|  |
| --- |
|  |
| SQL SERVER INTEGRATION SERVICES |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |

SSIS

[Overview of SSIS 4](#_Toc341351740)

[Typical Use of Integration Services 4](#_Toc341351741)

[SSIS Architecture 5](#_Toc341351742)

[Making Data Integration Approachable 7](#_Toc341351743)

[SQL Server Business Intelligence Development Studio 8](#_Toc341351744)

[Components of SSIS Package include 8](#_Toc341351745)

[Control Flow 8](#_Toc341351746)

[Container 9](#_Toc341351747)

[Tasks 10](#_Toc341351748)

[Precedence constraints 10](#_Toc341351749)

[Data Flow 10](#_Toc341351750)

[Data Flow Source 12](#_Toc341351751)

[Data Flow Transformation 13](#_Toc341351752)

[Data Flow Destination 15](#_Toc341351753)

[Connection Manager 16](#_Toc341351754)

[SSIS Tutorial: Working with Connection Managers 16](#_Toc341351755)

[Data Flow Transformation 17](#_Toc341351756)

[1. Aggregate : 19](#_Toc341351757)

[2. Audit : 19](#_Toc341351758)

[3. Character Map 20](#_Toc341351759)

[4. Derived Column 21](#_Toc341351760)

[5. Conditional split 21](#_Toc341351761)

[6. Copy Column: 23](#_Toc341351762)

[7. Data Conversion: 23](#_Toc341351763)

[8. Character Map: 24](#_Toc341351764)

[9. Import Column 25](#_Toc341351765)

[10. Export Column 29](#_Toc341351766)

[Exporting images with SSIS 29](#_Toc341351767)

[11. Loop Up: 32](#_Toc341351768)

[12. Merge: 33](#_Toc341351769)

[13. Merge Join: 33](#_Toc341351770)

[14. Multicast: 39](#_Toc341351771)

[15. Pivot 40](#_Toc341351772)

[16. Unpivot 42](#_Toc341351773)

[17. Row Count 44](#_Toc341351774)

[18. Row Sampling 44](#_Toc341351775)

[19. Percentage Sampling 45](#_Toc341351776)

[20. Slowly Changing Dimension 46](#_Toc341351777)

[21. Union All 48](#_Toc341351778)

[Control Flow Tasks in SSIS 49](#_Toc341351779)

[List of Database Control Flow Tasks 49](#_Toc341351780)

[SSIS maintenance plan tasks 49](#_Toc341351781)

[SSIS Container Tasks 50](#_Toc341351782)

[Bulk Insert Task 50](#_Toc341351783)

[Execute Process Task 51](#_Toc341351784)

[Execute SQL Task 52](#_Toc341351785)

[File System Task 58](#_Toc341351786)

[Script Task 59](#_Toc341351787)

[For Loop Container: 60](#_Toc341351788)

[ForEach Loop Container 62](#_Toc341351789)

[MULTIPLE FILES UPLOAD USING FOR EACH LOOP 67](#_Toc341351790)

[TRANSACTIONS IN SSIS 71](#_Toc341351791)

[(Web Service in SSIS) 79](#_Toc341351792)

[**Results** 83](#_Toc341351793)

[WMI Data Reader Task 84](#_Toc341351794)

[**Continuously watching files with WMI Event Watcher Task** 88](#_Toc341351795)

[MAINTENANCE PLAN TASKS 92](#_Toc341351796)

# Overview of SSIS

SQL Server Integration Services (SSIS) is a platform for building high performance data integration and workflow solutions. It allows creation of packages or SSIS packages which are made up of tasks that can move data from source to destination and alter it if required. SSIS is basically an ETL (Extraction, Transformation, and Load) tool whose main purpose is to do extraction, transformation and loading of data but it can be used for several other purposes,

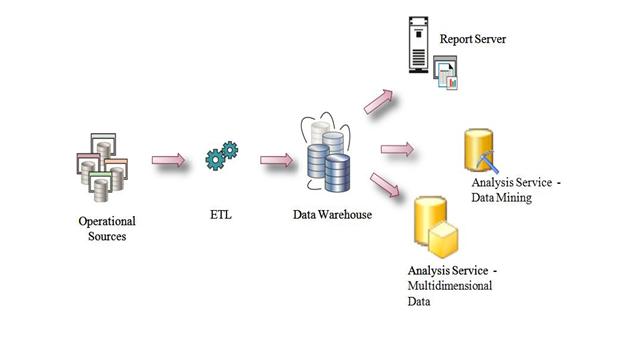
for example,

to automate maintenance of SQL Server databases,

update multidimensional cube data or

send e-mails detailing the status of the operation as defined by the user.

SSIS is a component of SQL Server 2005/2008/2012 and is the successor of DTS (Data Transformation Services) which had been in SQL Server 7.0/2000.



# Typical Use of Integration Services

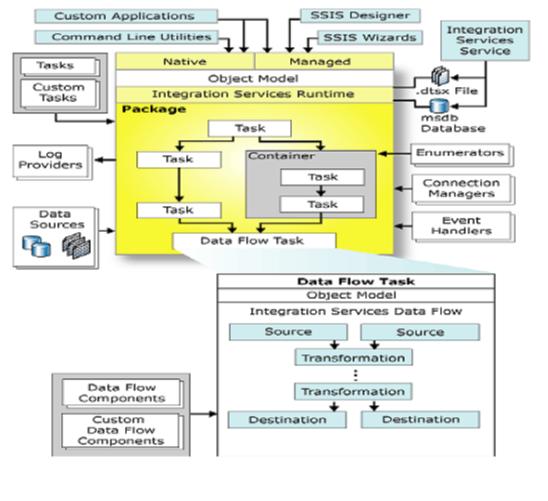
* Merging Data from Heterogeneous Data Stores
* Populating Data Warehouses and Data Marts
* Cleaning and Standardizing Data
* Building Business Intelligence into a Data Transformation Process
* Automating Administrative Functions and Data Loading

# SSIS Architecture

The SSIS architecture consists of two main components as given below:  
  
**SSIS Runtime Engine** – The SSIS runtime engine handles the control flow of a package. It saves the layout of packages, runs packages and provides support for logging, breakpoints, configuration, connections and transactions. The run-time engine is a parallel control flow engine that coordinates the execution of tasks or units of work within SSIS and manages the engine threads that carry out those tasks.  
  
The SSIS runtime engine executes the tasks inside a package in an orderly fashion. When the runtime engine encounters a data flow task in a package during execution it creates a data flow pipeline and lets that data flow task run in the pipeline.  
  
Note:  
The Integration Services service (a windows service) is not the same as the SSISruntime engine/service. It is not required if only the design and execute Integration Services packages are wanted. This windows service can be started to manage SSIS packages, for example to connect to multiple SSIS servers, start/stop package remotely/locally, manage the package store, import/export packages etc.  
  
**SSIS Data Flow Engine/Pipeline** – SSIS Data Flow Engine or Data Flow Pipeline or Transformation pipeline engine manages the flow of data from data sources, through transformations, and on to destination targets. When the Data Flow task executes, the SSIS data flow engine extracts data from one or more data sources, performs any necessary transformations on the extracted data and then delivers the data to one or more destinations.  
  
The Data flow engine is buffer oriented architecture, it pulls data from the source and stores it in a buffer (memory structure) and does the transformation in buffer/memory itself instead of processing on a row-by-row basis. The benefit of this in-memory processing is that processing is much faster as there is no need to physically copy/stage the data at each step of the data integration; the data flow engine manipulates data as it is transferred from source to destination.

Microsoft SQL Server Integration Services (SSIS) consist of three key parts:

* SSIS Services
* SSIS runtime engine and the runtime executables
* SSIS dataflow engine and the dataflow components



**Integration Services Service**

* Monitors running Integration Services packages and manages the storage of packages
* Integration Services object model
* Includes native and managed application programming interfaces (API) for accessing
* Integration Services tools, command-line utilities, and custom applications

**SSIS Run-time Engine & executables**

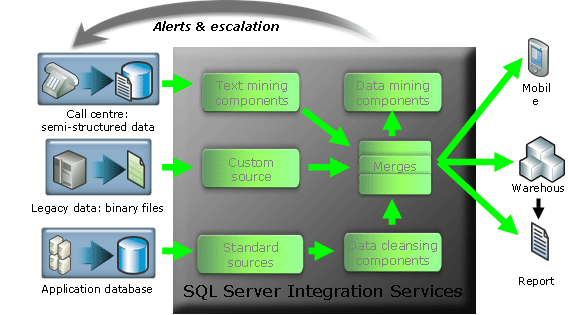
* Runs packages
* Supports logging, debugging, config, connections, & transactions
* SSIS Run-time executables
* Package, Containers, Tasks and Event Handlers

**SSIS Data-flow Engine & components**

* Provides In-Memory buffers to move data
* Calls Source Adaptors to files & DBs
* Provides Transformations to modify data
* Destination Adaptors to load data into data stores
* Components
* Source, Destination Adaptors & transformations

# Making Data Integration Approachable

The flexible and extensible architecture of SSIS allows it to address most of the technology challenges to data integration outlined earlier in this paper. As shown in Figure 11, SSIS eliminates (or at least minimizes) unnecessary staging. Because it performs complex data manipulation in a single pipeline operation, it is now possible to react to changes and patterns in the data fairly quickly, in a time frame that is actually meaningful for closing the loop and taking action. This is in contrast to traditional architectures that rely on data staging and that become impractical for closing the loop and taking meaningful action on data.



**Figure 11**

The extensible nature of SSIS makes it possible for organizations to leverage their existing investments in custom code for data integration by wrapping it as re-usable extensions to SSIS and by doing so to take full advantage of features such as logging, debugging, BI integration, etc. This greatly helps to overcome some of the organizational challenges outlined earlier in this paper.

The inclusion of SSIS in the SQL Server product makes the cost acquisition extremely reasonable as compared to other high-end data integration tools. Not only is the initial cost acquisition lowered, but via tight integration with Visual Studio and the rest of SQL Server BI tools, the cost of application development and maintenance is also significantly lowered in comparison to other similar tools. The extremely reasonable total cost of ownership (TCO) of SSIS (and the rest of SQL Server) makes enterprise-class data integration approachable to all segments of the market, taking it out of the exclusive domain of the largest (and richest) companies. At the same time, the architecture of SSIS is tuned to take advantage of modern hardware and to deliver performance and scale at the highest end of customer requirements. SSIS enables rich, scalable data integration to all customers, from the highest end enterprise to the small and medium business. In conjunction with the rest of the features in SQL Server, the Microsoft customer support infrastructure (ranging from broad, long beta testing, to rich online communities to premiere support contracts) and the consistency and integration with the rest of Microsoft product offerings, SSIS is truly a unique toolset that opens up new frontiers in data integration.

# SQL Server Business Intelligence Development Studio

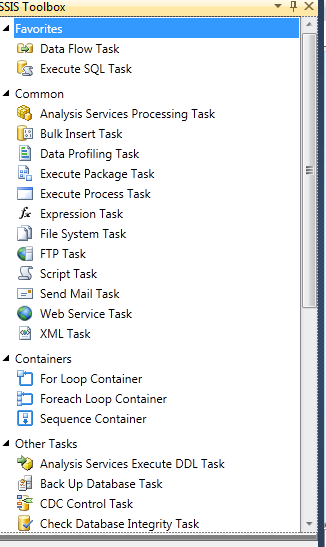
SQL Server Business Intelligence Development Studio (BIDS) allows users to create / edit SSIS packages using a drag-and-drop user interface. BIDS is very user friendly and allows you to drag-and-drop functionalities. There are a variety of elements that define a workflow in a single package. Upon package execution, the tool provides color-coded, real-time monitoring.

# Components of SSIS Package include

* Control Flow
* Data Flow

# Control Flow

Control flow deals with orderly processing of tasks, which are individual, isolated units of work that perform a specific action ending with a finite outcome (such that can be evaluated as either Success, Failure, or Completion). While their sequence can be customized by linking them into arbitrary arrangements with precedence constraints and grouping them together or repeating their execution in a loop with the help of containers, a subsequent task does not initiate unless its predecessor has completed.



**Elements of Control Flow include**

### Container

Containers provide structure in packages and services to tasks in the control flow. Integration Services include the following container types, for grouping tasks and implementing repeating control flows:

* The Foreach Loop container: It enumerates a collection and repeats its control flow for each member of the collection. The Foreach Loop Container is for situations where you have a collection of items and wish to use each item within it as some kind of input into the downstream flow.
* For Loop Container: It’s a basic container that provides looping functionality. A For loop contains a counter that usually increments (though it sometimes decrements), at which point a comparison is made with a constant value. If the condition evaluates to True, then the loop execution continues.
* Sequence Container: One special kind of container both conceptually and physically can hold any other type of container or Control Flow component. It is also called “container container”, or super container.

### Tasks

Tasks do the work in packages. Integration Services includes tasks for performing a variety of functions.

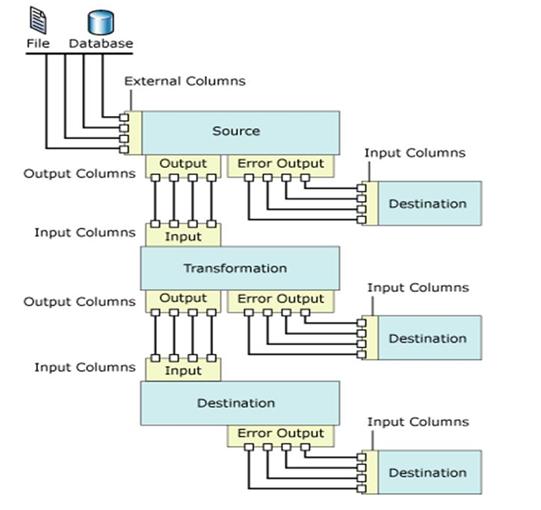
* The Data Flow task: It defines and runs data flows that extract data, apply transformations, and load data.
* Data preparation tasks: It copies files and directories, downloads files and data, saves data returned by Web methods, or works with XML documents.
* Workflow tasks: It communicates with other processes to run packages or programs, sends and receives messages between packages, sends e-mail messages, reads Windows Management Instrumentation (WMI) data, or watch for WMI events.
* SQL Server tasks: It accesses, copy, insert, delete, or modify SQL Server objects and data.
* Analysis Services tasks: It creates, modifies, deletes, or processes Analysis Services objects.
* Scripting tasks: It extends package functionality through custom scripts.
* Maintenance tasks: It performs administrative functions, such as backing up and shrinking SQL Server databases, rebuilding and reorganizing indexes, and running SQL Server Agent jobs.

### Precedence constraints

Precedence constraints connect containers and task in packages into an ordered control flow. You can control the sequence execution for tasks and containers, and specify conditions that determine whether tasks and containers run.

# Data Flow

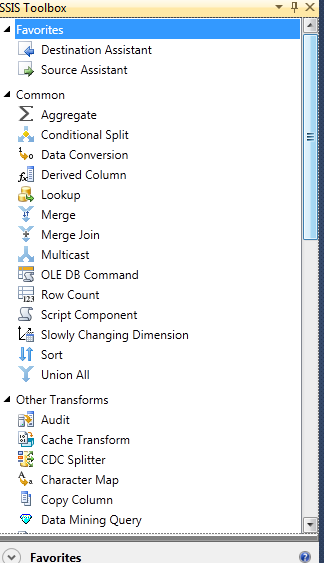
It’s processing responsibilities by employing the pipeline paradigm, carrying data record by record from its source to a destination and modifying it in transit by applying transformations. (There are exceptions to this rule, since some of them, such as Sort or Aggregate require the ability to view the entire data set before handing it over to their downstream counterparts). Items which are used to creating a data flow categorize into three parts.



**Elements of Data Flow include**

Elements of Data Flow are categorized into three parts:

1. **Data Flow Sources**: These elements are used to read data from different type of sources like (SQL Server, Excelsheet, etc.)
2. **Data Flow Transformations**: These elements are used to do process on data like (cleaning, adding new columns, etc.)
3. **Data Flow Destinations**: These elements are used save processed data into desired destination. (SQL Server, Excelsheet, etc.)



### Data Flow Source

Different items which can communicate in various types of source data are listed below:

* **DataReader Source**: The DataReader source uses an ADO.NET connection manager to read data from a DataReader and channel it into the Data Flow.
* **Excel Source**: The Excel source connects to an Excel file and, selecting content based on a number of configurable settings, supplies the Data Flow with data. The Excel Source uses the Excel connectionmanager to connect to the Excel file.
* **Flat File source**: Formats of which include CSV and fixed-width columns—are still popular. For many reasons, individual circumstances can dictate the use of CSV files over other formats,which is why the Flat File Source remains a popular Data Flow data source.
* **OLE DB Source**: The OLEDB Source is used when the data access is performed via an OLE DB provider. It’s a fairly simple data source type, and everyone is familiar with OLE DB connections.
* **Raw file Source**: The Raw File Source is used to import data that is stored in the SQL Server raw file format. It is a rapid way to import data that has perhaps been output by a previous package in the raw format.
* **XML Source**: The XML Source requires an XML Schema Definition (XSD) file, which is really the most important part of the component because it describes how SSIS should handle the XML document.

### Data Flow Transformation

Items in this category are used to perform different operations to make data in desired format.

