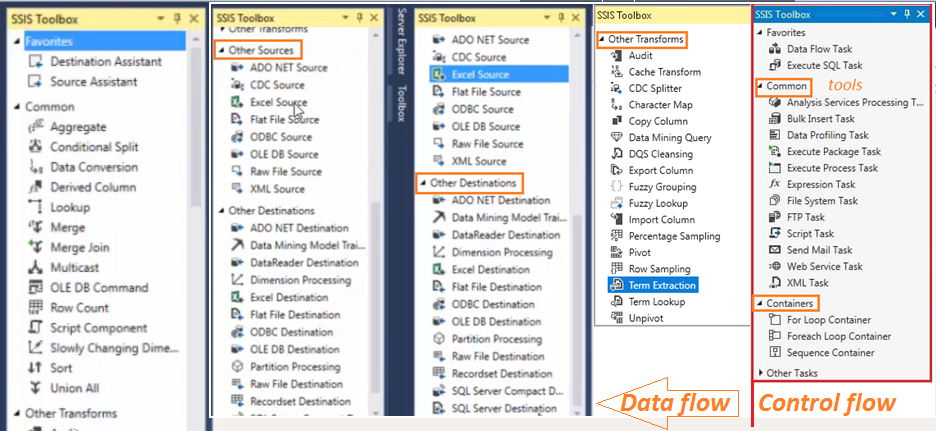
****

Depending on the functionality transformations are divided into the following five groups:

1. Business Intelligence Transformations
2. Split And Join Transformations
3. Row Transformations
4. Row-Set Transformations
5. Other Transformations.

**Business Intelligence Transformations**

1. **Fuzzy Group Transformation**: The Fuzzy Group Transformation is used to do data cleansing by finding rows that are likely duplicates and *reduce the number of duplicates within a dataset based on the Matching Decision.* This Transformation will accept only string data types while reducing the amount of duplicate data.
2. **Fuzzy Lookup Transformation**: The Fuzzy Lookup Transformation is used to return *close matches of reference* data for the incoming data stream. It can be matched and standardizes the data based on fuzzy logic.
3. **Term Extraction Transformation**: *Extracts terms* (nouns and noun phrases) from the input text into the transformation output column.
4. **Term Lookup Transformation**: Extracts terms from the *input column with TEXT data type and match them with the same or similar terms found in the lookup table*. Each term found in the lookup table is scanned for in the input column. If the term is found then the transformation returns the value as well as the number of times it occurs in the row. You can configure this transformation *to do a case-sensitive search*.
5. **Data Mining Query Transformation**: Queries a data mining model. Includes a query builder to *assist you with the development of Data Mining eXpressions (DMX) prediction queries*.
6. **Data Cleansing Transformation**: This transformation is used to *do automating data cleansing and monitoring* the overall status of the data cleansing process.

**Split and Join Transformations**

1. **Cache Transformation**: This transformation is used to *store data as a file or in memory* for use in a lookup transformation
2. **Conditional Split**: This transformation is used to accepts input and determine which destination to pipe the data into based on the result of an expression. It *redirects rows of data that meet specific conditions to different outputs*
3. **Look-Up Transformation**: This Look Up Transformation is used to *join the input data set to the reference table, view or row set* created by a SQL statement to lookup corresponding values. If some rows in the input data do not have corresponding rows in the lookup table then you must redirect such rows to a different output.
4. **Merge Transformation**: This transformation is used to *merge two sorted inputs into a single output* based on the values of the key columns in each data set. Merged columns must have either identical or compatible data types
5. **Merge-Join Transformation**: This transformation is used to merge two datasets into a single dataset using a *JOIN* function.
6. **Multicast Transformation**: This transformation is used to *send a copy of the data to an additional path* in the workflow. It duplicates the data in the dataflow that again we can send the data in parallel, or when we want to send the data to multiple destinations simultaneously.
7. **Union-All Transformation**: *Combines multiple inputs into a single output.* Rows are sorted in the order they're added to the transformation. You can ignore some columns from each output, but each output column must be mapped to at least one input column.

**Row Transformations**

1. **Character Map Transformation**: The Character Map transformation allows you to do *character operations on string* columns. It makes common string data changes for you.
2. **Copy Column Transformation**: This transformation is used to add a *copy of column to the transformation output*. You can later transform the copy. Makes a copy of a single or multiple columns that will be further transformed by subsequent tasks in the package
3. **Data Conversion Transformation**: This transformation is used *to convert a column data type* to a new (another) column data type.
4. **Derived Column Transformation**: This transformation is used to *apply expression to a data column and create a new derived column calculated* from an expression.
5. **OLEDB Command Transformation**: Runs a SQL *command for each input data row.* Normally your SQL statement will include a parameter (denoted by the question mark)
6. **Script Component Transformation**: This transformation is used to do a custom transformation. It uses a *script to transform the data and you can apply specialized business logic* to your data flow.

**Row Set Transformations**:

1. **Aggregate Transformation**: This transformation is used to aggregates the data from transformation or source and it *aggregates the values by group.*
2. **Row Sampling Transformation**: This transformation is used to capture a *sampling of the data* from the data flow by using a *row count* of the data flow's total rows. It Loads only a subset of your data, defined as the number of rows and it *randomly selects the data* and is delivered to somewhere.
3. **Percentage Sampling Transformation**: Loads only a *subset of your data*, defined as the percentage of all rows in the data source. It randomly selects percentage of rows.
4. **Sort Transformation**: This transformation is used to *sort the data* in the data flow by a given column and discard with duplicate values (optionally eliminating duplicates).
5. **Pivot Transformation**: This transformation is used to pivot *the data on a column into a more non-relational form*. It converts rows into columns.
6. **UnPivot Transformation**: This transformation is used to *unpivot the data from a non-normalized format* to a relational format.

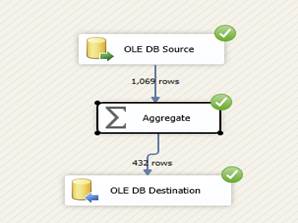
**Other SSIS Transformations**:

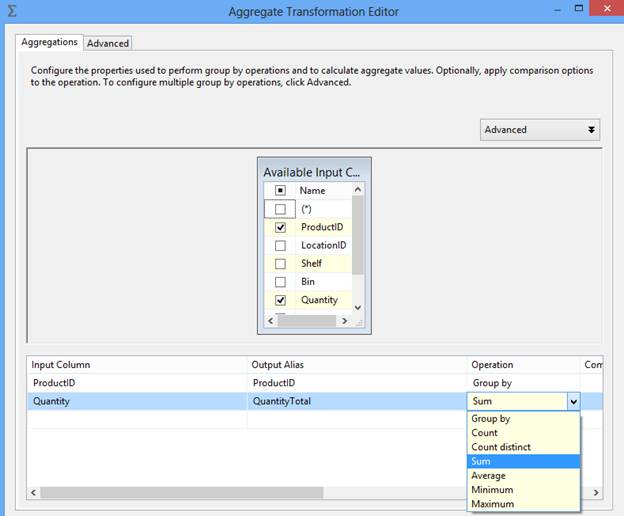
1. **Audit Transformation:** This transformation is used to *expose auditing information from the package* to the data pipe, such as package execution and Execution Time.
2. **Row Count Transformation:** This transformation is used to *count the rows* in the data flow and stores them as a variable.
3. **SCD Transformation:** This transformation *maintains the historical values of the dimension members* when new members are introduced. It automatically generates transformations for TYPE1 and TYPE2 SCD's.
4. **Export Transformation:** This transformation is used to *export the column from the data flow to the system*.
5. **Import Transformation:** This transformation is used to *read data from files and appends it to the data flow*.

**Dataflow - Common Transformation**

**Aggregate**

An Asynchronous full blocking transformation, Aggregate transformation allows to aggregate data from Data Flow to apply certain T-SQL functions that are done in a GROUP BY statement.

