of 5000 rows), or sample count.
5. Click **OK** or **Cancel** as appropriate.
6. To resample the data, click **Resample Data** on the **Explore <object name> Table** toolbar.

Implementing data flow using SSIS

Our goal in creating this package is to move data from a SQL Server database to an **Excel** file. As part of that goal, we also want to insert an additional column into the **Excel** file that's based on derived data.

To carry out our goal, we must add a **Data Flow** task to our control flow. The task lets us retrieve data from our data source, transform that data, and insert it into our destination, the **Excel** file. The **Data Flow** task is one of the most important and powerful components in SSIS and as such has it's own workspace, which is represented by the **Data Flow** tab in **SSIS Designer**, as shown in Figure 1.

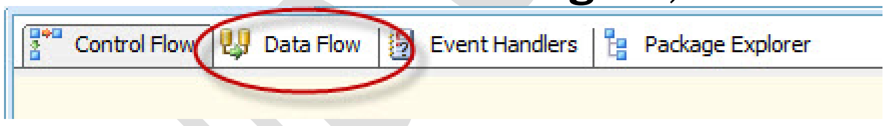


Figure 1: The Data Flow tab in SSIS Designer

Before we can do anything on the **Data Flow** tab, we must first add a **Data Flow** task to our control flow. To add the task, drag it from the **Control Flow Items** window to the Control Flow tab of the **SSIS Designer** screen, as illustrated in Figure 2.

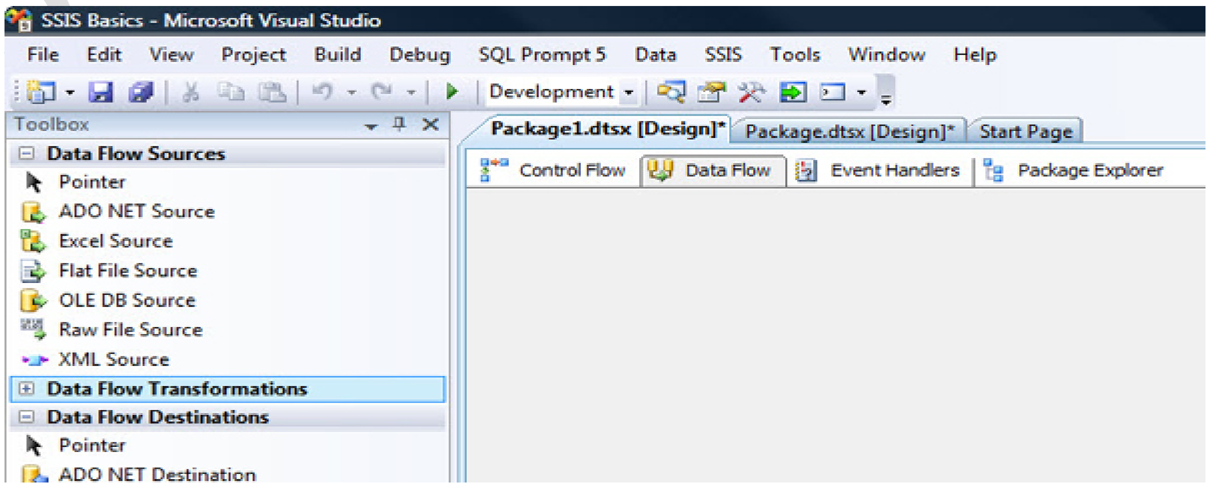
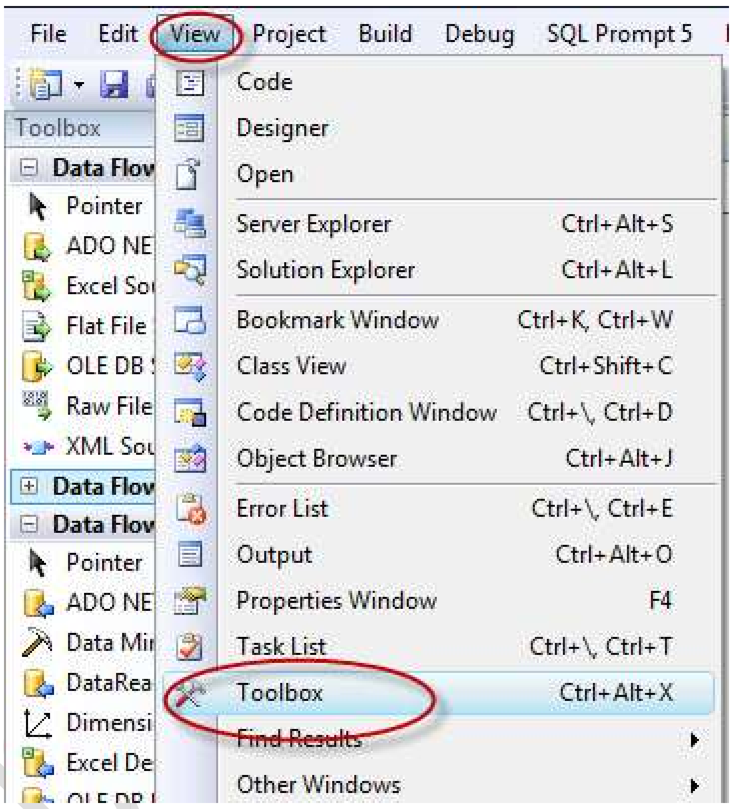


Figure 3: The Data Flow tab in SSIS Designer

Configuring the Data Flow

You configure a **Data Flow** task by adding components to the **Data Flow** tab. SSIS supports three types of data flow components:

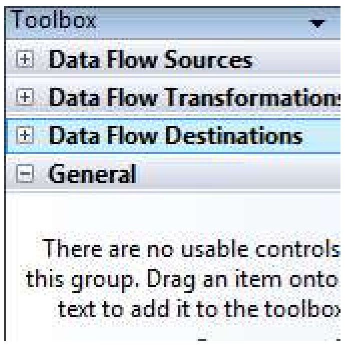
- **Sources:** Where the data comes from
- **Transformations:** How you can modify the data
- **Destinations:** Where you want to put the data



A **Data Flow** task will always start with a source and will usually end with a destination, but not always. You can also add as many transformations as necessary to prepare the data for the destination. For example, you can use the **Derived Column** transformation to add a computed column to the data flow, or you can use a **Conditional Split** transformation to split data into different destinations based on specified criteria. This and other components will be explained in future articles.

To add components to the **Data Flow** task, you need to open the **Toolbox** if it's not already open. To do this, point to the **View** menu and then click ToolBox, as shown in Figure 4.

Figure 4: Opening the Toolbox to view the data flow components


Toolbox
Data Flow Sources
Data Flow Transformation
Data Flow Destinations
General
There are no usable controls
this group. Drag an item onto
text to add it to the toolbo

At the left side of the **Data Flow** tab, you should now find the **Toolbox** window, which lists the various components you can add to your data flow. The **Toolbox** organizes the components according to their function, as shown in Figure 5.

Figure 5: The component categories as they appear in the Toolbox

To view the actual components, you must expand the categories. For example, to view the source components, you must expand the **Data Flow Sources** category, as shown in Figure 6

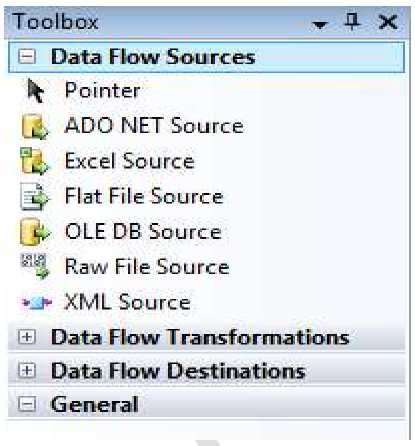
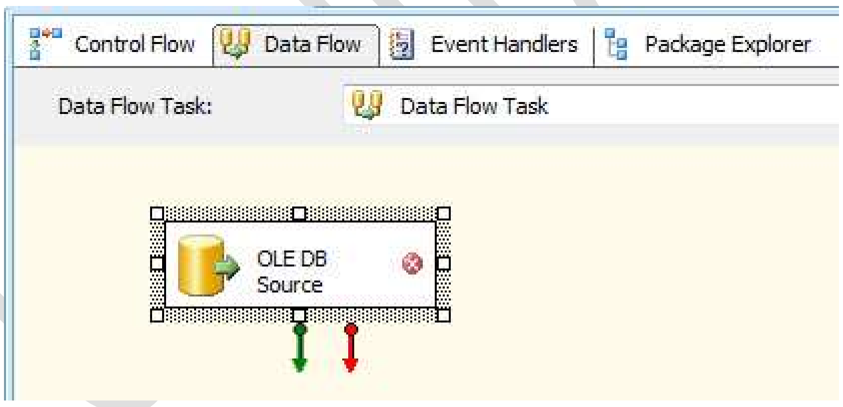