* **Aggregate**: The Aggregate transformation component essentially encapsulates number of aggregate functions as part of the Data Flow, like Count, Count distinct, Sum, Average, Minimum, Maximum, Group By with respect to one or more columns.
* **Audit**: The Audit transformation exposes system variables to the Data Flow that can be used in the stream. This is accomplished by adding columns to the Data Flow output. When you map the required system variable or variables to the output columns, the system variables are introduced into the flow and can be used.
* **Character Map**: It performs string manipulations on input columns Like Lowercase, Uppercase, etc.
* **Conditional Split**: The Conditional Split task splits Data Flow based on a condition. Depending upon the results of an evaluated expression, data is routed as specified by the developer.
* **Copy Column**: The Copy Column task makes a copy of a column contained in the input-columns collection and appends it to the output-columns collection.
* **Data Conversion**: It is converting data from one type to another. Just like Type Casting.
* **Data Mining Query**: The data-mining implementation in SQL Server 2005 is all about the discovery of factually correct forecasted trends in data. This is configured within SSAS against one of the provided data-mining algorithms. The DMX query requests a predictive set of results from one or more such models built on the same mining structure. It can be a requirement to retrieve predictive information about the same data calculated using the different available algorithms.
* **Derived Column**: One or more new columns are appended to the output-columns collection based upon the work performed by the task, or the result of the derived function replaces an existing column value.
* **Export Column**: It is used to extract data from within the input stream and write it to a file. There’s one caveat: the data type of the column or columns for export must be DT\_TEXT, DT\_NTEXT, or DT\_IMAGE.
* **Fuzzy Grouping**: Fuzzy Grouping is for use in cleansing data. By setting and tweaking task properties, you can achieve great results because the task interprets input data and makes “intelligent” decisions about its uniqueness.
* **Fuzzy Lookup**: It uses a reference (or lookup) table to find suitable matches. The reference table needs to be available and selectable as a SQL Server 2005 table. It uses a configurable fuzzy-matching algorithm to make intelligent matches.
* **Import Column**: It is used to import data from any file or source.
* **Lookup**: The Lookup task leverages reference data and joins between input columns and columns in the reference data to provide a row-by-row lookup of source values. This reference data can be a table, view, or dataset.
* **Merge**: The Merge task combines two separate sorted datasets into a single dataset that is expressed as a single output.
* **Merge Join**: The Merge Join transform uses joins to generate output. Rather than requiring you to enter a query containing the join, however (for example SELECT x.columna, y.columnb FROM tablea x INNER JOIN tableb y ON x.joincolumna = y.joincolumnb), the task editor lets you set it up graphically.
* **Multicast**: The Multicast transform takes an input and makes any number of copies directed as distinct outputs. Any number of copies can be made of the input.
* **OLE DB Command**: The OLE DB command transform executes a SQL statement for each row in the input stream. It’s kind of like a high-performance cursor in many ways.
* **Percentage Sampling**: The Percentage Sampling transform generates and outputs a dataset into the Data Flow based on a sample of data. The sample is entirely random to represent a valid cross-section of available data.
* **Pivot**: The Pivot transformation essentially encapsulates the functionality of a pivot query in SQL. A pivot query demoralizes a normalized data set by “rotating” the data around a central point—a value.
* **Row Count**: The Row Count task counts the number of rows as they flow through the component. It uses a specified variable to store the final count. It is a very lightweight component in that no processing is involved, because the count is just a property of the input-rows collection.
* **Row Sampling**: The Row Sampling task, in a similar manner to the Percentage Sampling transform I discussed earlier, is used to create a (pseudo) random selection of data from the Data Flow. This transform is very useful for performing operations that would normally be executed against a full set of data held in a table. In very high-volume OLTP databases, however, this just isn’t possible at times. The ability to execute tasks against a representative subset of the data is a suitable and valuable alternative.
* **Sort**: This transform is a step further than the equivalent ORDER BY clause in the average SQL statement in that it can also strip out duplicate values.
* **Script Component**: The Script Component is using for scripting custom code in transformation. It can be used not only as a transform but also as a source or a destination component.
* **Slowly Changing Dimension**: The Slowly Changing Dimension task is used to maintain dimension tables held in data warehouses. It is a highly specific task that acts as the conduit between an OLTP database and a related OLAP database.
* **Term Extraction**: This transformation extracts terms from within an input column and then passes them into the Data Flow as an output column. The source column data type must be either DT\_STR or DT\_WSTR.
* **Term Lookup**: This task wraps the functionality of the Term Extraction transform and uses the values extracted to compare to a reference table, just like the Lookup transform.
* **Union All**: Just like a Union All statement in SQL, the Union All task combines any number of inputs into one output. Unlike in the Merge task, no sorting takes place in this transformation. The columns and data types for the output are created when the first input is connected to the task.
* **Unpivot**: This task essentially encapsulates the functionality of an unpivot query in SQL. An unpivot query increases the normalization of a less-normalized or denormalized data set by “rotating” the data back around a central point—a value.

### Data Flow Destination

Finally, processed data will saved at destination with the help of these items.

* **Data Mining Model Training**: It trains data-mining models using sorted data contained in the upstream Data Flow. The received data is piped through the SSAS data-mining algorithms for the relevant model.
* **DataReader Destination**: The results of an SSIS package executed from a .NET assembly can be consumed by connecting to the DataReader destination.
* **Dimension Processing**: Dimension Processing is another SSAS-related destination component. It is used to load and process an SSAS dimension.
* **Excel Destination**: The Excel Destination has a number of options for how the destination Excel file should be accessed. (Table or View, TableName or ViewName variable, and SQL Command)
* **Flat File Destination**: The Flat File Destination component writes data out to a text file in one of the standard flat-file formats: delimited, fixed width, fixed width with row delimiter.
* **OLE DB Destination**: The OLE DB Destination component inserts data into any OLE DB–compliant data source.
* **Partition Processing**: The Partition Processing destination type loads and processes an SSAS partition. In many ways, it is almost exactly the same as the Dimension Processing destination—at least in terms of configuration. You select or create an SSAS connection manager, choose the partition to process, and then map input columns to the columns in the selected partition.
* **Raw File Destination**: The Raw File Destination is all about raw speed. It is an entirely native format and can be exported and imported more rapidly than any other connection type, in part because the data doesn’t need to pass through a connection manager.
* **Recordset Destination**: The Recordset Destination creates an instance of an ActiveX Data Objects (ADO) Recordset and populates it with data from specified input columns.
* **SQL Server Destination**: The SQL Server Destination provides a connection to a SQL Server database. Selected columns from the input data are bulk inserted into a specified table or view. In other words, this destination is used to populate a table held in a SQL Server database.
* **SQL Server Mobile Destination**: The SQL Server Mobile Destination component is used to connect and write data to a SQL Server Mobile (or SQL Server Compact Edition) database.

# Connection Manager

### SSIS Tutorial: Working with Connection Managers

SSIS uses connection managers to integrate different data sources into packages. SSIS includes a wide variety of different connection managers that allow you to move data around from place to place.

Table 16-1 lists the available connection managers.

|  |  |
| --- | --- |
| **Connection Manager** | **Handles** |
| ADO Connection Manager | Connecting to ADO objects such as a Recordset. |
| ADO.NET Connection Manager | Connecting to data sources through an ADO.NET provider. |
| Analysis Services Connection Manager | Connecting to an Analysis Services database or cube. |
| Excel Connection Manager | Connecting to an Excel worksheet. |
| File Connection Manager | Connecting to a file or folder. |
| Flat File Connection Manager | Connecting to delimited or fixed width flat files. |
| FTP Connection Manager | Connecting to an FTP data source. |
| HTTP Connection Manager | Connecting to an HTTP data source. |
| MSMQ Connection Manager | Connecting to a Microsoft Message Queue. |
| Multiple Files Connection Manager | Connecting to a set of files, such as all text files on a particular hard drive. |
| Multiple Flat Files Connection Manager | Connecting to a set of flat files. |
| ODBC Connection Manager | Connecting to an ODBC data source. |
| OLE DB Connection Manager | Connecting to an OLE DB data source. |
| SMO Connection Manager | Connecting to a server via SMO. |
| SMTP Connection Manager | Connecting to a Simple Mail Transfer Protocol server. |
| SQL Server Mobile Connection Manager | Connecting to a SQL Server Mobile database. |
| WMI Connection Manager | Connecting to Windows Management Instrumentation data. |

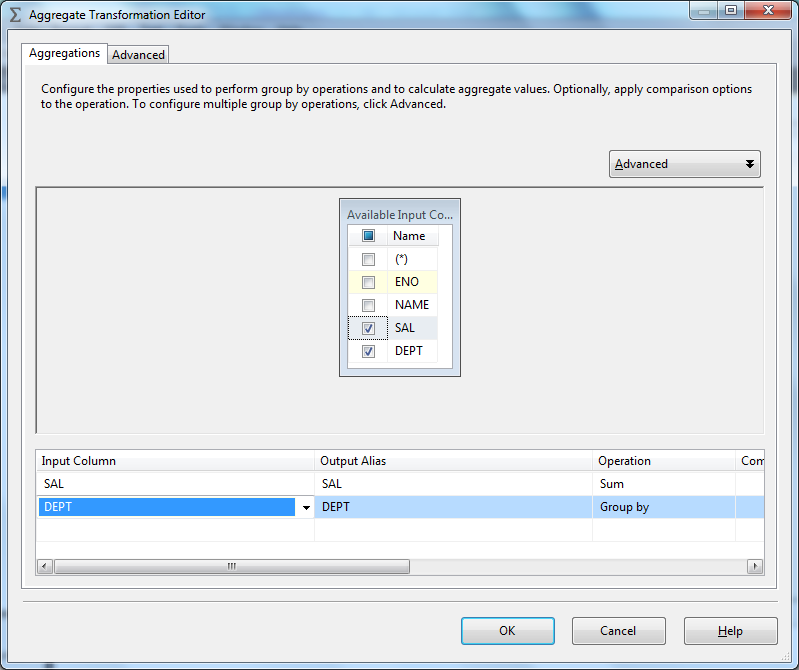
### Data Flow Transformation

Items in this category are used to perform different operations to make data in desired format.

* **Aggregate**: The Aggregate transformation component essentially encapsulates number of aggregate functions as part of the Data Flow, like Count, Count distinct, Sum, Average, Minimum, Maximum, Group By with respect to one or more columns.
* **Audit**: The Audit transformation exposes system variables to the Data Flow that can be used in the stream. This is accomplished by adding columns to the Data Flow output. When you map the required system variable or variables to the output columns, the system variables are introduced into the flow and can be used.
* **Character Map**: It performs string manipulations on input columns Like Lowercase, Uppercase, etc.
* **Conditional Split**: The Conditional Split task splits Data Flow based on a condition. Depending upon the results of an evaluated expression, data is routed as specified by the developer.
* **Copy Column**: The Copy Column task makes a copy of a column contained in the input-columns collection and appends it to the output-columns collection.
* **Data Conversion**: It is converting data from one type to another. Just like Type Casting.
* **Data Mining Query**: The data-mining implementation in SQL Server 2005 is all about the discovery of factually correct forecasted trends in data. This is configured within SSAS against one of the provided data-mining algorithms. The DMX query requests a predictive set of results from one or more such models built on the same mining structure. It can be a requirement to retrieve predictive information about the same data calculated using the different available algorithms.
* **Derived Column**: One or more new columns are appended to the output-columns collection based upon the work performed by the task, or the result of the derived function replaces an existing column value.
* **Export Column**: It is used to extract data from within the input stream and write it to a file. There’s one caveat: the data type of the column or columns for export must be DT\_TEXT, DT\_NTEXT, or DT\_IMAGE.
* **Fuzzy Grouping**: Fuzzy Grouping is for use in cleansing data. By setting and tweaking task properties, you can achieve great results because the task interprets input data and makes “intelligent” decisions about its uniqueness.
* **Fuzzy Lookup**: It uses a reference (or lookup) table to find suitable matches. The reference table needs to be available and selectable as a SQL Server 2005 table. It uses a configurable fuzzy-matching algorithm to make intelligent matches.
* **Import Column**: It is used to import data from any file or source.
* **Lookup**: The Lookup task leverages reference data and joins between input columns and columns in the reference data to provide a row-by-row lookup of source values. This reference data can be a table, view, or dataset.
* **Merge**: The Merge task combines two separate sorted datasets into a single dataset that is expressed as a single output.
* **Merge Join**: The Merge Join transform uses joins to generate output. Rather than requiring you to enter a query containing the join, however (for example SELECT x.columna, y.columnb FROM tablea x INNER JOIN tableb y ON x.joincolumna = y.joincolumnb), the task editor lets you set it up graphically.
* **Multicast**: The Multicast transform takes an input and makes any number of copies directed as distinct outputs. Any number of copies can be made of the input.
* **OLE DB Command**: The OLE DB command transform executes a SQL statement for each row in the input stream. It’s kind of like a high-performance cursor in many ways.
* **Percentage Sampling**: The Percentage Sampling transform generates and outputs a dataset into the Data Flow based on a sample of data. The sample is entirely random to represent a valid cross-section of available data.
* **Pivot**: The Pivot transformation essentially encapsulates the functionality of a pivot query in SQL. A pivot query demoralizes a normalized data set by “rotating” the data around a central point—a value.
* **Row Count**: The Row Count task counts the number of rows as they flow through the component. It uses a specified variable to store the final count. It is a very lightweight component in that no processing is involved, because the count is just a property of the input-rows collection.
* **Row Sampling**: The Row Sampling task, in a similar manner to the Percentage Sampling transform I discussed earlier, is used to create a (pseudo) random selection of data from the Data Flow. This transform is very useful for performing operations that would normally be executed against a full set of data held in a table. In very high-volume OLTP databases, however, this just isn’t possible at times. The ability to execute tasks against a representative subset of the data is a suitable and valuable alternative.
* **Sort**: This transform is a step further than the equivalent ORDER BY clause in the average SQL statement in that it can also strip out duplicate values.
* **Script Component**: The Script Component is using for scripting custom code in transformation. It can be used not only as a transform but also as a source or a destination component.
* **Slowly Changing Dimension**: The Slowly Changing Dimension task is used to maintain dimension tables held in data warehouses. It is a highly specific task that acts as the conduit between an OLTP database and a related OLAP database.
* **Term Extraction**: This transformation extracts terms from within an input column and then passes them into the Data Flow as an output column. The source column data type must be either DT\_STR or DT\_WSTR.
* **Term Lookup**: This task wraps the functionality of the Term Extraction transform and uses the values extracted to compare to a reference table, just like the Lookup transform.
* **Union All**: Just like a Union All statement in SQL, the Union All task combines any number of inputs into one output. Unlike in the Merge task, no sorting takes place in this transformation. The columns and data types for the output are created when the first input is connected to the task.
* **Unpivot**: This task essentially encapsulates the functionality of an unpivot query in SQL. An unpivot query increases the normalization of a less-normalized or denormalized data set by “rotating” the data back around a central point—a value.

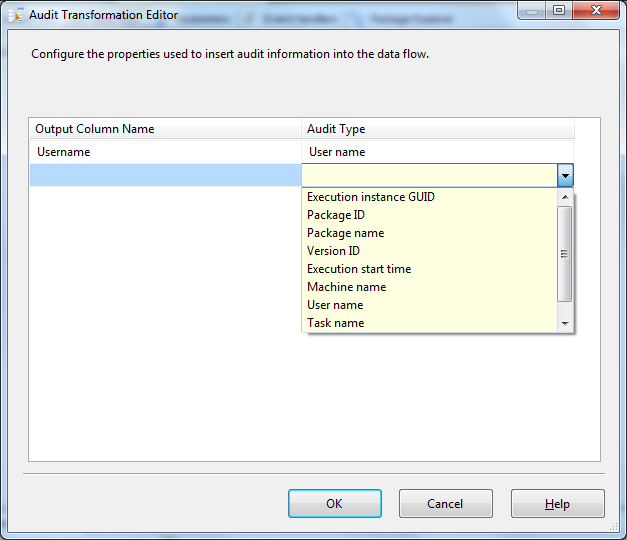
# Aggregate :

The Aggregate transformation component essentially encapsulates number of aggregate functions as part of the Data Flow, like Count, Count distinct, Sum, Average, Minimum, Maximum, Group By with respect to one or more columns.



# Audit :

The Audit transformation exposes system variables to the Data Flow that can be used in the stream. This is accomplished by adding columns to the Data Flow output. When you map the required system variable or variables to the output columns, the system variables are introduced into the flow and can be used.

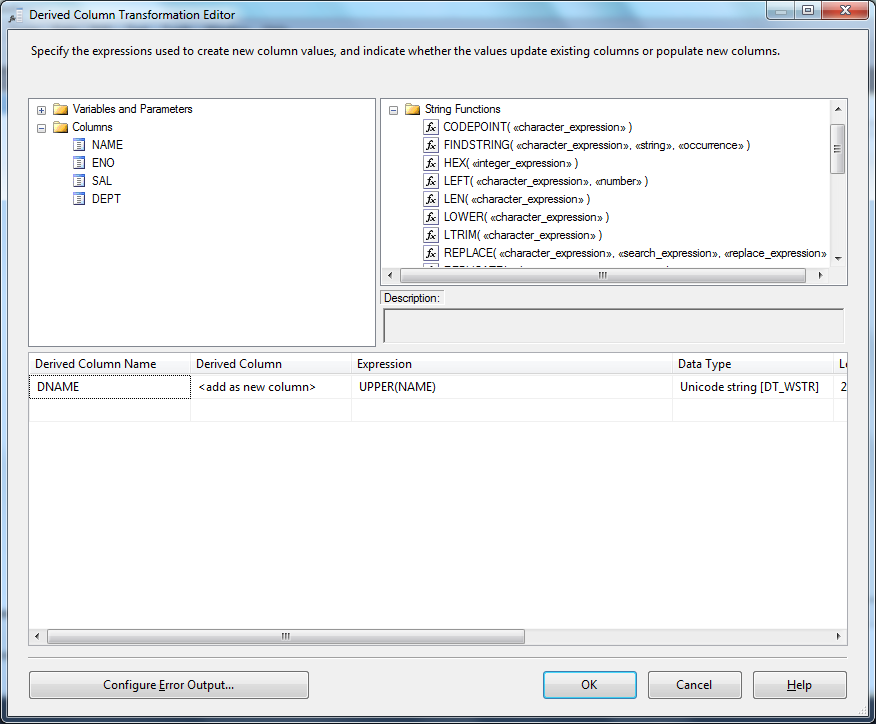


# Character Map

It performs string manipulations on input columns Like Lowercase, Uppercase, etc.