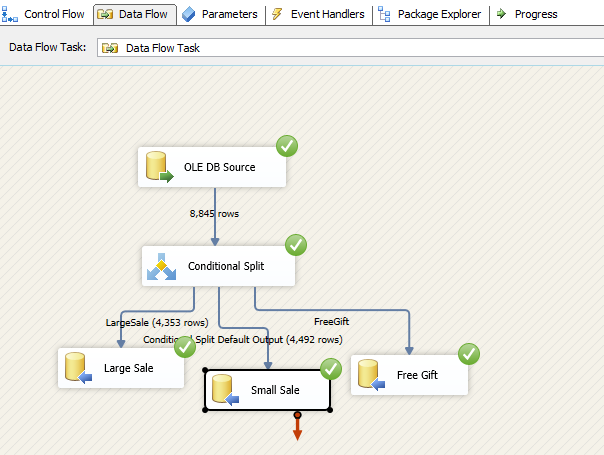


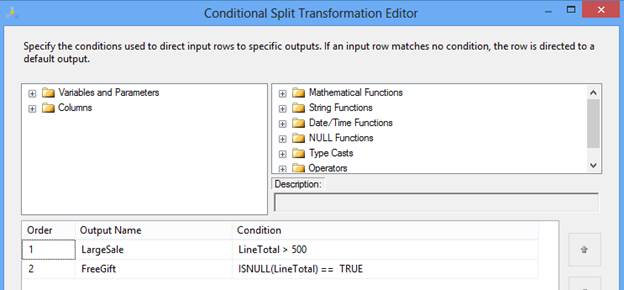


In the above example we have applied SUM aggregation but Aggregation transformation provides other options to aggregate data like Count, Count distinct, Average, Minimum and Maximum.

**Conditional Split**

Synchronous transformation, allows you to send the data from a single data path to various outputs or paths based on conditions that use the SSIS expressions.





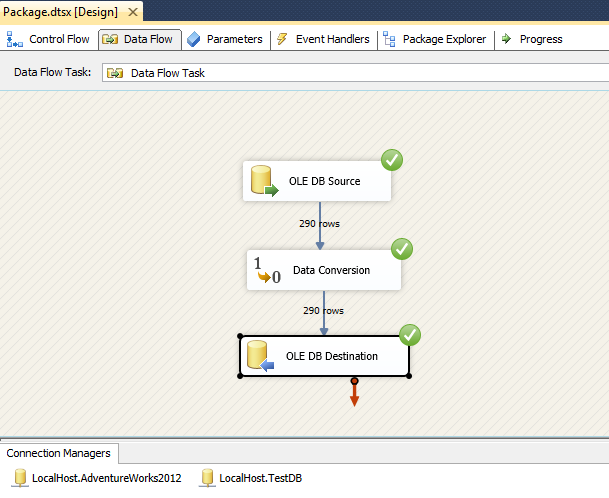
In the above example, we are splitting input records based on total order cost. If cost is more than 500, the record will be considered as part of a large sale. If LineTotal is NULL, we are assuming it's a free gift and no cost is associated with it. The rest we can consider part of small sale, in current implementation it is the default output of Conditional Split transformation.

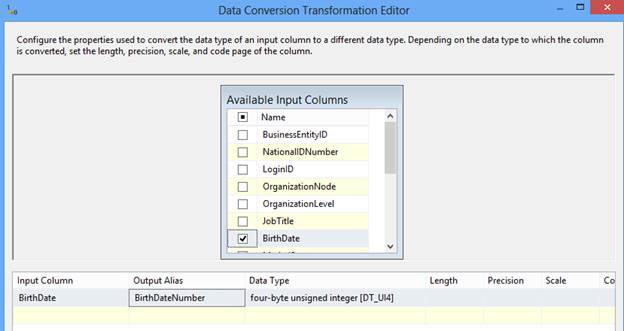
After execution of DFT the data will move in three different destinations as per ETL design.

## Data Conversion

Synchronous transformation is used for data conversion. It is a similar function to the Convert or Cast functions in T-SQL. It is a very useful transformation if we are pulling same data from multiple sources.

Data flow task design for Data conversion:

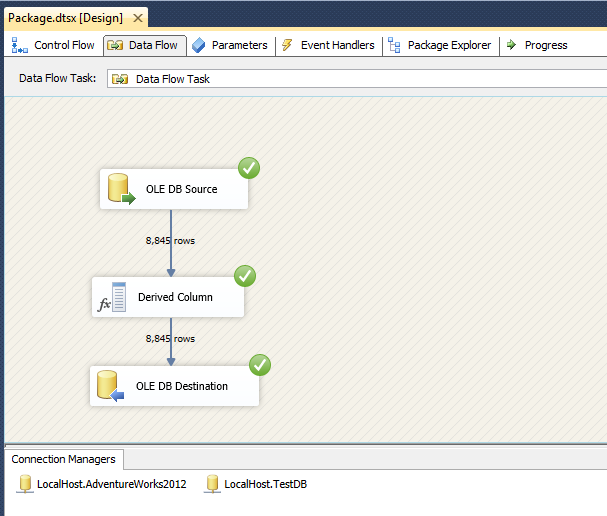


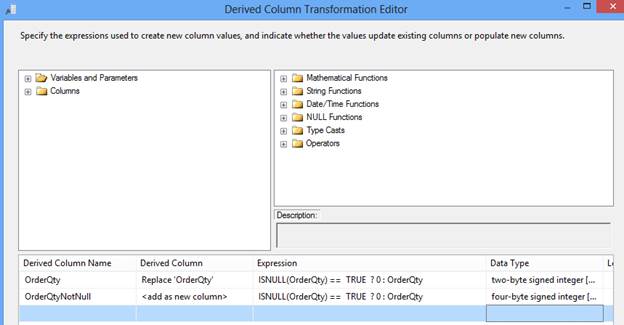


In this example we converted the BirthDate column of the datetime data type in another column BirthDateNumber of Integer data type.

## Derived Column

Synchronous transformation, this transformation creates a new column that is derived from the output of another column. This transformation provides you two options; either you can create a new column as a derived column or replace the existing column with a new derived column.

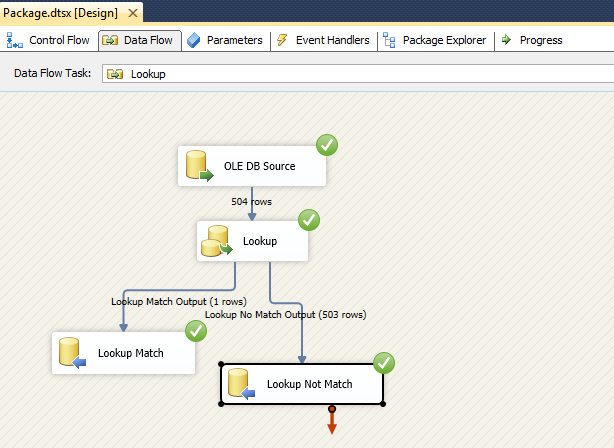




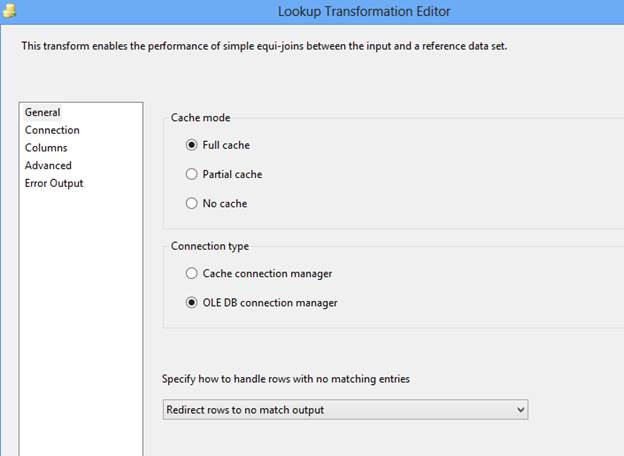
In this example, in the first row, check the if OrderQty value is NULL then update with 0 and in the second row apply the same operation as in the first row; the only difference is it will create one new column OrderQtyNotNull in the output. So, with the help of Derived Column transformation you can either update an existing column value or introduce a new column in the output.