Figure 6: Viewing the data flow source components

Adding an OLE DB Source



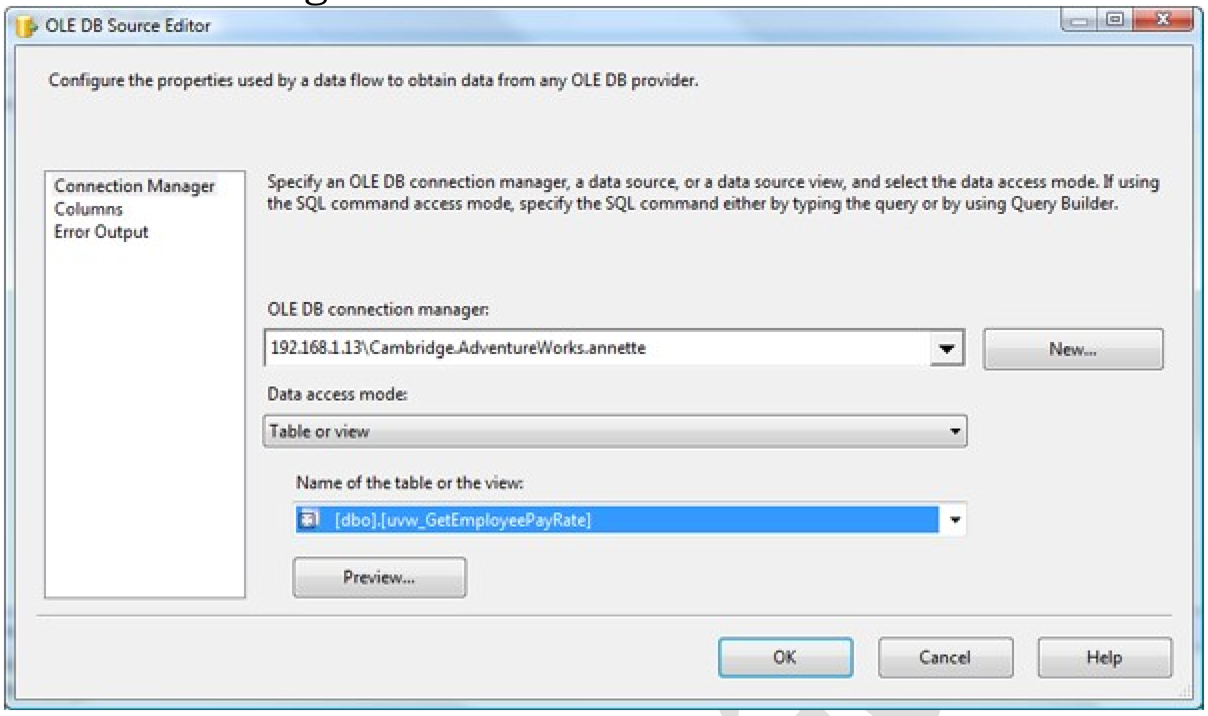
The first component we're going to add to the data flow is a source. Because we're going to be retrieving data from a SQL Server database, we'll use an **OLE DB** source. To add the component, expand the **Data Flow Sources** category in the **Toolbox**. Then drag an **OLE DB** source from to the **Data Flow** window. Your data flow should now look similar to Figure 7.

Figure 7: Adding an OLE DB source to your data flow

You will see that we have a new item named **OLE DB** Source. You can rename the component by right-clicking it and selecting rename. For this example, I renamed it **Employees**. There are several other features about the **OLE DB** source noting:

- A database icon is associated with that source type. Other source types will show different icons.
- A reversed red X appears to the right of the name. This indicates that the component has not yet been properly configured.
- Two arrows extend below the component. These are called *data paths*. In this case, there is one green and one red. The green data path marks the flow of data that has no errors. The red data path redirects rows whose values are

truncated or that generate an error. Together these data paths enable the developer to specifically control the flow of data, even if errors are present.
To configure the **OLE DB** source, right-click the component and then click **Edit**. The **OLE DB** Source **Edit**or appears, as shown in Figure 8.



OLE DB Source Editor

Configure the properties used by a data flow to obtain data from any OLE DB provider.

Connection Manager
Columns
Error Output

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager: 192.168.1.13\Cambridge.AdventureWorks.annette

Data access mode: Table or view

Name of the table or the view: [dbo].[uvw_GetEmployeePayRate]

Figure 8: Configuring the OLEDB source

From the **OLE DB connection manager** drop-down list, select the **OLE DB** connection manager we set up in the last article, the one that connects to the **AdventureWorks** database.
Next, you must select one of the following four options from the **Data access mode** drop-down list:

- Table or view
- Table name or view name variable
- SQL command

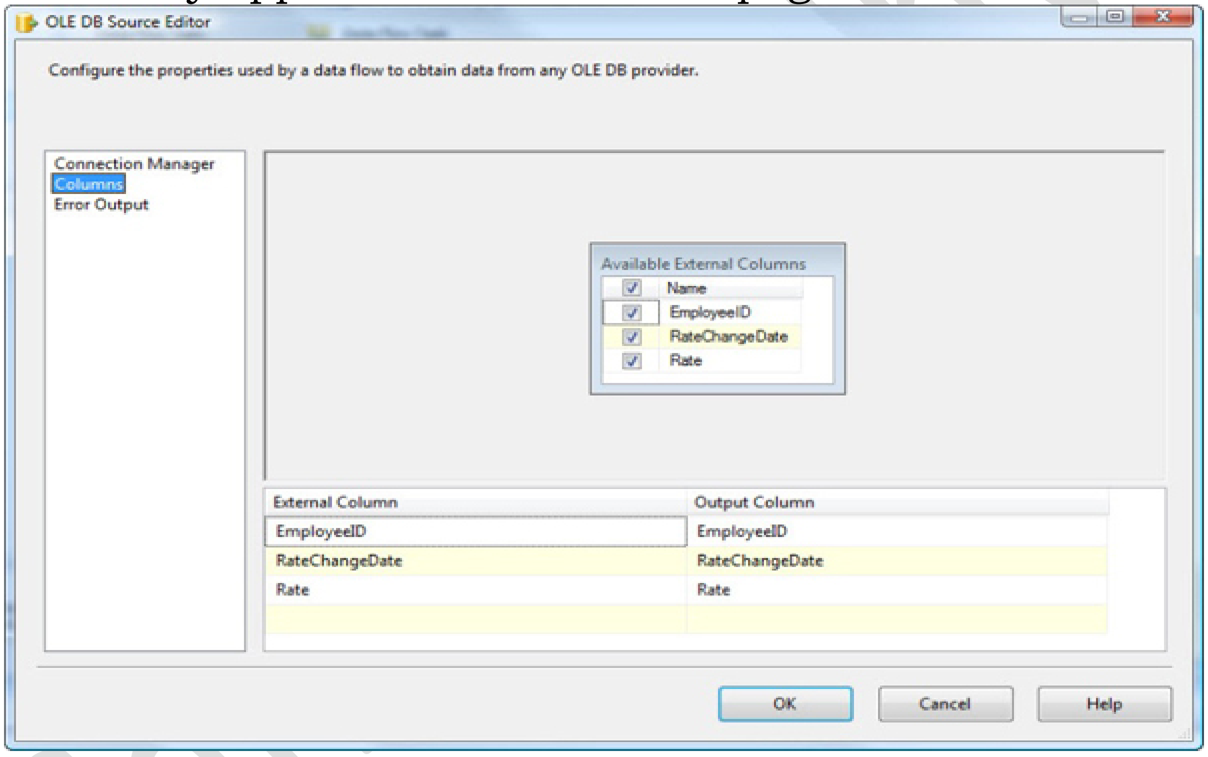
SQL command from variable
For this example, we'll select the **Table or View** option because we'll be retrieving our data through the **uvw_GetEmployeePayRate** view, which returns the latest employee pay raise and the amount of that raise. Listing 1 shows the Transact-SQL used to create the view in the **AdventureWorks** database.

```
CREATE VIEW uvw_GetEmployeePayRate
AS
    SELECT  H.EmployeeID ,
            RateChangeDate ,
            Rate
    FROM    HumanResources.EmployeePayHistory H
            JOIN ( SELECT   EmployeeID ,
                            MAX(RateChangeDate) AS [MaxDate]
```

```
                    FROM     HumanResources.EmployeePayHi
        story
                    GROUP BY EmployeeID
                  ) xx ON H.EmployeeID = xx.EmployeeID
                        AND H.RateChangeDate = xx.MaxDate
   GO
```

Listing 1: The uvw_GetEmployeePayRate view definition

After you ensure that Table or view is selected in the **Data access mode** drop-down list, select the **uvw_GetEmployeePayRate** view from the **Name of the table or the view** drop-down list. Now go to the **Columns** page to select the columns that will be returned from the data source. By default, all columns are selected. Figure 9 shows the columns (**EmployeeID**, **RateChangeDate**, and **Rate**) that will be added to the data flow for our package, as they appear on the **Columns** page.



