DTS Common Tasks

Objectives

- Learn how and when to use the Bulk Insert task.
- Find out how bulk insert uses format files.
- Use the Execute SQL task with parameterized queries.
- Understand global variables and why they are used.
- Explore the functionality of the Transform Data task and its underlying architecture, the data pump.
- Set up error handling.
- Understand data load settings for Microsoft SQL Server.

Bulk Insert Task

In Data Transformation Services, the Bulk Insert task, one of the managing data tasks, is the fastest way to load data into Microsoft SQL Server tables. Use this task whenever the source of the load is a text file. In situations where third-party systems have no ODBC drivers or OLE DB providers, an interim text file may be the best approach. Also, if the input data consists of a very large number of rows, such as a million, the Bulk Insert task is a good choice.

The Bulk Insert task uses the Transact-SQL BULK INSERT statement to perform its data load. This statement uses the architecture of the bulk copy program (bcp) that ships with almost all versions of SQL Server. This technology performs a very fast data load. This task is the fastest of any tasks that load data into SQL Server.

Since the Bulk Insert task is tied to SQL Server statements and architecture, the destination of the task is always a Microsoft SQL Server database. Within that database, the target can be either a table or a view.

A drawback to this task is that no transformations can be performed during the load. This is understandable because a transformation involves loading a source row, making a modification, and sending the result to the destination row. This obviously slows things down, so the speed of the bulk insert depends on bypassing that procedure. Another consideration is that rows with errors in the insert cannot be logged.

By default, the Bulk Insert task matches columns in the file in the same order as the target table or view. Although you want the file to be in the easiest and quickest format, that is not always possible because the text file format comes from a source that you may not control.

To handle this situation, the Bulk Insert task enables you to assign *format files* when the text file source and the destination table or view are different. Format files are covered in another section of this chapter.

Because data is being inserted into a SQL Server table or view, the user of this task needs the correct permissions. Specifically, the user must be a member of either the *sysadmin* or *bulkadmin* fixed server roles. A member of the sysadmin role can do anything in SQL Server.

A member of the bulkadmin role has Bulk Insert permissions. By using the lesser role, you ensure a higher degree of security in SQL Server. A user who is a member of the bulkadmin role must also have insert permission on the target table or view. The user also needs file system permissions in order to read the source file.

Bulk Insert Task Options

Several data load options can be set for the Bulk Insert task. You modify these options on the Options tab of the Bulk Insert Task Properties dialog box, which is shown in Figure 1. These options affect the performance of the task.

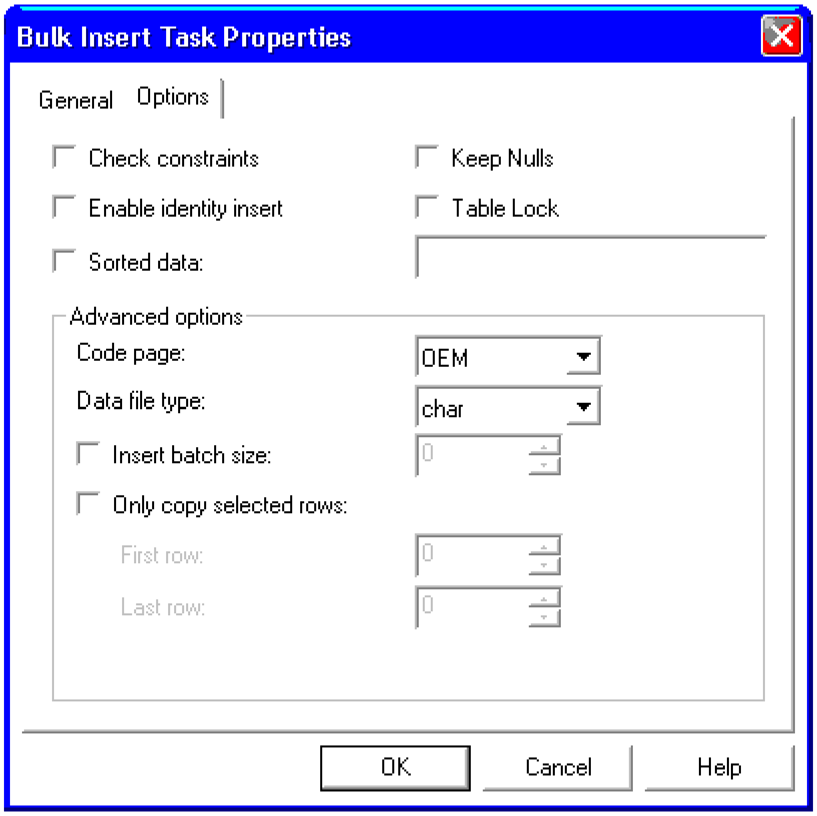


Figure 1. The Options tab of the Bulk Insert Task Properties dialog box.

Insert Batch Size

When the Bulk Insert task runs, the rows that are inserted into SQL Server are set as a batch. Each batch of rows is one *transaction*. A transaction is a unit of work—either all of the rows in the batch are successfully inserted, or none of the rows is inserted. Any error in the batch fails all of the rows in the batch.

WARNING! Transactions are actually controlled at the package and task level. If you are using transactions at the package level, then batches are not complete transactions. Instead, successful batches are held until the transaction of the package is complete.

The task itself does not fail just because one batch fails. The Bulk Insert task fails only after ten failures have occurred. Error settings like this are covered in another chapter of this course.

WARNING!	Task failure can lead to transaction failure of the package. This means that if the step for this task returns failure, it can cause the package transaction to fail as well.

The default batch size is zero, but you can change this. The zero setting has the following implications:

- **One Batch:** All of the rows in the source table are inserted into SQL Server as one batch.
- **Fast Performance:** This is generally true since batches and transactions create overhead. With only one batch, there is less overhead.
- **Task Failure:** Although the task never reaches the maximum number of errors, the task effectively fails because one row failure fails the batch, and that batch is the entire task.
- **Resource Impact:** The rows are not *committed* until the last row is processed. This impacts memory usage and the transaction log of the database. Resources are a tricky issue. For example, one aspect of a single batch is that you get fast performance. But if the batch is *too* large, performance suffers.

Changing the batch size is desirable when the number of rows is too large for the resources of your system, or when there is a potential for failures. Since a failure only fails the batch, smaller batches result in more successes on the insert. As for performance, this is a delicate balancing act that is highly dependent on the system configuration (amount of memory and disk space) and the SQL Server database configuration (size and settings of the transaction log).

When the batch size is set to one, then the data is loaded one row at a time. This way all of the rows that are good are inserted. The maximum number of errors allowed impacts this setting.

Another factor to consider is the data load. Where is the data coming from? Are there multiple data sources? Are you using an existing table? Are there constraints on the table? Are there users are on the system at the time?

Table Lock

By default, SQL Server uses row-level locks on the destination table. This adds overhead to the bulk insert operation because every lock has to be maintained until the end of the transaction. Locks are another SQL Server resource to consider.

Lock contention on the table is possible if other activities are occurring, such as parallel data loads in the DTS package.

If you set the Table Lock check box, then the task will obtain just one lock on the entire destination table. This reduces lock resource overhead, but the lock will remain in place until the task finishes. Overall performance will improve.

Sorted Data

Tables in SQL Server can have clustered indexes. A *clustered index* is an index that sets the physical order of the table in the logical order of the index. Put another way, the clustered index *is* the table.

A clustered index has the same lookup structure as nonclustered indexes. The difference is that the bottom level (or leaf level) of a clustered index is the actual table. Nonclustered indexes have pointers to the table at the leaf level. This means that when new rows are added to a table with a clustered index, SQL Server must locate the position of the row to do the insert. Normally it does so one row at a time, using the lookup structure of the clustered index.

If the file that is being inserted is in the clustered index order, then SQL Server can use a faster technique—it scans the table in the clustered index order, starting at the beginning of the table, and performs the inserts at the correct location. With this method, SQL Server does not have to use the lookup structure of the index.

In this case, set the Sorted data check box and then specify the column names that make up the clustered index. These column names are from the destination table.

If the table does not have a clustered index, then you do not need this option, since new data is added at the logical end of the table.

Other Options

The following options can also be set on this page. They either do not impact performance or have less effect.

- **Check Constraints:** Normally constraints on the target table are ignored in a bulk insert operation, which makes the insert go faster. But constraint errors could affect data integrity, so you may have to select this option.
- **Enable Identity Insert:** If the values for an identity column are to be supplied by the source file, then you should select this option. The default is not set; in that case, SQL Server generates the identity column values.

- **Keep NULL Values:** If a column has a default constraint, or a bound default, and a NULL value is supplied in the source file, the default is used. If you select this option, then the NULL is placed in the column, not its default value.
- **Code Page:** When the source file has any type of characters that are less than 32 or more than 127, then you should indicate the code page value so the characters will be parsed correctly.
- **Data File Type:** This has to do with the number of bytes per character. If you are using Unicode text files or native data types in the file, then change this option.
- **Only Copy Selected Rows:** This option, together with the first and last row numbers, enables you to specify a consecutive chunk of rows to insert. This option is useful when a failure occurs in the insert, and you wish to go back and add the rows that have no errors. This option is important if the source file has a header row. By its very nature, a bulk insert defines the file structure and does not need the headers. In that case, select this option and set the First row to 2. When the Last row is 0, it will insert rows until the end of file.

Format Files

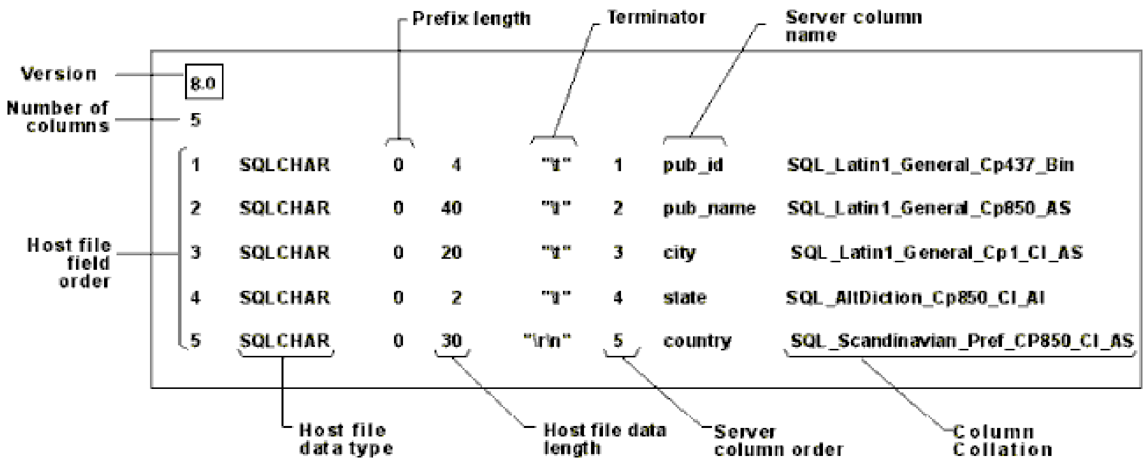
By default, data from the text file is matched column by column with the target table or view. The Bulk Insert task also loads all of the columns from the source file and expects the file’s columns to be delimited by commas. If there is any deviation from this default format, then you have to use a format file to specify the changes.

You use format files when the number of columns in the source file is not the same as the number in the target table or view, when the columns are not in the same order, or when the source file uses fixed-width fields or different delimiters.

NOTE Delimiters are not the same as separators, yet most people use the term *delimiter* when they mean *separator*. Even dialog box windows in SQL Server make this “error.” *Delimiter* describes the character(s) that delimit textual data. For example, quotation marks are the most common form of delimiter. *Separator* describes the character(s) that separate fields and rows in the text file. Commas, tabs, and spaces are the most common forms of field separator. New lines are the most common form of row separator. Nevertheless, since the screens use the term delimiter instead of separator, this course does as well.

Structure

Format files are tab-delimited files that specify the source file and destination table layouts. The beginning rows explain the overall structure of the format file and are followed by a row for each field or column within the source data file. Figure 2 shows a sample of a format file. This sample is from Microsoft SQL Server Books Online.



```
8.0
5
1   SQLCHAR   0   4    "\t"     1   pub_id     SQL_Latin1_General_Cp437_Bin
2   SQLCHAR   0   40   "\t"     2   pub_name   SQL_Latin1_General_Cp850_AS
3   SQLCHAR   0   20   "\t"     3   city       SQL_Latin1_General_Cp1_CI_AS
4   SQLCHAR   0   2    "\t"     4   state      SQL_AltDiction_Cp850_CI_AI
5   SQLCHAR   0   30   "\r\n"   5   country    SQL_Scandinavian_Pref_CP850_CI_AS
```

Figure 2. Sample bulk copy format file.

The information in the file is as follows:

- **Version:** The version of the bcp program, which is the same as the version of SQL Server. (SQL Server 2000 has an internal major version number of 8.)
- **Number of Columns:** Informs bcp of the number of rows in the rest of the format file, where each row describes each column of the source text file. The number of columns must be the same in every row in the source text file.
- **Host File Field Order:** The numerical position of the field or column in the source text file. These are ordinal positions that begin with 1.
- **Host File Data Type:** Indicates the data type in the text file. For ASCII data, use SQLCHAR. Integer data can be SQLINT. But since ASCII text files are readable, all the columns can be indicated as SQLCHAR. If the source file is a binary file from SQL Server, use the default data types of SQL Server.
- **Prefix Length:** Indicates that the length of the field is stored with the data. This is only used when bcp exports data from SQL Server. It is set to 0 for imports, which is all that the Bulk Insert task does.
- **Host File Data Length:** The maximum length, in bytes, of the data type. This value is used for truncation purposes. The source file column is terminated by a character.

- **Terminator:** Used to separate the fields. Each field can have its own delimiter. Notice that in Figure 2, the last field in the row has the newline indicators as its delimiter.
- **Server Column Order:** The order of the columns in the SQL Server destination table. Put a zero in this column to indicate that this source file column is to be omitted in the bulk insert.
- **Server Column Name:** The name of the destination column for that source file column. The name actually does not matter, but it must not be blank.
- **Collation:** The collation used for the character and Unicode data in the source file. If this column of the format file is empty, the Bulk Insert task will use the collation setting of the destination table or database.

All columns in the source text file must have an entry in the format file. If a source column is not to be transferred, set the Server column order to 0. Columns of the SQL Server table not found in the source text file, can be specified in the format file with a Host file field order of 0. These columns can also be left out of the format file altogether.

Creating a Format File

There are three ways to create a format file:

- **Manually:** Don't use this method if you can avoid it.
- **bcp:** This program has an interactive mode that prompts you for information. The program will look at the source file and ask you for the storage type, prefix length, field length, and terminators for each field in the text file. Although this is a lot better than manually creating the file, it is not a graphical utility but a character prompt program—if you make a mistake, you have to start over.
- **The DTS Bulk Insert Task:** There is a Generate button on the General page of the Bulk Insert Task Properties window. If you select the Use format file option, then you can click on the Generate button to create the format file. This is the easiest way to create a format file. However, it cannot handle all options. After generating the format file, manual editing may be required.

Applying the Bulk Insert Task

When the source data is in a text file and the destination is SQL Server, then the fastest way to load is via the Bulk Insert task. As long as there are no transformations and error logging is not necessary, this task is very useful.

Bulk insert input files must be in the correct format in order to reap the benefit of speed. If transformations or error logs are necessary, then you must use other tasks. For example, the Transform Data task can change the input data to match the destination table. It can also record error information and capture the invalid rows.