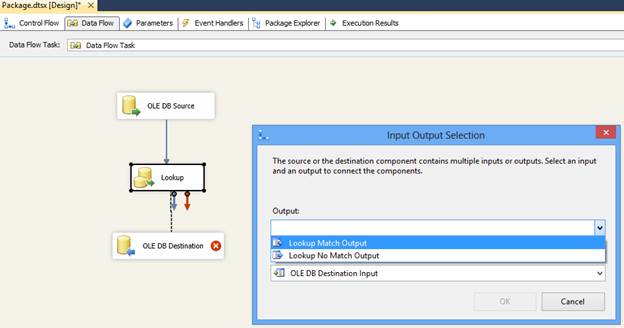
## Lookup

Synchronous transformation, allows you to perform an equi-join between values in the transformation input and values in the reference dataset similar to T-SQL. This transformation is used to join two datasets at a time.  To join more than two datasets we need to put multiple Lookup transformations, similar to a T-SQL join condition.

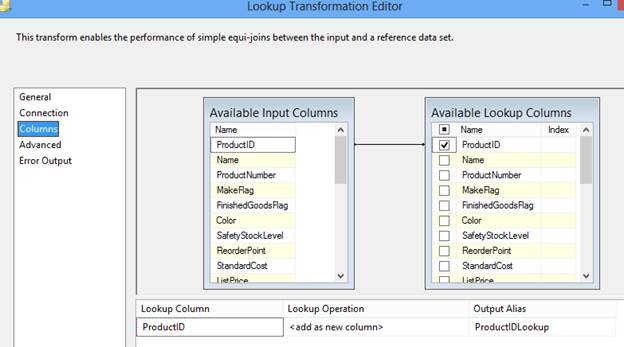


If there is no matching entry in the reference dataset, no join occurs. By default, the Lookup transformation treats rows without matching entries as errors. However, it can configure the Lookup transformation to redirect such rows to a no match output as shown in the images below:





The join can be a composite join, which means that multiple columns can be used in the join in the transformation input to columns in the reference dataset; for simplification we used only one column. Refer to the below image:

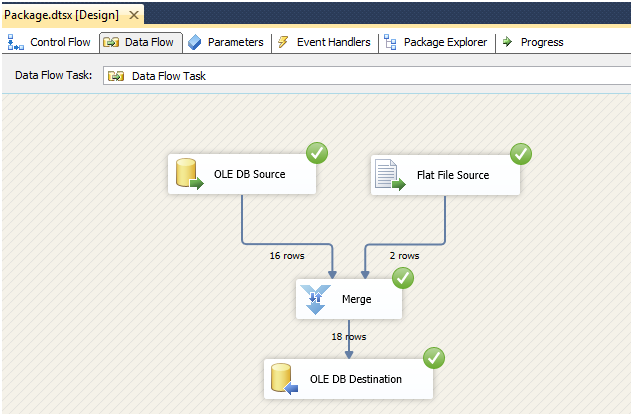


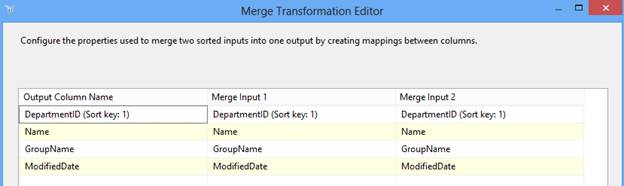
In above image, you can observe in the Lookup Operation that we specified “<add as new column>”; its mean values from the reference dataset are added as a new column to the transformation output. For example, the Lookup transformation can extract the ProductID details from a table using a value from an input column, and then add the ProductIDLookup to the transformation output. The values from the reference table can replace column values or can be added to new columns.

Lookup transformations provides several modes of operations, Full cache, Partial cache or No cache, that allows a trade-off between performance and resource usage.

## Merge

An Asynchronous partial blocking transformation merges two sorted data sets into a single dataset. This transformation is very useful when during ETL its needs to merge data from two different data sources. Merge transformation can’t merge a column that has a numeric data type with a column that has a character data type.





In the above example, we are merging data from two sources; OLEDB and Flat File. The Merge transformation automatically maps columns that have the same metadata. You can then manually map other columns that have compatible data types.

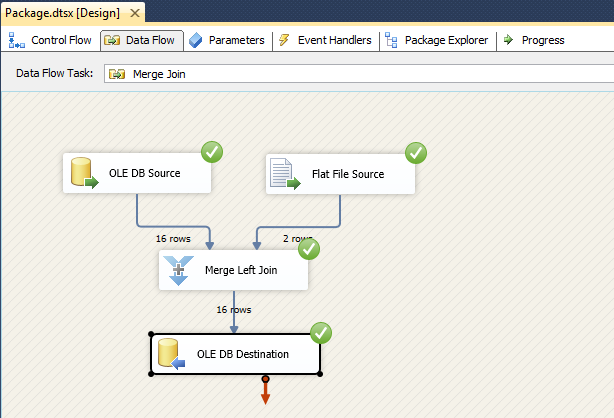
This transformation has two inputs and one output. It does not support an error output.

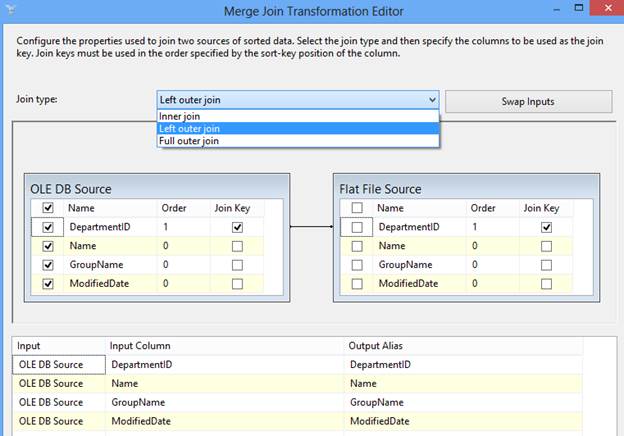
## Merge Join

An Asynchronous partial blocking transformation, allows joining data from two sorted datasets using a FULL, LEFT, or INNER join.

It also has two inputs and one output and like Merge transformation, does not support an error output.

Data Flow task design for Merge Join:



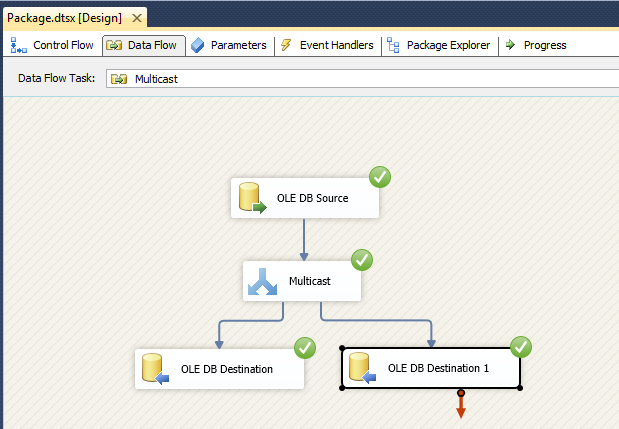


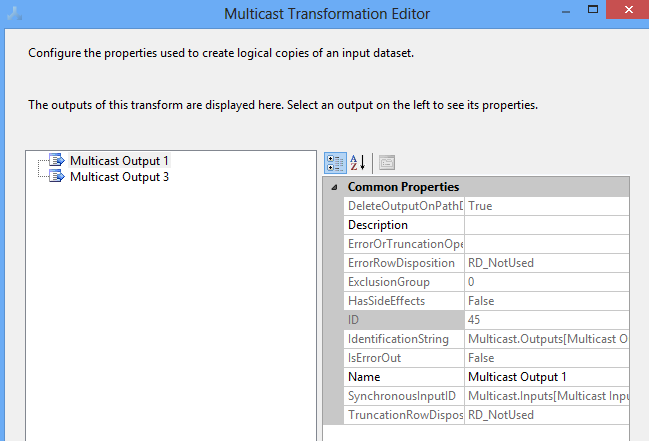
In above example, we merged data from two different sources; OLEDB and Flat File, applying a Left outer join on DepartmentID.

## Multicast

Synchronous transformation allows you to distribute its input to one or more outputs. This transformation is similar to the Conditional Split transformation. Both transformations direct an input to multiple outputs. The difference between the two is that the Multicast transformation directs every row to every output, and the Conditional Split directs a row to a single output.