Configure the properties used by a data flow to obtain data from any OLE DB provider.

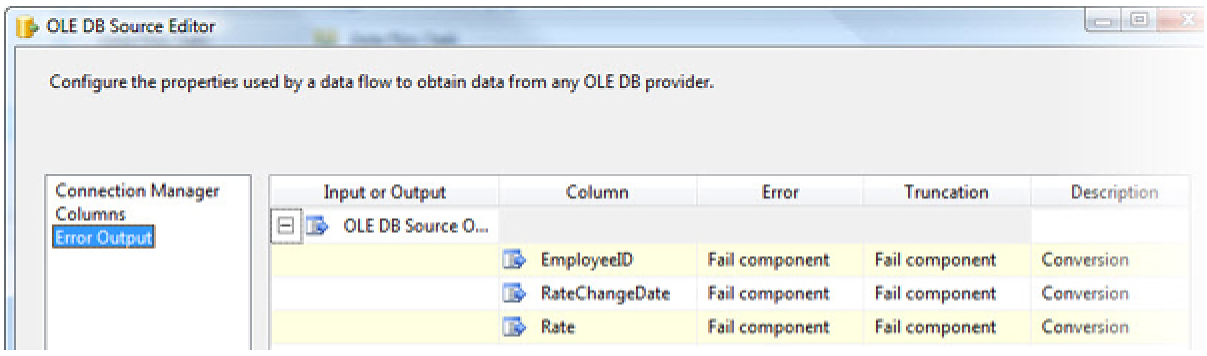
	Name
✓	EmployeeID
✓	RateChangeDate
✓	Rate

External Column	Output Column
EmployeeID	EmployeeID
RateChangeDate	RateChangeDate
Rate	Rate

Figure 9: The Columns page of the OLE DB Source Editor

If there are columns you don't wish to use, you can simply uncheck them in the **Available External Columns** box.

Now click on the **Error Output** page (shown in Figure 10) to view the actions that the SSIS package will take if it encounters errors.



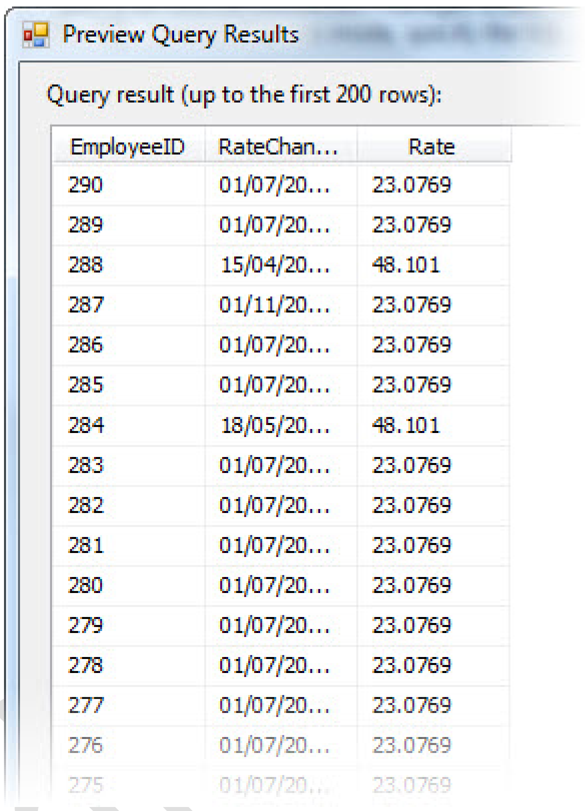
Configure the properties used by a data flow to obtain data from any OLE DB provider.

Input or Output	Column	Error	Truncation	Description
OLE DB Source O...				
	EmployeeID	Fail component	Fail component	Conversion
	RateChangeDate	Fail component	Fail component	Conversion
	Rate	Fail component	Fail component	Conversion

Figure 10: The Error Output page of the OLE DB Source Editor

By default, if there is an error or truncation, the component will fail. You can override the default behavior, but explaining how to do that is beyond the scope of this article. You'll learn about error handling in future articles.

Now return to the Connection Manager page and click the Preview button to view a sample dataset in the Preview Query Results window, shown in Figure 11. Previewing the data ensures that what is being returned is what you are expecting.



Query result (up to the first 200 rows):

EmployeeID	RateChan...	Rate
290	01/07/20...	23.0769
289	01/07/20...	23.0769
288	15/04/20...	48.101
287	01/11/20...	23.0769
286	01/07/20...	23.0769
285	01/07/20...	23.0769
284	18/05/20...	48.101
283	01/07/20...	23.0769
282	01/07/20...	23.0769
281	01/07/20...	23.0769
280	01/07/20...	23.0769
279	01/07/20...	23.0769
278	01/07/20...	23.0769
277	01/07/20...	23.0769
276	01/07/20...	23.0769
275	01/07/20...	23.0769

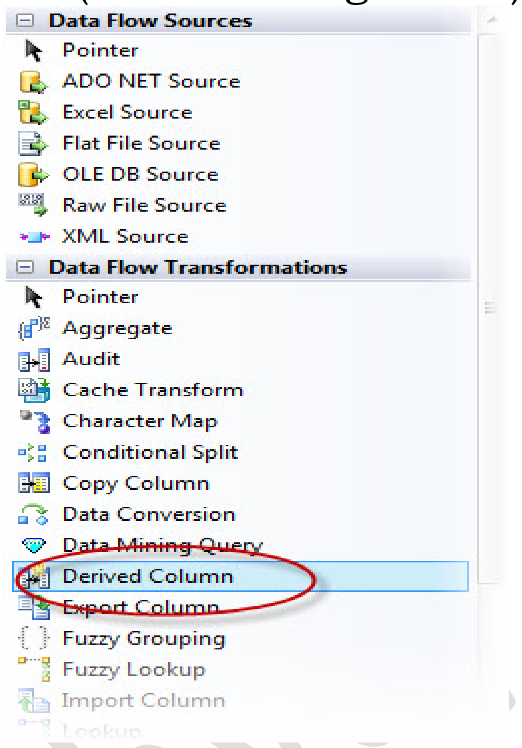
Figure 11: Previewing a sample dataset

After you've configured the **OLE DB** Source component, click **OK**.

Adding a Derived Column Transformation

The next step in configuring our data flow is to add a transformation component. In this case, we'll add the **Derived Column** transformation to create a column that calculates the annual pay increase for each employee record we retrieve through the **OLE DB** source.

To add the component, expand the **Data Flow Transformations** category in the **Toolbox** window, and drag the **Derived Column** transformation (shown in Figure 12) to



the **Data Flow** tab design surface.

Figure 12: The Derived Column transformation as its listed in the Toolbox

Drag the green data path from the **OLE DB** source to the **Derived Column** transformation to associate the two components, as shown in Figure 13. (If you don't connect the two components, they won't be linked and, as a result, you won't be able to edit the transformation.)

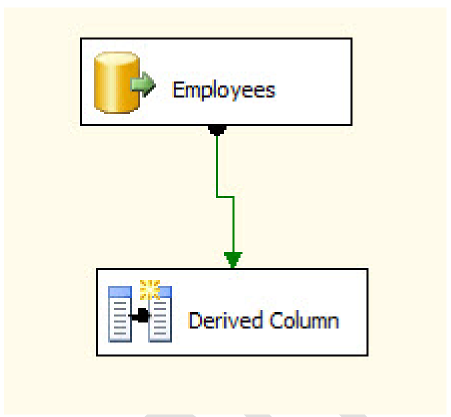


Figure 13: Using the data path to connect the two components

The next step is to configure the **Derived Column** component. Double-click the component to open the **Derived Column** Transformation **Edit**or, as shown in Figure 14.