If the operation is to go in the opposite direction (transfer data from SQL Server into a text file), then you cannot use the Bulk Insert task. Again, the Transform Data task can handle transfers from SQL Server into a text file. Another solution would be to use the bcp program to export the data. In that case, you can use the Execute Process task to run the bcp program, or to run a command file that executes the bcp program.

Staging Tables

One way to use the speed of bulk insert is in a DTS package that uses different tasks to accomplish the overall objective. In this scenario, a package could use the Bulk Insert task to load the data as fast as possible into a staging table that has very few, if any, constraints or indexes on it. Once the data is in the staging table, a Transform Data task transfers and transforms the data into the actual destination tables and performs the error handling as well.

Staging tables are intermediate stops on the way to completing a data load. When you populate a data warehouse or data mart, a staging table, even a staging database, can serve as a useful tool in the process. Staging tables can combine the data from multiple data sources into one central location, so that you can use more standard techniques for building the data mart.

Staging tables bring the data together from disparate files. Then you need only one process to move the data to the final form. This technique makes the data load process more modular by splitting the data load into multiple steps. The more modular the process, the easier it is to make any necessary changes.

Breaking the process into steps makes error recovery easier. Two steps are used for a data load: first use bulk insert to copy the data into a staging table, then transform the data into its final form and table. If the second step fails for any reason, you don’t have to return to the original data sources; instead, use the staging tables and rerun the transformation.

Bulk Insert Package

Assume a situation in which two text files with employee information need to be loaded into the whistler data mart. Both of these files are comma-delimited and identical. One of these files will be loaded into a staging table, which has the exact same column structure as the input file. The other file will be loaded directly into a dimension table by using a format file. The reason for two copies of the same file is so that the data loads can run in parallel.

The only purpose behind this package is to show how the Bulk Insert task works.

Planning

Before beginning any design of a DTS Package, you must lay out a plan so that you know what the package should do. The three necessary items for data copies are: source, destination, and method.

- **What is the source?** In this scenario, the sources are comma-delimited files named employee.txt and employee_delimited.txt. Both files are in the folder for this chapter. Both files have a comma-delimited row with the column names in it.
- **What is the destination?** Employee_delimited.txt is used for the staging table, employee_stage. Employee.txt is destined for the employee_dim table. Both tables are in the whistler SQL Server database.
- **What is the transfer method?** There are no transformations. Both transfers will use the Bulk Insert task because all of the conditions for using that task are in place: no transformations are necessary, the sources are text files, Microsoft SQL Server tables are the destination. Since the number of columns in the employee.txt file differs from the number of columns in the employee_dim table, a format file is used.

Try It Out!

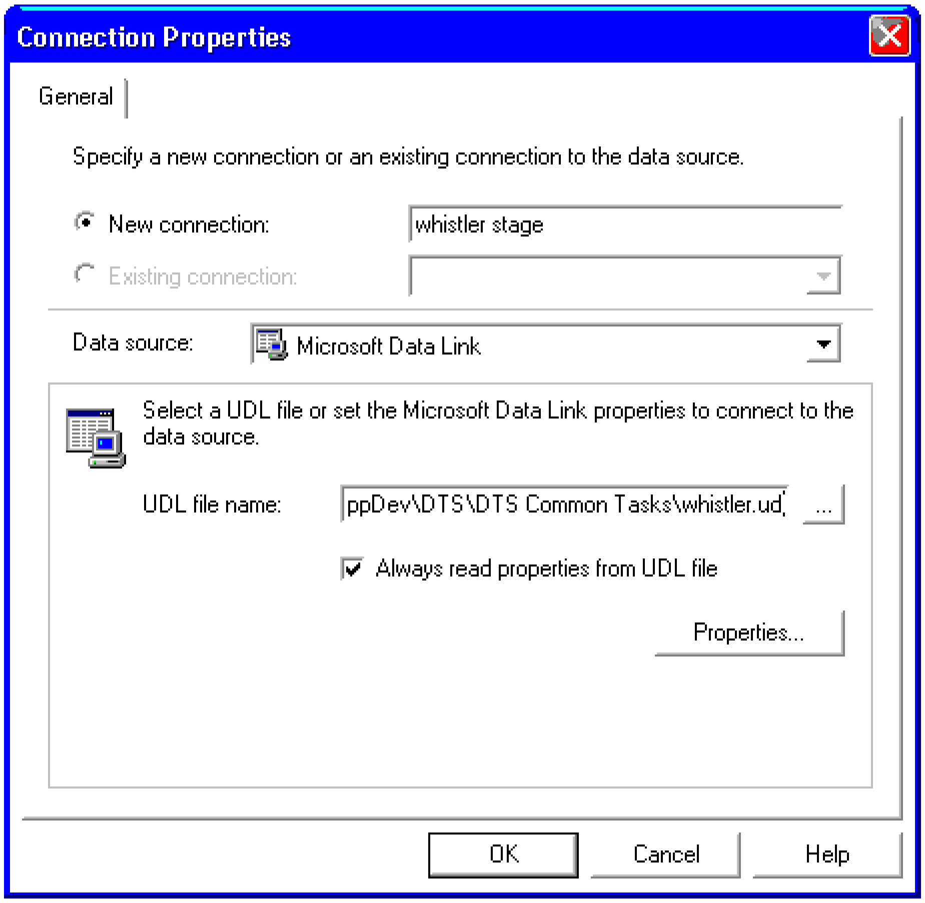
In these steps exercise, you will create a DTS package that copies the data from two different text files into two different tables.

1. In the Enterprise Manager, right-click the Data Transformation Services folder, and choose **New Package** from the context menu.

2. Click the **Microsoft Data Link** button on the Connection toolbar.

3. This brings up the Connection Properties dialog box with the data source selected and the New Connection option selected. Name the connection **whistler stage** in the text box next to the New Connection option. Then set the file name to **whistler.udl**, from the folder for this chapter.

4. Select the **Always read properties from UDL file** check box. This means that any changes to the UDL file are read into the DTS package connection. The window should look like Figure 3. Click the **OK** button.


Connection Properties

General

Specify a new connection or an existing connection to the data source.

(●) New connection: whistler stage

( ) Existing connection:

Data source: Microsoft Data Link

Select a UDL file or set the Microsoft Data Link properties to connect to the data source.

UDL file name: ppDev\DTS\DTS Common Tasks\whistler.udl

[✓] Always read properties from UDL file

Figure 3. The Connection Properties dialog box.

5. Click the **Bulk Insert Task** button on the Task toolbar.

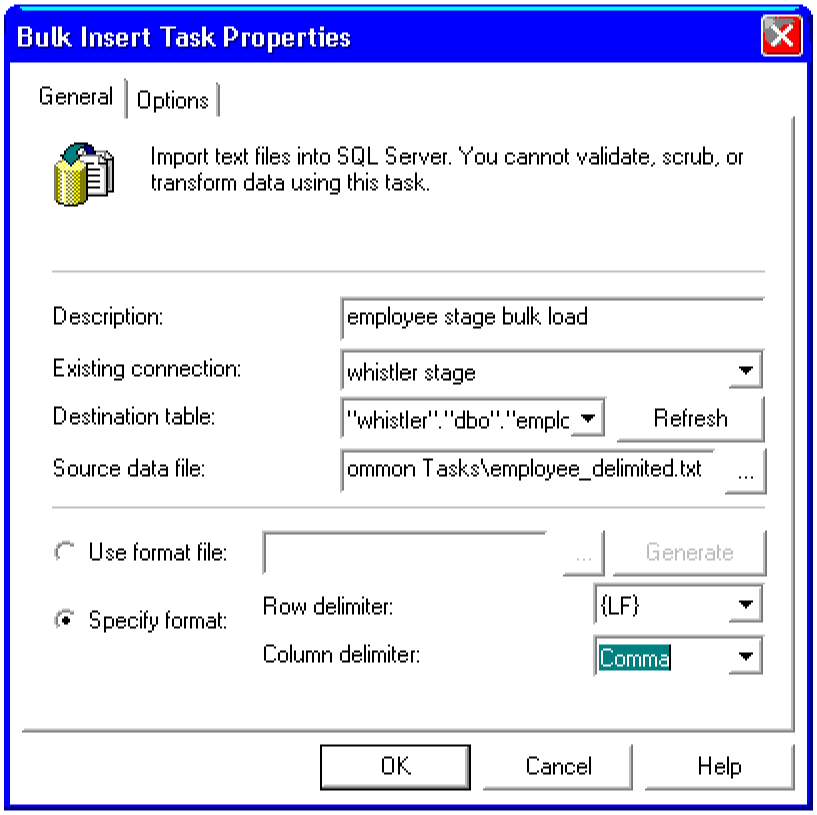
6. Type **employee stage bulk load** in the Description text box.

7. Choose **whistler stage** from the Existing connection drop-down list.

8. Choose the **employee_stage** table in the Destination table drop-down list.

9. Select the **employee_delimited.txt** file from the folder for this chapter as the Source data file.

10. Click the Specify format option, and select **Comma** from the Column delimiter drop-down list. Your screen should look like Figure 4.



Import text files into SQL Server. You cannot validate, scrub, or transform data using this task.

Description: employee stage bulk load
Existing connection: whistler stage
Destination table: "whistler"."dbo"."emplc
Source data file: ommon Tasks\employee_delimited.txt
Use format file:
[✓ Specify format:]
Row delimiter: {LF}
Column delimiter: Comma

Figure 4. The General page of the Bulk Insert Task Properties dialog box.

11. Click the **Options** tab and select the **Only copy selected rows** option.

12. Set the First row number to **2**, leave the Last row number at **0**, and click **OK**.

13. Notice that there are no arrows in the package. Notice also that unlike data transformations, the Bulk Insert task only uses one connection, the SQL Server destination database. Looking at Figure 4, you can see that the source information is a property of the task, and therefore no connection is necessary. Finally, although source text file connections can use the header row of the file, bulk insert cannot, and the row must be ignored.

14. Click the **Execute** button on the Designer toolbar, and then click **OK** in the Results message box.

15. If any errors appear in the Executing Package dialog box, be sure to double-click on them to find the error message. When you are finished, click **Done** button.

16. Click the **Save** button on the Designer toolbar. Name the package **bulk insert**, choose **SQL Server** from the location drop-down list, and click the **OK** button.

17. Verify the results by using Query Analyzer. Execute the following statement:

```
SELECT * FROM whistler.dbo.employee_stage
```

The table should have 11 rows with data in every column.

Format File Load

18. Return to the DTS Package Designer. Click the **Microsoft Data Link** button on the Connection toolbar.

19. This brings up the Connection Properties dialog box with the data source selected and the New Connection option selected. Name the connection **whistler dim** in the text box next to the New Connection option. Then set the file name to **whistler.udl**, from the folder for this chapter.

20. Select the **Always read properties from UDL file** check box, and then click **OK**.

21. Click the **Bulk Insert Task** button in the Task toolbar.

22. Type **employee dim bulk load** in the Description text box.

23. Choose **whistler dim** from the Existing connection drop-down list.

24. Choose the **employee_dim** table in the Destination table drop-down list.

25. Select the **employee.txt** file from the folder for this chapter as the Source data file.

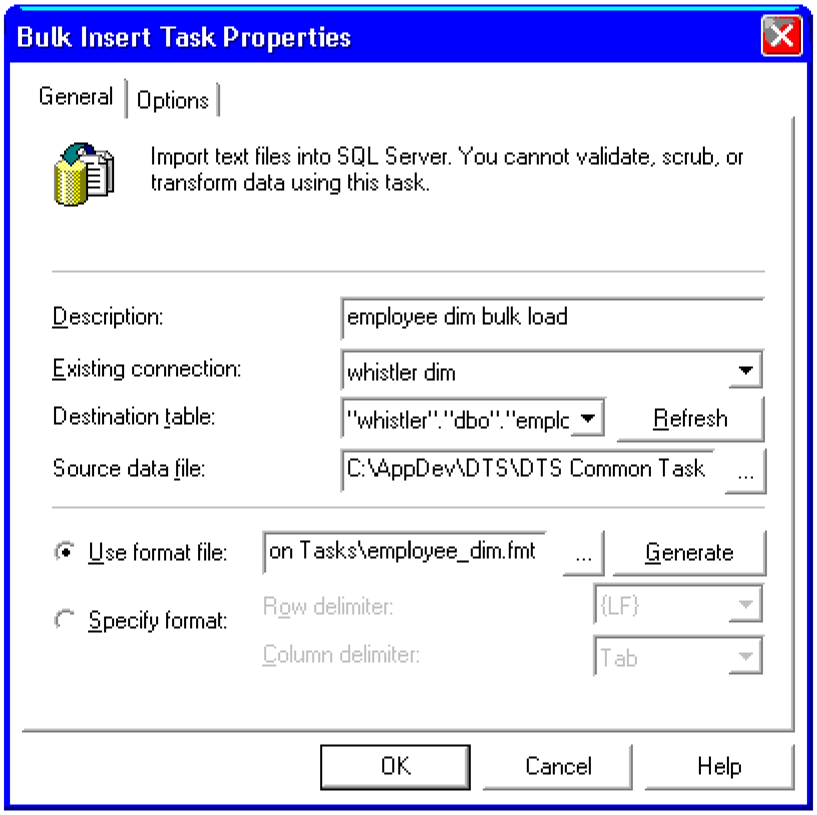
26. Click the **Use format file** option. As the format file, select the **employee_dim.fmt** file from the folder for this chapter. Your window should look like Figure 5. The contents of the format file are as follows:

```
7.0
9
1 SQLINT 0 0 "" 0 employee_dim_key
2 SQLCHAR 0 30 "," 2 employee_id
3 SQLCHAR 0 30 "," 3 last_name
4 SQLCHAR 0 30 "," 4 first_name
5 SQLCHAR 0 30 "," 5 title
6 SQLCHAR 0 30 "," 6 city
7 SQLCHAR 0 30 "," 7 region
8 SQLCHAR 0 30 "," 8 country
9 SQLCHAR 0 30 "\r\n" 9 postal_code
```

Notice that the first row has a zero in the server column order position and that the termination character is set to null (empty string). That is because there is no column in the source file for the employee_dim_key column. Yet it still has a host file field order of one. Finally, you'll notice that the server column order positions start at 2, because the first column of the table is skipped.

WARNING!	It might be nice to see how DTS can generate a format file, but the Generate process cannot handle a situation in which the number of columns differs between source file and table. You can build the format file using the employee_stage table and then modify it using Notepad to accommodate the extra column in employee_dim.

WARNING!	When you use the Generate button, a text file properties dialog box series runs. The second dialog box of the series has a check box for First row has column names. This check box is not used to build the format file. It is used to create column names in the last dialog box of the series.



Import text files into SQL Server. You cannot validate, scrub, or transform data using this task.

Description: employee dim bulk load

Existing connection: whistler dim

Destination table: "whistler"."dbo"."emplc

Source data file: C:\AppDev\DTS\DTS Common Task

(● Use format file: on Tasks\employee_dim.fmt

○ Specify format: Row delimiter: {LF} Column delimiter: Tab

Figure 5. The General tab of the Bulk Insert Task Properties dialog box, showing the use of a format file.

27. Click the **Options** tab and select the **Only copy selected rows** option.

28. Set the First row number to **2**. Leave the Last row number at **0**. Click the **OK** button. Your finished package will look something like Figure 6.