In nutshell, a Multicast transformation is used to create/distribute exact copies of the source dataset to one or more destination datasets.

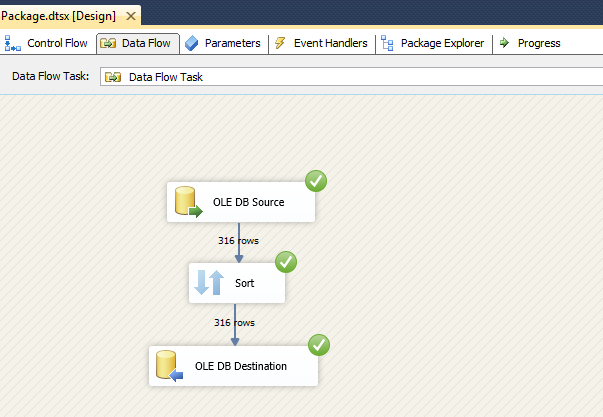


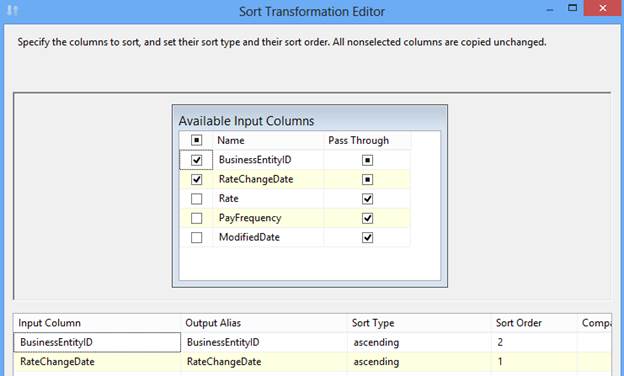


In the above example, we are distributing log data to two different destinations

## Sort

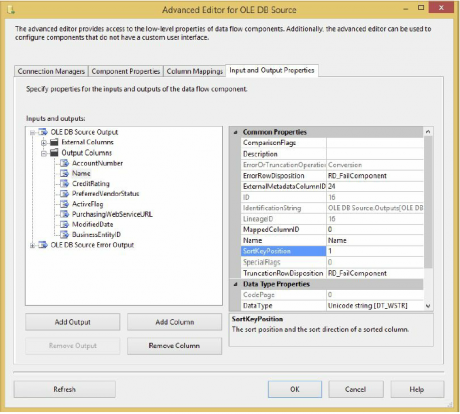
An Asynchronous full blocking transformation allows sort or arrange input data in ascending or descending order and copies the sorted data to the transformation output. You can apply multiple sorts to an input; the column with the lowest number is sorted first, the sort column with the second lowest number is sorted next.





In above example, we arranged input data in ascending order of RateChangeDate first and BusinessEntityID column second.

Sort transformation has one input and one output. It does not support error outputs.



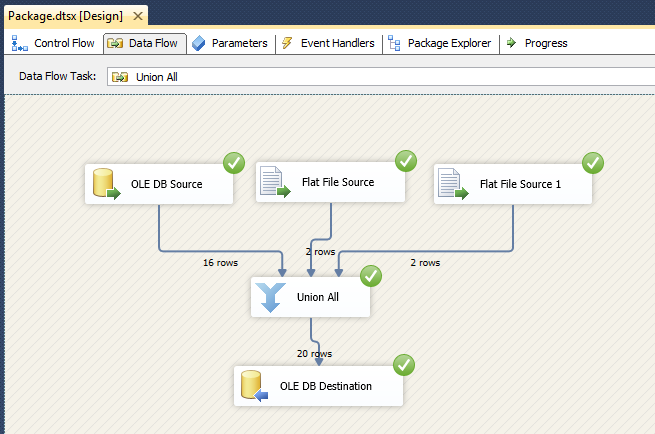
If you have an ORDER BY clause in your T-SQL statement in the OLE DB Source or the ADO.NET Source, you can notify SSIS that the data is already sorted, obviating the need for the Sort Transformation in the Advanced Editor. After ordering the data in your SQL statement, right-click the source and select Advanced Editor. From the Input and Output Properties tab, select OLE DB Source Output. In the Properties pane, change the IsSorted property to True.

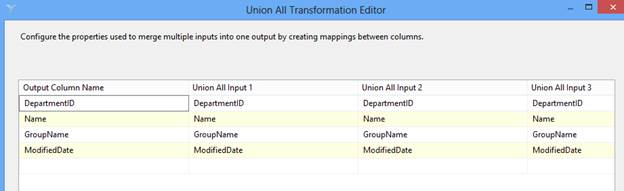
Then, under Output Columns, select the column you are ordering on in your SQL statement, and change the SortKeyPosition to 1 if you’re sorting only by a single column ascending, as shown in Figure. If you have multiple columns, you could change this SortKeyPosition value to the column position in the ORDER BY statement starting at 1. A value of -1 will sort the data in descending order.

## Union All

An Asynchronous partial blocking transformation, allows you to combine multiple (more than two) input and produce one output. Its add inputs to transformation output one after the other and doesn’t sort the data.

Data Flow task design of Union All:

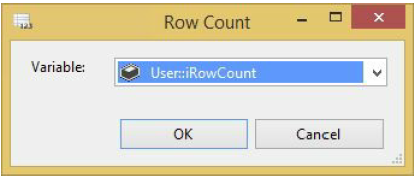




In above example, we used three sources as input and combine all using the Union All transformation before inserting into  the destination. Here, we took two different type of sources; OLEDB and Flat File.

### Row Count

The Row Count Transformation provides the capability to count rows in a stream that is directed to its input source. This transformation must place that count into a variable that could be used in the Control Flow — for insertion into an audit table, for example.  If you need to know how many rows are split during the Conditional Split Transformation, direct the output of each side of the split to a separate Row Count Transformation. You can then use this variable to log information into storage, to build e-mail messages, or to conditionally run steps in your packages.



In the Control Flow tab, add a variable named **iRowCount**. Ensure that the variable is package scoped and of type Int32

### Slowly Changing Dimension

A dimension table contains a set of discrete values with a description and often other measurable attributes such as price, weight, or sales territory. The classic problem is what to do in your dimension data when an attribute in a row changes — particularly when you are loading data automatically through an ETL process. This transformation can shave days off of your development time in relation to creating the load manually through T-SQL, but it can add time because of how it queries your destination and how it updates with the OLE DB Command Transform (row by row).

### Copy Column

The Copy Column Transformation is a very simple transformation that copies the output of a column to a clone of itself. This is useful if you wish to create a copy of a column before you perform some elaborate transformations. You could then keep the original value as your control subject and the copy as the modified column. To configure this transformation, go to the Copy Column Transformation Editor and check the column you want to clone. Then assign a name to the new column.

### Data Mining Query

The Data Mining Query Transformation typically is used to fill in gaps in your data or predict a new column for your Data Flow. This transformation runs a Data Mining Extensions (DMX) query against an SSAS data-mining model and adds the output to the Data Flow. It also can optionally add columns, such as the probability of a certain condition is true. A few great scenarios for this transformation would be the following:

* You could take columns, such as the number of children, household income, and marital income, to predict a new column that states whether the person owns a house or not.
* You could predict what customers would want to buy based on their shopping cart items.
* You could fill the gaps in your data where customers didn’t enter all the fields in a questionnaire.

The possibilities are endless with this transformation.

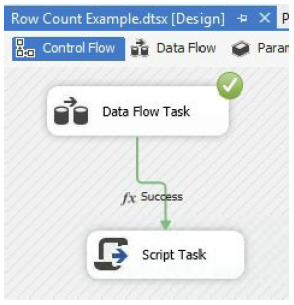
### DQS Cleansing

The Data Quality Services (DQS) Cleansing Transformation performs advanced data cleansing on data flowing through it. With this transformation, you can have your business analyst (BA) create a series of business rules that declare what good data looks like in the Data Quality Client (included in SQL Server). The BA will use a tool called the Data Quality Client to create domains that define data in your company, such as what a Company Name column should always look like. The DQS Cleansing Transformation can then use that business rule.