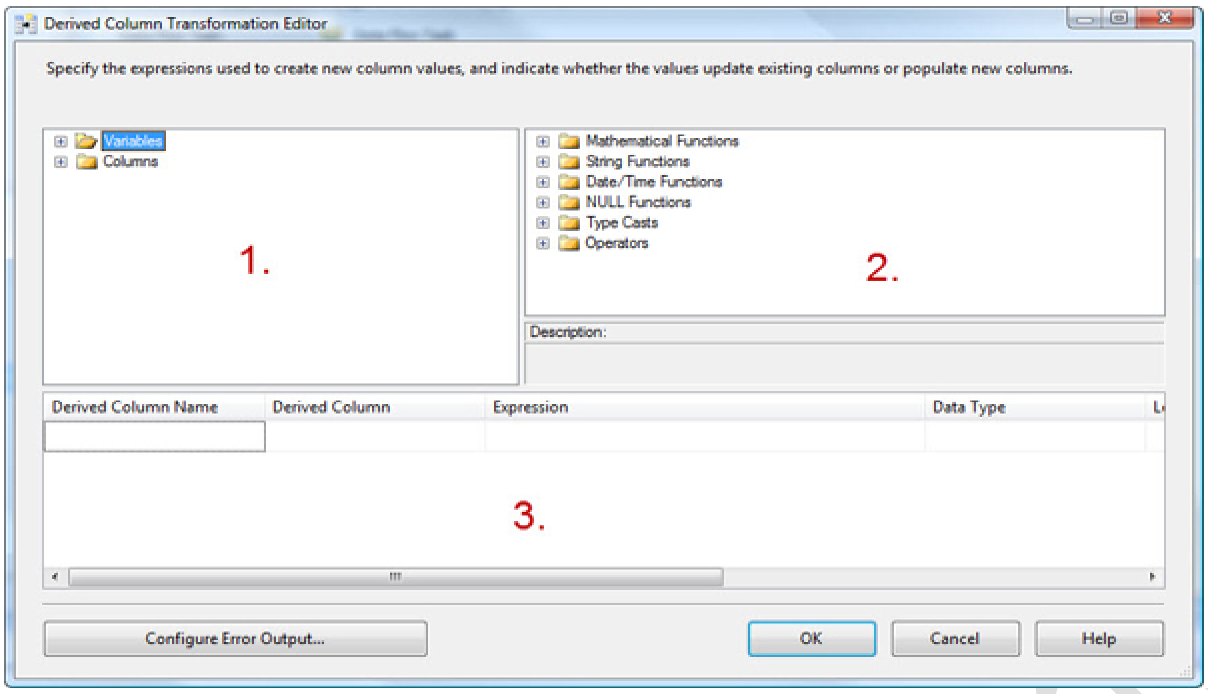

Derived Column Transformation Editor
Specify the expressions used to create new column values, and indicate whether the values update existing columns or populate new columns.
Variables
Columns
1.
Mathematical Functions
String Functions
Date/Time Functions
NULL Functions
Type Casts
Operators
2.
Description:
Derived Column Name
Derived Column
Expression
Data Type
3.
Configure Error Output...
OK
Cancel
Help

Figure 14: Configuring the Derived Column transformation

This editor is made up of three regions, which I've labeled **1**, **2** and **3**:

1. Objects you can use as a starting point. For example you can either select columns from your data flow or select a variable. (We will be working with variables in a future article.)
2. Functions and operators you can use in your derived column expression. For example, you can use a mathematical function to calculate data returned from a column or use a date/time function to extract the year from a selected date.
3. Workspace where you build one or more derived columns. Each row in the grid contains the details necessary to define a derived column.

For this exercise, we'll be creating a derived column that calculates a pay raise for employees. The first step is to select the existing column that will be the basis for our new column. To select the column, expand the **Columns** node, and drag the **Rate** column to **the Expression** column of the first row in the derived columns grid, as shown in Figure 15.

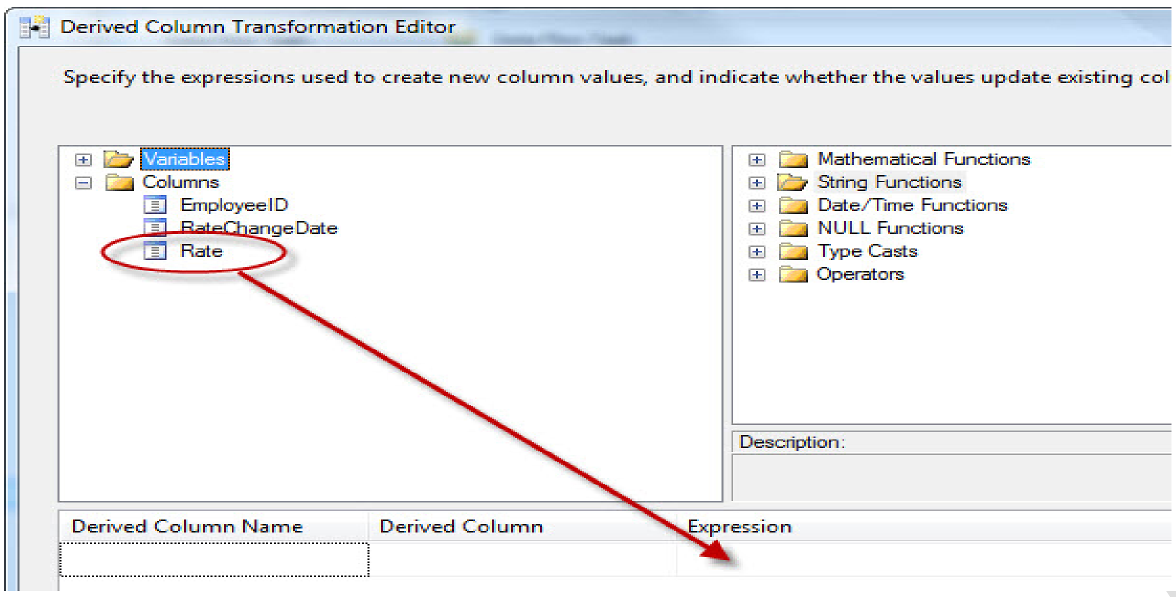
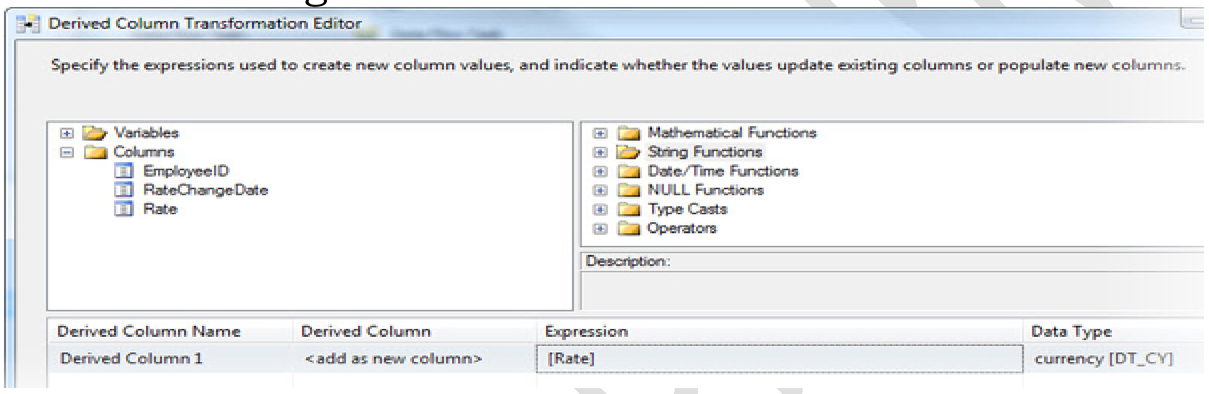


Figure 15: Adding a column to the Expression column of the derived column grid

When you add your column to the **Expression** column, SSIS prepopulates the other columns in that row of the grid, as shown in Figure 16.


Derived Column Transformation Editor
Specify the expressions used to create new column values, and indicate whether the values update existing columns or populate new columns.
Variables
Columns
EmployeeID
RateChangeDate
Rate
Mathematical Functions
String Functions
Date/Time Functions
NULL Functions
Type Casts
Operators
Description:

Derived Column Name	Derived Column	Expression	Data Type
Derived Column 1	<add as new column>	[Rate]	currency [DT_CY]

Figure 16: Prepopulated values in derived column grid

As you can see, SSIS has assigned our derived column the name Derived Column 1 and set the Derived Column value to <add as new column>. In addition, our [Rate] field now appears in the **Expression** column, and the **currency[DT_CY]** value has been assigned to the **Data Type** column.