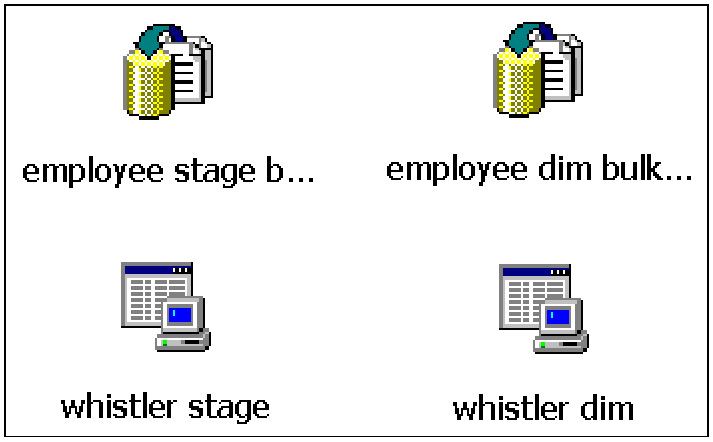


Figure 6. The completed bulk insert DTS package.

29. Click the **Save** button on the Designer toolbar.

30. Click the **Execute** button on the toolbar and then click **OK** in the Results message box.

31. If any errors appear in the Executing Package dialog box, be sure to double-click on them to find the error message. When you are finished, click the **Done** button.

32. Verify the results by using Query Analyzer. Execute the following statements:

```
SELECT * FROM whistler.dbo.employee_stage
SELECT * FROM whistler.dbo.employee_dim
```

There should be 22 rows in the employee_stage table, since this is the second time it has loaded. There should be 31 rows in the employee_dim table because it had 20 rows from another package that loaded data into it. If you run this job again, the row numbers will increase by 11 each time because there is no delete rows step in this package.

Execute SQL Task

The Execute SQL task is the Data Transformation Services task that runs SQL and extended SQL statements. This task enables you to extract and manipulate data on any data source that understands these statements—which includes most relational database sources.

Examples of SQL statements that are useful in a data load package are: dropping indexes before the load, deleting any data that might be in staging tables, or running a stored procedure to update other tables.

In order to use the Execute SQL task on a data source, you need to consider the following points:

- **SQL:** The data source must understand the SQL syntax.
- **Performance:** The performance of the task is tied to the performance of the SQL statement on the data source. DTS can do nothing to optimize the task.
- **Multiple Statements:** The Execute SQL task can be set to run more than one SQL statement. Multiple statements in one task run one at a time, just as if they were statements in a program.

TIP: In some relational databases, such as SQL Server, multiple statements can be run as a single batch, which improves overhead performance. In SQL Server, use the GO statement at the end of the batch.

- **Designer:** The Execute SQL Task Properties dialog box has a Build Query button that leads to a Query Designer so you can create statements graphically.

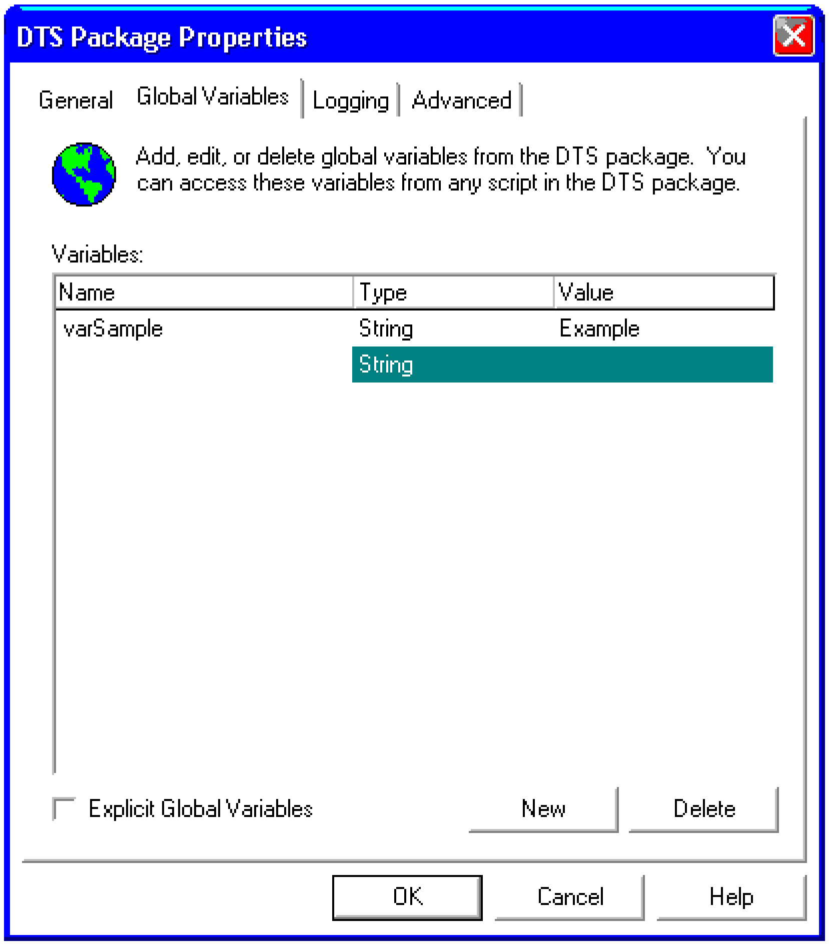
Parameterized Queries

The Execute SQL task can use parameterized queries to create dynamic and reusable packages. The only requirement for a data source is that the OLE DB provider that works with that data source must use the ICommand interface. DTS global variables are used to pass values into the parameters of the query.

Global Variables

DTS global variables are user-defined storage locations that contain specific values. By using global variables, you can design packages that pass information from one step to another, even across package executions.

You define global variables, along with an initial value, on the Global Variables tab of the DTS Package Properties dialog box, as shown in Figure 7.



Add, edit, or delete global variables from the DTS package. You can access these variables from any script in the DTS package.

Variables:

Name	Type	Value
varSample	String	Example
	String	

Figure 7. The Global Variables tab of the DTS Package Properties dialog box.

This page has a three-column list. The first column holds the variable names, the second column has the data types of the variables, and the last column has the initial value of that variable. The list is also accessible from within the Parameter Mappings dialog box of the Execute SQL Task Properties dialog box.

The package properties version of this window has an additional option called Explicit Global Variables. Set this option when you want variables to be defined here rather than in scripts. When you clear the option, you can define global variables in any script in a DTS package, just by using the DTSGlobalVariables collection. Another dynamic way to use global variables is with the /A switch of the dtsrun executable. With this method, you set the variable value when the package is executed.

Global variables can store a value before execution of the Execute SQL task, so that the variable can be used as an input parameter. You can also use global variables to store the results of an output parameter in a stored procedure.

Dynamic Queries

You can use input parameters in the SQL statements and map global variables to them in order to create dynamic queries. At runtime, the global variable values are used in place of the parameter placeholders in the SQL statement.

The steps involved in using parameters and global variables are:

- **Question Marks:** Question marks are used as parameters in the query. A parameterized query might look like:

```
SELECT * FROM product_dim
WHERE product_name = ?
AND category_name = ?
```

- **Position:** The way to identify a parameter is by its position in the query. When there is more than parameter, the first question mark is parameter 1, the second question mark is parameter 2, and so on. In the example above, Parameter 1 is some product name and Parameter 2 is some category name.
- **Global Variables:** You need to create the global variables that hold the values for the parameters in a query. For example, variables called ProductName and CategoryName hold values for the query above.
- **Mapping:** In the properties for the task, you map the global variables to the parameters in the query by position. In the example, the ProductName variable is mapped to the first parameter and the CategoryName variable is mapped to the second parameter.

Query Results

You can use output parameters in a query to get the results of a query and then store those results in global variables. Row values and rowsets can be stored in global variables.

Row Values

If a query in an Execute SQL task returns just one row, then global variables can be mapped to the column names used in the query. For example, consider a query that returns the units in stock and reorder level of a particular product in a product table. An example of this query follows:

```
SELECT units_in_stock, reorder_level
FROM products
WHERE product_id = 35
```

This query returns one row with two columns, units_in_stock and reorder_level. Instead of mapping the results into a destination source, you want to capture the values within the DTS package so they can be used in another task. That other task might be another parameterized query or perhaps an ActiveX Script task that does some calculations based on those values. Or these values might be needed in the ActiveX script of a step that determines whether or not a task runs.

In the properties of the Execute SQL Task dialog box, you can map these column names to global variables. Then any other task or script within the package can use the values.

Entire Rowsets

If the query returns multiple columns and rows, then the rowset can be passed into a global variable. That global variable becomes a Microsoft ActiveX Data Object (ADO) recordset—an in-memory lookup table that can be accessed via ActiveX scripts in other steps and tasks.

The advantage of this technique is that it allows you to access a set of data for internal processing within the package several times. You will not need to requery the data source every time the package needs to use the rowset. In many cases, data like this is saved in a staging table. Any task that needs the data also needs a connection to the staging table. Instead, use the global variable rowset to make multiple accesses easier.

The disadvantage is the amount of data in the rowset. Memory is a precious resource and it is not a good idea to use it for a large table when there are so many ways of accessing tables from within DTS. Performance also suffers when memory usage is high.

If you do use this technique, then all of the properties and methods of an ADO recordset are available through the global variable. You can navigate through the recordset, do searches on it, query it, and manipulate the contents. You do all of this in tasks and steps that can use ActiveX scripts.

Table_delete

The whistler database contains a stored procedure called table_delete that deletes all the rows from a parameter-specified table name. The procedure has one input parameter for the name of the table to empty. A SQL statement is built in a string variable, and then the sp_executesql system stored procedure is used to run that statement. The code for table_delete is as follows:

```
CREATE PROCEDURE table_delete
@Tablename nvarchar(30)AS
DECLARE @SQLString NVARCHAR(500)

SET @SQLString = N'DELETE FROM ' + @Tablename
EXEC sp_executesql @SQLString
```

When this procedure executes, a table name must be supplied for the parameter because no default value exists for the parameter.

You need to create a DTS package to run the stored procedure. The package and task need to be as flexible as the SQL Server procedure.

Planning

Before creating any package, you need a plan that defines the tasks and steps and connections to use. The DTS Package Designer is a blank slate, so you need to know where to start. Since this is not a data transfer, the planning is different.

Remember the following points in your planning:

- **Connection:** The stored procedure is in the SQL Server whistler database. A Microsoft Data Link file will be used to establish the connection.
- **Task:** The stored procedure is executed as part of a SQL statement, so the Execute SQL task will be used. The statement will execute the stored procedure using a question mark parameter for the procedure parameter.
- **Variable:** A global variable will be created to hold the value of the table name. The variable will be assigned to a table name from the whistler database.
- **Mapping:** The global variable must be mapped to the task's parameter.

Try It Out!

In these steps you will create a DTS package for deleting the contents of a table. The name of the table will be specified in a global variable and mapped to a parameterized query in a task to do the deletion.

1. Create a new package in Data Transformation Services.

2. Click the **Microsoft Data Link** button on the Connection toolbar.

3. This brings up the Connection Properties dialog box with the data source selected and the New Connection option selected. Name the connection **whistler** in the text box next to the New Connection option. Then set the file name to **whistler.udl**, from the folder for this chapter.

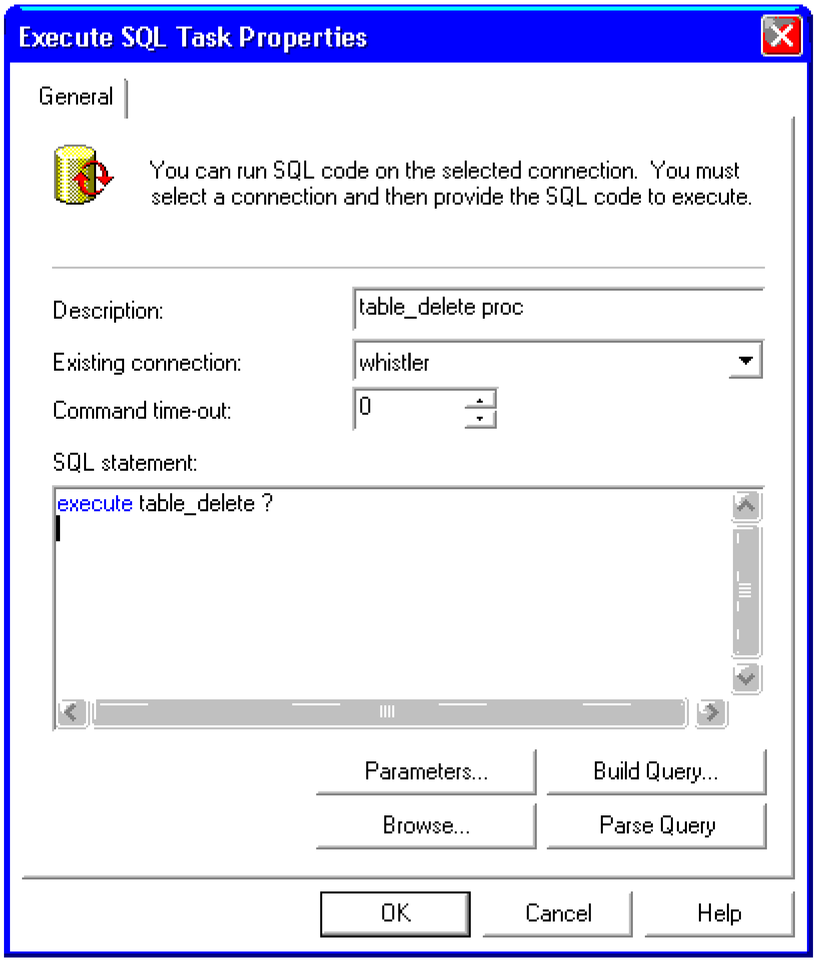
4. Select the **Always read properties from UDL file** check box, and then click **OK**.

5. Click the **Execute SQL Task** button on the Task toolbar.

6. Type **table_delete proc** in the Description text box.

7. Choose **whistler** from the Existing connection drop-down list.

8. Type **execute table_delete ?** in the SQL statement box. Your window should look like Figure 8.



You can run SQL code on the selected connection. You must select a connection and then provide the SQL code to execute.

Description: table_delete proc

Existing connection: whistler

Command time-out: 0

SQL statement:

```
execute table_delete ?
```

Figure 8. The Execute SQL Task Properties dialog box.

9. Click the **Parse Query** button, and then click **OK** in the Designer message box.

10. Click the **Parameters** button. This brings up the Parameter Mapping dialog box.

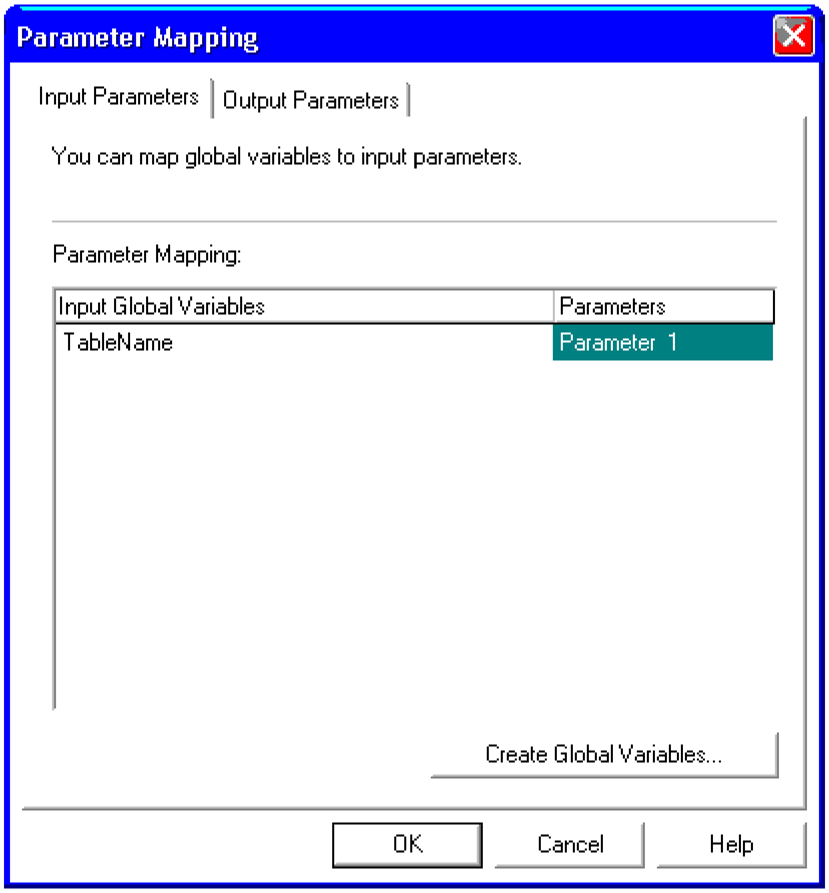
11. On the Input Parameters page, click the **Create Global Variables** button.