This transformation will score the data for you and tell you what the proper cleansed value should be. Advanced-Data Cleansing in SSIS Topic covers this transformation in much more detail.

### Script Component

The Script Component enables you to write custom .NET scripts as transformations, sources, or destinations. Once you drag the component over, it will ask you if you want it to be a source, transformation, or destination.



Some of the things you can do with this transformation include the following:

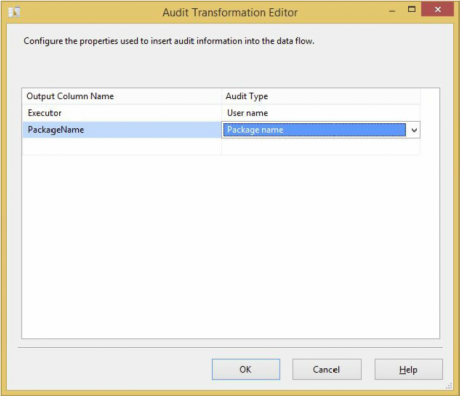
* Create a custom transformation that would use a .NET assembly to validate credit card numbers or mailing addresses.
* Validate data and skip records that don’t seem reasonable. For example, you can use it in a human resource recruitment system to pull out candidates that don’t match the salary requirement at a job code level.
* GIACT - validates routing number and bank account no
* Check image OCR
* vendor insurance information

Scripts used as sources can support multiple outputs, and you have the option of precompiling the scripts for runtime efficiency.

### Audit

The Audit Transformation allows you to add auditing data to your Data Flow. Because of acts such as HIPPA and Sarbanes-Oxley (SOX) governing audits, you often must be able to track who inserted data into a table and when. This transformation helps you with that function. The task is easy to configure. For example, to track what task inserted data into the table, you can add those columns to the Data Flow path with this transformation. The functionality in the Audit Transformation can be achieved with a Derived Column Transformation, but the Audit Transformation provides an easier interface.

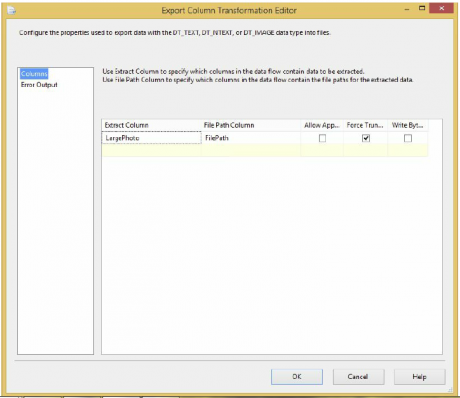
All other columns are passed through to the path as an output, and any auditing item you add will also be added to the path. Simply select the type of data you want to audit in the Audit Type column (shown in Figure 4-22), and then name the column that will be outputted to the flow. Following are some of the available options:



* **Execution instance GUID**: GUID that identifies the execution instance of the package
* **Package ID:** Unique ID for the package
* **Package name**: Name of the package
* **Version ID**: Version GUID of the package
* **Execution start time**: Time the package began
* **Machine name:** Machine on which the package ran
* **User name**: User who started the package
* **Task name:** Data Flow Task name that holds the Audit Task
* **Task ID:** Unique identifier for the Data Flow Task that holds the Audit

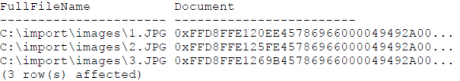
### Export Column

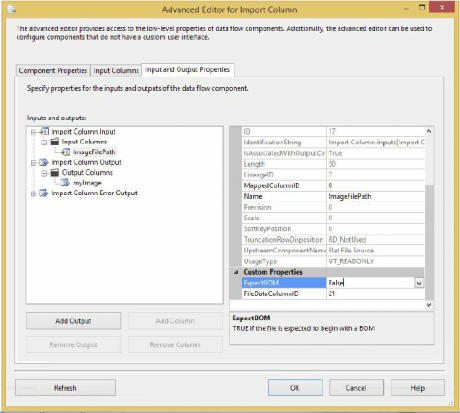
The Export Column Transformation Task is used to extract blob-type data from fields in a database and create files in their original formats to be stored in a file system or viewed by a format viewer, such as Microsoft Word or Microsoft Paint. The trick to understanding the Export Column Transformation is that it requires an input stream field that contains digitized document data, and another field that can be used for a fully qualified path. The Export Column Transformation will convert the digitized data into a physical file on the file system for each row in the input stream using the fully qualified path.



### Import Column

The Import Column Transformation is a partner to the Export Column Transformation. These transformations do the work of translating physical files from system file storage paths into database blob-type fields, and vice versa. The trick to understanding the Import Column Transformation is knowing that its input source requires at least one column that is the fully qualified path to the file you are going to store in the database, and you need a destination column name for the output of the resulting blob and file path string.

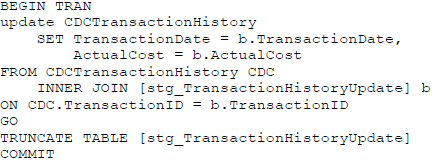




### OLE DB Command

The OLE DB Command Transformation is a component designed to execute a SQL statement for each row in an input stream. This task is analogous to an ADO Command object being created, prepared, and executed for each row of a result set. The input stream provides the data for parameters that can be set into the SQL statement, which is either an in-line statement or a stored procedure call.

This transformation should be avoided whenever possible. It’s a better practice to land the data into a staging table using an OLE DB Destination and perform an update with a set-based process in the Control Flow with an Execute SQL Task. The Execute SQL Task’s statement would look something like this if you loaded a table called stg\_TransactionHistoryUpdate and were trying to do a bulk update:



If you have 2,000 rows running through the transformation, the stored procedure or command will be executed 2,000 times. It might be more efficient to process these transactions in a SQL batch, but then you would have to stage the data and code the batch transaction. The main problem with this transformation is performed.

Percentage and Row Sampling

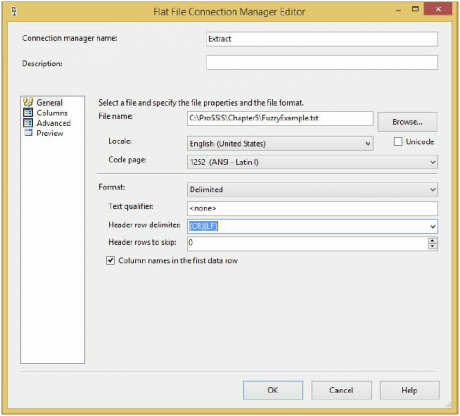
The Percentage Sampling and Row Sampling Transformations enable you to take the data from the source and randomly select a subset of data. The transformation produces two outputs that you can select. One output is the data that was randomly selected, and the other is the data that was not selected. You can use this to send a subset of data to a development or test server. The most useful application of this transformation is to train a data mining model. You can use one output path to train your data-mining model, and the sampling to validate the model.

To configure the transformation, select the percentage or number of rows you wish to be sampled. As you can guess, the Percentage Sampling Transformation enables you to select the percentage of rows, and the Row Sampling Transformation enables you to specify how many rows you wish to be outputted randomly. Next, you can optionally name each of the outputs from the transformation. The last option is to specify the seed that will randomize the data. If you select seed and run the transformation multiple times, the same data will be outputted to the destination. If you uncheck this option, which is the default, the seed will be automatically incremented by one at runtime, and you will see random data each time.

### Fuzzy Lookup

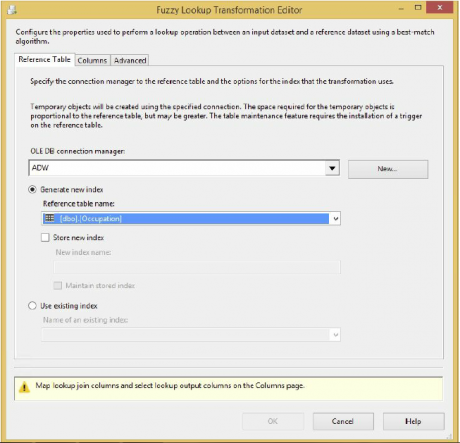
The Fuzzy Lookup and Fuzzy Grouping Transformations add one more road to take at the crossroads of bad data. These transformations allow the addition of a step to the process that is easy to use, consistent, scalable, and reusable, and they will reduce your unmatched rows significantly — maybe even altogether. If you’ve already allowed bad data in your dimension tables, or you are just starting a new ETL process, you’ll want to put the Fuzzy Grouping Transformation to work on your data to find data redundancy. This transformation can examine the contents of a suspect field in a staged or committed table and provide possible groupings of similar words based on provided tolerances. This matching information can then be used to clean up that table.