You can change the **Derived Column** Name value by simply typing a new name in the box. For this example, I've renamed the column **NewPayRate**.

For the **Derived Column** value, you can choose to add a new column to your data flow (which is the default value, **<add as new column>**) or to replace one of the existing columns in your data flow. In this instance, we'll add a new column, but there may be times when overwriting a column is required.

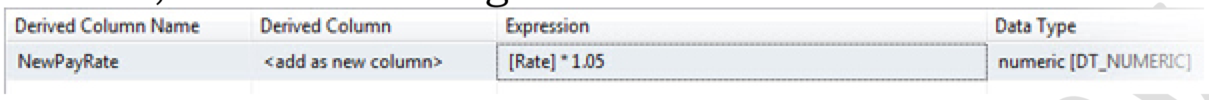
The data type is automatically created by the system and can't be changed at this stage.

Our next step is to refine our expression. Currently, because only the **Rate** column is included in the expression, the derived column will return the existing values in that column. However, we want to calculate a new pay rate. The first step, then, is to add an operator. To view the list of available operators, expand the list and scroll through them. Some of the operators are for string functions and some for math functions.

To increase the employee's pay rate by 5%, we'll use the following calculation:

[Rate] * 1.05

To do this in the **Expression** box, either type the multiplication operator (*), or drag it from the list of operators to our expression (just after the column name), and then type 1.05, as shown in Figure 17.



Derived Column Name	Derived Column	Expression	Data Type
NewPayRate	<add as new column>	[Rate] * 1.05	numeric [DT_NUMERIC]

Figure 17: Defining an expression for our derived column

You will see that the Data Type has now changed to numeric [DT_NUMERIC].

Once you are happy with the expression, click on OK to complete the process. You will be returned to the Data Flow tab. From here, you can rename the Derived Column transformation to clearly show what it does. Again, there are two data paths to use to link to further transformations or to connect to destinations.

Adding an Excel Destination

Now we need to add a destination to our data flow to enable us to export our results into an **Excel**spreadsheet.

To add the destination, expand the **Data Flow** Destinations category in the **Toolbox**, and drag the **Excel**destination to the **SSIS Designer** workspace, as shown in Figure 18.

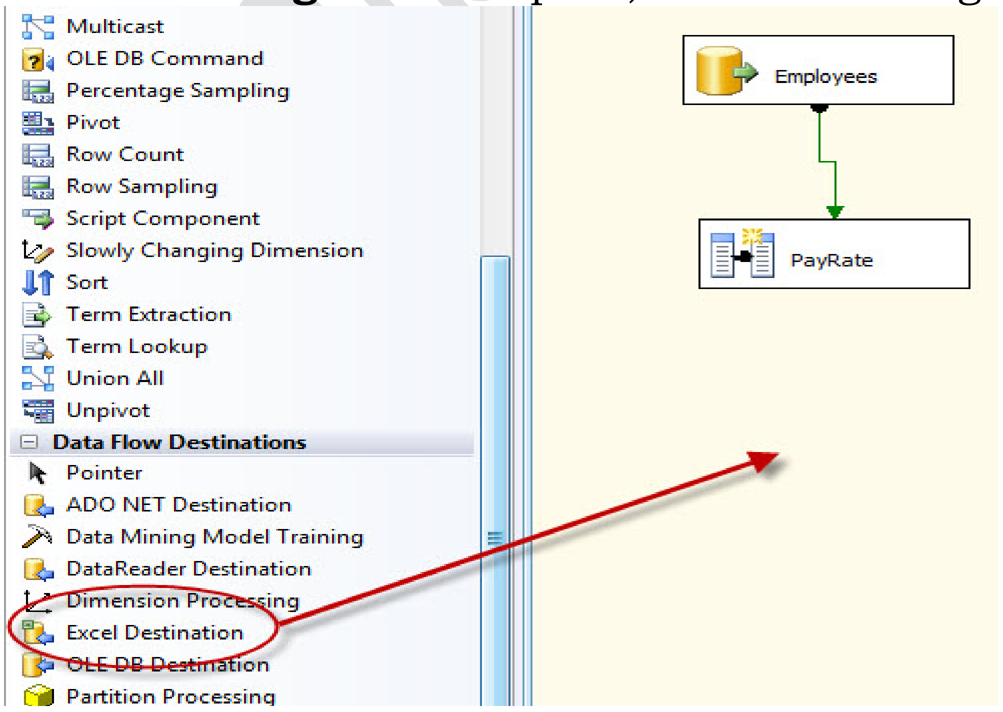


Figure 18: Adding an Excel destination to your data flow

Now connect the green data path from the **Derived Column** transformation to the **Excel** destination to associate the two components, as shown in Figure 19.

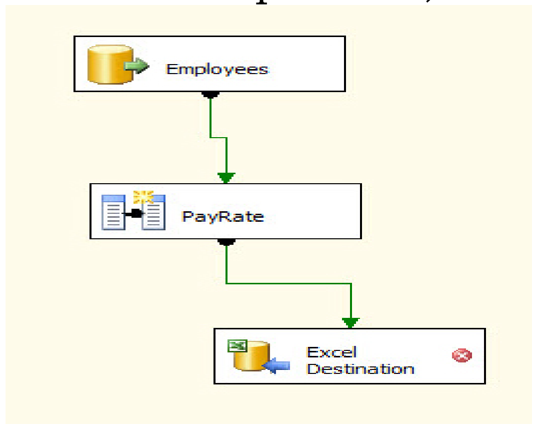
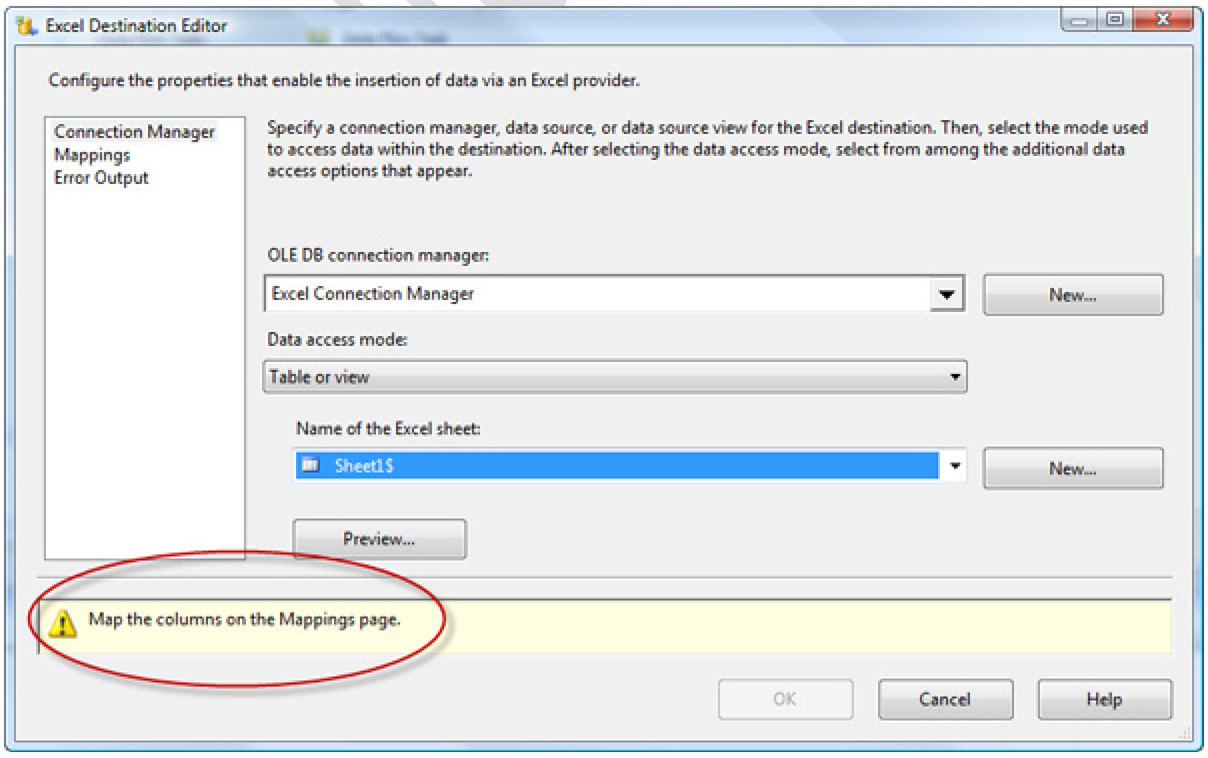


Figure 19: Connecting the data path from the transformation to the destination

As you can see, even though we have connected the **PayRate** transformation to the **Excel** destination, we still have the reversed red X showing us that there is a connection issue. This is because we have not yet selected the connection manager or linked the data flow columns to those in the **Excel**destination.