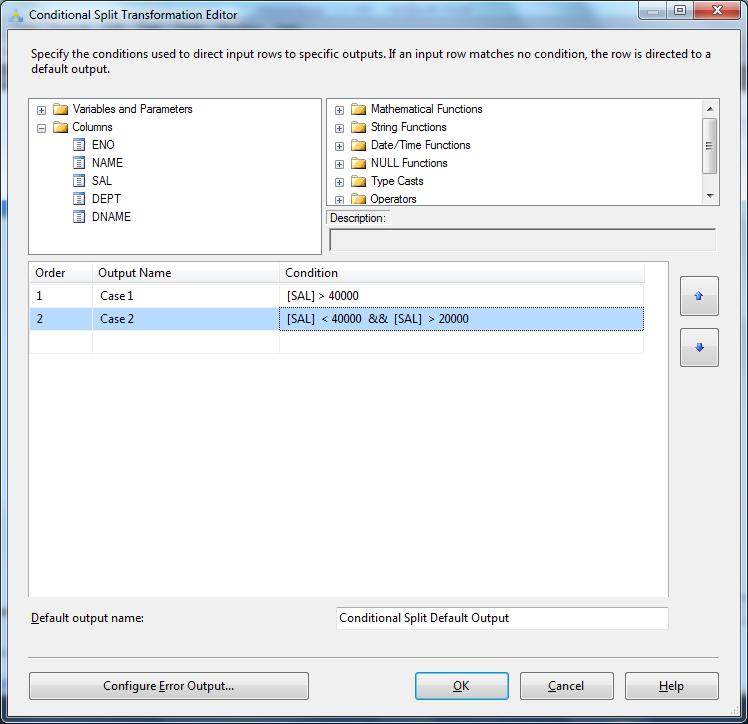
# 

# Derived Column



# Conditional split

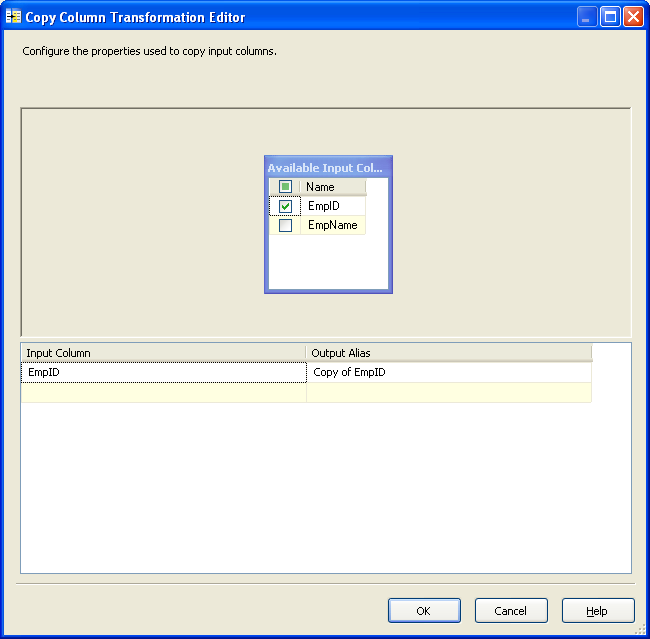
Add a conditional split and double click on that to configure as below



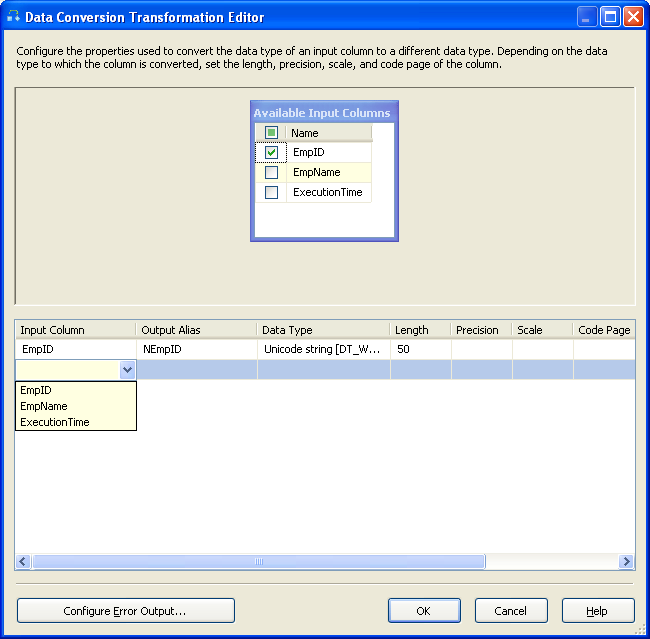
Note : Default Conditional Output : All the remaining rows will be go into default output

# Copy Column:

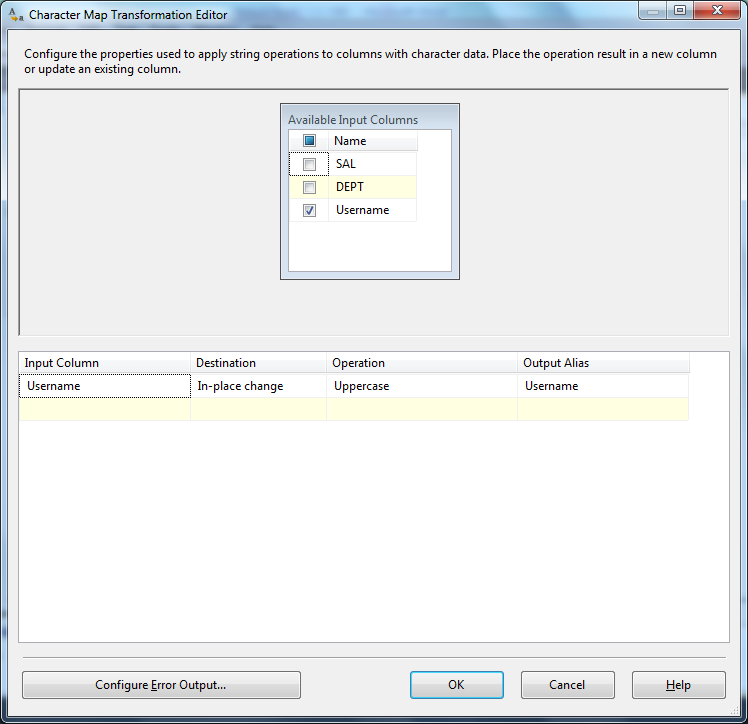
Creates a copy of the column



# Data Conversion:



# Character Map:



# Import Column

**Case**  
I want to import files (images) with SSIS to a SQL Server table. And export those images to an other folder.  
  
**Solution**  
The easiest way to import files/images with SSIS, is with the Import Column Transformation.   
  
**Importing images with Import Column**  
For the first example I have created a CSV file with filepaths of images named bikes.csv:

|  |
| --- |
| C:\Users\Joost\Pictures\MountainBike1.jpg C:\Users\Joost\Pictures\MountainBike2.jpg C:\Users\Joost\Pictures\MountainBike3.jpg |

My target table looks like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE [dbo].[ImageStore](   [id] [int] IDENTITY(1,1) NOT NULL,   [imageName] [nvarchar](50) NULL,   [imageFile] [varbinary](max) NULL  ) ON [PRIMARY] |

Note: If you want to store images in a SQL Server table use the VARBINARY datatype because the IMAGE datatype is due to be removed from SQL Server.  
  
**A1) Flat File Source**  
Go to your Data Flow and use a Flat File Source to read the CSV file from above.

|  |
| --- |
| <http://4.bp.blogspot.com/-QfUl8guYYto/TjOnAlofHXI/AAAAAAAAAW8/y5Fi0Yhddcc/s1600/importimages1.jpg> |
| Import file with only one column: ImagePath |

**A2) Get filename from path**  
I also want to store the original filename in my table. I use an expression in a Derived Column to get the filename from the filepath. Add a Derived Column and add a new column named FileName. The expression is:  
**RIGHT(ImagePath,FINDSTRING(REVERSE(ImagePath),"\\",1) - 1)**

|  |
| --- |
| [http://1.bp.blogspot.com/-Jay4JbonCbs/TjOpmAsKo_I/AAAAAAAAAXA/SeV1D9sqmOI/s320/importimages2.jpg](http://1.bp.blogspot.com/-Jay4JbonCbs/TjOpmAsKo_I/AAAAAAAAAXA/SeV1D9sqmOI/s1600/importimages2.jpg) |
| Extract filename from filepath |

**A3) Import Column**  
Now the actual importing of the image files into SSIS. Add a Import Column to the Control Flow.

|  |
| --- |
| [http://3.bp.blogspot.com/-IRLGMNiWhVw/TjOq-GP4XtI/AAAAAAAAAXE/Fb4CNAeDgYI/s320/importimages3.jpg](http://3.bp.blogspot.com/-IRLGMNiWhVw/TjOq-GP4XtI/AAAAAAAAAXE/Fb4CNAeDgYI/s1600/importimages3.jpg) |
| Import Column |

**A4) Input Columns**  
Edit the Import Column transformation and go to the Input Columns tab. Select the column that contains the filepath of the images.

|  |
| --- |
| [http://4.bp.blogspot.com/-Lsv5kzOT0F4/TjOs45SnOCI/AAAAAAAAAXI/EPZbjbkPJuQ/s320/importimages4.jpg](http://4.bp.blogspot.com/-Lsv5kzOT0F4/TjOs45SnOCI/AAAAAAAAAXI/EPZbjbkPJuQ/s1600/importimages4.jpg) |
| Select column |

**A5) Input and Output Properties**  
Go to the Input and Output Properties tab and add a new **output column** of the type DT\_IMAGE and give it the name ImageData. Remember the generated ID because you need it for the next step.

|  |
| --- |
| [http://4.bp.blogspot.com/-cO6tZJwO7Xk/TjO0K02Q8mI/AAAAAAAAAXQ/tut8QgMCZVY/s320/importimages5.jpg](http://4.bp.blogspot.com/-cO6tZJwO7Xk/TjO0K02Q8mI/AAAAAAAAAXQ/tut8QgMCZVY/s1600/importimages5.jpg) |
| New output column: DT\_IMAGE |

**A6) Connect input and output**  
Now go to the Input Columns. Select the input column and change the **FileDataColumnId** to the ID of the output column (24 in my case).

|  |
| --- |
| [http://1.bp.blogspot.com/-r22WEIbqkVE/TjO0w7NLlXI/AAAAAAAAAXU/1WjE-l-SSHI/s320/importimages6.jpg](http://1.bp.blogspot.com/-r22WEIbqkVE/TjO0w7NLlXI/AAAAAAAAAXU/1WjE-l-SSHI/s1600/importimages6.jpg) |
| FileDataColumnId |

**A7) OLE DB Destination**  
Now you're ready to import the data in the database table. Add an OLE DB Destination and select the table ImageStore. Connect FileName to imageName and ImageData to imageFile.

|  |
| --- |
| [http://1.bp.blogspot.com/-3CU3CDyMZjU/TjPV6aLHMDI/AAAAAAAAAXY/kBNWm2D_5nU/s320/importimages7.jpg](http://1.bp.blogspot.com/-3CU3CDyMZjU/TjPV6aLHMDI/AAAAAAAAAXY/kBNWm2D_5nU/s1600/importimages7.jpg) |
| OLE DB Destination |

**A8) The result**  
Run the package and watch the table for the result.

# Export Column

### Exporting images with SSIS

**Case**  
In a previous article I showed you how to import images (or other files) with SSIS into a SQL Server table. This article shows you how to get them out again with the Export Column Transformation.  
  
**Solution**  
This example assumes that you have a filled table named ImageStore which I created in the previous article.

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE TABLE [dbo].[ImageStore]( [id] [int] IDENTITY(1,1) NOT NULL,   [imageName] [nvarchar](50) NULL,   [imageFile] [varbinary](max) NULL ) ON [PRIMARY] |

**1) Variables**  
Add a string variable to the package to store the path of the export folder.  
I called mine **ExportFolder**. Fill it with a value like: C:\Users\Joost\Pictures\Export\

|  |
| --- |
| <http://1.bp.blogspot.com/-uiXqk8KRHaY/TjP8McnhGXI/AAAAAAAAAX4/IBNB99P0SN8/s1600/importimages15.jpg> |
| Variable to store the export folder |

**2) OLE DB Source**  
Add an OLE DB Source component that reads from the table ImageStore. We need the columns imageName and imageFile.

|  |
| --- |
| <http://1.bp.blogspot.com/-lgxKSvi3eUw/TjQAlDryE2I/AAAAAAAAAX8/mJ4SUIq3Pb8/s1600/importimages16.jpg> |
| OLE DB Source component |

**3) Derived Column**  
With the folderpath from the variable and the filename from the column imageName we create an export filepath which is needed for the Export Column component. Name it NewFilePath.

|  |
| --- |
| [http://3.bp.blogspot.com/-HxUcvMkXiLw/TjQBU9fOhMI/AAAAAAAAAYA/IcgUl9cFxnQ/s400/importimages17.jpg](http://3.bp.blogspot.com/-HxUcvMkXiLw/TjQBU9fOhMI/AAAAAAAAAYA/IcgUl9cFxnQ/s1600/importimages17.jpg) |
| Expression: @[User::ExportFolder] + imageName |

**4) Export Column**  
Add an Export Column component and select the imageFile column as **Extract Column** and the NewFilePath column as **File Path Column**.

|  |
| --- |
| [http://2.bp.blogspot.com/-DpQz6ybk7yE/TjQCaFXqHGI/AAAAAAAAAYE/6qhXBocO3JM/s320/importimages18.jpg](http://2.bp.blogspot.com/-DpQz6ybk7yE/TjQCaFXqHGI/AAAAAAAAAYE/6qhXBocO3JM/s1600/importimages18.jpg) |
| Export Column |

**5) The result**  
Run the package and watch the folder for the result.

|  |
| --- |
| [http://2.bp.blogspot.com/-lmO12NK9PNg/TjRWeTqs5pI/AAAAAAAAAYI/Exb5EvfItp4/s400/importimages19.jpg](http://2.bp.blogspot.com/-lmO12NK9PNg/TjRWeTqs5pI/AAAAAAAAAYI/Exb5EvfItp4/s1600/importimages19.jpg) |
| The result |

Note: it also works with other file types, not just images.

# Loop Up:

Joins 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.

Obtaining additional data columns. For example, the majority of employee demographic information might be available in a flat file, but other data such as department where each employee works, their employment start date and job grade might be available from a table in relational database.

SOURCETABLE DATA: source table

naturalkey modifydate

-------------------------------- -----------------------

a 2006-01-01 00:00:00

b 2012-01-15 15:32:00

b 2012-01-15 15:33:00

LOOKUPDIMENSION DATA : lookup table.

naturalkey roweffectivedate rowexpireddate surrogatekey

-------------------------------- ----------------------- ----------------------- ------------

a 2005-12-11 00:00:00 2006-02-28 00:00:00 1

LOOK UP OUTPOUT

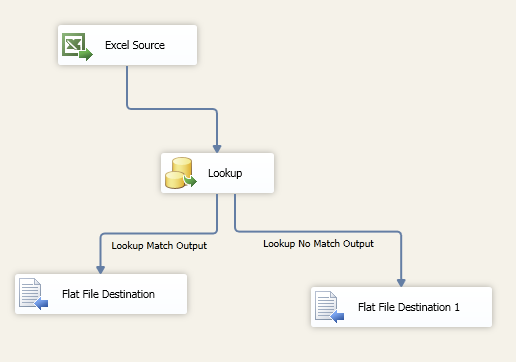
a,2006-01-01 00:00:00,2005-12-11 00:00:00,2006-02-28 00:00:00,1

NOTLOOKUP OUTPUT

-------------------

b,2012-01-15 15:32:00

b,2012-01-15 15:33:00



# Merge:

Merges 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. For example you can merge VARCHAR (30) and VARCHAR (50) columns. You cannot merge INT and DATETIME columns.

Combining the columns from multiple data sources into a single row set prior to populating a dimension table in a data warehouse. Using Merge transformation saves the step of having a temporary staging area. With prior versions of SQL Server you had to populate the staging area first if your data warehouse had multiple transactional data sources.

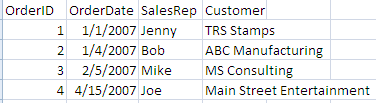
# Merge Join:

Joins two sorted inputs using INNER JOIN, LEFT OUTER JOIN or FULL OUTER JOIN algorithm. You can specify columns used for joining inputs.

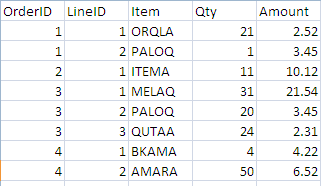
Combining the columns from multiple data sources into a single row set prior to populating a dimension table in a data warehouse. Using Merge Join transformation saves the step of having a temporary staging area. With prior versions of SQL Server you had to populate the staging area first if your data warehouse had multiple transactional data sources.   
  
Note that Merge and Merge Join transformations can only combine two data sets at a time. However, you could use multiple Merge Join transformations to include additional data sets.

Here we have two source files an OrderHeader and an OrderDetail.  We want to merge this data and load into one table in SQL Server called Orders.

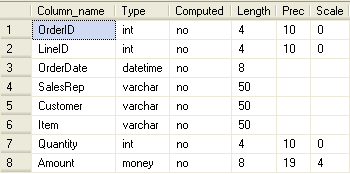
OrderHeader source file.



OrderDetail source file



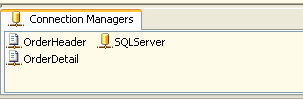
Orders table



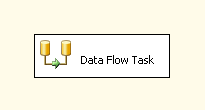
Building the SSIS Package

First create a new SSIS package and create the three Connections that we will need.