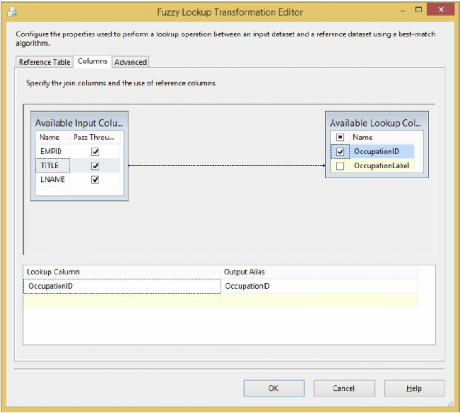
If you are correcting data during an ETL process, use the Fuzzy Lookup Transformation — my suggestion is to do so only after attempting to perform a regular lookup on the field. This best practice is recommended because Fuzzy Lookups don’t come cheap. They build specialized indexes of the input stream and the reference data for comparison purposes. You can store them for efficiency, but these indexes can use up some disk space or take up some memory if you choose to rebuild them on each run. Storing matches made by the Fuzzy Lookups over time in a translation or pre-dimension table is a great design. Regular Lookup Transformations can first be run against this lookup table and then divert only those items in the Data Flow that can’t be matched to a Fuzzy Lookup. This technique uses Lookup Transformations and translation tables to find matches using INNER JOINs. Fuzzy Lookups whittle the remaining unknowns down if similar matches can be found with a high level of confidence. Finally, if your last resort is to have the item diverted to a subject matter expert, you can save that decision into the translation table so that the ETL process can match it next time in the first iteration.



Using the Fuzzy Lookup Transformation requires an input stream of at least one field that is a string. Internally, the transformation has to be configured to connect to a reference table that will be used for comparison. The output to this transformation will be a set of columns containing the following:

* **Input and Pass-Through Field Names and Values**: This column contains the name and value of the text input provided to the Fuzzy Lookup Transformation or passed through during the lookup.
* **Reference Field Name and Value**: This column contains the name and value(s) of the matched results from the reference table.
* **Similarity**: This column contains a number between 0 and 1 representing similarity to the matched row and column. The similarity is a threshold that you set when configuring the Fuzzy Lookup Task. The closer this number is to 1, the closer the two text fields must match.
* **Confidence**: This column contains a number between 0 and 1 representing the confidence of the match relative to the set of matched results. Confidence is different from similarity, because it is not calculated by examining just one word against another but rather by comparing the chosen word match against all the other possible matches. For example, the value of Knight Brian may have a low similarity threshold but high confidence that it matches to Brian Knight. Confidence gets better the more accurately your reference data represents your subject domain, and it can change based on the sample of the data coming into the ETL process.



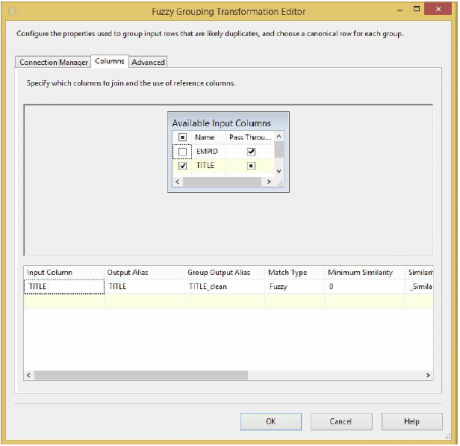


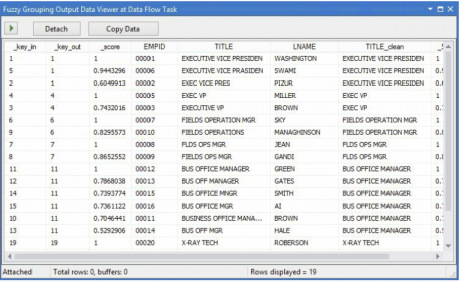
### Fuzzy Grouping

The Fuzzy Grouping Transformation can look through a list of similar text and group the results using the same logic as the Fuzzy Lookup. You can use these groupings in a transformation table to clean up source and destination data or to crunch fact tables into more meaningful results without altering the underlying data. The Fuzzy Group Transformation also expects an input stream of text, and it requires a connection to an OLE DB Data Source because it creates in that source a set of structures to use during the analysis of the input stream.

The Fuzzy Lookup Editor has three configuration tabs:

* **Connection Manager**: This tab sets the OLE DB connection that the transform will use to write the storage tables that it needs.
* **Columns**: This tab displays the Available Input Columns and allows the selection of any or all input columns for fuzzy grouping analysis. Below Screenshot shows a completed Columns tab. Each column selected is analyzed and grouped into logical matches, resulting in a new column representing that group match for each data row. Each column can also be selected for Pass-Through — meaning the data is not analyzed, but it is available in the output stream. You can choose the names of any of the output columns: Group Output Alias, Output Alias, Clean Match, and Similarity Alias Score column.

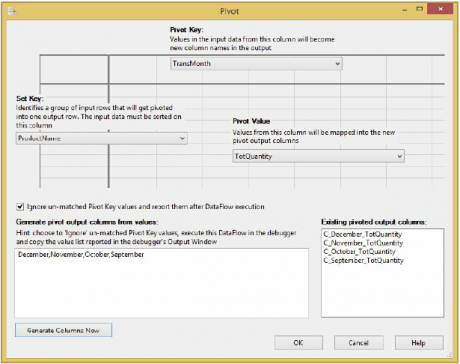




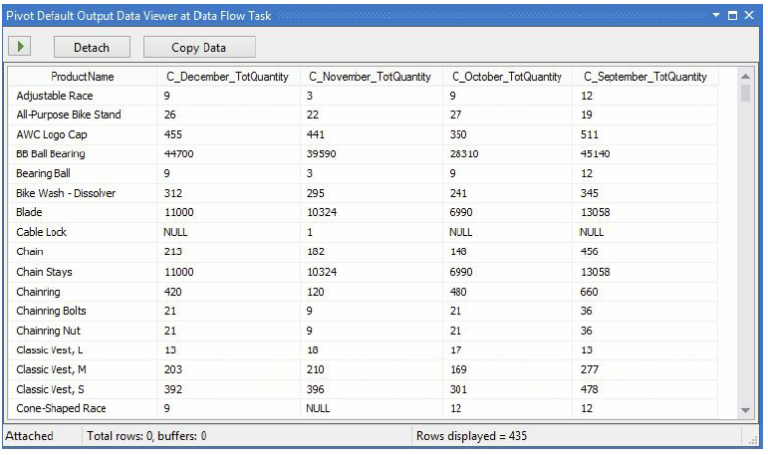
The title “Executive Vice President” isn’t a complete title, and really should be grouped with “Exec VP,” but this is pretty good for about five minutes of work.

### Pivot Transform

Do you ever get the feeling that pivot tables are the modern-day Rosetta Stone for translating data to your business owners? You store it relationally, but they ask for it in a format that requires you to write a complex case statement to generate. Well, not anymore. Now you can use an SSIS transformation to generate the results. A pivot table is a result of crosstabulated columns generated by summarizing data from a row format.

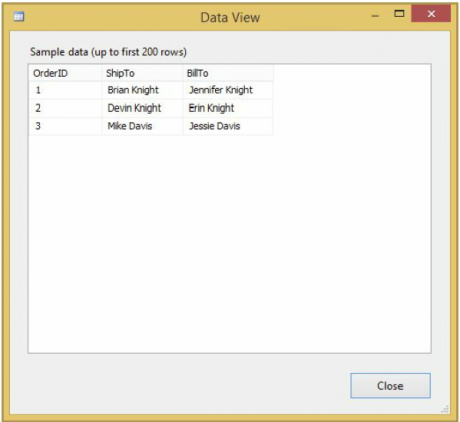


* **Pivot Key**: A pivot column is the element of input data to “pivot.” The word “pivot” is another way of saying “to create a column for each unique instance of.” However, this data must be under control.
* **Set Key**: Set key creates one column and places all the unique values for all rows into this column. Just like any GROUP BY statement, some of the data is needed to define the group (row), whereas other data is just along for the ride.
* **Pivot Value**: These columns are aggregations for data that provide the results in the matrix between the row columns and the pivot columns.