Next, right-click the **Excel** destination, and click **Edit**. This launches the **Excel Destination Editor** dialog box, shown in Figure 20. On the **Connection Manager** page, under **OLE DB connection manager**, click on the **New** button then under **Excel File Path** click on the Browse button and select the file you created in the previous article and click on OK, then under **Name of the Excel Sheet** select the appropriate sheet from the file.


Excel Destination Editor

Configure the properties that enable the insertion of data via an Excel provider.

Specify a connection manager, data source, or data source view for the Excel destination. Then, select the mode used to access data within the destination. After selecting the data access mode, select from among the additional data access options that appear.

OLE DB connection manager: Excel Connection Manager

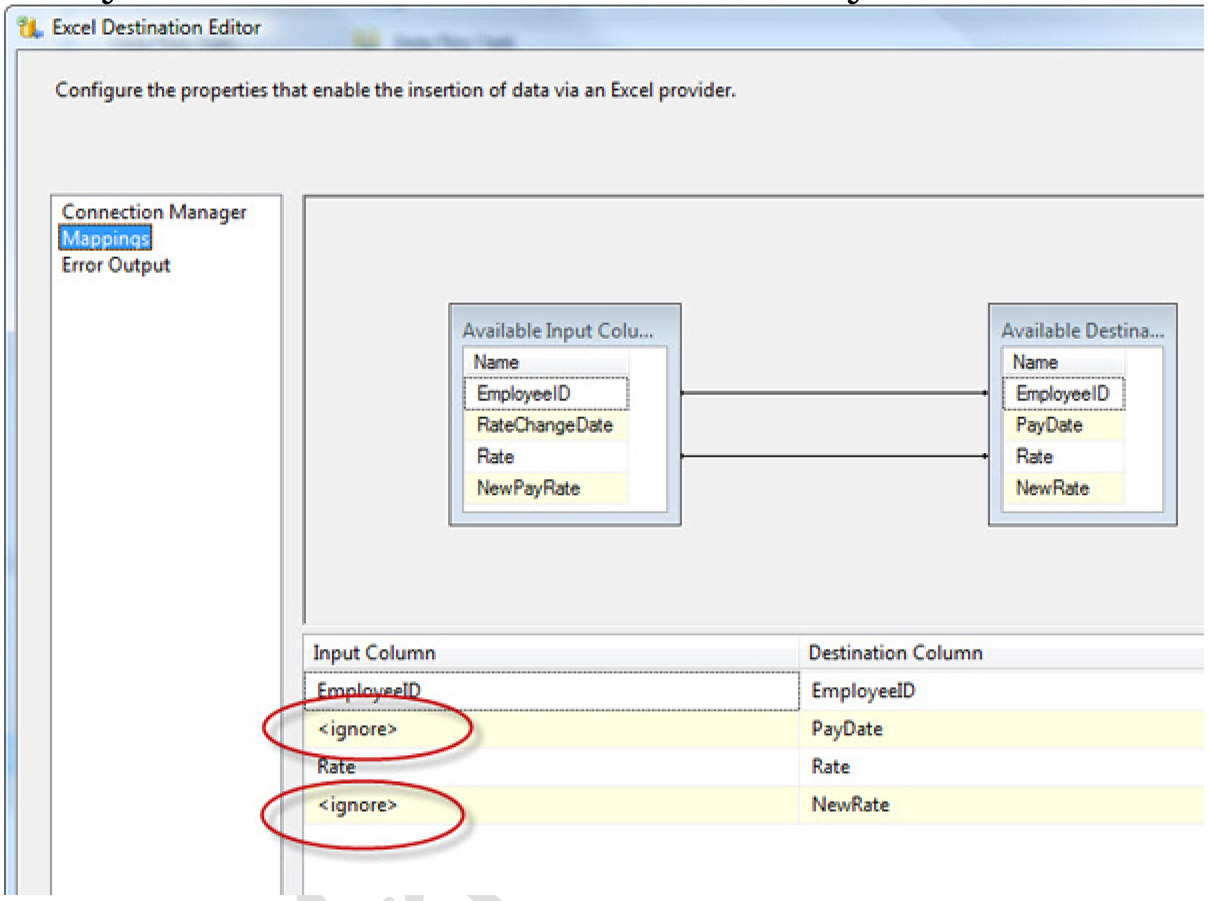
Data access mode: Table or view

Name of the Excel sheet: Sheet1$

Map the columns on the Mappings page.

Figure 20: Configuring the Excel destination component

At the bottom of the **Connection Manager** page, you'll notice a message that indicates we haven't mapped the source columns with the destination columns. To do this, go to the **Mappings** page (shown in Figure 21) and ensure that the columns in the data flow (the input columns) map correctly to the columns in the destination **Excel** file. The package will make a best guess based on field names; however, for this example, I have purposefully named my columns in the excel spreadsheet differently from those in the source database so they wouldn't be matched automatically.

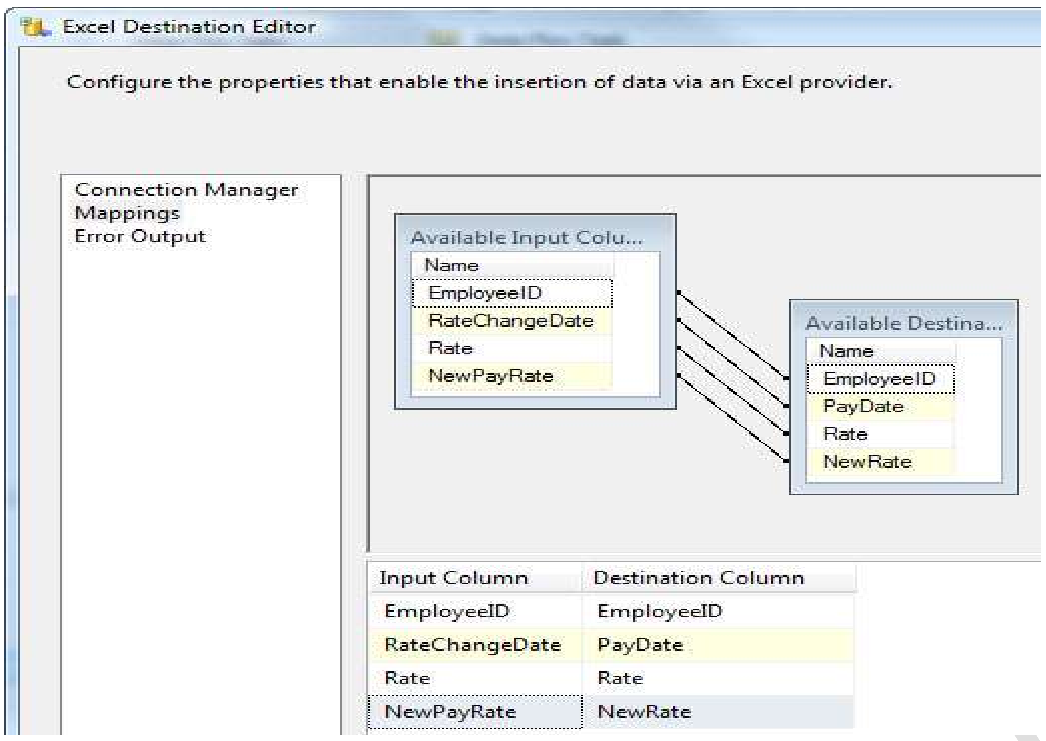

Excel Destination Editor

Configure the properties that enable the insertion of data via an Excel provider.

Input Column	Destination Column
EmployeeID	EmployeeID
<ignore>	PayDate
Rate	Rate
<ignore>	NewRate

Figure 21: The Mappings page of the Excel Destination Editor

To match the remaining columns, click the column name in the Input Column grid at the bottom of the page, and select the correct column. As you select the column, the list will be reduced so that only those columns not linked are available. At the same time, the source and destination columns in the top diagram will be connected by arrows, as shown in Figure 22.