12. The Global Variables dialog box is very similar to the package properties global variables tab. Type **TableName** as the variable name.

13. Select **String** from the Type column drop-down list.

14. Type **employee_dim** in the Value column for the variable you created, and then click **OK**. There is no need to put quotes around the value you typed; that will be handled within the DTS package.

15. Back in the Parameter Mapping dialog box, choose **TableName** from the drop-down list in the Input Global Variables column next to the parameter called Parameter 1. Your window should look like Figure 9.



You can map global variables to input parameters.

Parameter Mapping:

Input Global Variables	Parameters
TableName	Parameter 1

Figure 9. The Parameter Mapping dialog box.

16. Click **OK** in this dialog box, and then click **OK** on the Execute SQL Task Properties dialog box.

17. Click the **Save** button on the Designer toolbar. Name the package **table_delete**, set the location drop-down list to **SQL Server**, and click **OK**.

18. Click the **Execute** button on the toolbar and then click **OK** in the Results message box.

19. If any errors appear in the Executing Package dialog box, be sure to double-click on them to find the error message. When you are finished, click the **Done** button.

20. Verify the results by using Query Analyzer. Execute the following statement:

```
SELECT * FROM whistler.dbo.employee_dim
```

The employee_dim table should contain zero rows.

Transform Data Task

The Transform Data task is probably the most common task in Data Transformation Services. After all, this task bears the name of the product and is the task that the DTS Import/Export Wizard creates whenever data is being copied. The wizard may use other tasks as well, but it uses the Transform Data task for copying data.

The Transform Data task not only copies and modifies data, but it can also do error handling and exception reporting. Although this task can be used with any OLE DB or ODBC data source, it has special options for Microsoft SQL Server. These options enable you to optimize the task for faster execution on SQL Server.

The DTS Data Pump

At the heart of the transformation is the DTS data pump. The data pump is an internal Component Object Model (COM) object that has a set of movement interfaces. DTS uses these interfaces to move and modify the data.

The data pump architecture has the following features:

- **Copying:** The data pump does high-speed batch copying of data, regardless of whether the data is being modified. The application of *batch* copying is dependent on the destination data product. If the destination supports batches, then the data pump uses that feature.
- **Transformations:** The data pump has several built-in transformations for string data, such as Uppercase, Lowercase, and Middle of String. By using these, you get better performance than by using scripts.
- **Scripting:** The data pump enables you to code transformations in any ActiveX scripting language. Scripting extends the built-in transformations, so you can implement complex logic in the data pump process.
- **Extensibility:** Since the DTS data pump is a COM object, you can write custom tasks in any COM-compliant language. DTS packages can use these custom tasks in the same way as built-in tasks, using icons and property dialog boxes.
- **Phases:** Data in the data pump goes through different phases as it is copied and transformed. DTS supplies hooks into these phases so that you can write ActiveX scripts to tap into them. This means that you have additional places to customize and enhance transformations. (Multiple phases are covered in another chapter of this course.)

Data Pump Process

At the most basic level, the data pump reads rows from a source, transforms those rows as needed, and then writes rows to the destination. This process seems like a normal transformation process, and it is except that the data pump is the underlying architecture of transformation.

The process gets more complicated when there are multiple batches or error handling. If there is just one batch, no error processing, and the destination is a SQL Server database, the basic process goes as follows:

1. **Connect:** The data pump establishes connections with the data source and data destination first. All connections, whether OLE DB or ODBC, are made through the OLE DB architecture.

2. **Metadata:** After the connections are made, the data pump finds the attributes of the source and destination columns by reading the metadata that OLE DB exposes. The metadata consists of column names, their data types, and nullable properties. The data pump stores this information and uses it as it copies and transforms the data.

3. **Transformation Data:** The data pump gathers information from the task definition about the transformation. This includes the string transformations that are being used, ActiveX scripts, and custom mapping between the source and destination columns.

4. **Read and Process:** The data pump then reads rows from the source, one at a time, and applies the transformation. It's important to understand that the data pump does this process on a row-by-row basis; this is why the Bulk Insert task performs better in some cases. On the other hand, this means that row information can be processed in some very inventive ways.

5. **Writes:** Once the transformation is complete, the data pump attempts to write the row to the destination buffer. Once all of the rows are processed, the entire buffer is committed to the destination source. Using a buffer means that, by default, an error in the process can cause rejection of the entire batch. However, this behavior can be modified.

Data Pump Tasks

Several different tasks in Data Transformation Services use the DTS data pump. As a group, these tasks are known as Transforming Data tasks.

- **Transform Data Task:** The most basic data pump implementation. This task copies data from data sources to destinations, and optionally can add modifications as it copies the data.
- **Data Driven Query Task:** Performs flexible Transact-SQL actions on the data including the use of stored procedures and adding, removing, and modifying data. This task is covered in another chapter of this course.

- **ParallelDataPumpTask Object:** Performs the same operations as the Transform Data and Data Driven Query tasks, but does them on hierarchical data. This data may come from special ADO object models using the Shape statement, or even XML data, which usually has a hierarchical structure. This object can only be accessed programmatically.

Creating Transformations

Copying data from a source to a destination involves a transformation of some sort. Even if there are no changes to the data, the copy operation itself is a transformation. Remember that the data pump architecture involves reading and processing data, even if there are no changes to the data. Even in a copy, column mapping occurs to match the source columns to the destination columns.

Transformation Types

The data pump uses transformations to process source data as it is copied to the destination. Transformations are defined in the Transform Data task. In other chapters of this course, the simplest transformation, copy column, is used to copy data from the source to the destination.

Transformations can be assigned to individual columns that are mapped, to groups of columns, or to uneven mappings, such as one source column that is copied to multiple destination columns, or multiple source columns to one destination column. In other words, every match between source and destination columns can have a separate transformation.

All of the transformation types in the following list are available through the DTS data pump object and the Transform Data task.

- **Copy Column:** The standard source-to-destination copy. No data modification occurs in the copy process.
- **ActiveX:** Involves user-defined complex logic. Use it to combine multiple columns into one, or to transform one column into multiple columns. It also provides a way to define conditional modifications to data or to change data of one type to another.
- **Datetime String:** Transforms a source date value into a new destination format.
- **Uppercase String:** Converts the source string data into all uppercase characters and copies data into the destination.
- **Lowercase String:** Converts the source string data into all lowercase characters and copies data into the destination.

- **Middle of String:** Extracts a substring from the source data before copying it to the destination. This transformation has options for changing the case and for trimming white space.
- **Trim String:** Converts the source string data by trimming white space characters and copies the data to the destination. The options for this transformation are leading, trailing, or embedded white space.
- **Read File:** If a column in the source data has a file name, this transformation copies the contents of that file to the destination column.
- **Write File:** Involves two source columns, one of which has a file name, and the other with the source data. The transformation creates the named file by copying the source data from the second column into the file indicated in the first column.
- **Custom:** If you write a C++ program that uses the data pump object, you can refer to that program through this transformation type.

You assign one of these transformation types to each mapping in the Transform Data task.

Column Mappings

After you have planned your transformations, you create the column mappings. Column mappings determine which column(s) from the source match column(s) on the destination. Each mapping represents a separate data pump operation. Each data pump operates for each row as it is processed.

You create column mappings on the Transformations tab of the Transform Data Task Properties dialog box. The mappings appear as arrows that point to the Destination list box columns.

There are three general categories of column mappings and each is represented by a particular type of graphic:

- **One-to-One:** A single source column that maps to a single destination column. The graphic is one arrow that originates from one column name in the Source list and points to one column name in the Destination list. Remember that this arrow represents just one data pump operation.
- **Symmetric Many-to-Many:** A mapping of a number of columns in the source to an equal number of columns in the destination. The user sets the exact match order of the columns. The graphic is again a single arrow from the Source list to the Destination list. The difference is that the arrow has branches at each end that point to the column names that participate in the mapping. This type of mapping involves only one data pump operation.
- **Asymmetric:** Involves unequal numbers of columns on the source side and destination side. There are three types of asymmetric

mappings: fewer source columns mapped to more destination columns; more source columns mapped to fewer destination columns; and mappings with no source or no destination. Whatever the choice, one arrow represents one data pump operation.

One example of an asymmetric mapping is reading FirstName and LastName source columns and concatenating them into a FullName destination column. Another example is reading data from global variables into destination columns. The global variables might hold the datetime and operator name of the package execution.

A Transform Data task can involve any number of each of these mapping types. Each arrow represents one data pump operation.

Transformation Optimization

When you design the column mappings, you want to maximize each data pump operation so that the total number of data pumps is as low as possible. To maximize the operation, reduce the separate column mappings within the task.

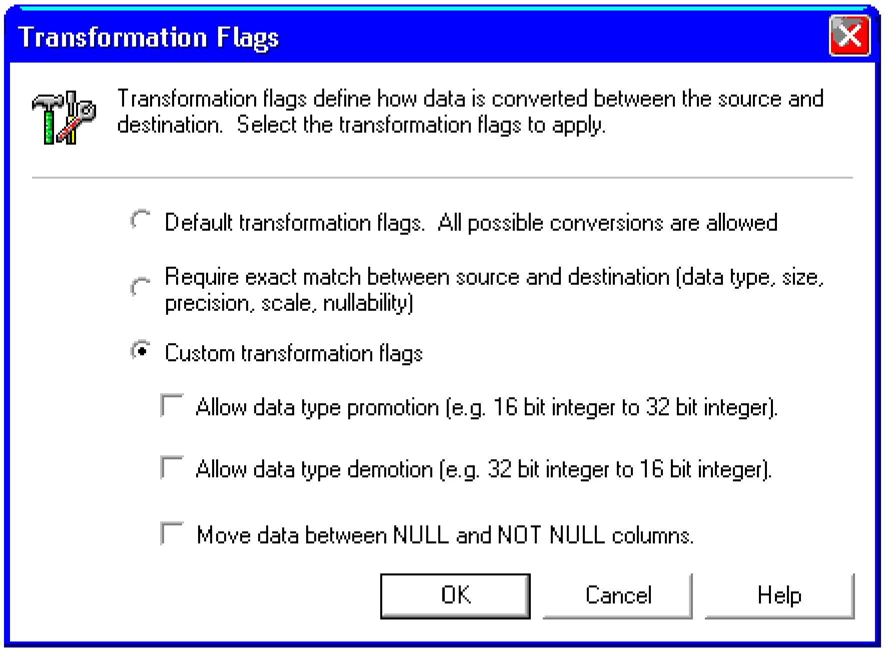
Remember that each mapping—each arrow—represents one data pump operation. Therefore, you obtain optimal performance by using many-to-many mappings. Grouping common transformations improves performance.

Sometimes groups can be created from different transformations by using their similarity and the options of a particular transformation type. For example, if you have an Uppercase string transformation and a Trim string transformation, you can combine them into one Trim string transformation and specify the uppercase option. (This assumes that trimming the original Uppercase transformation has no adverse effect on the data.)

Data Type Conversion

One last item in standard transformations concerns data type conversions—specifically the conversion of data types involving *promotion*, *demotion* and null behavior. This is especially important when data is copied between different data source products, and the data types do not exactly match.

When the source and destination data types match exactly, there are no problems with the copy. The question is: what to do if they do not match exactly? You need to address this question for each transformation in the task, or each arrow representing a column mapping. In the Transform Data Task Properties window, you make this choice with transformation flags. To do this, right-click on a transformation (arrow) and choose the **Flags** option from the context menu. The Transformation Flags dialog box will appear as shown in Figure 10.



Transformation Flags

Transformation flags define how data is converted between the source and destination. Select the transformation flags to apply.

( ) Default transformation flags. All possible conversions are allowed

( ) Require exact match between source and destination (data type, size, precision, scale, nullability)

(●) Custom transformation flags

[ ] Allow data type promotion (e.g. 16 bit integer to 32 bit integer).

[ ] Allow data type demotion (e.g. 32 bit integer to 16 bit integer).

[ ] Move data between NULL and NOT NULL columns.

Figure 10. The Transformation Flags dialog box.

There are three ways to go with these flags:

- **Allow all Possible:** Default transformation flags is the default setting and it allows all possible conversions. DTS performs best-fit conversions between the source and destination data types and widths as well. For example, if the source string data is wider than the destination column, DTS truncates the data. DTS will do what it can to make the copy succeed.
- **Exact Only:** Require exact match between source and destination is the most stringent option. If there is no match in type, size, precision, scale, or nullable properties, then an error occurs in that transformation. This is extreme, but strict enforcement is sometimes desirable.
- **Custom:** Custom transformation flags allow you to find a middle ground between all possible and exact match only. By selecting this option, you can set any of its three suboptions to allow promotion, demotion, or nullability. Selecting all three check boxes is the same as selecting **the allow all possible** option.
  - **Promotion** means changing from lower precision to greater precision, such as converting from a 16-bit integer to a 32-bit integer.
  - **Demotion** means changing to a lower precision data type. This is the opposite of promotion and could result in an overflow situation.

- **Nullability** involves copying data from source columns that allow NULLs to destination columns that do not allow NULLs.

WARNING! Keep in mind that similar data types in different products can differ in important ways. OLE DB providers for sources perform the implementation of those types. Therefore, you need to know how OLE DB implements those types when choosing a conversion option.

Building Transformations

To learn how to set up transformations, you will create a DTS package to transfer data from a tab-delimited source text file into a new table in the whistler SQL Server database. The text file consists of employee-type information with a header row. The new table, called test_data, will have columns that exactly match the columns in the source file. Some of the data has to be transformed before being copied into the test_data file. The source columns and their transformations are in Table 1.

Source Column	Transformation
EmployeeID	Direct Copy
LastName	Direct Copy
FirstName	Direct Copy
Title	Lowercase
City	Direct Copy
Region	Uppercase and Middle of String: Positions 1 and 2
Country	Uppercase
PostalCode	Direct Copy
BirthDate	Convert Date Format to dd MMM yyyy

Table 1. Transformation requirements for data copy.

Try It Out!

In these steps you will create a DTS package to copy and transform data from a text file to a SQL Server table. The columns of the text file will be copied to the columns of the table, but some of the columns will be changed in various ways.

1. In the Enterprise Manager, right-click the Data Transformation Services folder and choose **New Package** from the context menu.