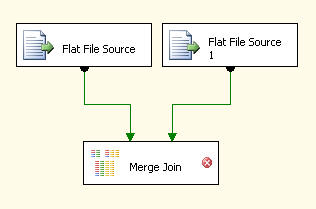
1. Flat File Source 1 - OrderHeader
2. Flat File Source 2 - OrderDetail
3. OLE DB Destination - SQLServer



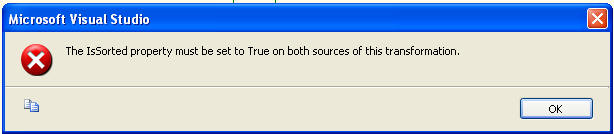
Then add a DATA FLOW task.



Next we need to build our load from these two flat file sources and then use the MERGE JOIN task to merge the data.  So the Data Flow steps would look something like this.

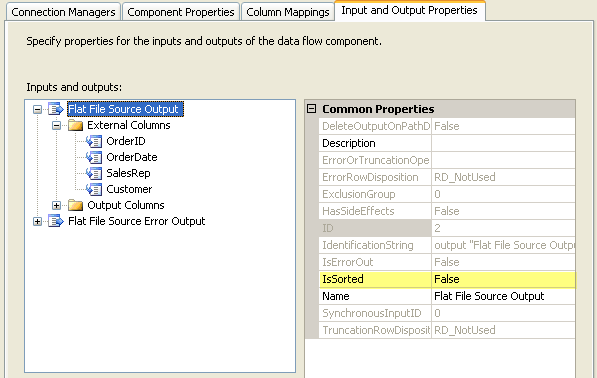


At this point if you try to edit the MERGE JOIN task you will get the below error.  The reason for this is because the data needs to be sorted for the MERGE JOIN task to work.  We will look at two options for handling this sorting need.

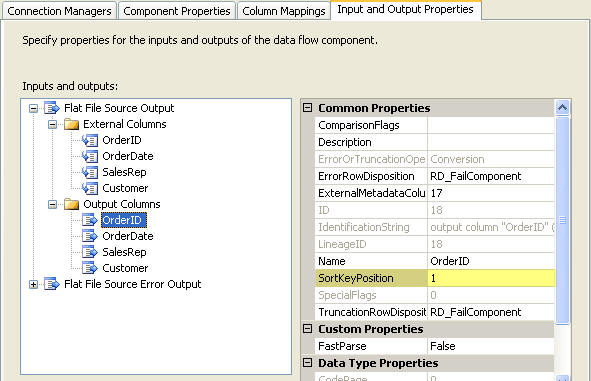


Option #1 - Data is presorted prior to loading the data.

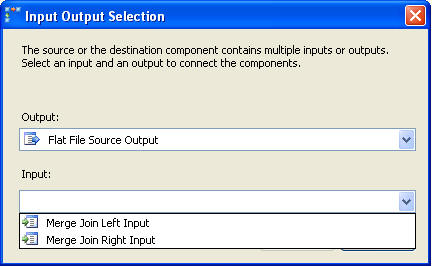
Let's assume that are data is sorted prior to loading.  We therefore need to tell SSIS this is the case as well as show which column the data is sorted on.  First if you right click on "Flat File Source" and select the "Show Advanced Editor".  On the Input and Output Properties tab you need to change the "IsSorted" to True for both of the Flat File Sources.



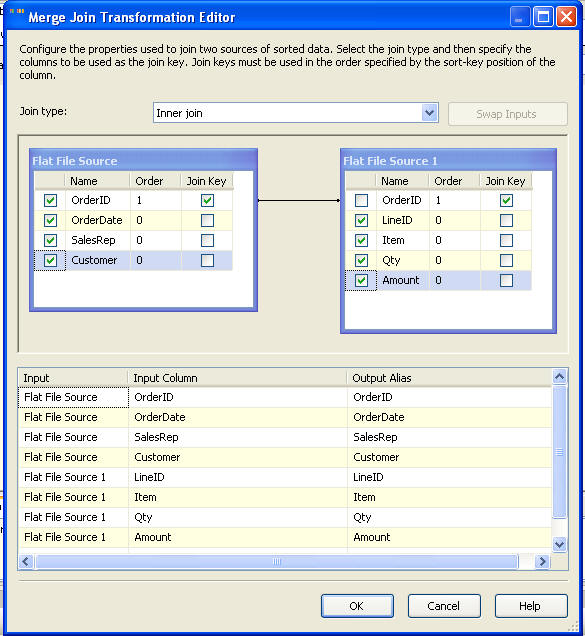
Next you need to let SSIS know which column is the SortKey.  Here we are specifying the OrderID column.  This also needs to be done for both of the flat file sources.



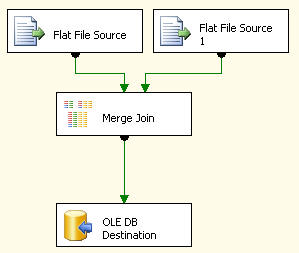
Once this is complete you will be able to move on with the setup and select the input process as shown below.



From here you can select the columns that you want to have for output as well as determine what type of join you want to employ between these two files.



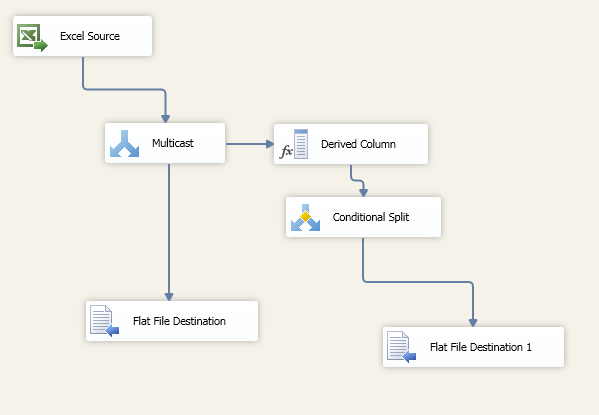
Lastly you would need to add your OLE Destination, select the table and map the columns to finish the process.



# Multicast:

The Multicast transformation splits a single data file into multiple files. Multicasts are handy if you need to send multiple copies of data to different destinations or perform different subsequent transformations. The transformation has one input and multiple outputs. There is very little configuration needed or required, all you can do is add a name and description for each output once you have created it

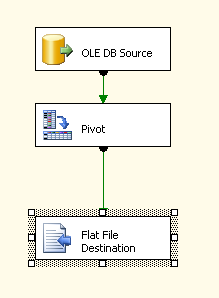
Similar to the conditional split transformation, but the entire data set is piped to multiple destinations.

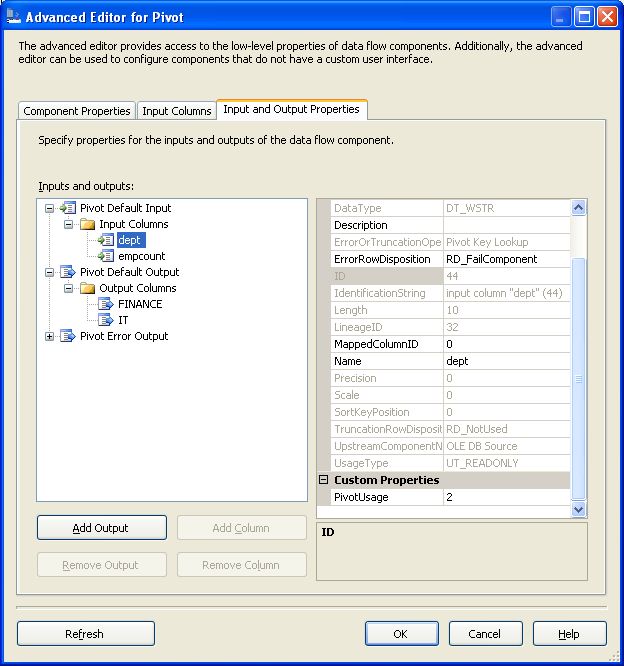


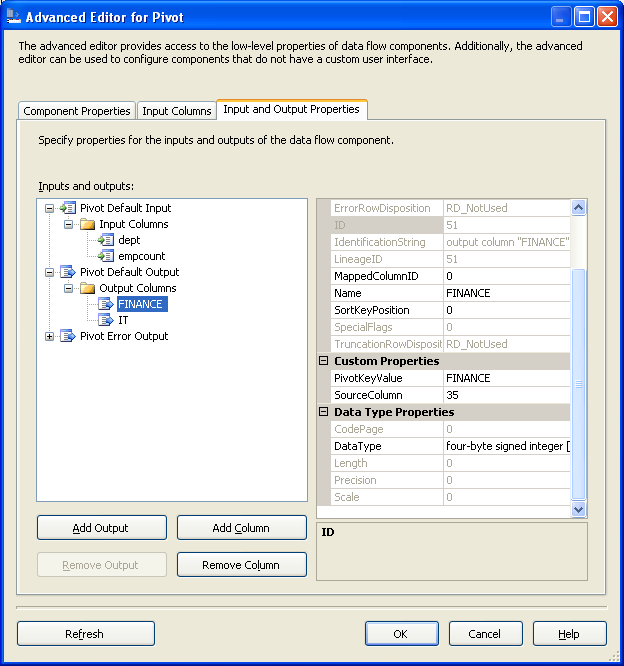
# Pivot

For input columns, we need to configure the pivot usage attribute.

|  |  |
| --- | --- |
| **Option** | **Description** |
| 0 | The column is not pivoted, and the column values are passed through to the transformation output. |
| 1 | The column is part of the set key that identifies one or more rows as part of one set. All input rows with the same set key are combined into one output row. |
| 2 | The column is a pivot column. At least one column is created from each column value. |
| 3 | The values from this column are placed in columns that are created as a result of the pivot. |

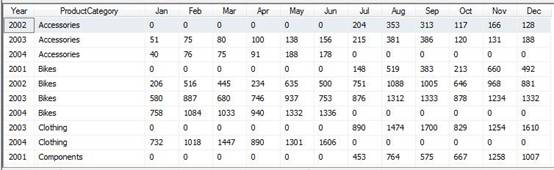




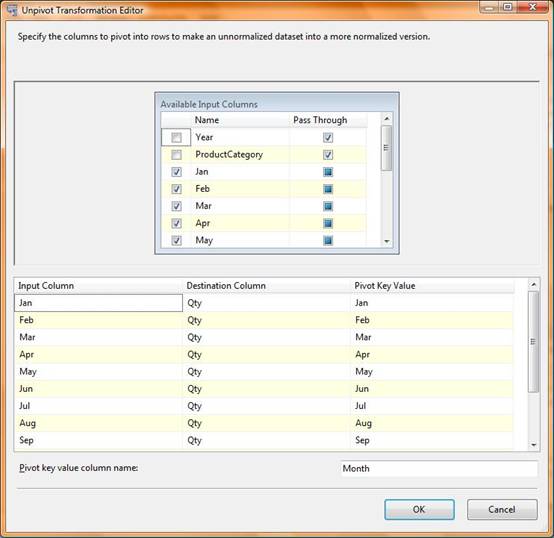


# Unpivot

First, we will add a package to the existing SSIS solution. Then we will add a Flat File Connection Manager and point the pivoteddata.txt file to it. Finally, we will add a Flat file source and point the previously added Flat File Connection Manager to it. Below depicts the data output you should see from the Flat File Source.



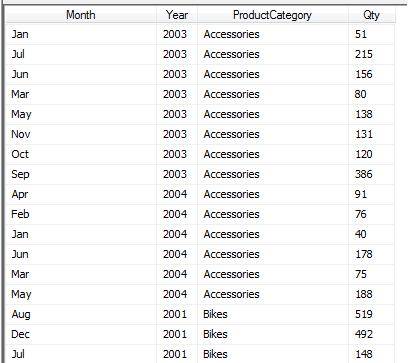
Our next steps is to do the unpivoting. Unlike the pivot configuration, the unpivot configuration is relatively simple.



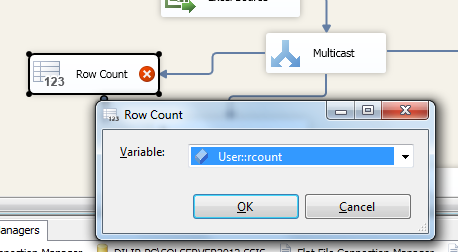
You will need to select the pivot columns, which in this case the pivot columns are Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov and Dec. The pivot column name will be in column titled Month, which you can configure at the bottom of the screen. The values of these pivoted key columns (Jan, Feb etc.) into column called Qty which can be configured at destination column.

In this unpivot example there is a small issue. Here the columns which have a value of “0” in the Qty column will be transformed into rows. For example, the Qty of accessories in January of 2002 is ‘0’ and there will be and unnecessary row for that record. To alleviate these records, a conditional split with a simple condition is used. The condition is Qty > 0, which will eliminate any row with “0” in the Qty field.

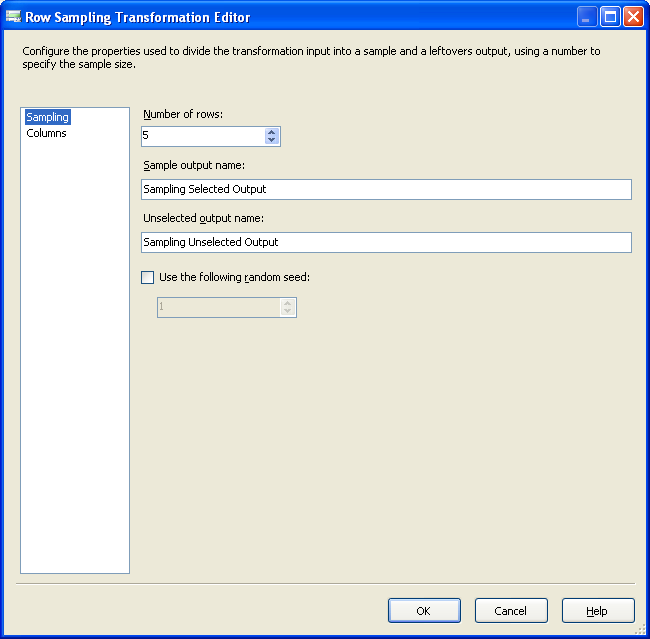
Below is the output of the unpivoted transformation.



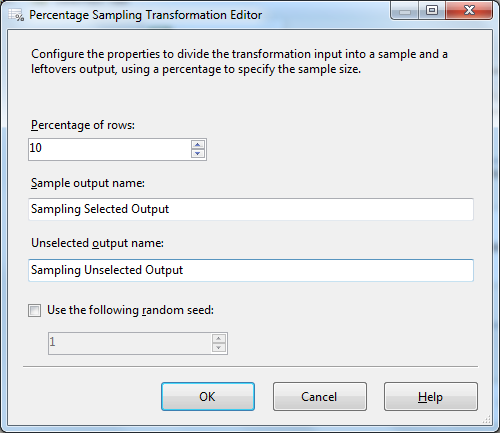
# Row Count



# Row Sampling



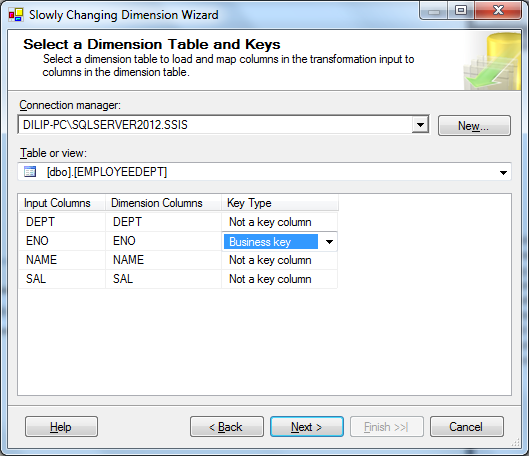
# Percentage Sampling



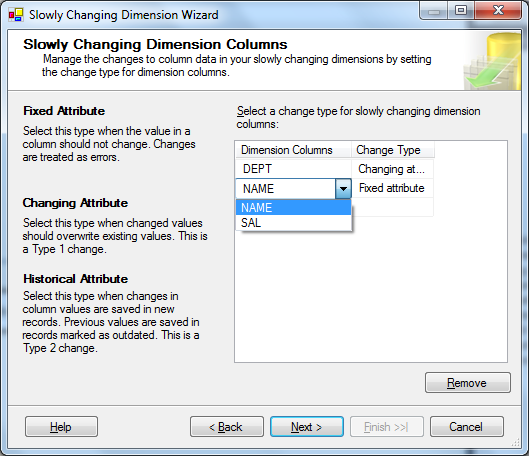
# Slowly Changing Dimension

The simplest explanation is that it compares the attributes (column values) of rows of incoming data against a reference table, using a unique key – called the Business Key – to identify the record to compare against. What can make it complex is the range of comparison options and possible outputs for the component. The component checks attributes for three scenarios:

1. **New** record – no record with that business key exists in the reference table
2. **Changed** attributes – a record with that business key exists and compared attributes have changed
3. **Unchanged** attributes – a record with that business key exists and compared attributes have not changed

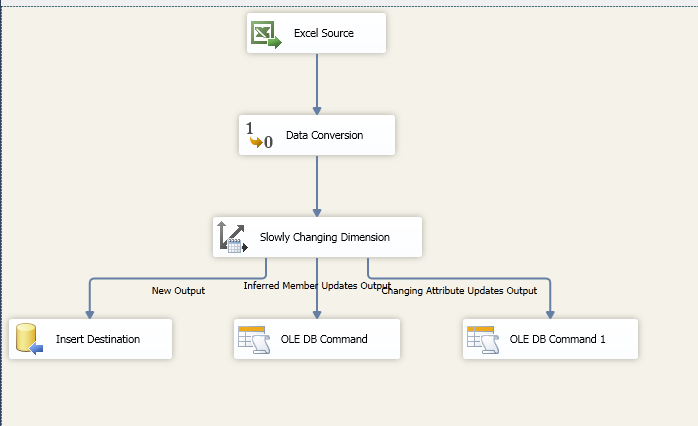


Business Key :

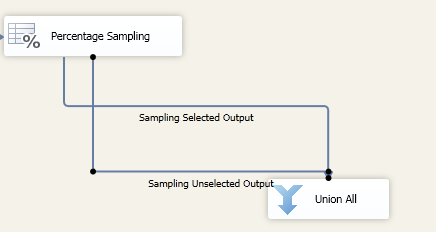


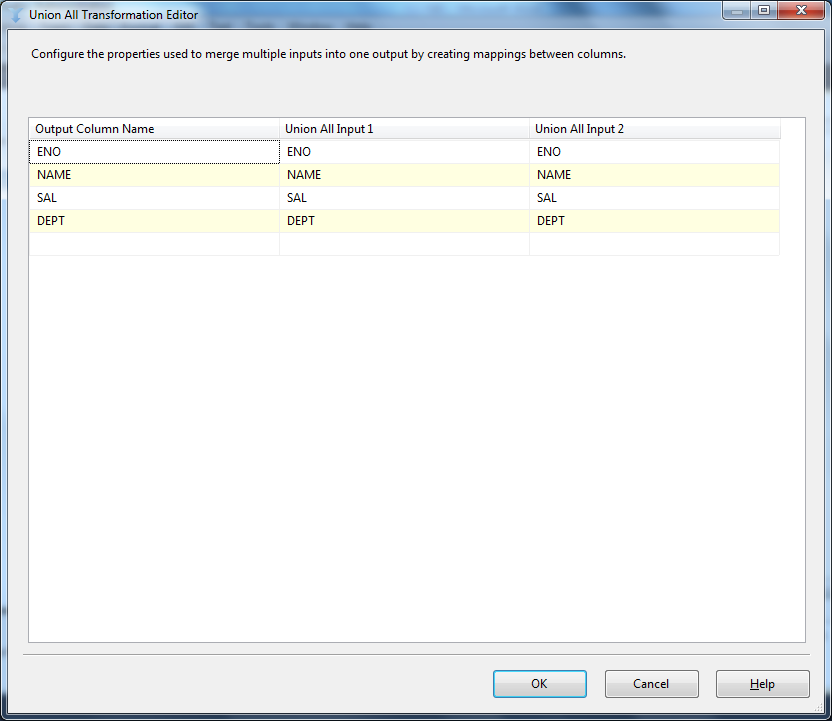
On the current screen you will have to select the dimension column and specify type of column.