Excel Destination Editor

Configure the properties that enable the insertion of data via an Excel provider.

Connection Manager
Mappings
Error Output

Input Column	Destination Column
EmployeeID	EmployeeID
RateChangeDate	PayDate
Rate	Rate
NewPayRate	NewRate

Figure 22: Mapping the columns between the data flow and the destination

Once you've properly mapped the columns, click OK. The Data Flow tab should now look similar to the screenshot in Figure 23.

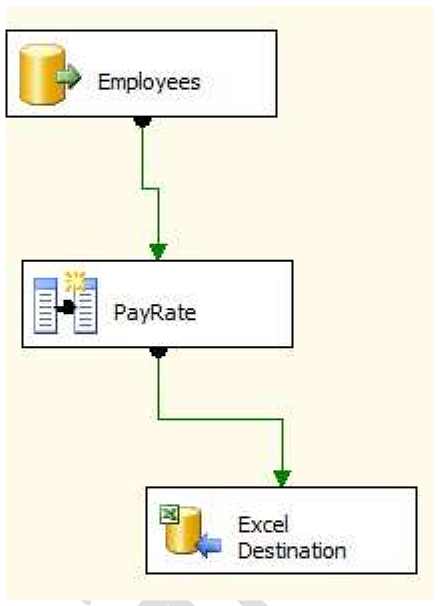


Figure 23: The configured data flow in your SSIS package

Running an SSIS Package in BIDS

Now all we need to do is execute the package and see if it works. To do this, click the Execute button. It's the green arrow on the toolbar, as shown in Figure 24.

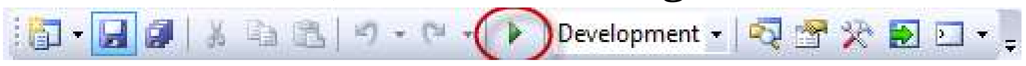
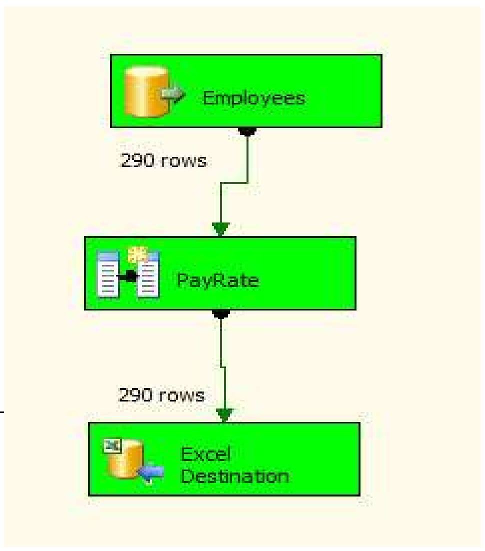


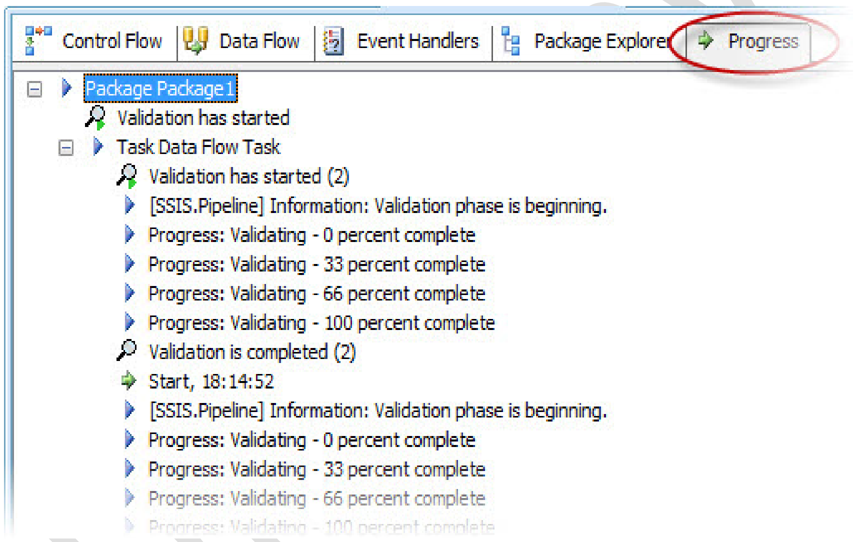
Figure 24: Clicking the Execute button to run your SSIS package



As the package progresses through the data flow components, each one will change color. The component will turn yellow while it is running, then turn green or red on completion. If it turns green, it has run successfully, and if it turns red, it has failed. Note, however, that if a component runs too quickly, you won't see it turn yellow. Instead, it will go straight from white to green or red.

The Data Flow tab also shows the number of rows that are processed along each step of the way. That number is displayed next to the data path. For our example package, 290 rows were processed between the Employees source and the PayRate transformation, and 290 rows were processed between the transformation and the Excel destination. Figure 25 shows the data flow after the three components ran successfully. Note that the number of processed rows are also displayed.

Figure 25: The data flow after if has completed running

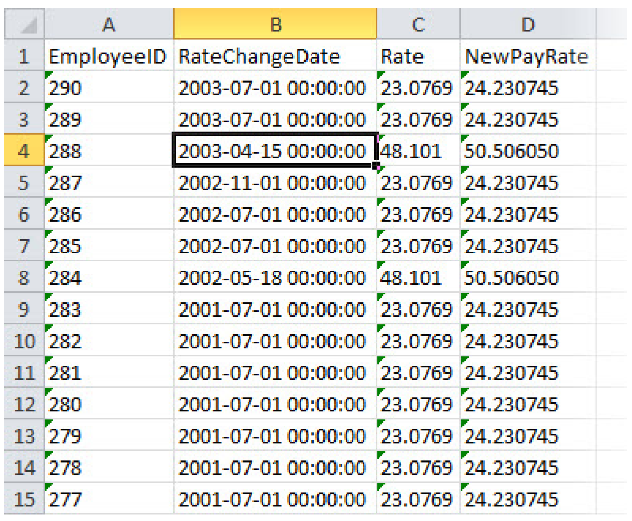


```
Package Package1
  Validation has started
  Task Data Flow Task
    Validation has started (2)
    [SSIS.Pipeline] Information: Validation phase is beginning.
    Progress: Validating - 0 percent complete
    Progress: Validating - 33 percent complete
    Progress: Validating - 66 percent complete
    Progress: Validating - 100 percent complete
    Validation is completed (2)
    Start, 18:14:52
    [SSIS.Pipeline] Information: Validation phase is beginning.
    Progress: Validating - 0 percent complete
    Progress: Validating - 33 percent complete
    Progress: Validating - 66 percent complete
    Progress: Validating - 100 percent complete
```

You can also find details about the package's execution on the **Progress** tab (shown in Figure 26). The tab displays each step of the execution process. If there is an error, a red exclamation mark is displayed next to the step's description. If there is a warning, a yellow exclamation mark is displayed. We will go into resolving errors and how to find them in a future article.

Figure 26: The Progress tab in SSIS Designer

Now all that's needed is to check the Excel file to ensure that the data was properly added. You should expect to see results similar to those in Figure 27.



	A	B	C	D
1	EmployeeID	RateChangeDate	Rate	NewPayRate
2	290	2003-07-01 00:00:00	23.0769	24.230745
3	289	2003-07-01 00:00:00	23.0769	24.230745
4	288	2003-04-15 00:00:00	48.101	50.506050
5	287	2002-11-01 00:00:00	23.0769	24.230745
6	286	2002-07-01 00:00:00	23.0769	24.230745
7	285	2002-07-01 00:00:00	23.0769	24.230745
8	284	2002-05-18 00:00:00	48.101	50.506050
9	283	2001-07-01 00:00:00	23.0769	24.230745
10	282	2001-07-01 00:00:00	23.0769	24.230745
11	281	2001-07-01 00:00:00	23.0769	24.230745
12	280	2001-07-01 00:00:00	23.0769	24.230745
13	279	2001-07-01 00:00:00	23.0769	24.230745
14	278	2001-07-01 00:00:00	23.0769	24.230745
15	277	2001-07-01 00:00:00	23.0769	24.230745

Figure 27: Reviewing the Excel file after package execution Bottom of Form

Introduction to Control Flow

In computer science, **control flow** (or **flow of control**) is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated. The emphasis on explicit control flow distinguishes an *imperative programming* language from a *declarative programming* language.