Connections

2. Click the **Text File (Source)** button on the Connection toolbar.

3. This brings up the Connection Properties dialog box with the data source selected and the New Connection option selected. Name the connection **Test data file** in the text box next to the New Connection option. Then set the file name to **test_data.txt**, from the folder for this chapter. Click the **Properties** button.

4. In the Text File Properties dialog box, select the **First row has column names** check box and make sure that the Text qualifier drop-down list is set to **Double Quote**. Click the **Next** button.

5. In the Specify Column Delimiter dialog box, notice that DTS reads the file and figures out that tabs are being used as delimiters. Check the Preview of the data, making sure that DTS has the columns and data correct. Note the format of the dates. Click **Finish** and then click **OK** in the Connection Properties dialog box.

6. Click the **Microsoft OLE DB Provider for SQL Server** button on the Connection toolbar

7. This brings up the Connection Properties dialog box with the data source selected and the New Connection option selected. Name the connection **whistler** in the text box next to the New Connection option. Then set the database drop-down list to **whistler** and then click **OK**.

Task Creation

8. On the Designer menu, select **Task|Transform Data Task**.

9. Click the **Test data file** connection icon, and then click the **whistler** connection icon. Depending on icon placement, your design window should look like Figure 11.

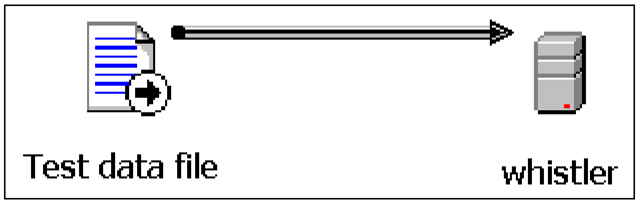


Figure 11. The DTS package with the Transform Data task.

10. Double-click the **Transform Data Task** arrow that connects the connection icons.

11. On the Source tab of the Transform Data Task Properties dialog box, type **load test_data** as the description, and make sure that the Table/View text box is pointing to the **test_data.txt** file. Click the **Preview** button, and after reviewing the data, click **OK**.

Destination Table Creation

12. Click the Destination tab.

13. The table that is the target of this copy does not currently exist in the whistler database, so you need to create it. The Destination tab has a button for that purpose, so click **Create**.

14. When you create a table in the properties of the Transform Data task, DTS assumes that the table will have the same column names as the source data. DTS also assigns the varchar data type to every column with a maximum length of 255. Although the test_data table will have the same column names, the data types will be different. The SQL statement to create test_data is in the test_data_table.sql file in the folder for this chapter. Using **Notepad**, open the **test_data_table.sql** file.

NOTE 255 is the old SQL Server (before 7.0) maximum length for a character or variable character data type. It's interesting that it is also the default value in DTS.

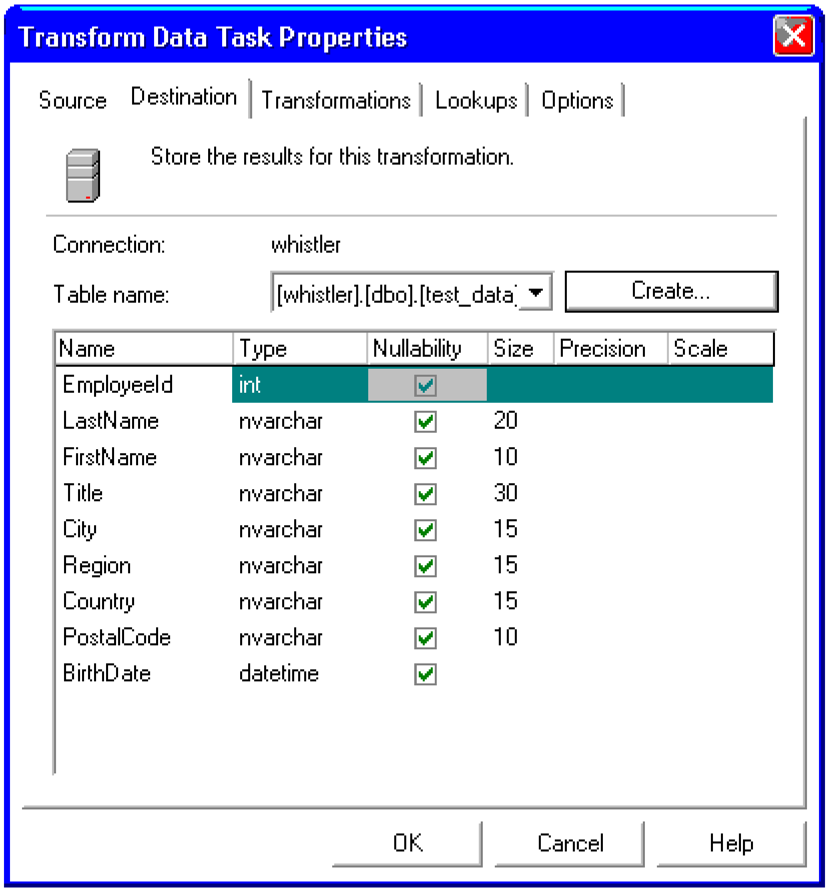
15. Select all of the text in the file and copy it to the Clipboard.

16. In the Create Destination Table dialog box, highlight all of the text in the SQL Statement box, and paste in the statement from the Clipboard. The statement will be the following code:

```
CREATE TABLE [dbo].[test_data] (
        [EmployeeId] [integer]NULL,
   [LastName] [nvarchar] (20) NULL ,
   [FirstName] [nvarchar] (10) NULL ,
   [Title] [nvarchar] (30) NULL ,
   [City] [nvarchar] (15) NULL ,
   [Region] [nvarchar] (15) NULL ,
   [Country] [nvarchar] (15) NULL ,
   [PostalCode] [nvarchar] (10) NULL ,
   [BirthDate] [datetime] NULL
) ON [PRIMARY]
```

17. Click the **OK** button. The Destination tab will look like Figure 12. When you click **OK**, the new table is created immediately. It is not created later when the package and task are executed.

TIP: If you want to create a table in the package at runtime, use the Execute SQL task.



Store the results for this transformation.

Connection: whistler

Table name: [whistler].[dbo].[test_data]

Name	Type	Nullability	Size	Precision	Scale
EmployeeId	int	✓			
LastName	nvarchar	✓	20		
FirstName	nvarchar	✓	10		
Title	nvarchar	✓	30		
City	nvarchar	✓	15		
Region	nvarchar	✓	15		
Country	nvarchar	✓	15		
PostalCode	nvarchar	✓	10		
BirthDate	datetime	✓			

Figure 12. The Destination tab for the test_data table.

Copy Columns

18. Click the Transformations tab and click the **Delete All** button to drop all of the default transformations.

NOTE Now that you know how to do transformations, you understand that the default choice is one-to-one column mappings. This is the least efficient way to copy data, because each arrow represents one data pump operation. In this sample, every row would have nine separate data pump operations.

19. Remember that most of the row is transferred using direct copies. The most efficient manner to do this is to group the columns into a many-to-many copy column transformation. In the Source list, click the **EmployeeID** column, click and hold the **CTRL** key, and click the **LastName**, **FirstName**, **City**, and **PostalCode** columns.

20. In the Destination list, click the **EmployeeID** column, click and hold the **CTRL** key, and click the **LastName**, **FirstName**, **City**, and **PostalCode** columns.

21. With all of these source and destination columns highlighted, click the **New** button.

22. In the Create New Transformation dialog box, choose **Copy Column** from the list and click **OK**.

23. On the General tab of the Transformation Options dialog box, type **copy column transformation** in the Name text box and click the **Properties** button.

24. In the Column Order dialog box, verify that the columns are properly matched. If they are out of order, you can fix them by clicking the drop-down lists in the columns. Click the **OK** button. Click **OK** again in the Transformation Options dialog box.

25. Back on the Transformations tab, select **copy column** transformation in the Name drop-down list and click the **Test** button.

26. Clicking this button actually performs the copy and transformation currently selected. Furthermore, it writes the output from the transformation into a temporary destination text file. This allows you to find errors before completing the task and package so that you can debug the task. Click the **OK** button in the Results message box.

WARNING! Since DTS is using a temporary file, there is no indication that the actual copy and transformation will work on the destination. This only happens when the package is executed. Furthermore, not all transformations can be tested this way.

27. In the Testing Transformation dialog box, click the **View Results** button to see the copied data. After reviewing the results, click **OK** and then click **Done**.

String Conversions

28. There are still four more transformations to create. Each is a one-to-one transformation. Each transformation begins the same way: select the column name in the Source list, select the matching column name in the Destination list, and click the **New** button.

29. Convert the title to lowercase characters. Start the **Title** transformation as specified in Step 28.

30. In the Create New Transformation dialog box, choose **Lowercase String** from the list and click **OK**.

31. On the General tab of the Transformation Options dialog box, type **lowercase transformation** in the Name text box and click **OK**.

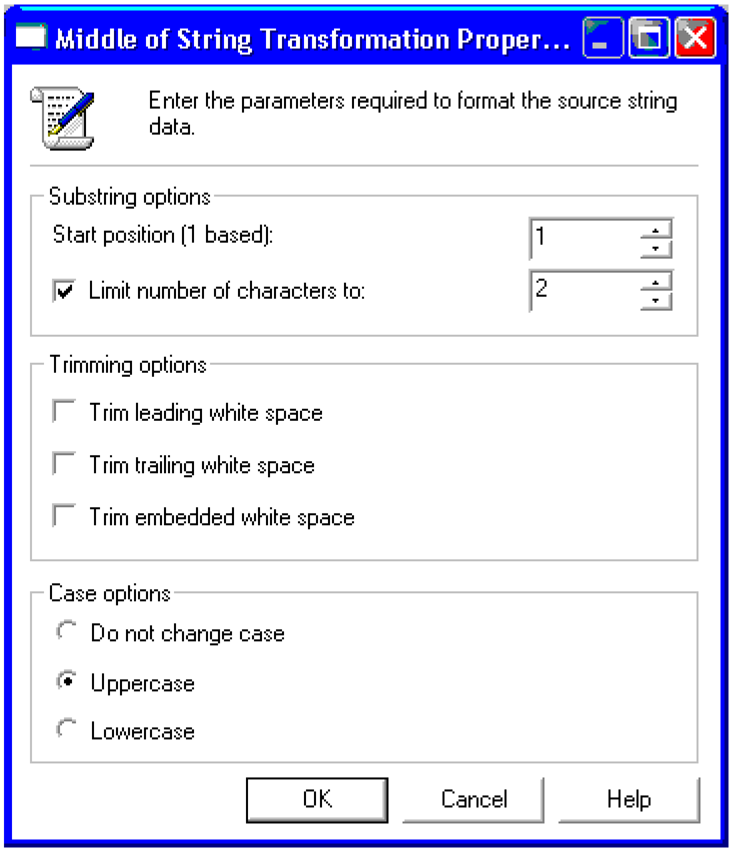
TIP: There is no need to check the properties, which is the column order, because there is only one column in the transformation.

32. Only the first two characters of Region need to be copied, but they also must be converted to uppercase characters. Start the **Region** transformation as specified in Step 28.

33. In the Create New Transformation dialog box, choose **Middle of String** from the list and click the **OK** button.

34. On the General tab of the Transformation Options dialog box, type **middle of string and uppercase transformation** in the Name text box and click the **Properties** button.

35. In the Middle of String Transformation Properties dialog box, set the Start position (1-based) to **1**. Select the **Limit number of characters to** check box, and set the value to **2**. Select the **Uppercase** option. Your screen should look like Figure 13.


Middle of String Transformation Proper...
Enter the parameters required to format the source string data.
Substring options
Trimming options
Case options
OK
Cancel
Help

Start position (1 based): 1

[✓] Limit number of characters to: 2

[ ] Trim leading white space

[ ] Trim trailing white space

[ ] Trim embedded white space

Do not change case [✓ Uppercase] Lowercase

Figure 13. The Middle of String Transformation Properties dialog box.

36. Click the **OK** button and then click **OK** in the Transformation Options dialog box.

37. Back on the Transformations tab, select **middle of string and uppercase transformation** in the Name drop-down list and click the **Test** button. Click **OK** in the Results message box and click the **View Results** button on the Testing Transformation dialog box.

38. Review the output, click **OK**, and then click **Done**.

39. Convert Country to uppercase characters. Start the **Country** transformation as specified in Step 28.

40. In the Create New Transformation dialog box, choose **Uppercase String** from the list and click **OK**.

TIP:	You might have noticed by now that the Transformation Options dialog box contains a sequence number. This number indicates the order of the data pump operations. The sequence is determined by the order you use to create the transformations.

41. On the General tab of the Transformation Options dialog box, type **uppercase transformation** in the Name text box and click **OK**.

DateTime Conversion

42. The format of the source BirthDate column needs to be changed since the text file is using pipes in between date parts. The output should be something more standard. Start the **BirthDate** transformation as specified in Step 28.

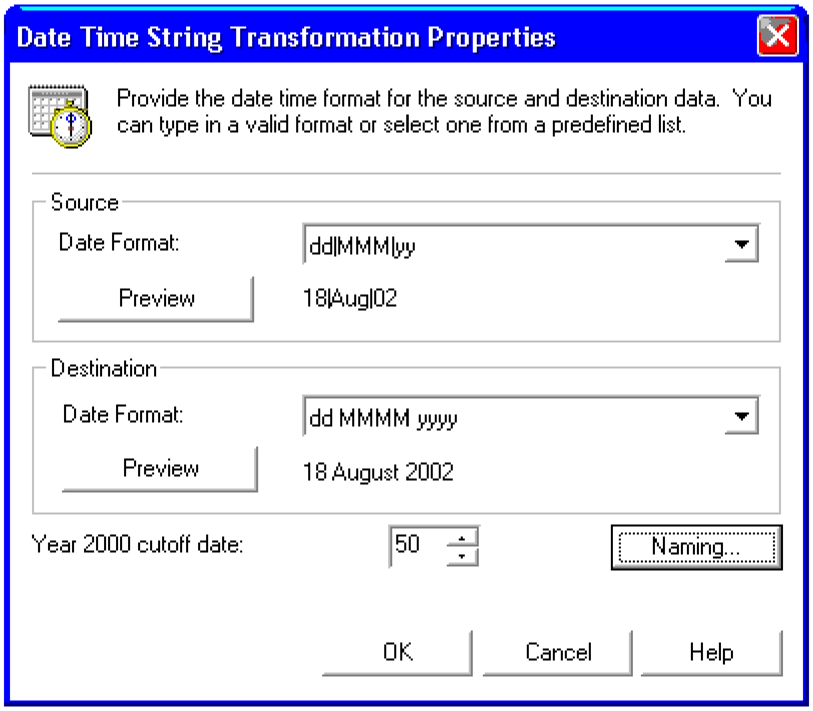
43. In the Create New Transformation dialog box, choose **DateTime String** from the list and click **OK**.

44. On the General tab of the Transformation Options dialog box, type **datetime transformation** in the Name text box and click the **Properties** button.

45. This brings up the Date Time String Transformation Properties dialog box. Here you set the source and destination formats for the dates/times. There is a check box to handle year 2000 issues as well. You can select the formats from a predefined list or type them in for nonstandard formats. Clear the Source **Date Format** entry, and type in **dd|MMM|yy**. Click the **Preview** button to see the current date in that format.

46. Clear the Destination **Date Format** entry, and type in **dd MMMM yyyy**. Click the **Preview** button.

47. Change the Year 2000 cutoff date to **50**. This means that any two-digit year greater than 50 will be interpreted as 19xx, while values less than 50 will be 20xx. Your screen should look like Figure 14.