The column would be either   
                a. Fixed column which means the data should not change and SCD would not allow the change to happen. For the current example we would map fixed column to name as name would not change.  
  
                b. Changing column is the column whose data changes over a period of time. In our case it would be city column  
  
                c. Historical attribute is the column that would hold the flag values. In other words the column that would be used to distinguish the current record and historical record in the type to slowly changing dimension.



# Union All





# Control Flow Tasks in SSIS

## List of Database Control Flow Tasks

|  |  |
| --- | --- |
| **Task** | **Purpose** |
| ActiveX Script | Execute an ActiveX Script |
| Analysis Services Execute DDL | Execute DDL query statements against an Analysis Services server |
| Analysis Services Processing | Process an Analysis Services cube |
| Bulk Insert | Insert data from a file into a database |
| Data Mining Query | Execute a data mining query |
| Execute DTS 2000 Package | Execute a Data Transformation Services Package (DTS was the SQL Server 2000 version of SSIS) |
| Execute Package | Execute an SSIS package |
| Execute Process | Shell out to a Windows application |
| Execute SQL | Run a SQL query |
| File System | Perform file system operations such as copy or delete |
| FTP | Perform FTP operations |
| Message Queue | Send or receive messages via MSMQ |
| Script | Execute a custom task |
| Send Mail | Send e-mail |
| Transfer Database | Transfer an entire database between two SQL Servers |
| Transfer Error Messages | Transfer custom error messages between two SQL Servers |
| Transfer Jobs | Transfer jobs between two SQL Servers |
| Transfer Logins | Transfer logins between two SQL Servers |
| Transfer Master Stored Procedures | Transfer stored procedures from the master database on one SQL Server to the master database on another SQL Server |
| Transfer SQL Server Objects | Transfer objects between two SQL Servers |
| Web Service | Execute a SOAP Web method |
| WMI Data Reader | Read data via WMI |
| WMI Event Watcher | Wait for a WMI event |
| XML | Perform operations on XML data |

## SSIS maintenance plan tasks

|  |  |
| --- | --- |
| **Task** | **Purpose** |
| Back Up Database | Back up an entire database to file or tape |
| Check Database Integrity | Perform database consistency checks |
| Execute SQL Server Agent Job | Run a job |
| Execute T-SQL Statement | Run any T-SQL script |
| History Cleanup | Clean out history tables for other maintenance tasks |
| Maintenance Cleanup | Clean up files left by other maintenance tasks |
| Notify Operator | Send e-mail to SQL Server operators |
| Rebuild Index | Rebuild a SQL Server index |
| Reorganize Index | Compacts and defragments an index |
| Shrink Database | Shrinks a database |
| Update Statistics | Update statistics used to calculate query plans |

## SSIS Container Tasks

|  |  |
| --- | --- |
| **Container** | **Purpose** |
| For Loop | Repeat a task a fixed number of times |
| Foreach | Repeat a task by enumerating over a group of objects |
| Sequence | Group multiple tasks into a single unit for easier management |

# 

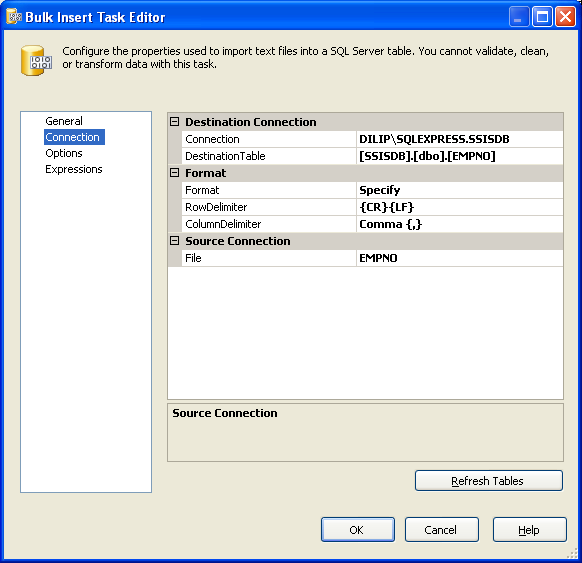
# Bulk Insert Task

Bulk insert task is used to copy large amount of data into SQL Server tables from text files. For example, imagine a data analyst in your organization provides a feed from a mainframe system to you in the form of a text file and you need to import this into a SQL server table. The easiest way to accomplish this is in SSIS package is through the bulk insert task.

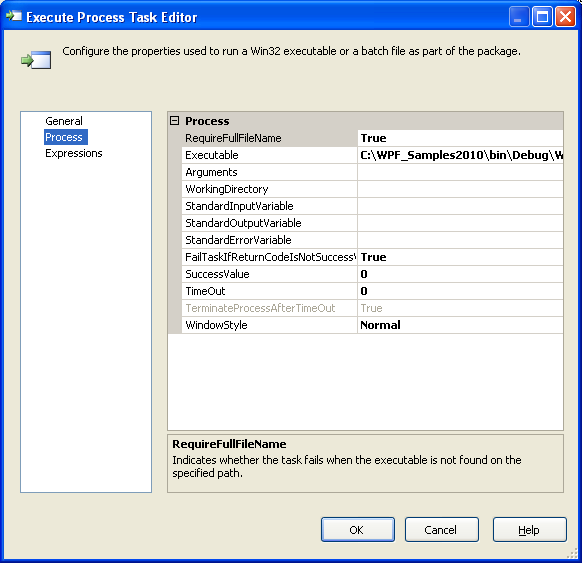
Bulk Insert doesn’t have an option to truncate and load; hence you must use an Execute SQL Task to delete the data already present in the table before loading flat file data.

 It is an easy to use and configure task but with few ***cons.***

1. It only allows appending the data into the table and you cannot perform truncate and load.
2. Only Flat file can be used as source and not any other type of databases.
3. Only SQL Server Databases can be used as destination. It doesn’t support any other files/ RDBMS systems.
4. A failure in the Bulk Insert task does not automatically roll back successfully loaded batches.
5. Only members of the SYSADMIN fixed server role can run a package that contains a Bulk Insert task.

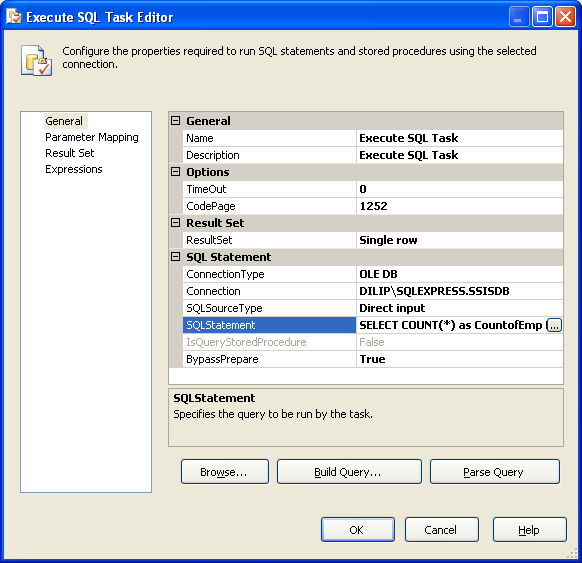


# Execute Process Task

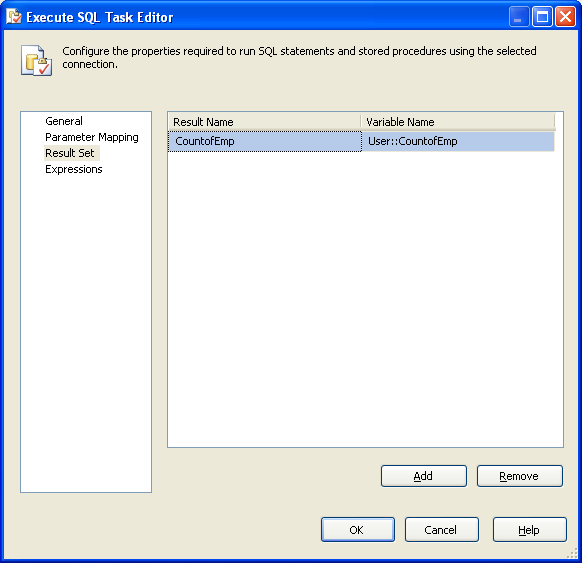


# 

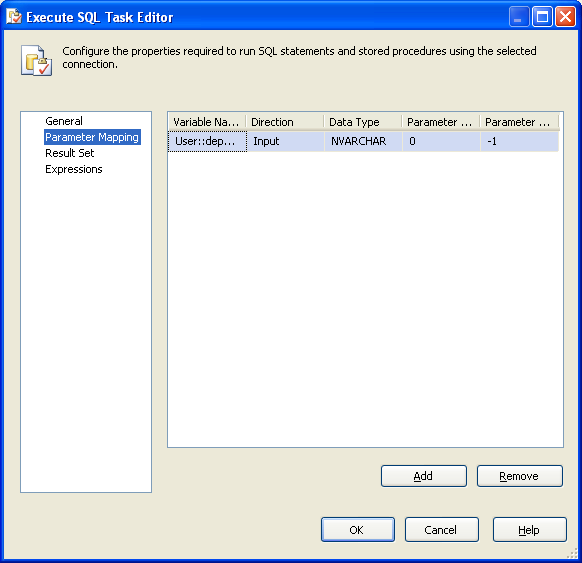
# Execute SQL Task

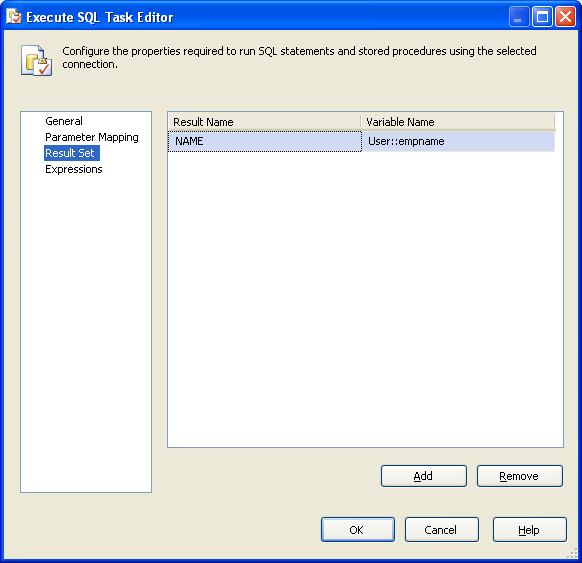


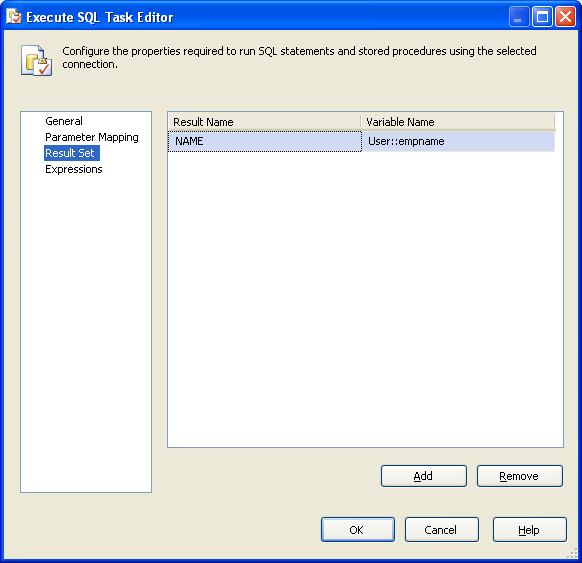
Sql Statement : SELECT COUNT(\*) as CountofEmp from emp\_dept



select NAME from emp\_dept where DEPTNAME = ?





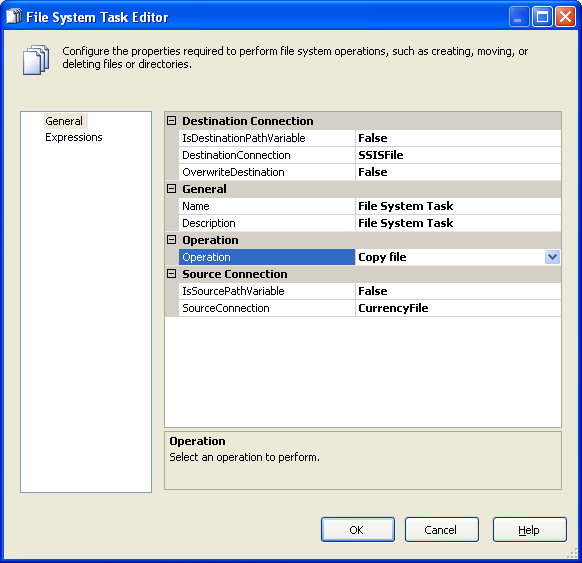


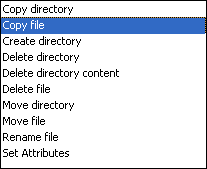
# File System Task

Several times we need to do some file operations like copy, move, delete or rename file while performing the integration tasks.

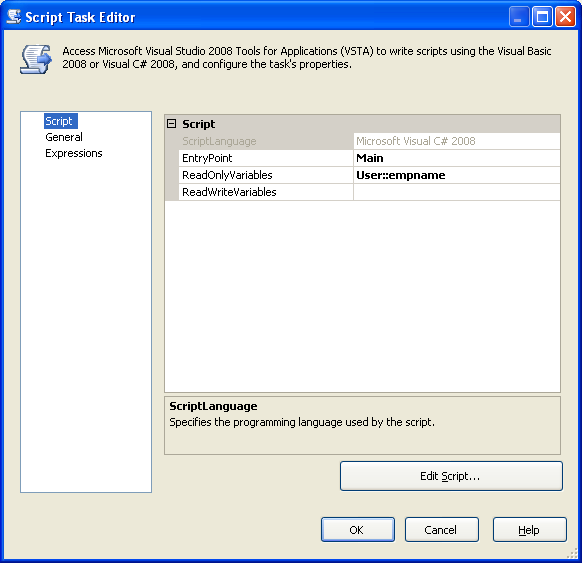
In SSIS we can accomplish these by File System Task control. The File System Task control is used for performing operations on files and directories such as move, create, delete, rename file and directories.