Within an imperative programming language, a *control flow statement* is a statement the execution of which results in a choice being made as to which of two or more paths to follow. For non-strict functional languages, functions and language constructs exist to achieve the same result, but they are usually not termed control flow statements.

A set of statements is in turn generally structured as a block, which in addition to grouping, also defines a lexical scope.

Interrupts and signals are low-level mechanisms that can alter the flow of control in a way similar to a subroutine, but usually occur as a response to some external stimulus or event (that can occur asynchronously), rather than execution of an *in-line* control flow statement.

At the level of machine language or assembly language, control flow instructions usually work by altering the program counter. For some central processing units (CPUs), the only control flow instructions available are conditional or unconditional branch instructions, also termed jumps.

Creating Dynamic Packages

Scenario

In data warehouse projects it is often necessary to import a huge amount of data provided in CSV format into a database, without making essential structural changes to the source data format.

Possible requirements for this Extract Transform Load (ETL) processes are:

- Adding a BatchId to the imported records in the staging tables
- Check the structure of the source file
- Logging detailed information about the imported data (number of rows imported, time to import the data, etc.)

To achieve this with MS SSIS, it is necessary to build a SSIS package for every source file. This can be very time consuming due do the fact that there may be hundreds of source files to be imported and it is very cumbersome to do all the mapping tasks again and again.

In this articles I want to show, how it is possible to build these SSIS packages dynamically. I will show how to store the required metadata and how to build all the packages by providing a template package with the desired functionality.

The SSIS package

One of the first things to consider is how the packages should look like. Which features should they contain, and how should the data be transformed on the way from the source file to the destination table.

For the sake of simplicity I will use a rather simple template package. But I will mention how one can implement more features into the ETL process without breaking the concepts I show in this articles.

Basis features of the template package:

- Signature check – before the procession of the input file takes place, the structure of the file is validated
- BatchId –assign a unique Id to each imported record

Using Containers

Containers are objects in SQL Server Integration Services that provide structure to packages and services to tasks. They support repeating control flows in packages, and they group tasks and containers into meaningful units of work. Containers can include other containers in addition to tasks.

Packages use containers for the following purposes:

- Repeat tasks for each element in a collection, such as files in a folder, schemas, or SQL Server Management Objects (SMO) objects. For example, a package can run Transact-SQL statements that reside in multiple files.
- Repeat tasks until a specified expression evaluates to **false**. For example, a package can send a different e-mail message seven times, one time for every day of the week.
- Group tasks and containers that must succeed or fail as a unit. For example, a package can group tasks that delete and add rows in a database table, and then commit or roll back all the tasks when one fails.

Container Types

Integration Services provides four types of containers for building packages. The following table lists the container types.

1. Foreach Loop Container: - Runs a control flow repeatedly by using an enumerator.
2. For Loop Container: - Runs a control flow repeatedly by testing a condition.
3. Sequence Container: - Groups tasks and containers into control flows that are subsets of the package control flow.
4. Task Host Container: - Provides services to a single task.

Summary of Container Properties

All container types have a set of properties in common. If you create packages using the graphical tools that Integration Services provides, the Properties window lists the following properties for the Foreach Loop, For Loop, and Sequence containers. The task host container properties are configured as part of configuring the task that the task host encapsulates. You set the Task Host properties when you configure the task.

DelayValidation A Boolean value that indicates whether validation of the container is delayed until run time. The default value for this property is **False**.

Description The container description. The property contains a string, but may be blank.

Disable A Boolean value that indicates whether the container runs.The default value for this property is **False**.

DisableEventHandlers A Boolean value that indicates whether the event handlersassociated with the container run. The default value for this property is **False**.

FailPackageOnFailure A Boolean value that specifies whether the package fails if an error occurs in the container. The default value for this property is **False**.

ForcedExecutionValue If **ForceExecutionValue**is set to **True**, the object that contains the optional execution value for the container. The default value of this property is **0**.

ForcedExecutionValueType The data type of **ForcedExecutionValue**. The default value of
this property is **Int32**.

ForceExecutionResult A value that specifies the forced result of running the package or container. The values are **None**, **Success**, **Failure**, and**Completion**. The default value for this property is **None**.

ForceExecutionValue A Boolean value that specifies whether the optional execution value of the container should be forced to contain a particular value. The default value of this property is **False**.

ID The container GUID, which is assigned when the package is created. This property is read-only.

Name The name of the container.

Foreach Loop Container

The Foreach Loop container defines a repeating control flow in a package. The loop implementation is similar to Foreachlooping structure in programming languages. In a package, looping is enabled by using a Foreach enumerator.

The Foreach Loop container repeats the control flow for each member of a specified enumerator. SQL Server Integration Services provides the following enumerator types:Foreach ADO enumerator to enumerate rows in tables. For example, you can get the rows in an ADO recordset.
The Recordset destination saves data in memory in a recordset that is stored in a package variable of Object data type. You typically use a Foreach Loop container with the Foreach ADO enumerator to process one row of the recordset at a time. The variable specified for the Foreach ADO enumerator must be of Object data type. For more information about the Recordset destination, see Use a Recordset Destination.
Foreach ADO.NET Schema Rowset enumerator to enumerate the schema information about a data source. For example, you can enumerate and get a list of the tables in the AdventureWorks2012 SQL Server database.
Foreach File enumerator to enumerate files in a folder. The enumerator can traverse subfolders. For
example, you can read all the files that have the *.log file name extension in the Windows folder and its subfolders.
ForeachFrom Variable enumerator to enumerate the enumerable object that a specified variable contains.
The enumerable object can be an array, an ADO.NET DataTable, an Integration Services enumerator, andso on. For example, you can enumerate the values of an array that contains the name of servers
Foreach Item enumerator to enumerate items that are collections. For example, you can enumerate the names of executables and working directories that an Execute Process task uses.

For Loop Container

The For Loop container defines a repeating control flow in a package. The loop implementation is similar to the For looping structure in programming languages. In each repeat of the loop, the For Loop container evaluates an expression and repeats its workflow until the expression evaluates to False. The For Loop container usesthe following elements to define the loop: An optional initialization expression that assigns values to the loop counters. An evaluation expression that contains the expression used to test whether the loop should stop or continue. An optional iteration expression that increments or decrements the loop counter.

Configure a For Loop Container

1. In SQL Server Data Tools (SSDT), open the Integration Services project that contains the package you want.
2. In Solution Explorer, double-click the package to open it.
3. Click the Control Flow tab.
4. To open the Toolbox, click Toolbox on the View menu.
5. Expand Control Flow Items and Maintenance Tasks.
6. Drag tasks and containers from the Toolbox to the design surface of the Control Flow tab.
7. Connect a task or container within the package control flow by dragging its connector to another component in the control flow.
8. To save the updated package, click Save Selected Items on the File menu.

Foreach Loop Container

The Foreach Loop container defines a repeating control flow in a package. The loop implementation is similar to Foreach looping structure in programming languages. In a package, looping is enabled by using a Foreach enumerator. The Foreach Loop container repeats the control flow for each member of a specified enumerator

Configure a Foreach Loop Container

1. In SQL Server Data Tools (SSDT), open the Integration Services project that contains the package you want.
2. Click the Control Flow tab and double-click the Foreach Loop.
3. In the Foreach Loop Editor dialog box, click General and, optionally, modify the name and description of the Foreach Loop.
4. Click Collection and select an enumerator type from the Enumerator list.
5. Specify an enumerator and set enumerator options as follows.

To use the Foreach File enumerator, provide the folder that contains the files to enumerate, specify a filter for the file name and type, and specify whether the fully qualified file name should be returned. Also, indicate whether to recurse through subfolders for more files.

To use the Foreach Item enumerator, click Columns, and, in the For Each Item Columns dialog box, click Add to add columns. Select a data type in the Data Type list for each column, and click OK.

Sequence Container

The Sequence container defines a control flow that is a subset of the package control flow. Sequence containers group the package into multiple separate control flows, each containing one or more tasks and containers that run within the overall package control flow.

Configuration of the Sequence Container

The Sequence container has no custom user interface and you can configure it only in the Properties window of SQL Server Data Tools (SSDT) or programmatically.

Task Host Container

The task host container encapsulates a single task. In SSIS Designer, the task host is not configured separately; instead, it is configured when you set the properties of the task it encapsulates. For more information about the tasks that the task host containers encapsulate.

Configuration of the Task Host

You can set properties in the Properties window of SQL Server Data Tools (SSDT) or programmatically**.**