Date Time String Transformation Properties
Provide the date time format for the source and destination data. You can type in a valid format or select one from a predefined list.
Source
Date Format:
Preview
Destination
Date Format:
Preview
Year 2000 cutoff date:
Naming...
OK
Cancel
Help

Source Date Format: ddlMMMlyy — 18lAugl02

Destination Date Format: dd MMMM yyyy — 18 August 2002

Year 2000 cutoff date: 50

Figure 14. The Date Time String Transformation Properties dialog box.

48. Click the **Naming** button. The Calendar Names dialog box shows the strings that are used for months, based on numerical values. Click **OK** to close the Calendar Names dialog box, click **OK** again to close the Date Time String Transformations Properties dialog box, and then click **OK** in the Transformation Options dialog box, to return to the Transformations tab.

WARNING!	Destination datetime formats and strings have no impact on Microsoft SQL Server. SQL Server has a datetime format that handles dates. This column format overrides any options selected in these dialog boxes. The only option that will make a difference is the Year 2000 cutoff date.

49. Back on the Transformations tab, select **datetime transformation** in the Name drop-down list and click the **Test** button. Click **OK** in the Results message box and click the **View Results** button in the Testing Transformation dialog box.

50. Notice that there are two dates in 20xx format, 2037 and 2048. Since these are birthdates, they cannot be correct, so the Year 2000 cutoff date needs to be changed. Click **OK** and then click **Done**.

51. Back on the Transformations tab, select **datetime transformation** in the Name drop-down list and click the **Edit** button. Click the **Properties**

button and change the Year 2000 cutoff date to **35**. Click the **OK** button and click the **OK** button in the Transformation Options dialog box. The Transformations tab should look like Figure 15.

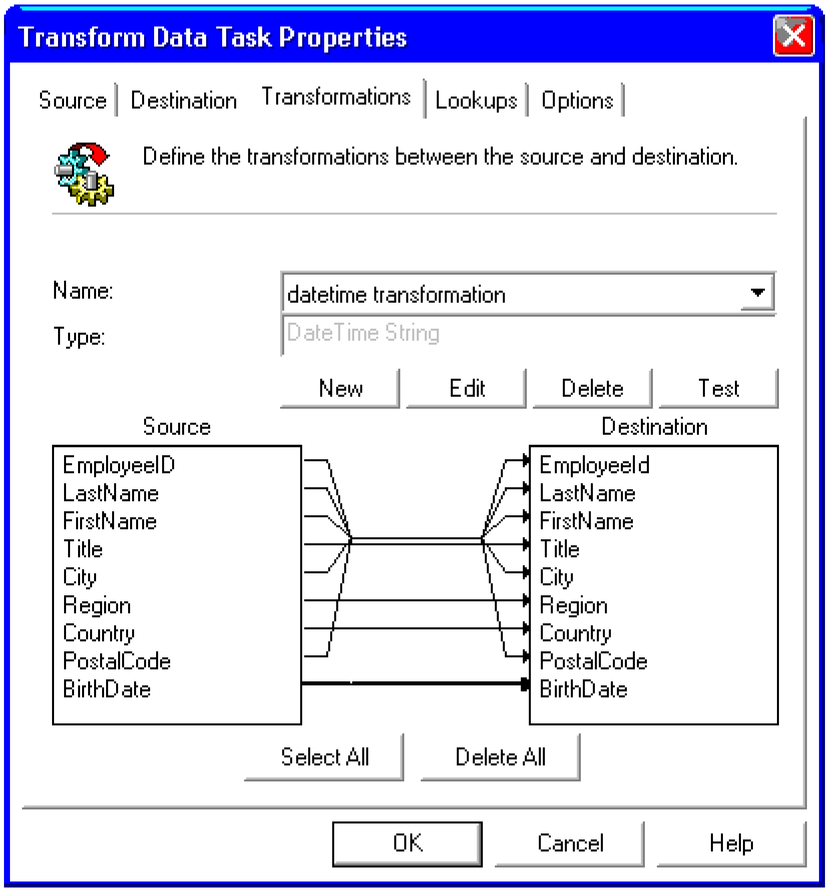


Figure 15. The Transformations tab of the Transform Data Task Properties dialog box.

52. Select **datetime transformation** in the Name drop-down list and click the **Test** button. Click the **OK** button in the Results message box and click the **View Results** button in the Testing Transformation dialog box.

53. The dates should be okay now, with all the years in 19xx format. Click the **OK** button and then click **Done**. Leave the Transform Data Task Properties dialog box open.

Error Handling

When you develop a DTS package, you need a plan for error handling. This plan has to handle incorrectly formatted source data that cannot be inserted into the destination source.

If incorrect properties are set for transformations, then virtually every row will fail and the step of the task will fail. The plan must handle the type of error that affects only some of the rows. This error type indicates an inconsistency in the source data or individual errors in the data.

These errors happen mainly as a result of transformations or inserts.

- **Transformations:** If the data modification is invalid, DTS will raise an error. This normally will not happen with standard data pump operations, but can happen in the logic of an ActiveX script.
- **Inserts:** If the destination constraints are violated, the row cannot be inserted into the destination and an insert error can result.

Error Files

Exception files can capture the rows that are causing the errors so that you can examine the data and debug the problem areas. Data Transformation Services does not capture transformation errors by default. You have to set up the exception files on the Options tab of the Transform Data Task Properties dialog box. The top of this tab contains an Exception file section with the following entries and options:

- **Name:** Type in the full path for the exception file name. The file name must include the extension; there is no default extension. This file will contain information about the errors and/or exceptions. There is an ellipsis button for selecting the path and file.
- **7.0 Format:** For backward compatibility with Microsoft SQL Server 7.0. When you select this option, there will be only one file that contains the errors as well as the exceptions. Clear this check box when you want to use separate files.
- **Error Text:** Indicates that a log file will be created the first time that the package is executed. This file is appended with information every time the package runs. It stores execution information, including the package name, execution start and ending times, and any errors.
- **Source Error Rows:** Refers to a file that contains the source rows in which an error occurred. When an error occurs, the source row is written to the file if the row is available. The exception file is not created if no errors occur. The file has an extension of .Source and it is recreated if a package execution has an error. Delimiters and qualifiers can be set for both exception files.
- **Dest Error Rows:** Refers to a file that contains the destination rows when an error was encountered. When an error occurs, the destination row is written to the file if the row is available. The exception file is not created if no errors occur. The file has an extension of .Dest and it is recreated if a package execution has an error.

WARNING! The important point is that the rows are written into the exception files *if the row is available* when the error happens. Source rows are available most of the time, except during multiple-row processing. Destination rows will not be available for transformation errors. The error occurred during the modification, so an output row was never created.

There is a definite advantage in using the SQL Server 2000 format of separate files for error information and exceptions. You can examine and edit the exception files, then use them as a source for another transformation. This means that you can have a package that takes the destination exception rows and copies them into a SQL table that has no constraints. You can then examine the data from within SQL Server.

Error Management

Managing errors is a crucial part of DTS transformations. As the amount of data to load increases, the possibility of errors increases. If there are some errors, then it does not pay to reject the entire load. The objective is to load all the good rows, while saving the exceptions. This technique may lead to some manual processing of the data, but most of the load will be automated.

The Transform Data task has an option that sets the maximum number of errors that can happen before the task fails. This option allows valid data to be loaded while invalid data is rejected.

The data pump's default behavior for error handling is:

- No errors allowed for a given task.
- The first error causes termination of the task.
- The first error causes the task and step to fail.
- The package succeeds regardless of the step failure.

The objective in error management is to override one or more of these default behaviors. The Transform Data task has an option to allow a maximum number of errors. As DTS processes the task, it maintains an error count, and once the number of errors reaches the maximum error setting, DTS terminates the task and the step fails. If the task completes and the number of errors is less than the maximum error setting, then the step succeeds.

When you set the Max error count option, keep the following items in mind:

- **Setting:** The default value is zero, which means one error causes failure. It can be set to a value between zero and 9,999. If you need a larger setting, it can be done through the Disconnected Edit dialog box of the package, which enables you to modify the properties of all objects in a package. This is covered in another chapter of this course.

- **Termination:** Processing of the task terminates when DTS detects that the number of errors has reached the setting value.
- **Failure:** If the Max error count setting is reached, the step fails.
- **Package:** Even if the step fails, by default the package still succeeds. This default can be modified, and is covered in another chapter of this course.

SQL Server Optimization

If the destination is Microsoft SQL Server, you can use settings on the options page to optimize the data load. These options cover fast load and use of batches. Batches add another element to error handling.

Fast Load

The options tab has several settings for improving performance. Although each of these settings is independent of the others, they all fall under the heading of fast load. You must select the Use fast load check box before you can set any of the other options.

The OLE DB provider for SQL Server supports the IRowsetFastLoad interface, which is exclusive to SQL Server. Other OLE DB providers use the standard IRowsetChange interface, which causes the data pump to load each row through an INSERT SQL statement.

The Use fast load option enables the data pump to use the faster interface, which creates batches of data before inserting the data into the SQL Server tables. The data is inserted via the bulk load capability of the Bulk Insert task.

After you select the Use fast load check box, you can set any of the following options:

- **Keep NULL Values:** When you select this option, NULL values from the data source are inserted into a target column even if the column has a default constraint or a bound default object. This option is also available in the Bulk Insert task.
- **Check Constraints:** When you clear this option, SQL Server inserts the new data without checking the constraints on the target table. This option carries with it the potential for inserting massive amounts of invalid data, and should be used carefully.
- **Enable Identity Insert:** When you select this option, identity column values are taken from the data source. If you clear the option, SQL Server generates the values for the identity column.

- **Table Lock:** Normally, SQL Server uses row-level locks for inserting new data. Locks are an overhead resource that can be a drain on the system. Rather than have many locks for those new rows, you can place just one lock on the entire table during the insert.

Batch Size

If you set the Use fast load option, then you can configure batch size to collect the insert rows into groups before they get to SQL Server. Use the Insert batch size setting on the Options tab to configure batch size.

Earlier, you found out how the data pump works. The DTS data pump normally transforms each row on an individual basis and then writes the row into the destination buffer. It only commits these changes to the destination data source after all of the records in a batch are processed.

By default, the batch size is zero, which means that all of the rows must be transformed before any rows are written to the destination. If you are copying 100,000 rows of data, DTS will handle all of these rows as one batch, storing them in the buffer and then writing them after reading all of them. If the Max error count has not changed and 99,999 rows have been processed, and an error occurs in the last row, none of the records are written.

By adjusting the batch size, you can positively impact the data load performance. However, as in many cases, the batch size can be too low or too high and have a negative impact on performance. As always, you are balancing different system resources against each other, trying to find the optimal settings.

The batch size can be set to any value between zero and 9,999. If you want a larger batch size, use the Disconnected Edit dialog box of the package.

Batch Errors

Once you use fast load and configure batch size, a new wrinkle is added to the error handling strategy. Different types of errors affect batches in different ways. The error counting is also impacted as well as the outcome when the maximum error count is reached.

There are two general types of errors: transformation and insert. If a transformation error occurs, DTS will act as follows:

- The row fails.
- The row is not added to the batch.
- The error count increases by one.

Consider this situation: you are transforming ten rows, and your batch size is five. The fifth row has a transformation error. This row is not added to the batch, the error count increases by one, and the batch is not written. When the sixth row is processed successfully, then the batch is written to the target table.

When an insert error, such as a primary key violation, occurs, DTS will act as follows:

- The entire batch fails.
- The error count increases by one.

Consider the same situation as above: you are transforming ten rows, and your batch size is five. The fifth row has a uniqueness violation. In this case, all five rows fail to be inserted and the error count increases by one.

TIP: Some errors look like they would be destination insert errors, when in fact they are treated as transformation errors. The data pump reads in certain characteristics of the target, such as whether a column allows NULL values. Therefore, when the data pump processes a row it treats a NULL violation as a transformation error. It does not wait to try to insert the row.

The nature of fast load impacts the recording of errors. The error is always written to the error text file, so that you can debug and fix errors. But the exception rows are not available during a fast load. The .Source and .Dest files will not have entries that correspond to the error in the text file. This is especially true of insert errors. If you want the rows written to the exception files, turn off fast load.

Configuring Options

To demonstrate what happens with data loads, the test_data load package will be modified so that you can see how error handling, batches, and exception files work.

Try It Out!

In these steps, you will modify the DTS package created earlier. These modifications will allow you to see how error handling works.

1. Return to the DTS Package Designer, where the Transform Data Task Properties dialog box is still open, and select the Options tab.

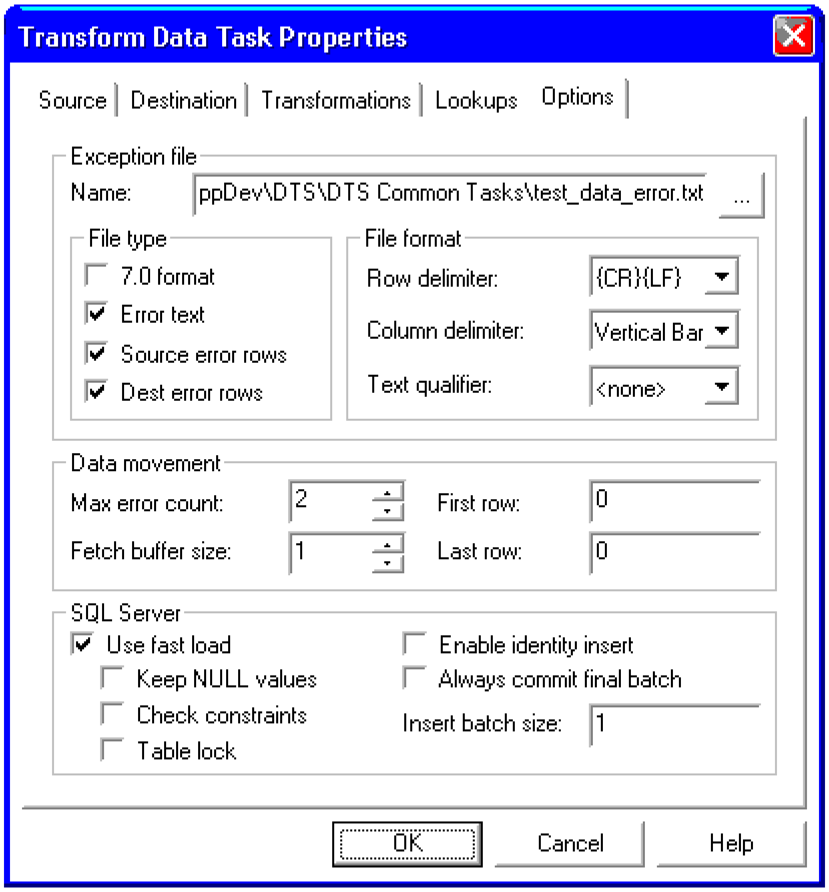
2. In the Name text box for an exception file, type **test_data_error.txt** and the path to the folder for this chapter.

3. Under File type, clear the 7.0 format check box and set the **Error text**, **Source error rows**, and **Dest error rows** options.

4. The maximum number of allowable errors will be two and the batch size will be set to one. This means that data will be committed after each row is processed. The step fails if two errors occur. Under Data movement, set the **Max error count** to **2**.