File System Task





# Script Task



public void Main()

{

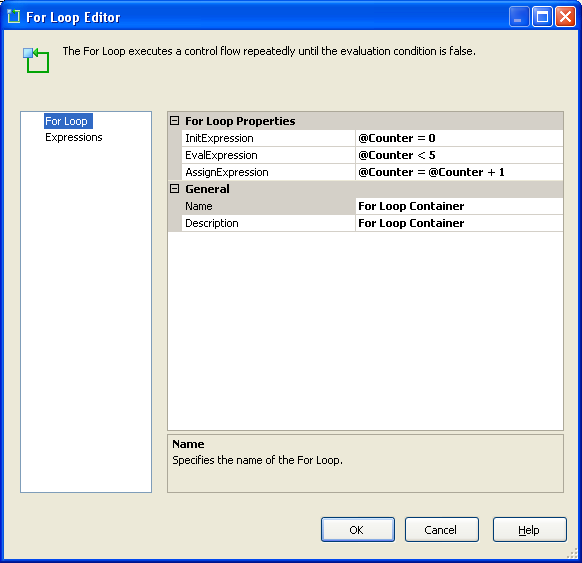
// TODO: Add your code here

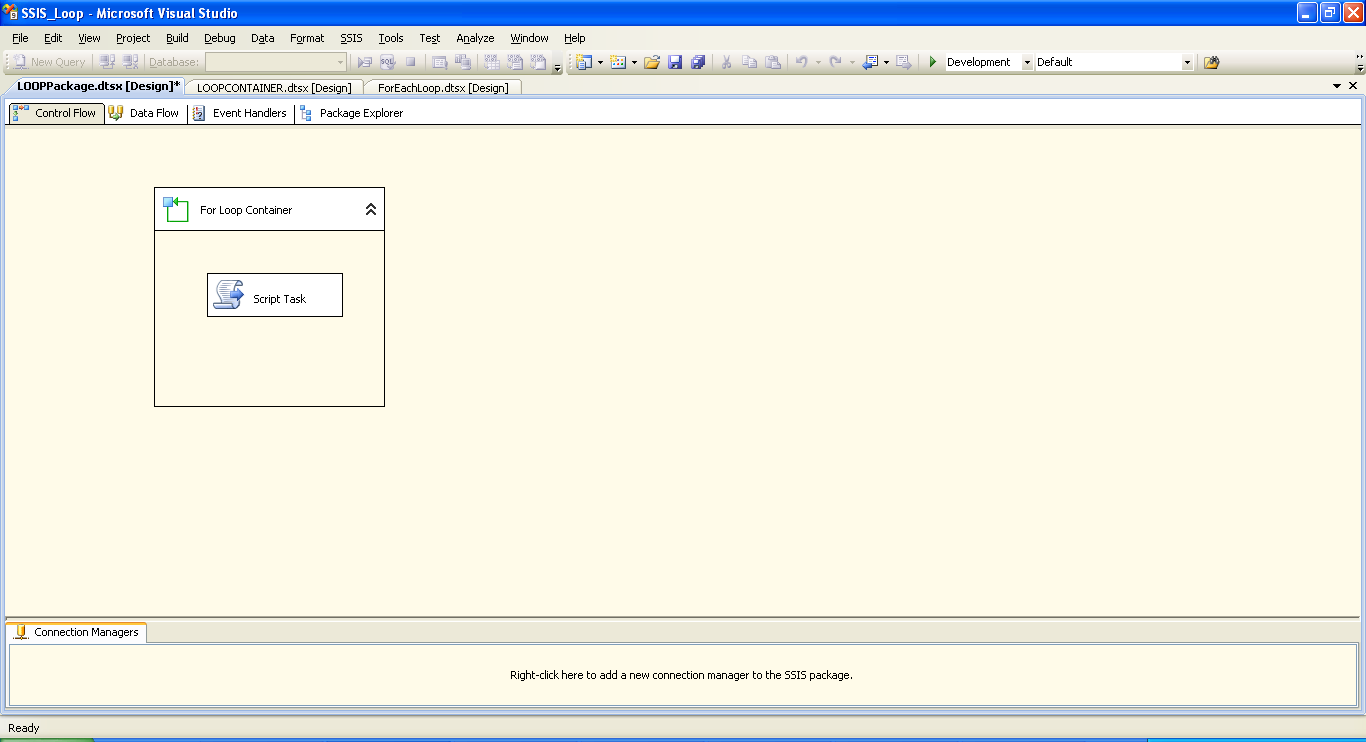
MessageBox.Show(Dts.Variables["empname"].Value.ToString());

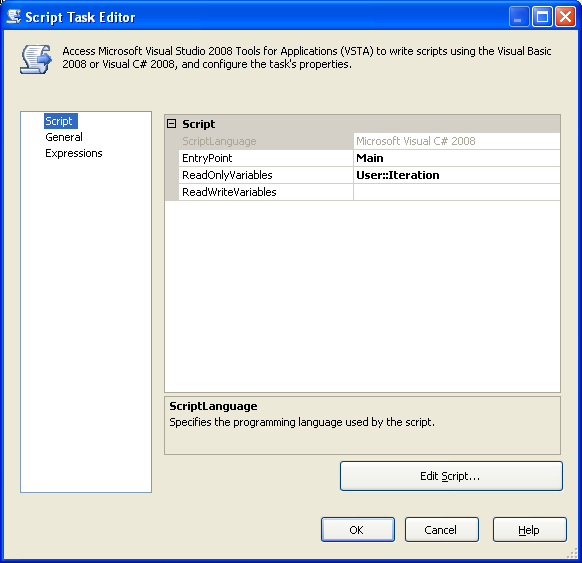
Dts.TaskResult = (int)ScriptResults.Success;

}

# For Loop Container:







public void Main()

{

// TODO: Add your code here

MessageBox.Show(Dts.Variables["Iteration"].Value.ToString());

Dts.TaskResult = (int)ScriptResults.Success;

}

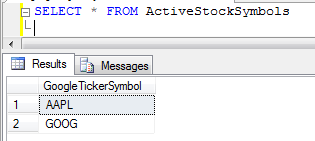
INSERT INTO [Employee\_Dim] ([emp\_name],[emp\_id],[emp\_DOB])VALUES('EmpName' + Cast(? as varchar),101 + Cast(? as int),getdate())

# ForEach Loop Container

Often times in SSIS I find I need to loop through a list from a control table and do the same item repetitively based on variables in that control table.  Today I will walk you through step by step how to use an execute SQL task and the Foreach loop Container to do just that in SSIS.

(Note this is the first in a series of post I will do on creating a simple data warehouse that constantly updates with almost real-time stock data.  Look for more on this later)

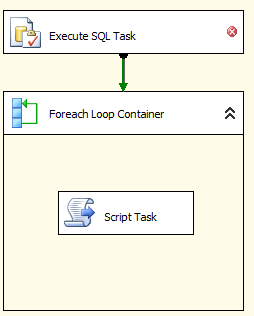
**1st Step – Getting a dataset**



My simple stock symbol dataset

For this step we just need a query that returns a simple dataset for us to loop though. I create a table with 2 rows in it that have stock ticker symbols.  The end result is that we will call a task inside the for loop for each symbol in this list.

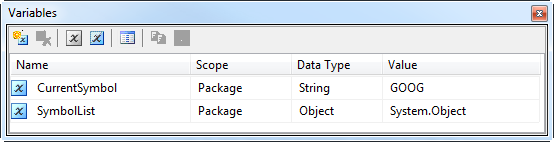
**2nd Step – Setting up the package**



Pull in an Execute SQL task, a for each loop and a script task into the main design window, also attach the Execute SQL task to the for loop.  Hopefully you package looks like this image here.

**3rd Step – Setting up some variables**

Next we need to setup a few control variable before we start configuring our package tasks, bring up the variable window (view->otherwindows->variables for those of you how forget like me) and create 2 variables that match the image below.

[](http://bithoughts.net/wp-content/uploads/2011/11/variables.png)

Note the datatype of the Symbol list (Object) you must set this correctly!

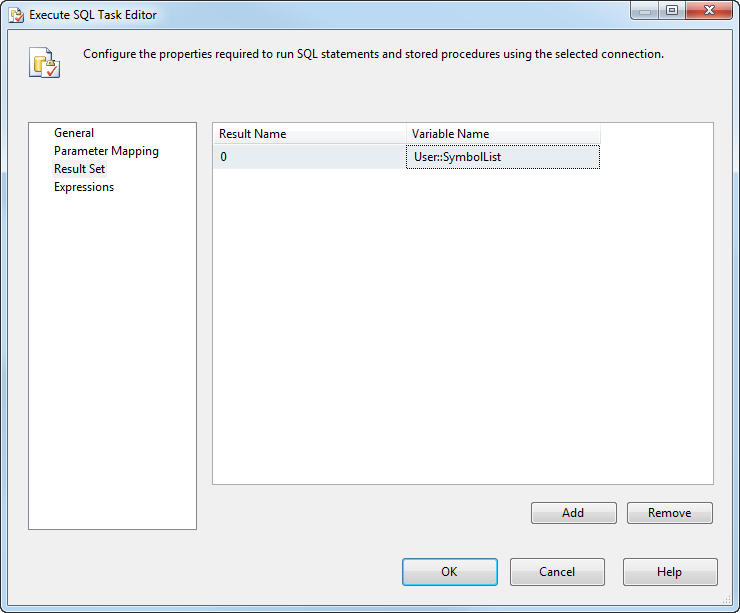
**4th Step – Configuring the Execute SQL Task**

Lets configure the Execute SQL task to pull in a full result set and assign that result set to the variable we just created.



Make sure resultset is set to "Full Result Set" and don't forget to rename the task!

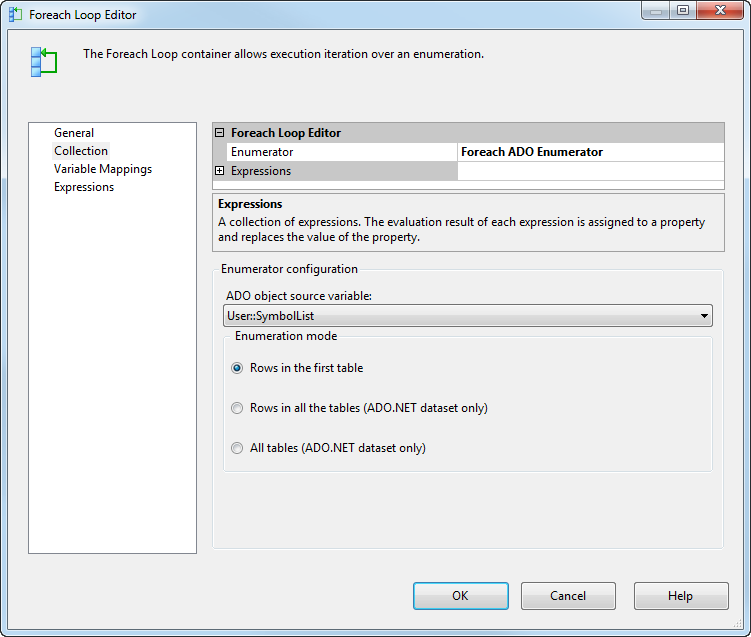
Next select the Result Set Tab and assign the result set to our “special” object variable

[](http://bithoughts.net/wp-content/uploads/2011/11/SQLTaskResultSet.png)

Set the Result Name to '0' and select the variable you created as an object earlier, in my case User::SymbolList

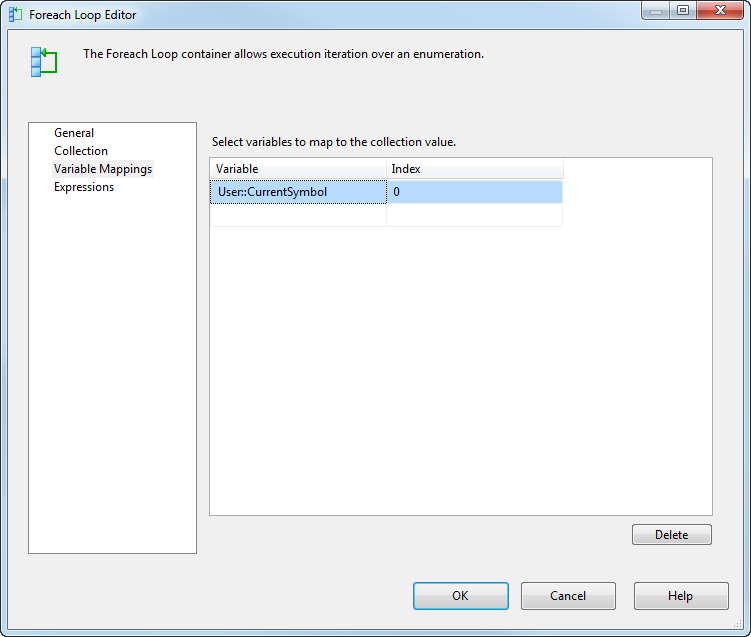
**5th Step – Foreach Loop Container configuration**

Now lets configure the Foreach Loop container to loop though the resultset.  Double click the container and goto the collection tab.  Set the enumerator to Foreach ADO Enumerator and set the source variable to the SymbolList variable created earlier.

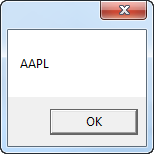


Your collection tab should look something like this.

Next select the variable mappings and assign the CurrentSymbol variable to Index 0. The Index number here corresponds to the row number in the resultset. 0 is the first column, 1 the second etc if your resultset had more than 1 row.

[](http://bithoughts.net/wp-content/uploads/2011/11/ForEachVariableMapping.png)

**6th step – Testing and Success!**

[](http://bithoughts.net/wp-content/uploads/2011/11/MessageBox.png)

# MULTIPLE FILES UPLOAD USING FOR EACH LOOP

Files: “C: Temp” folder

1. Nightly\_01.txt

CustomerName, PhNo, email

AAA,111-222-3000,aaa@test.com

BBB,111-222-3000,bbb@test.com

CCC,111-222-3000,ccc@test.com

DDD,111-222-3000,ddd@test.com

1. Nightly\_02.txt

CustomerName, PhNo,email

EEE,111-222-3000,eee@test.com

FFF,111-222-3000,fff@test.com

GGG,111-222-3000,ggg@test.com

HHH,111-222-3000,hhh@test.com

SQL SERVER TABLE :

CREATETABLE [dbo].[Staging\_Customers](

[CustName] [varchar](255)NULL,

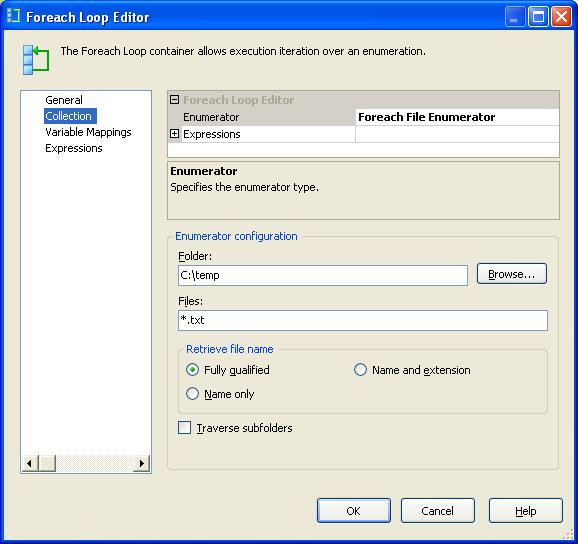
[CustPhone] [varchar](255)NULL,

[CustEmail] [varchar](255)NULL

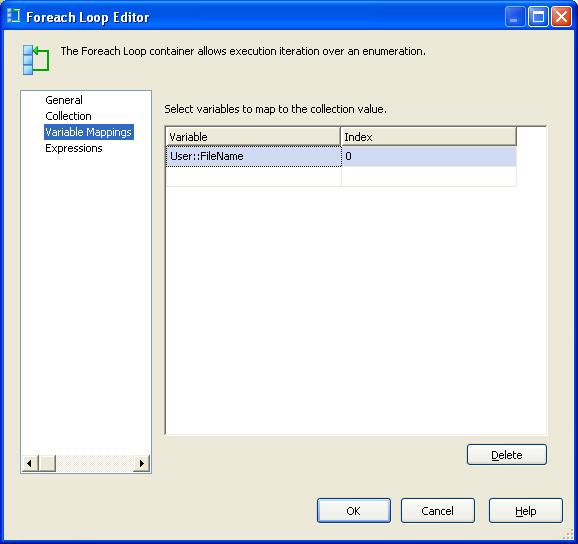
)ON [PRIMARY]

To use the package you need to place the 3 enclosed text files into your "C:\Temp" folder and make sure there are no other "\*.txt" files in there. Then simply run the package. It does a number of things:

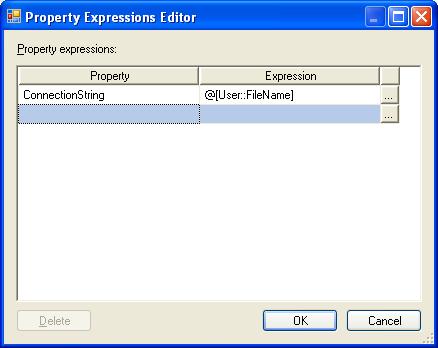
1. Enumerates all the files in "C:\Temp" matching the wildcarded string "\*.txt"



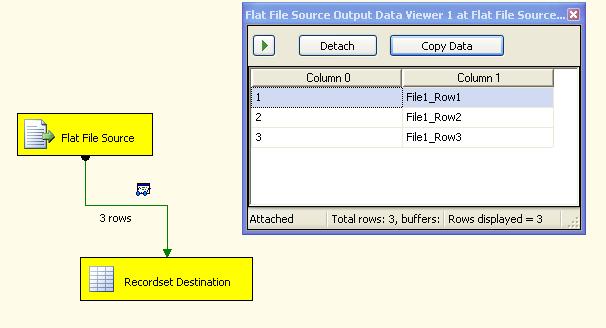
1. Maps the file path of the currently enumerated file to a variable User::Filename



1. Uses a property expression on the Connection String property of a Flat File Connection Manager to point to the file indicated by User::Filename



1. Process the file (which in this demo simply passes all the data into a Record set destination)



I hope this alleviates some of the pain people are having in using the file enumerator in the For each container. It’s a wonderfully powerful feature of SSIS and will have many uses as SSIS gets unleashed unto the masses.

# TRANSACTIONS IN SSIS

Transactions in SSIS allows you to commit the group of data flow tasks (or) allows you to roll back the group of dataflow tasks as single logical unit of work.

Transactions supported by every executable (i.e. package level, task level, container level)

For this 2 properties have to be set for executable

1. Transaction option

2. Isolation level

**Transaction option is of 3 types**

1. Not supported: executable will not support transactions

2. Supported: executable will join the transaction if transaction exists.(default)

3. Required: executable join the transaction if exists else it creates new transaction

Not Supported: It will not support transaction. Also items within it will not support transactions.

Supported: The Items within it will not support transaction.

Required: Create a New Transaction. Items within it will support transaction.

**Process:**

If any tasks are executed, then tasks are executed under transaction if transaction option for executable is set to supported.