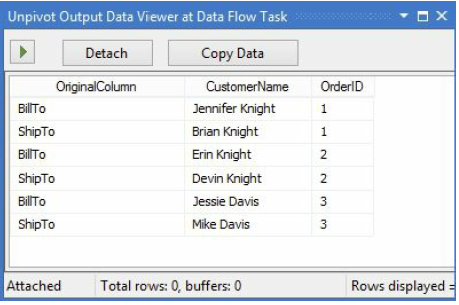


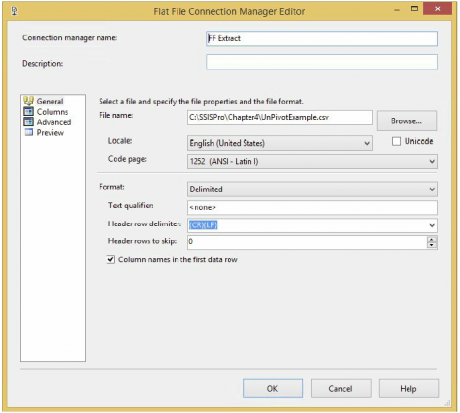
### Unpivot

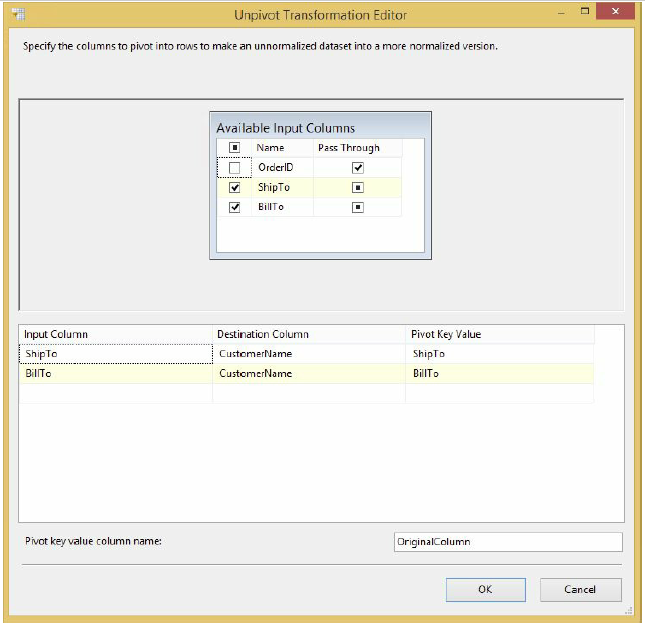
As you know, mainframe screens rarely conform to any normalized form. For example, a screen may show a Bill To Customer, a Ship To Customer, and a Dedicated To Customer field. Typically, the Data Source would store these three fields as three columns in a file (such as a virtual storage access system, or VSAM). Therefore, when you receive an extract from the mainframe you may have three columns, as shown in below ScreenShot.



Your goal is to load this file into a Customer table in SQL Server. You want a row for each customer in each column, for a total of six rows in the Customer table, as shown in the CustomerName and OrderID columns in below Screen Shoot.



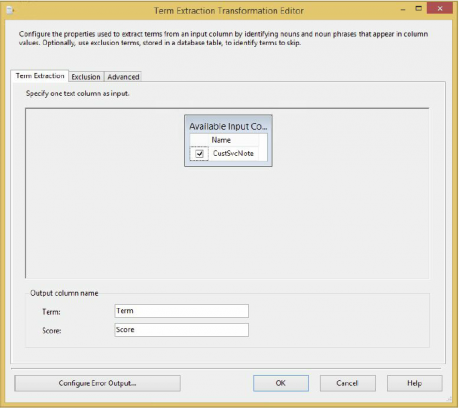




Term Extraction

If you have ever done some word and phrase analysis on websites for better search engine placement, you are familiar with the job that this transformation performs. The Term Extraction Transformation is a tool to mine the free-flowing text for the English word and phrase frequency. You can feed any text-based input stream into the transformation and it will output two columns: a text phrase and a statistical value for the phrase relative to the total input stream. The statistical values or scores that can be calculated can be as simple as a count of the frequency of the words and phrases, or they can be a little more complicated, such as the result of a formula named the TFIDF score. The TFIDF acronym stands for Term Frequency and Inverse Document Frequency, and it is a formula designed to balance the frequency of the distinct words and phrases relative to the total text sampled. If you’re interested, here’s the formula:

**TDIDF(of term or phrase) = (frequency of term) \*log((# rows in sample) /(#rows with term or phrase))**



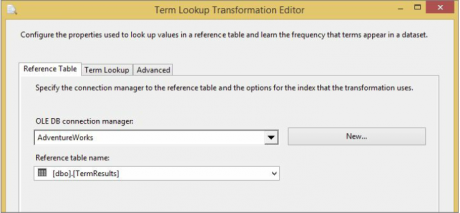
Even though we’re not going to set these tabs, the Exclusion tab enables you to specify noise words for the Term Extraction to ignore. The Advanced tab enables you to control how many times the word must appear before you output it as evidence. Close the Term Extraction Transformation Editor. Ignore the cautionary warnings about rows sent to error outputs. You didn’t configure an error location where bad rows should be saved, but it’s not necessary for this example.

The Advanced tab, which allows for some configuration of the task, is divided into four categories:

* Term Type: Settings that control how the input stream should be broken into bits called tokens. The Noun Term Type focuses on the transformation on nouns only, Noun Phrases extracts noun phrases, and Noun and Noun Phrases extracts both.
* Score Type: Choose to analyze words either by frequency or by a weighted frequency.
* Parameters: Frequency threshold is the minimum number of times a word or phrase must appear in tokens. The maximum length of the term is the maximum number of words that should be combined together for evaluation.
* Options: Check this option to consider case-sensitivity or leave it unchecked to disregard.

### Term Lookup

The Term Lookup Transformation uses the same algorithms and statistical models as the Term Extraction Transformation to break up an incoming stream into noun or noun phrase tokens, but it is designed to compare those tokens to a stored word list and output a matching list of terms and phrases with simple frequency counts. Now a strategy for working with both term based transformations should become clear. Periodically use the Term Extraction Transformation to mine the text data and generate lists of statistical phrases. Store these phrases in a word list, along with phrases that you think the term extraction process should identify. Remove any phrases that you don’t want to be identified. Use the Term Lookup Transformation to reprocess the text input to generate your final statistics. This way, you are generating statistics on known phrases of importance. A real-world application of this would be to pull out all the customer service notes that had a given set of terms or that mention a competitor’s name.



* Reference Table: This is where you configure the connection to the reference table. The Term Lookup Task should be used to validate each tokenized term that it finds in the input stream.
* Term Lookup: After selecting the lookup table, you map the field from the input stream to the reference table for matching.
* Advanced: This tab has one setting to check whether the matching is case sensitive.

### Synchronous versus Asynchronous Transformations

Transformations are divided into two main categories: synchronous and asynchronous. In SSIS, you want to ideally use all synchronous components.

**Synchronous transformations** are components such as the Derived Column and Data Conversion Transformations, where rows flow into memory buffers in the transformation, and the same buffers come out. No rows are held, and typically these transformations perform very quickly, with minimal impact to your Data Flow.

**Asynchronous transformations** can cause a block in your Data Flow and slow down your runtime. There are two types of asynchronous transformations: partially blocking and fully blocking.

**Partially blocking transformations**, such as the Union All, create new memory buffers for the output of the transformation.

**Fully blocking transformations**, such as the Sort and Aggregate Transformations, do the same thing but cause a full block of the data. In order to sort the data, SSIS must first see every single row of the data. If you have a 100MB file, then you may require 200MB of RAM in order to process the Data Flow because of a fully blocking transformation. These fully blocking transformations represent the single largest slowdown in SSIS and should be considered carefully in terms of any architecture decisions you must make.