5. Under SQL Server, select the **Use fast load** option and clear the **Check constraints** option.

6. Type **1** in the Insert batch size text box. Your window should look like Figure 16. Click the **OK** button.


Transform Data Task Properties

Source | Destination | Transformations | Lookups | Options

Exception file
Name: ppDev\DTS\DTS Common Tasks\test_data_error.txt

File type
- [ ] 7.0 format
- [x] Error text
- [x] Source error rows
- [x] Dest error rows

File format
Row delimiter: {CR}{LF}
Column delimiter: Vertical Bar
Text qualifier: <none>

Data movement
Max error count: 2
Fetch buffer size: 1
First row: 0
Last row: 0

SQL Server
- [x] Use fast load
  - [ ] Keep NULL values
  - [ ] Check constraints
  - [ ] Table lock
- [ ] Enable identity insert
- [ ] Always commit final batch

Insert batch size: 1

Figure 16. The Options tab of the Transform Data Task Properties dialog box.

7. Save the package, naming it **transform_data** and use the **SQL Server** location.

8. Click the **Execute** button on the Designer toolbar. Click **OK** in the Results message box and then click **Done**.

9. Verify the results by using Query Analyzer. Execute the following statement:

```
SELECT * FROM whistler.dbo.test_data
```

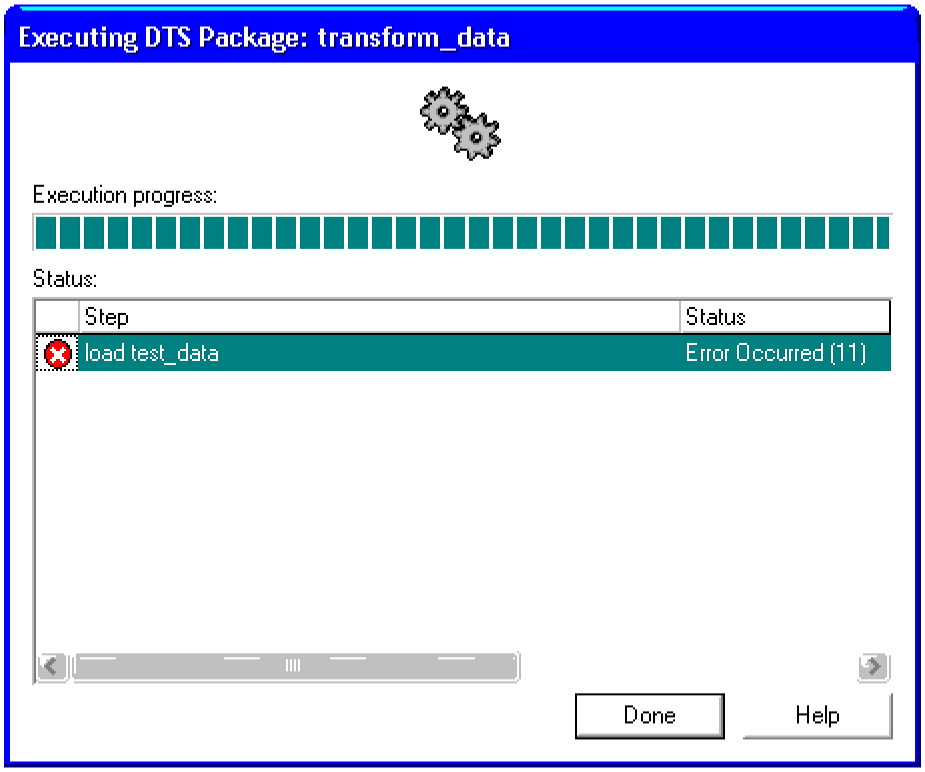
The test_data table should contain 11 rows. Note that the titles are all lowercase, the countries are all uppercase, and the regions are two letters and uppercase, except where there are NULL values. Leave Query Analyzer open.

10. Open Windows Explorer and locate the **test_data_error.txt** file in the folder for this chapter. Double-click the file to open and review it. Note that although no errors occurred, the file has package and step information, including the names of the exception files. Note that those files do not exist.

11. Back in the Query Analyzer, load the script file called **create_an_error.sql** from the folder for this chapter. Review the script and then execute it. This script empties test_data and creates a unique index on the PostalCode column.

12. Return to the transform_data package in the DTS Package Designer. Click the **Execute** button on the Designer toolbar. Note that one task failed. Click **OK** in the Results message box.

13. The Executing DTS Package dialog box shows that a step failed and that the Status is Error Occurred, as shown in Figure 17. Note that it still processed 11 rows.



Step	Status
load test_data	Error Occurred (11)

Figure 17. The Executing DTS Package dialog box showing an error in the load test_data step.

14. Double-click the **load test_data** step. The error message indicates an error at row 11, and that the number of errors is two, which is the maximum error count. The last actual error message is also shown. Click **OK** and then click **Done**.

15. In Windows Explorer, double-click the **test_data_error.txt** file to view its contents. Note that rows 10 and 11 failed with the same error message, "Cannot insert duplicate key row…" The error file will have all of the error messages, not just the one that caused the maximum error count. Note that there are no exception files, because you are using fast load.

16. In the Query Analyzer, type the following statement:

```
SELECT * FROM whistler.dbo.test_data
```

The test_data table should contain nine rows because the batch size is one, so the insert errors only failed the rows that had duplicate keys. Another reason is that the package did not fail, and it is not using transactions.

17. Return to the DTS Package Designer. Double-click the **Transform Data Task** arrow, click the **Options** tab, and change the Insert batch size entry to **0**. Click the **OK** button.

18. In the Query Analyzer, type and execute the following statement:

```
DELETE FROM whistler.dbo.test_data
```

19. In the **transform_data** package, click the **Execute** button on the Designer toolbar. The same error will occur. Click **OK** in the Results message box, double-click the **load test_data** step, and read the error message. Note that the maximum error count was not indicated in the message. Click **OK** and then click **Done**.

20. Close and save the package.

21. In the Query Analyzer, type and execute the following statement:

```
SELECT * FROM whistler.dbo.test_data
```

    The table is empty. Since the batch size is zero, all of the rows processed as one batch; one insert failure fails the entire batch.

    If you examine the error file, you'll see only the insert error at row ten. When DTS received the insert error, it did not attempt to insert row 11 because it is part of the failed batch as well.

Summary

- The DTS Bulk Insert task is the fastest way to load data into a SQL Server destination table or view. The source data must be a text file in order to use this task.
- Drawbacks of the Bulk Insert task are that it cannot do any transformations or error handing.
- By default, the Bulk Insert task expects that all of the source text file columns match the destination table. If there are any differences in the number of columns or the order of the columns, you must create a format file to handle the differences. A format file is a tab-delimited text file with a standard layout.
- You can use the DTS Execute SQL task to run SQL statements on any data source that understands the syntax. This task can provide excellent flexibility through the use of parameterized queries. Parameters are created in the query by using question marks, and are identified by the position or order in the query.
- Global variables are objects in a DTS package that can store values to be passed from step to step. Global variables are well suited for supplying values to input parameters in the Execute SQL task, and for storing returned values from those queries as well.
- The Transform Data task is one of the most basic of tasks in DTS packages. This task can copy data from any OLE DB or ODBC source and pass it to any OLE DB or ODBC destination. It can also perform a number of transformations as the data is copied.
- The Transform Data task uses the DTS data pump, an object that is designed to read, modify, and write data. The data pump options enable the Transform Data task to perform built-in transformations as well as scripted and custom modifications.
- The Transform Data task can perform error handling and recording, and supports the creation of exception files for recording the rows in error. The task can allow a certain number of errors before failing.
- When the destination of the Transform Data task is a SQL Server table, there are extra options for optimizing performance. These include a fast load feature as well as the ability to group new rows into batches.

Questions

1. True/False: The Bulk Insert task can be used to insert data into a text file format.

2. How do you create format files?

3. True/False: The Execute SQL task can be used with Microsoft Access.

4. List the steps for creating a parameterized query in the Execute SQL task.

5. You can define a new table from within the properties of the Transform Data task. When is this table created?

6. What is the most efficient method of column mapping? Why?

Answers

1. True/False: The Bulk Insert task can be used to insert data into a text file format.
   False. The Bulk Insert task reads from a text file and inserts data into SQL Server destinations.

2. How do you create format files?
   By using the bulk copy program (bcp) or the Generate option from the Bulk Insert Package Properties dialog box. You can also create or edit them manually using Notepad.

3. True/False: The Execute SQL task can be used with Microsoft Access.
   True. It can be used with any data source that can handle SQL statements.

4. List the steps for creating a parameterized query in the Execute SQL task.
   1) Create a SQL statement with parameters, using question marks for the parameters. 2) Create as many global variables as you have parameters. 3) Assign values to the global variables. 4) Map the parameters to the global variables, based on the order of the parameters in the SQL statement.

5. You can define a new table from within the properties of the Transform Data **task**. When is this table created?
   In the task creation, at the moment you click on the OK button on the Create Destination Table dialog box.

6. What is the most efficient method of column mapping? Why?
   Many-to-many column mappings. All of the columns in a many-to-many are handled in one data pump operation, thus minimizing the number of data pump operations.

Lab 4: DTS Common Tasks

TIP: The results of this lab are packages that can be found in this lab's completed folder.

Lab 4 Overview

In this lab you'll learn how to create DTS packages that implement the most commonly used tasks as part of the process of building a data warehouse or data mart. These tasks are the Bulk Insert Task, the Execute SQL Task, and the Transform Data Task. The Bulk Insert Task will be used to load data into staging tables for use in the data marts. The Execute SQL Task will be used to create a time dimension table through SQL statements. The Transform Data Task will be used to populate the warehouse dimension table from an Excel worksheet.

To complete this lab, you'll need to work through three exercises:

- Load Staging Tables
- Create Time Dimension
- Load Warehouse Dimension

Each exercise includes an "Objective" section that describes the purpose of the exercise. You are encouraged to try to complete the exercise from the information given in the Objective section. If you require more information to complete the exercise, the Objective section is followed by detailed step-by-step instructions.

Load Staging Tables

Objective

In this exercise, you'll use the DTS Package Designer to create two packages for loading data into staging tables for the Whistler data mart. The staging tables are located in the whistler SQL Server database. The first load is of sales data from a legacy order entry system. This data is relatively clean and stored in a tab-delimited file. The second file is a comma-delimited file of the most recent inventory at Northwind Traders. It also comes from a legacy source system. The sales data will be loaded into the sales_staging file in the SQL Server whistler database and the inventory data will be loaded into the inventory_staging table in the same database. The inventory data file needs a format file that can be generated using the Bulk Insert Task. The connections for SQL Server will use a Microsoft Data Link (UDL) file.

Things to Consider

- Both of the staging tables need to be empty before loading the data from the text files. Neither table has any constraints on it.
- The exercise assumes that you are using Windows authentication, and that you have administrative privileges on your SQL Server. If you need to use SQL Server authentication, then enter the appropriate user name and password.

Setup Instructions

This lab uses the whistler database that may already contain some data from previous labs. Prior to starting this lab you will need to restore the whistler database from a backup that has the image of the database needed for this lab. Before starting the restore, make sure that there are no connections using the whistler database. To do the restore (if the database is not already connected to your SQL Server installation, follow the instructions in the Introduction for attaching it):

1. In the Enterprise Manager, expand the **Databases** folder, right-click the **whistler** database, and select **All Tasks|Restore Database** from the context menu.

2. In the Restore database dialog box, on the General tab, on the Restore line, select the **From device** option and click the **Select Devices** button.

3. In the Choose Restore Devices dialog box click the **Add** button.

4. In the Choose Restore Destination dialog box, click the ellipsis button next to the File name text box.

5. In the Backup Device Location dialog box, find the folder for this lab, click the **whistler.bak** file, and click the **OK** button.

6. Click the **OK** button to close the Choose Restore Destination dialog box, click the **OK** button to close the Choose Restore Devices dialog box, and click the **OK** button in the Restore database dialog box to start the restore.

7. Once the restore has finished, click the **OK** button in the message box indicating the success of the restore.

Step-by-Step Instructions

1. Open the **sales.txt** file found in this lab's folder.

2. Review the column and row delimiters, and then close the file.

3. In the Enterprise Manager, right-click the Data Transformation Services folder, and choose **New Package** from the context menu.

4. Click the **Microsoft Data Link** button on the Connection toolbar.

5. Name the connection **whistler** in the text box next to the New Connection option. Then set the file name to **whistler.udl**, from this lab's folder.

6. Set the **Always read properties from UDL file** check box and then click the **OK** button.

7. Click the **Bulk Insert Task** button on the Task toolbar.

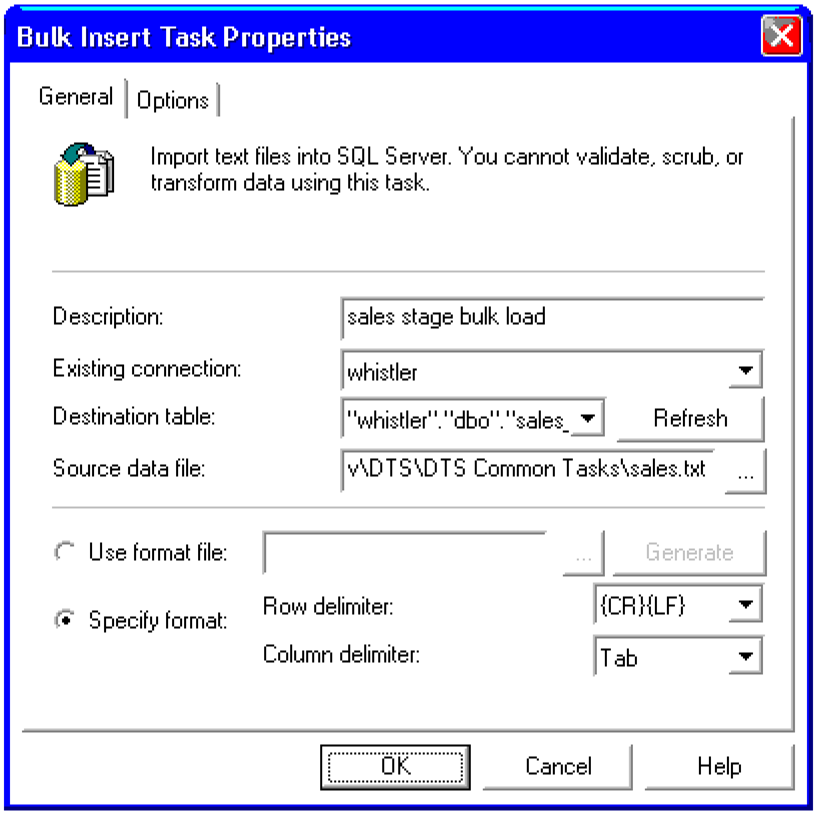
8. Type **sales stage bulk load** in the Description text box.

9. Choose **whistler** from the Existing connection drop-down list.

10. Choose the **sales_stage** table in the Destination table drop-down list.

11. Select the **sales.txt** file from this lab's folder as the Source data file.

12. Click the **Specify format** option, select **{CR}{LF}** from the Row delimiter drop-down list, and select **Tab** from the Column delimiter drop-down list. Your screen should look like Figure 18.


Bulk Insert Task Properties

General | Options

Import text files into SQL Server. You cannot validate, scrub, or transform data using this task.

Description: sales stage bulk load

Existing connection: whistler

Destination table: "whistler"."dbo"."sales_ | Refresh

Source data file: v\DTS\DTS Common Tasks\sales.txt | ...

( ) Use format file: | ... | Generate

(●) Specify format: Row delimiter: {CR}{LF} Column delimiter: Tab

Figure 18. The Bulk Insert Task Properties dialog box.

13. When you review the sales.txt file, notice that the first row has column headings. You want to skip that row. Click the Options tab and select the **Only copy selected rows** option.