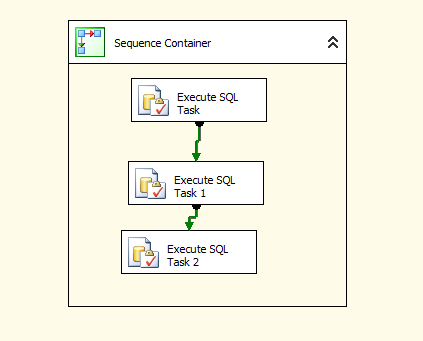
Suppose sequence container having 3 sql execute tasks and it is sequence container is having transaction option =required , Isolation level =serilizable (default).

Then it creates new transaction when executed the sequence container.

All the tasks in sequence container is having transaction option set to supported means these asks run under transaction created by sequence container.

If the tasks having transaction option set to Required, it will join the existing transaction,

if no transaction is there, it will create new transaction and executes

[](http://mssqlbuzz.files.wordpress.com/2011/10/image00116.png)

**Example:**

Create 3 tables with names as

1.Customer

2.Email

3.Vehicle

Syntaxes:

create table customer(custno int,custname varchar(10))

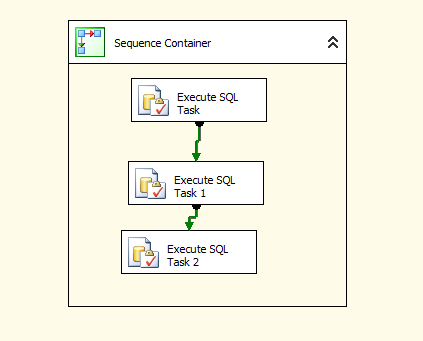
create table email(custno int,emailid varchar(20))

create table vehicle(custno int,vin char(5) not null,model char(3))

Where VIN-Vehicle identification number

Open Business intelligence development studio and click on file->new->project->integration services project

Place sequence container and drag 3 execute sql tasks like below

[](http://mssqlbuzz.files.wordpress.com/2011/10/image00116.png)

Double click on first execute sql task and set the properties

In general page, connection type =OLEDB

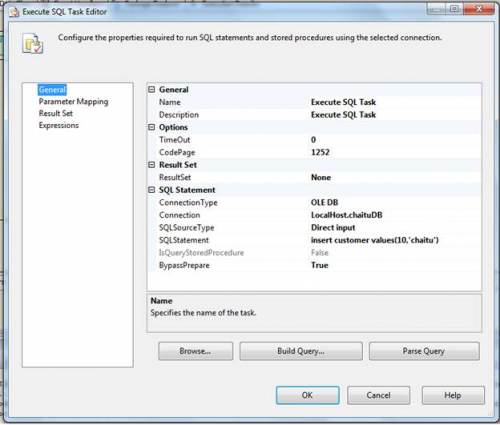
Connection: select new connection, a window occurs, specify the data source information give server name and database information and test the connection

General page,

Sql source type=Direct input

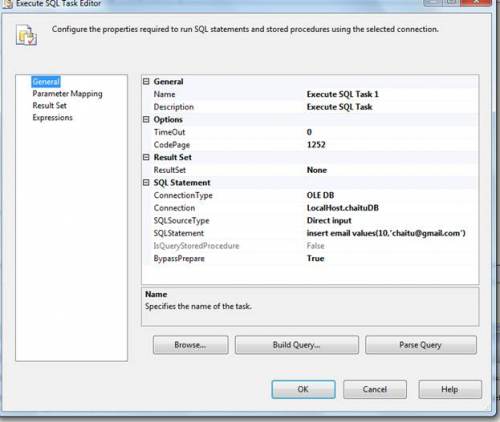
Sql statement=insert customer values (10,’chaitu’)

This is looks like below.

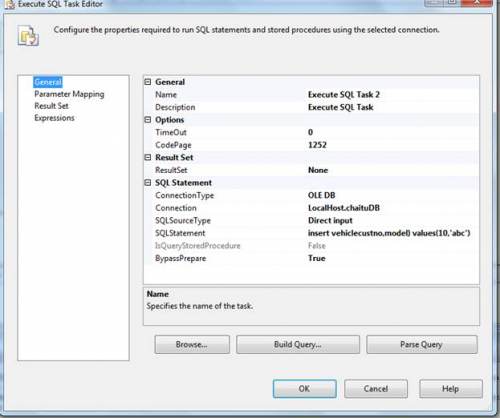
[](http://mssqlbuzz.files.wordpress.com/2011/10/image004.jpg)

Same can be done for other execute sql tasks with same connection but different sql statement

**Set Execute sql task 1 Properties**

[](http://mssqlbuzz.files.wordpress.com/2011/10/image005.jpg)

**Set Execute SQL task 2 properties**

[](http://mssqlbuzz.files.wordpress.com/2011/10/image006.jpg)

**insert vehiclecustno,model) values(10,’abc’)**

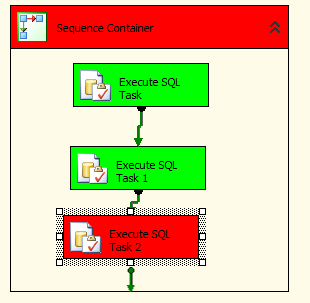
**We have passed values to the insert statement for 3rd task but the table is having not null column. We are not passing information to that column. So this task must fail.**

Transaction in SSIS makes use of the Distributed Transaction Coordinator (MSDTC) service which must be running in services. MSc. MSDTC also allows you to perform distributed transactions e.g. updating a any RDBMS database. If you execute an SSIS package that utilizes the built-in transaction support and MSDTC is not running, you will get an error message like the following:

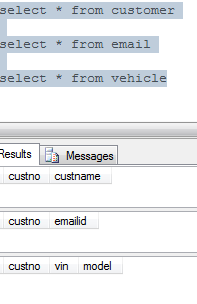
Error: 0xC001401A at Transaction: The SSIS Runtime has failed to start the distributed transaction due to error 0x8004D01B "The Transaction Manager is not available.". The DTC transaction failed to start. This could occur because the MSDTC Service is not running.

Execute the package by pressing f5

We got below error like this

[](http://mssqlbuzz.files.wordpress.com/2011/10/image0075.png)

First two execute sql tasks are success. 3rd task is failed. So transaction is roll backed. Because these 3 tasks are running under single transaction

[](http://mssqlbuzz.files.wordpress.com/2011/10/image0084.png)

To implement package scope transactions, place another sequence container move execute sql task 2 to that sequence container. Go to properties window of package by clicking f4

Transaction property: required

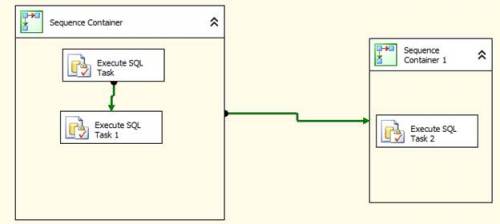
Isolation: serilizable(default)

**Sequence containers properties are:**

Transaction property: Supportes

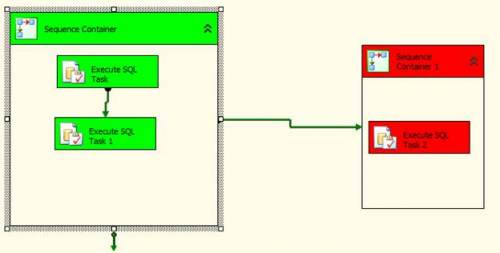
Isolation: serilizable(default)

It will be looks like below

[](http://mssqlbuzz.files.wordpress.com/2011/10/image009.jpg)

Execute the package by clicking on F5

It will be failed as 3rd task is failed.

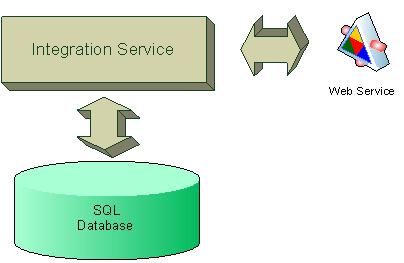
[](http://mssqlbuzz.files.wordpress.com/2011/10/image010.jpg)

So first and second task inserts data, which is roll backed.

# (Web Service in SSIS)

The SSIS framework provides a Web Service task which executes a Web Service method. You can use the Web Service task for the following purposes:

* Writing to a variable the values that a Web Service method returns.
* Writing to a file the values that a Web Service method returns.



This article defines a step by step approach to using a Web Service call in SQL Integration Services.

First, create a web service project.

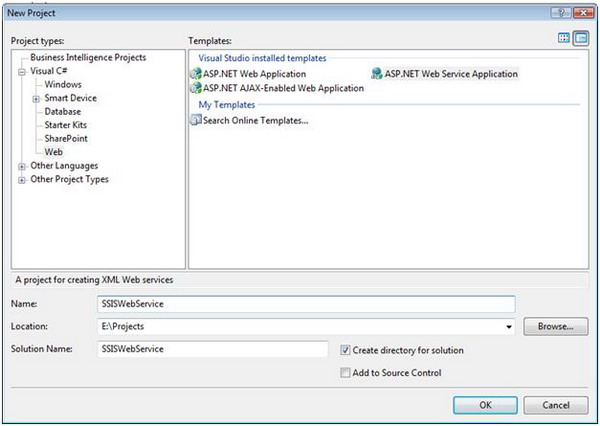


Figure 2 - Create a Web Service

Now, expose a method call in the Web Service:

public class SSISService : System.Web.Services.WebService

{

[WebMethod]

public string HelloMoon()

{

return "Hello Moon";

}

[WebMethod]

public int Multiply(int nParam1, int nParam2)

{

return nParam1 \* nParam2;

}

}

Create an SSIS package:

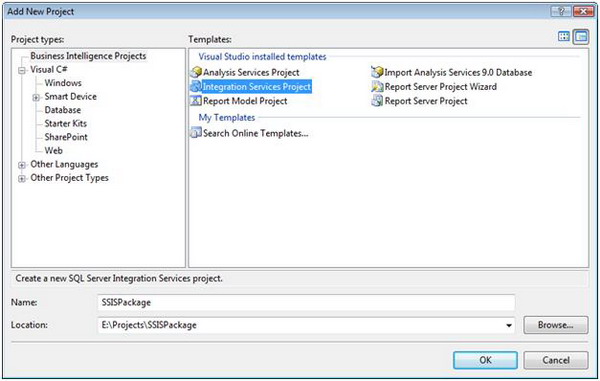


Figure 3 - Create an SSIS Package

Add a web service task:

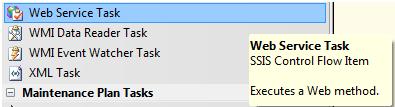


Figure 4 - SSIS Toolbox



Figure 5 - Web Service Task

Next, modify the Web Service task:

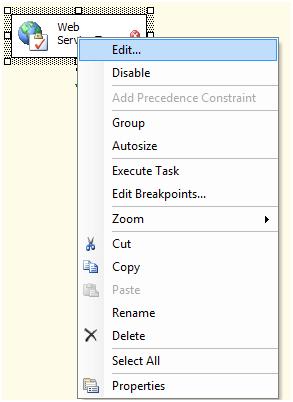


Figure 6 - Edit Web Service Task

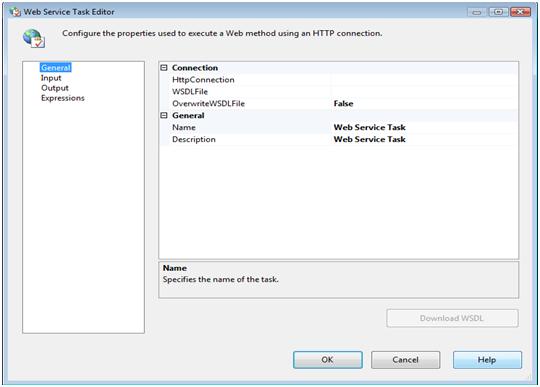


Figure 7 - Web Service Task Editor

Now, define the HTTP connection:

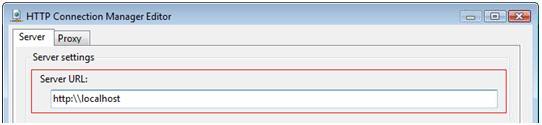


Figure 8 - HTTP Connection Manager Editor

The next step is to define the WSDL file:

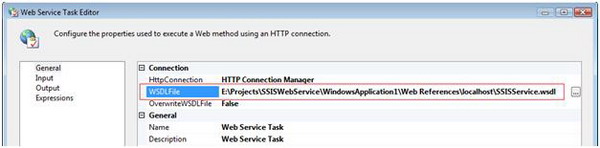


Figure 9 - Web Service Task Edit (Definition of the WSDL file)

Define the Web Service Task inputs:

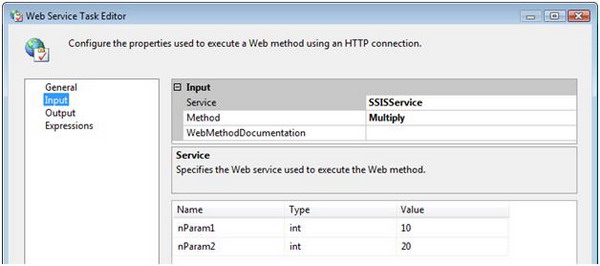


Figure 10 - Web Service Task Editor (Definition of web service input properties)

Now, define the Web Service output. The output of the Web Service can be written to variables or to an output file. This sample outputs the results from the Web Service to a file system file that is defined using the File Connection Manager Editor (Figure 11).

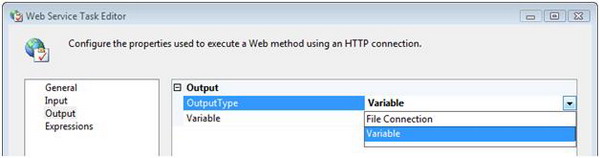


Figure 11 - Web Service Task Editor (Definition of Web Service output properties)

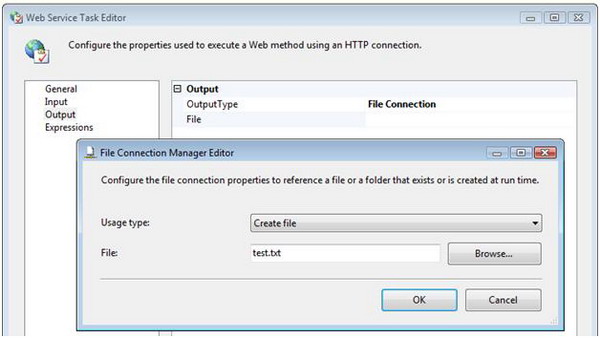


Figure 12 - File Connection Manger

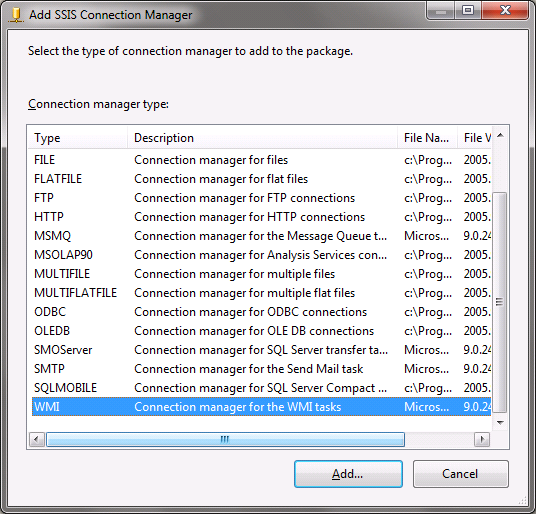
**Results**

The following is the encoded output from the Web Service, stored in a *test.txt* file

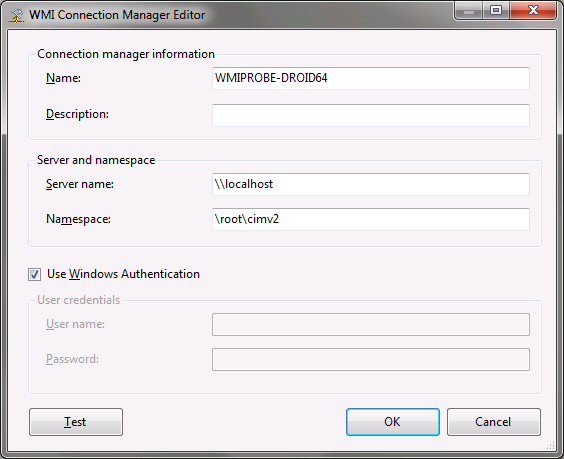
# ****WMI Data Reader Task****

The package uses the WMI Data Reader Task to gather the disk information. It requires a WMI connection and a WQL query to define what information to gather and from where.

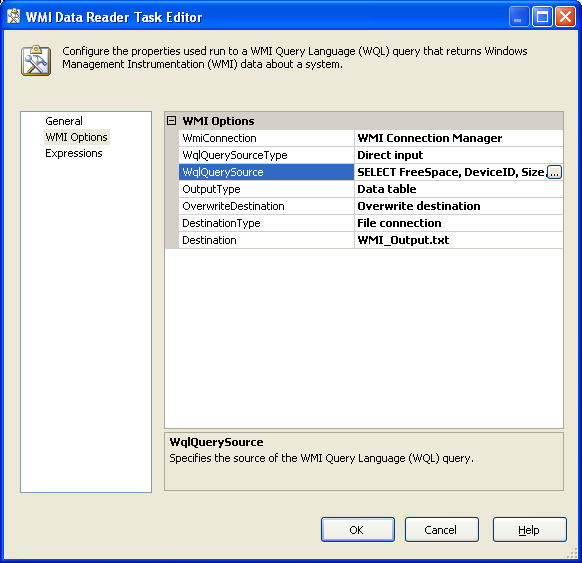
Let’s start with the WMI connection configuration. Right click in the BIDS Connection Managers pane and click on New Connection… which will pop up the Add SSIS Connection Manager dialog.