**Synchronous components (non-blocking)**

A simple explanation of Synchronous transformation is that a synchronous transformation processes incoming rows and passes them on in the data flow one row at a time. The output is synchronous with input, meaning that it occurs at the same time. Therefore, to process a given row, the transformation does not need information about other rows in the data set.

The output of a synchronous component uses the same buffer as the input. Reusing the input buffer is possible because the output of a synchronous component always contains exactly the same number of records as the input. Synchronous (non-blocking) transformations always offer the highest performance.

Synchronous transformations are either stream-based or row-based. Streaming transformations are calculated in memory and do not require any data from outside resources to transform the data. These are the fastest transformations around. Row-based transformations run a little bit slower because they require calling a service or looking up data from another source to calculate their values.

We can summarize Synchronous Transformations as below

* The output is synchronous with the input
* Input and Output record count stays the same (rows entering = rows exiting the transformation)
* Does not need information about any other rows inside the data set
* Can operate on the same buffer
* Better performing than an asynchronous transformation
* Synchronous transformations are either stream-based or row-based.

**Asynchronous components (Semi-blocking and Fully-blocking)**

We use Asynchronous transformation when it is not possible to process each row independently of all other rows. An example is the Sort transformation, where the component has to process the complete set of rows to generate a sort output. The output of an asynchronous component uses a new buffer. It’s not possible to reuse the input buffer because an asynchronous component can have more or less output records then input records.

All source adapters are asynchronous, they create two buffers; one for the success output and one for the error output. All destination adapters, on the other hand, are synchronous.

**Semi-Blocking Asynchronous Transformations** require a subset of the data to be collected before they can be sent to the destination(s). The shape of the data can change. A subtotal or sampling of data may be extracted from the source(s).

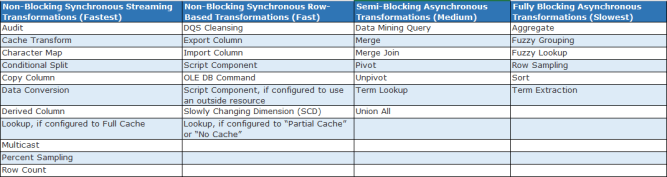
**Fully Blocking Asynchronous Transformations**are the slowest transformations. They require all the data to be pulled from the source(s) before they can be sent to the destination(s). All source data must be loaded into memory first – ouch! As much as we should try to avoid these, they can sometimes still be required, such as sorting data pulled from a flat-file source.

If there is more data than the memory available, it will use the **%TEMP%** directory to cache some of the data. If you want to use a different location, you can set the BufferTempStoragePath property of the Data Flow Task to point to a different folder location.

We can summarize Asynchronous Transformations as below

* Does not process rows independently in the dataset
* Rather than output rows as they are processed, the transformation must output data asynchronously, or at a different time
* Record counts usually change from input to output
* Must create a new buffer upon the output of the transformation
* Generally poorer performance than synchronous transformation
* Typically a Semi-Blocking or Blocking Transformation

**Below table categorizes various Synchronous and Asynchronous Transformations available in SSIS**

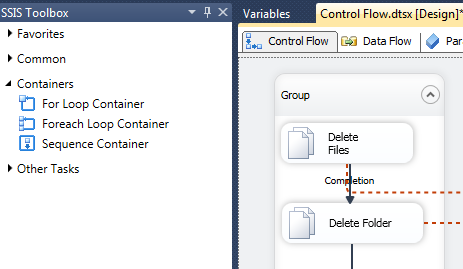
[](https://i2.wp.com/simplebiinsights.com/wp-content/uploads/2019/01/ssis_sysncvsashyn.png?ssl=1)

Below is the list of transformations under both categories, which will help you to design ETL and data warehouse system.

|  |  |  |
| --- | --- | --- |
| Synchronous Transformations | Asynchronous Transformations | |
| Partial blocking | Fully blocking |
| Audit | Data Mining Query | Aggregate |
| Character Map | Merge | Fuzzy Grouping |
| Conditional Split | Merge Join | Fuzzy Lookup |
| Copy Column | Pivot | Row Sampling |
| Data Conversion | Term Lookup | Sort |
| Derived Column | Union All | Term Extraction |
| Export Column | Unpivot |  |
| Import Column |  |  |
| Lookup |  |  |
| Multicast |  |  |
| OLE DB Command |  |  |
| Percent Sampling |  |  |
| Row Count |  |  |
| Script Component |  |  |
| Slowly Changing Dimension |  |  |

**SSIS Container**

* group related tasks together
* or define iterative processes.



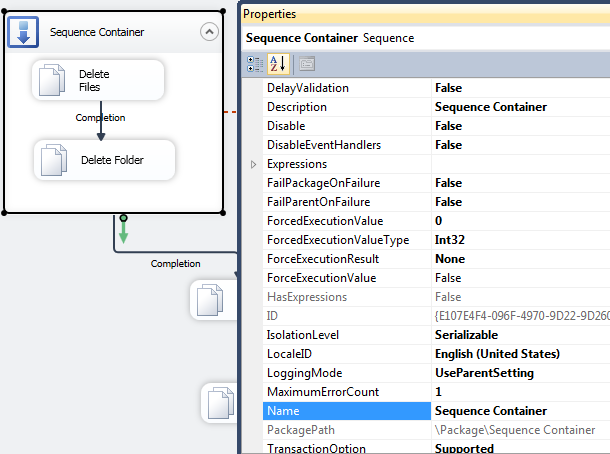
### 3.1 - Task : Each [control flow task](https://datacadamia.com/dit/ssis/control_flow#task) has its own implicit container

### 3.2 - Sequence

Sequence containers group tasks and other containers.

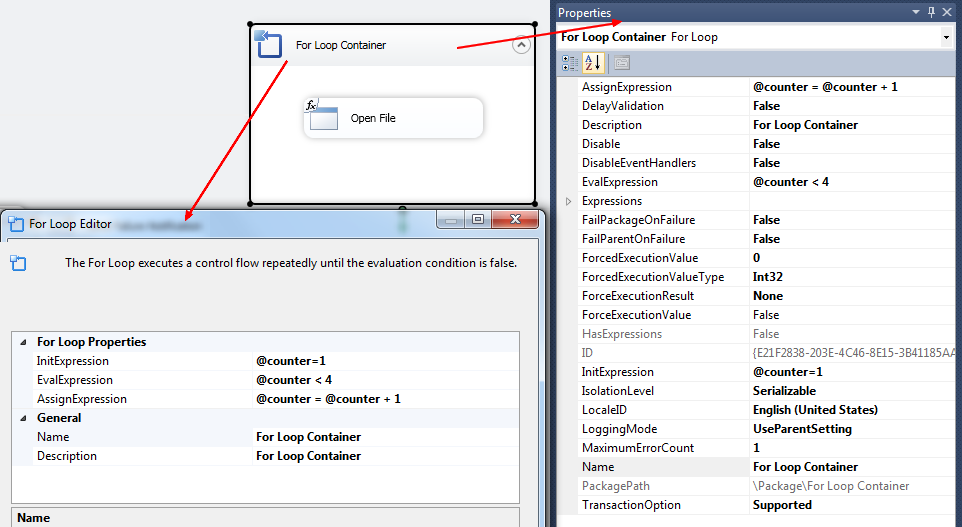
It enables to:

* set properties at the container level that apply to all elements within the container.
* Disable a logical subset of the package for debugging purposes.
* Create a scope for variables.
* Manage transactions at a granular level.



### 3.3 - For Loop

For Loop containers performs a loop until a condition is met.



The defined variable can be used with the following syntax:

@[User::counter]

### 3.4 - Foreach Loop

Foreach Loop containers performs a loop with an collection.

* File - Files in a folder.
* Item – A property collection for an SSIS object.
* ADO. A Recordset.
* ADO.NET Schema Rowset. Tables in a dataset or rows in a table.
* Variable – Array.
* Nodelist - Elements and attributes in an XML document.
* SMO – A collection of SQL Server Management Objects.