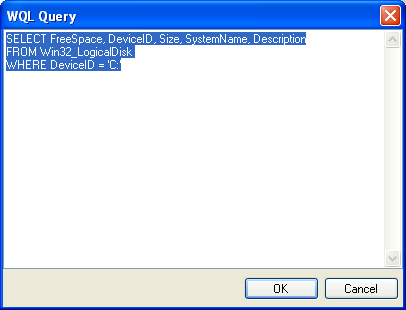


Select WMI and click the Add… button. The WMI Connection Manager Editor will be displayed.



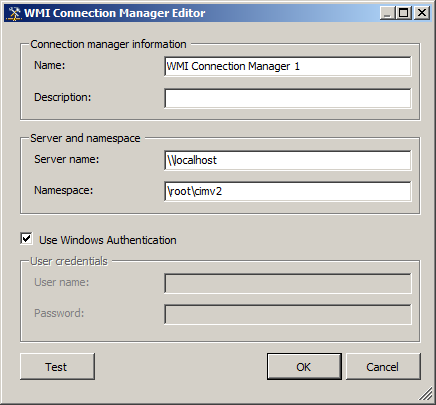
Configuring the WMI connection for the local computer is fairly simple. Set it up as shown above with the name of your choice of course and click OK. This configuration will only work if the account that the package will run under has rights to read data from the WMI provider. If there are security issues reading WMI data, other credentials will need to be provided or the account that runs the package given the rights to query the WMI provider.





**WMI EVENTS TASKS**

To set up the task drag a WMI Event Watcher task into the control flow of a package.



After you have the connection set up,select Direct Input as the Source type and then enter the following code into WQL query.

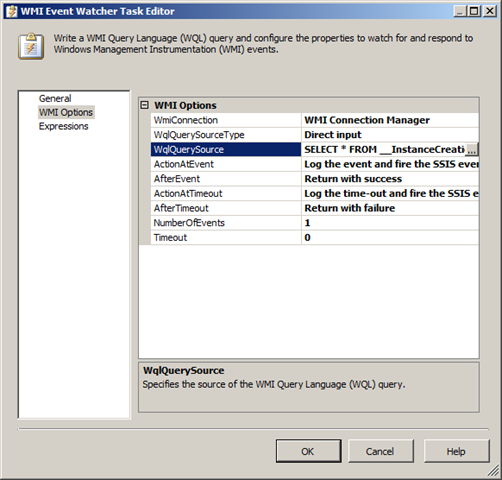
SELECT \* FROM \_\_InstanceCreationEvent WITHIN 10

WHERE TargetInstance ISA "CIM\_DirectoryContainsFile"

AND TargetInstance.GroupComponent = "Win32\_Directory.Name=\"d:\\\\NewFiles\""

This WQL statement tells the task to look in the C:\Projects\Temp folder and to look for the creation of a file every 10 seconds.  This will cause the package to run for infinity if we do not set a timeout period. So it is a good practice to set a time out for the task. The timeout option is the last option in the WMI Event Watcher Task Editor. Once the File exist in the folder location the WMI Event Watcher will turn green indicating success. Notice the double slashes in the code, this is necessary due to escape characters.

The "WITHIN 10" clause of the WQL statement can be change to less time, but this will cause more reads across the network and can cause performance issues. Set the timeout to the most time you think should be allotted so performance is not effected. But not too long, if you have it set to 300 seconds or 5 minutes, and the file is placed in the folder 1 second after the WMI Event Watcher has done it's check it will take almost 5 minutes before the file creation is detected. And if someone opens the file or deletes it before the task detects it the package may fail.



Case Study

**Continuously watching files with WMI Event Watcher Task**

**Case**  
A client wants to have a continuously running package watching for new files in a folder, so he can automatically process new files throughout the day without manually starting a package each time.  
  
**Solution**

The solution includes two aspects: watching for new files and the continuously running part. Let us start with the watching for new files part.

**Watching for new files**

There are some third party filewatchers available, but we will use the standard WMI Event Watcher Task.

1) Drag the WMI Event Watcher Task to your Control Flow and give it a suitable name.

|  |
| --- |
| http://4.bp.blogspot.com/_yHVTMii_mvA/TQ4dHrYDTXI/AAAAAAAAAB4/r-waC-x-rMc/s1600/ContinuouslyFileWatcher1.jpg |
| WMI Event Watcher Task |

﻿﻿﻿﻿

2) Go to the WMI Options tab and create a new WmiConnection. Select **Use Windows Authentication** for this exercise. It will use the account that is running your package.

|  |
| --- |
| http://3.bp.blogspot.com/_yHVTMii_mvA/TQ4fN_NcoyI/AAAAAAAAAB8/m0_R9fdNW0k/s400/ContinuouslyFileWatcher2.jpg |
| WMIConnection |

3) For the WqlQuerySourceType we will use the default Direct input so you will have to enter the WMI query under WqlQuerySource.

|  |
| --- |
| [http://3.bp.blogspot.com/_yHVTMii_mvA/TQ4g7X4h0NI/AAAAAAAAACA/_-K3b5OIYJA/s320/ContinuouslyFileWatcher3.jpg](http://3.bp.blogspot.com/_yHVTMii_mvA/TQ4g7X4h0NI/AAAAAAAAACA/_-K3b5OIYJA/s1600/ContinuouslyFileWatcher3.jpg) |
| Wql (SQL for WMI) |

4) Enter the Wql below in the box. There are two points of attention. First is the WITHIN 10 part. This indicates the number of seconds between each check. A too low number could overload your system. Second is the Directory name; notice the extra backslashes in the path.

SELECT \* FROM \_\_InstanceCreationEvent WITHIN 10

WHERE TargetInstance ISA "CIM\_DirectoryContainsFile"

AND TargetInstance.GroupComponent = "Win32\_Directory.Name=\"d:\\\\NewFiles\""

**Continuously running**  
5) The easiest way to create a continuously running package is to use a For Loop and set the following EvalExpression: "true == true". This will result in an infinite loop.

|  |
| --- |
| [http://4.bp.blogspot.com/_yHVTMii_mvA/TQ5XTKZKSDI/AAAAAAAAACI/PRZCnqmzqo0/s320/ContinuouslyFileWatcher4.jpg](http://4.bp.blogspot.com/_yHVTMii_mvA/TQ5XTKZKSDI/AAAAAAAAACI/PRZCnqmzqo0/s1600/ContinuouslyFileWatcher4.jpg) |
| Infinite loop |

﻿

﻿   
6) Drag the WMI Event Watcher Task into the Infinite loop.

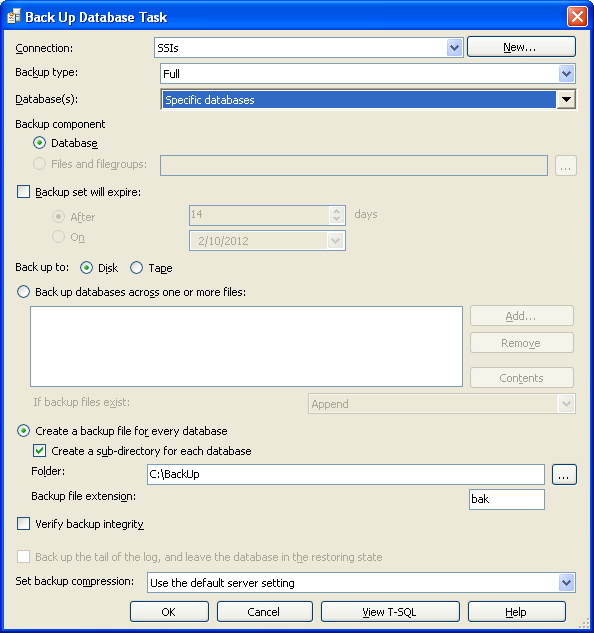
|  |
| --- |
| <http://1.bp.blogspot.com/_yHVTMii_mvA/TQ5YtPI9WuI/AAAAAAAAACM/iXRPUdB7jxw/s1600/ContinuouslyFileWatcher5.jpg> |
| Continuously watching |

7) Now add your own tasks behind the WMI Event Watcher Task. A possible solution could be a For Each Loop that loops through all files in d:\NewFiles\, processes them and moves them to an archive folder with the File System Task.

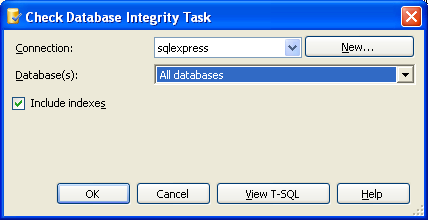
|  |
| --- |
| [http://1.bp.blogspot.com/_yHVTMii_mvA/TQ5cFLcao8I/AAAAAAAAACQ/6eA3_Sv5-B8/s320/ContinuouslyFileWatcher6.jpg](http://1.bp.blogspot.com/_yHVTMii_mvA/TQ5cFLcao8I/AAAAAAAAACQ/6eA3_Sv5-B8/s1600/ContinuouslyFileWatcher6.jpg) |
|  |

# MAINTENANCE PLAN TASKS

**Back Up DataBase Tasks:**



**Check Database Integrity Task:**



USE [Employment Equity ]

GO

DBCC CHECKDB(N'Employment Equity ') WITH NO\_INFOMSGS

**Execute Sql Server Agent Job Task:**

Use the **Execute SQL Server Agent Job Task** dialog to execute Microsoft SQL Server Agent jobs within a maintenance plan. This option will not be available if you have no SQL Server Agent jobs on the selected connection.

This task uses the **.sp\_start\_job** statement.

**Connection**

Select the server connection to use when performing this task.

**New**

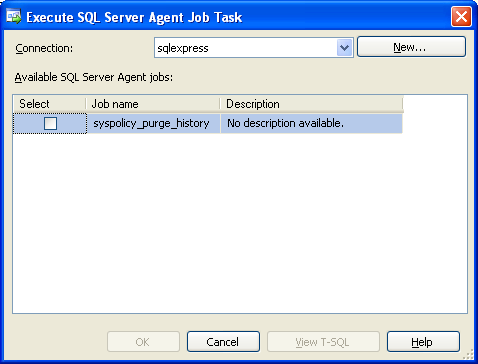
Create a new server connection to use when performing this task. The **New Connection** dialog box is described below.

**Available SQL Agent jobs**

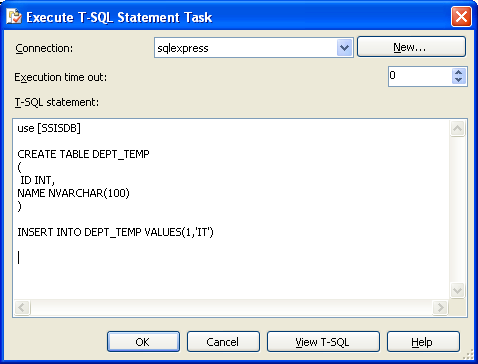
Select the job to execute. The grid provides the **Job name** and **Description** to identify the jobs.

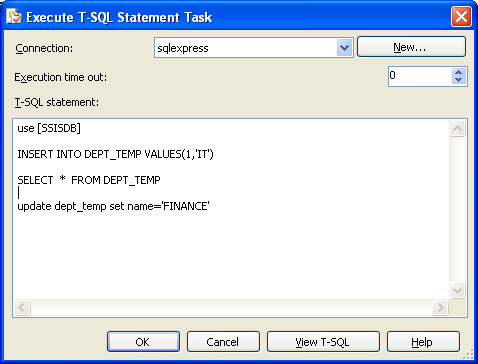
**View T-SQL**

View the Transact-SQL statements performed against the server for this task, based on the selected options.

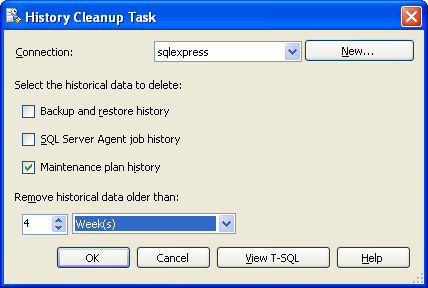
****

**Execute T-Sql Statement Task:**

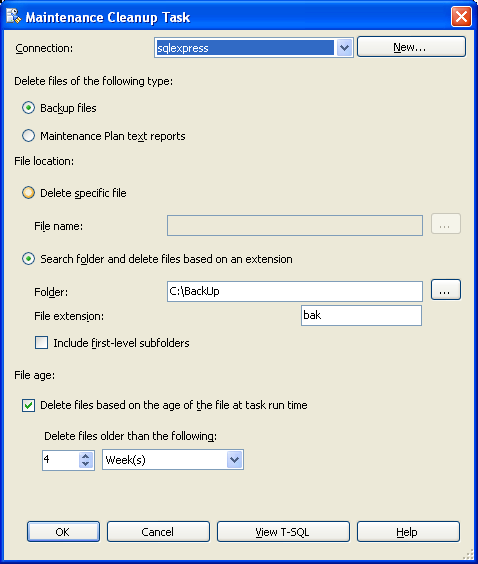




**History Cleanup Task:**



**Maintenance Cleanup Task:**



**Notify Operator Task:**

The Notify Operator task sends notification messages to SQL Server Agent operators. A SQL Server Agent operator is an alias for a person or group that can receive electronic notifications.

By using the Notify Operator task, a package can notify one or more operators via e-mail, pager, or **net send**. Each operator can be notified by different methods. For example, OperatorA is notified by e-mail and pager, and OperatorB is notified by pager and **net send**. The operators who receive notifications from the task must be members of the **OperatorNotify** collection on the Notify Operator task.

The Notify Operator task is the only database maintenance task that does not encapsulate a Transact-SQL statement or a DBCC command.

Connection

Select the server connection to use when performing this task.

New

Create a new server connection to use when performing this task. The New Connection dialog box is described below. This task is not available only for SQL Server 2000 or earlier versions.

Operators to notify

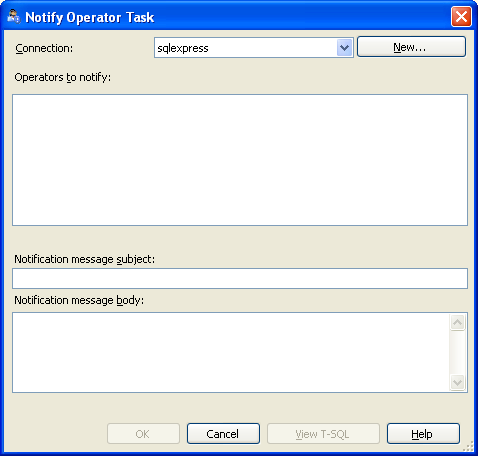
Specify the recipient of the e-mail.

Notification message subject

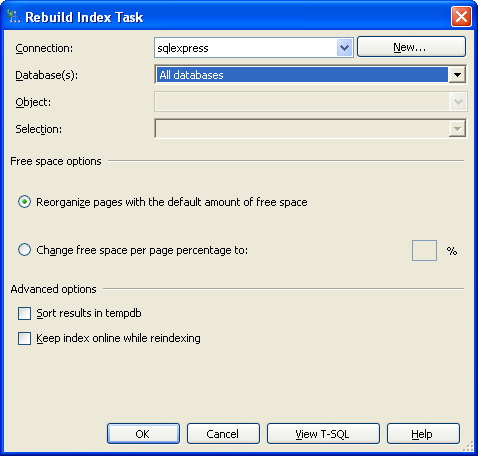
Specify the text to include in the notification message subject line.

Notification message body

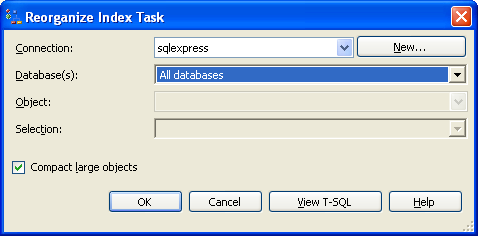
Specify the text to include in the notification message body.



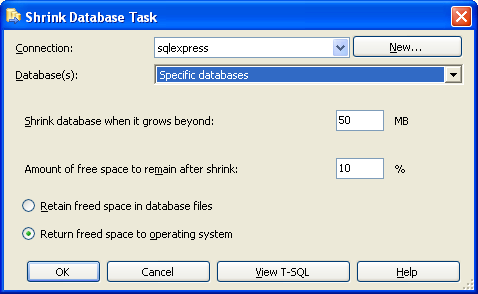
**Rebuild Index Task:**



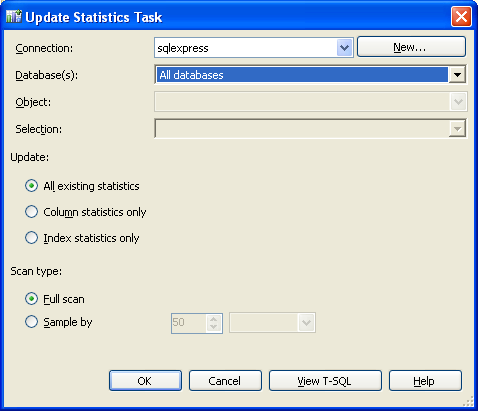
**Reorganize Index Task:**

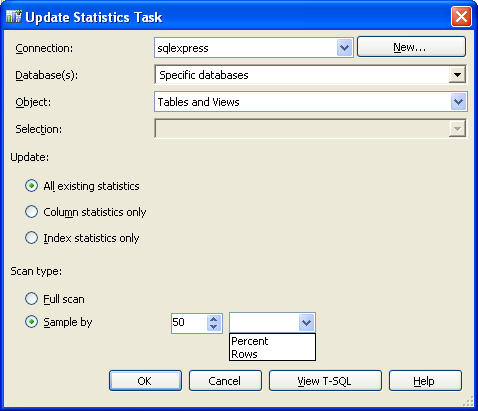


**Shrink Database Task:**



**Update Statistics Task:**





FTP DETAILS :

ftp://enosislearning.com/

userid : enosislearning

Password : enosis@123!

<add key="ftpaddress" value="ftp://111.118.177.195"/>

<add key="uname" value="ftpdetail195"/>

<add key="pass" value="dwqdegf@#21"/>