14. Set the First row number to **2**. Leave the Last row number at **0**. Click the **OK** button.

Add a Task to Empty the Staging Table

15. Click the **Microsoft Data Link** button on the Connection toolbar.

16. Name the connection **Truncate** in the text box next to the New Connection option. Then set the file name to **whistler.udl**, from this lab's folder.

17. Set the **Always read properties from UDL file** check box and then click **OK**.

18. Click the **Execute SQL Task** button on the Task toolbar.

19. Type **truncate stage** in the Description text box, and choose **Truncate** from the Existing connection drop-down list.

20. Type **TRUNCATE TABLE sales_stage** in the SQL statement box, and then click **OK**.

21. Click the **truncate stage** task icon, press the **CTRL** key, and click the **sales stage bulk load** task icon.

22. From the Designer menu, choose **Workflow|On Success**. The design sheet of your package should look like Figure 19, depending on where you placed the icons and arrow.

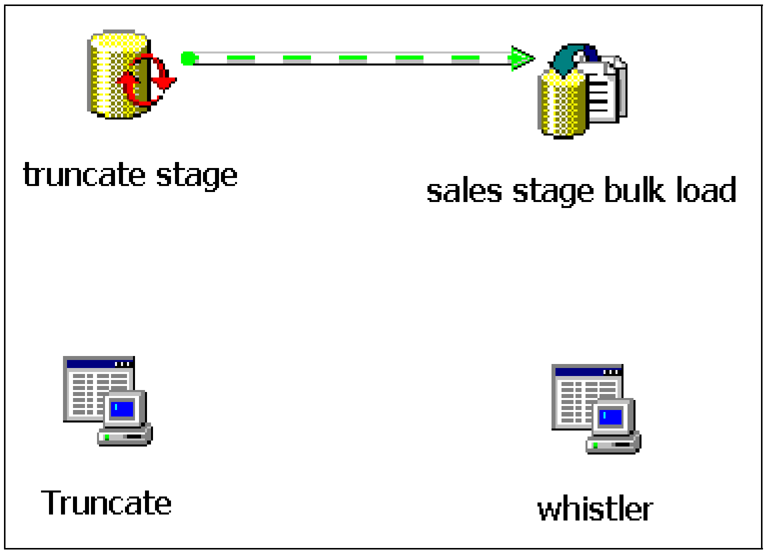


Figure 19. The completed sales_stage DTS package.

23. Click the **Save** button on the Designer toolbar. Name the package **sales_stage**, set the Location drop-down to **SQL Server**, and click the **OK** button.

24. Click the **Execute** button on the Designer toolbar. Click the **OK** button in the Results message box.

25. If any errors show up in the Executing Package dialog box, be sure to double-click on them to find the error message. When finished, click the **Done** button and close the package.

26. Verify the results by using Query Analyzer. Type in and execute the following statement:

```
SELECT * FROM whistler.dbo.sales_stage
```

There should be 2155 rows in the table.

The Inventory Staging Transfer

27. Open the **inventory.txt** file found in this lab's folder.

28. Review the column and row delimiters and the order of the source columns. Compare this order with the order of the columns in the destination table and then close the file.

29. In the Enterprise Manager, right-click the Data Transformation Services folder, and choose **New Package** from the context menu.

30. Click the **Microsoft Data Link** button on the Connection toolbar.

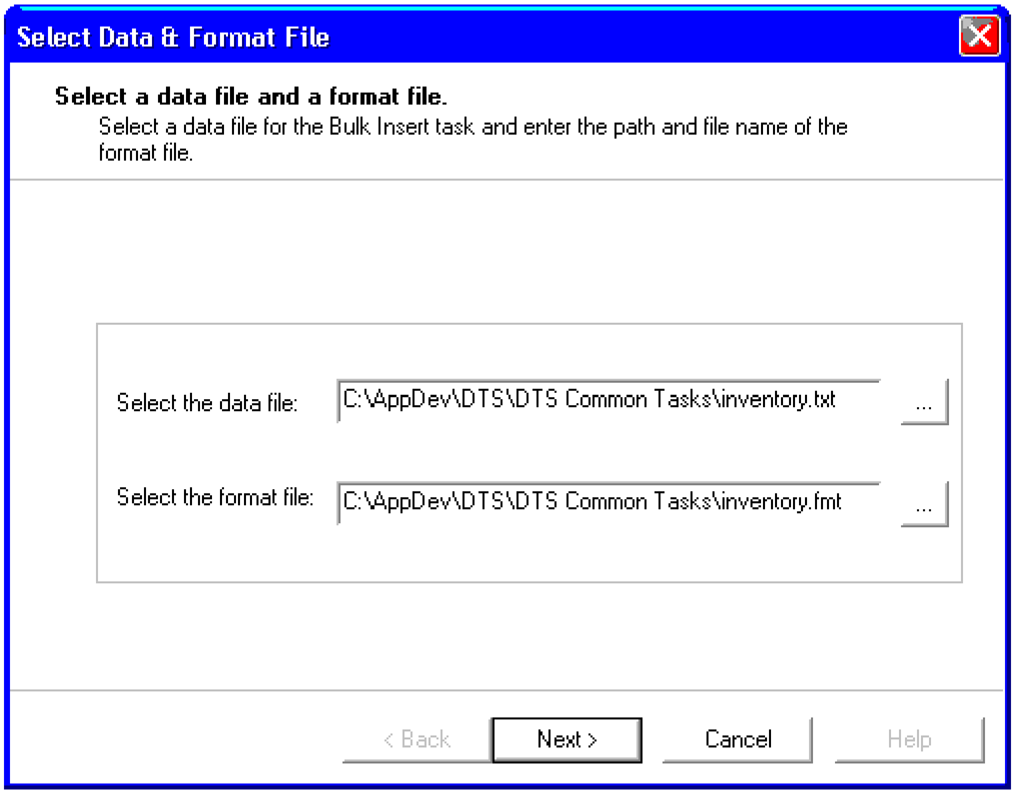
31. Name the connection **whistler** in the text box next to the New Connection option. Then set the file name to **whistler.udl**, from this lab's folder. Set the **Always read properties from UDL file** check box and then click **OK**.

32. Click the **Bulk Insert Task** button on the Task toolbar.

33. Type **inventory stage bulk load** in the Description text box. Choose **whistler** from the Existing connection drop-down list. Choose the **inventory_stage** table in the Destination table drop-down list. Select the **inventory.txt** file from this lab's folder as the Source data file.

34. Click the **Use format file** option and click the **Generate** button.

35. This opens the Select Data & Format File dialog box. For **Select the data file**, make sure that the **inventory.txt** file from the folder for this is indicated. In the **Select the format file** text box, type in **inventory.fmt** with the same path as the data file. Your screen will look something like Figure 20. Click the **Next** button.



Select a data file and a format file.
Select a data file for the Bulk Insert task and enter the path and file name of the format file.

Select the data file: C:\AppDev\DTS\DTS Common Tasks\inventory.txt

Select the format file: C:\AppDev\DTS\DTS Common Tasks\inventory.fmt

Figure 20. The Select Data & Format File dialog box.

36. In the Select File Format dialog box of the Text File Properties dialogs, most of the default settings are fine. Set the **First row has column names** checkbox and click the **Next** button.

37. In the Specify Column Delimiter dialog box, make sure that **Comma** is selected and click the **Finish** button. You have just created a format file for the bulk insert, but the column order is incorrect, so the format file has to be edited.

38. Start **Windows Explorer**, and navigate to this lab's folder. Open the **inventory.fmt** file with Notepad.

39. Change the server column order for **warehouse_code** from 2 to **3**. That's the number just in front of the column name.

40. Change the server column order for **quantity_on_hand** from 3 to **2**. The text in the file should look like the following:

```
7.0
4
1 SQLCHAR 0 4 "," 1 product_id
2 SQLCHAR 0 15 "," 3 warehouse_code
3 SQLCHAR 0 2 "," 2 quantity_on_hand
4 SQLCHAR 0 16 "\r\n" 4 inventory_date
```

41. Save the file and close Notepad.

42. Back in the Bulk Insert Task Properties dialog box click the Options tab.

43. Select the **Only copy selected rows** option, set the First row number to **2**, and leave the Last row number at **0**. Click the **OK** button.

44. Click the **Microsoft Data Link** button on the Connection toolbar.

45. Name the connection **Truncate** in the textbox next to the New Connection option, set the file name to **whistler.udl**, from this lab's folder, and set the **Always read properties from UDL file** check box. Click the **OK** button.

46. Click the **Execute SQL Task** button on the Task toolbar.

47. Type **truncate stage** in the Description text box, choose **Truncate** from the Existing connection drop-down list, and type **TRUNCATE TABLE inventory_stage** in the SQL statement box. Click the **OK** button.

48. Click the **truncate stage** task icon, press the **CTRL** key, click the **inventory stage bulk load** task icon, and choose **Workflow|On Success** from the menu. The design sheet of your package should look like Figure 21, depending on where you placed the icons and arrow.

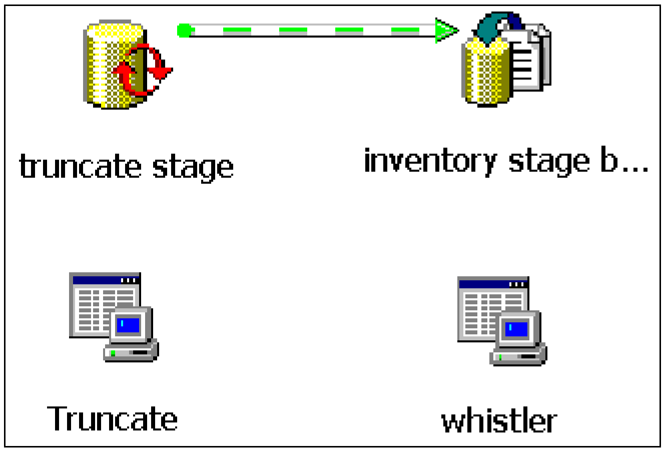


Figure 21. The completed inventory_stage DTS package.

49. Click the **Save** button on the Designer toolbar. Name the package **inventory_stage**, set the location drop-down list to **SQL Server**, and click the **OK** button.

50. Click the **Execute** button on the Designer toolbar. Click the **OK** button in the Results message box.

51. If any errors show up in the Executing Package dialog box, be sure to double-click on them to find the error message. When finished, click the **Done** button and close the package.

52. Verify the results by using Query Analyzer. Type in and execute the following statement:

```
SELECT * FROM whistler.dbo.inventory_stage
```

There should be 231 rows in the table.

Create Time Dimension

Objective

In this exercise, you'll create a time dimension in the Whistler data mart. The time_dim table in the whistler SQL Server database has a breakdown of dates into separate columns for year, month, day, and so on. The lowest level of detail in the time_dim dimension table is one day. You will use a stored procedure named time_dim_build to load the table. This procedure takes two input parameters: start date and end date. When the stored procedure runs, it fills the time_dim table with dates between the start date and end date.

Things to Consider

- The Execute SQL Task is best suited for this activity, using a parameterized query. The DTS package that is created will be called time_dim and saved to SQL Server.
- The input dates will be 1 Jan 1995 and 1 Jan 2005, so that the dimension is well prepared for historical as well as future dates.
- Time_dim has an identity column that will server as the identifier for the rows in the table.
- The exercise assumes that you are using Windows authentication, and that you have administrative privileges on your SQL Server. If you need to use SQL Server authentication, then enter the appropriate user name and password.

Step-by-Step Instructions

1. In the Enterprise Manager, expand the Databases folder, expand the whistler folder, and click **Stored Procedures**.

2. In the right pane, double-click the **time_dim_build** stored procedure, and review the code for the procedure. You will see that there are @start_date and @end_date parameters. Click the **OK** button.

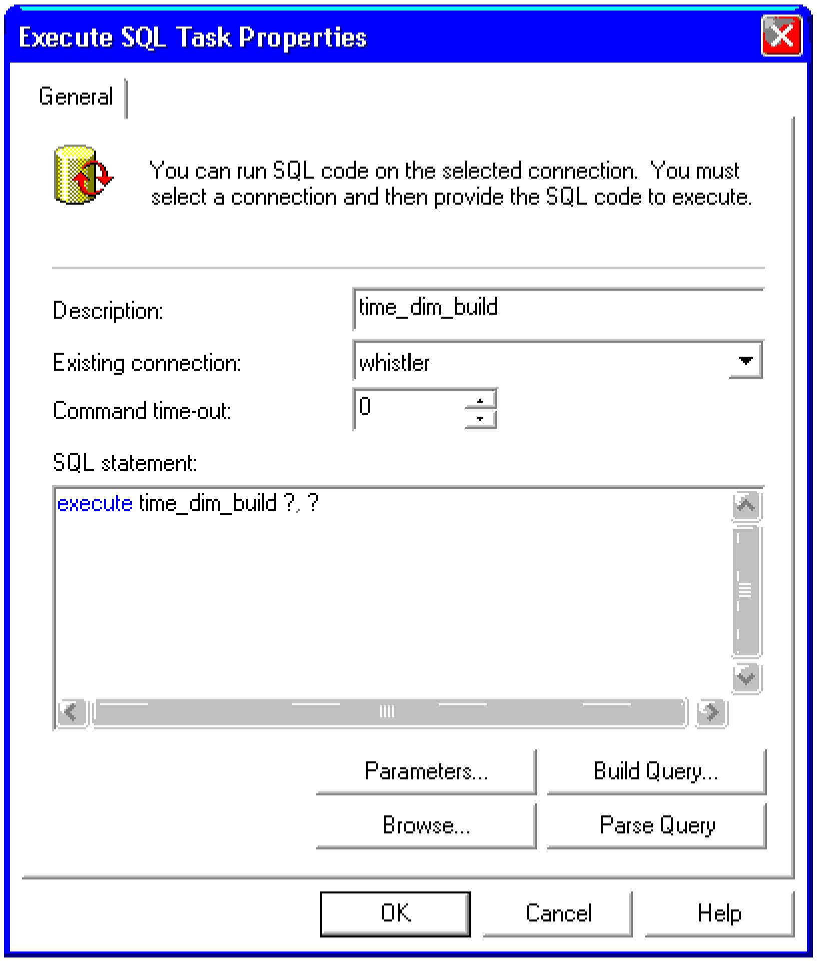
3. Right-click the Data Transformation Services folder, and choose **New Package** from the context menu.

4. Create a Microsoft Data Link connection named **whistler** that uses the **whistler.udl** file from this lab's folder. Don't forget to select the **Always read properties from UDL file** check box.

5. Click the **Execute SQL Task** button on the Task toolbar.

6. Type **time_dim_build** in the Description text box, and choose **whistler** from the Existing connection drop-down list.

7. Type **execute time_dim_build ?, ?** in the SQL statement box. The Execute SQL Task Properties dialog box should look like Figure 22.


Execute SQL Task Properties

General

You can run SQL code on the selected connection. You must select a connection and then provide the SQL code to execute.

Description: time_dim_build
Existing connection: whistler
Command time-out: 0
SQL statement:

```
execute time_dim_build ?, ?
```

Figure 22. The Execute SQL Task Properties dialog box for the time_dim_build task.

8. Click the **Parse Query** button to make sure of the syntax and click the **OK** button in the Designer message box.

9. Click the **Parameters** button.

10. This opens the Parameter Mapping dialog box, but you do not have anything to map yet. Make sure that the Input Parameters tab is selected and click the **Create Global Variables** button.

11. In the Global Variables dialog box, create two new variables so that they can be mapped to the parameters of the query.

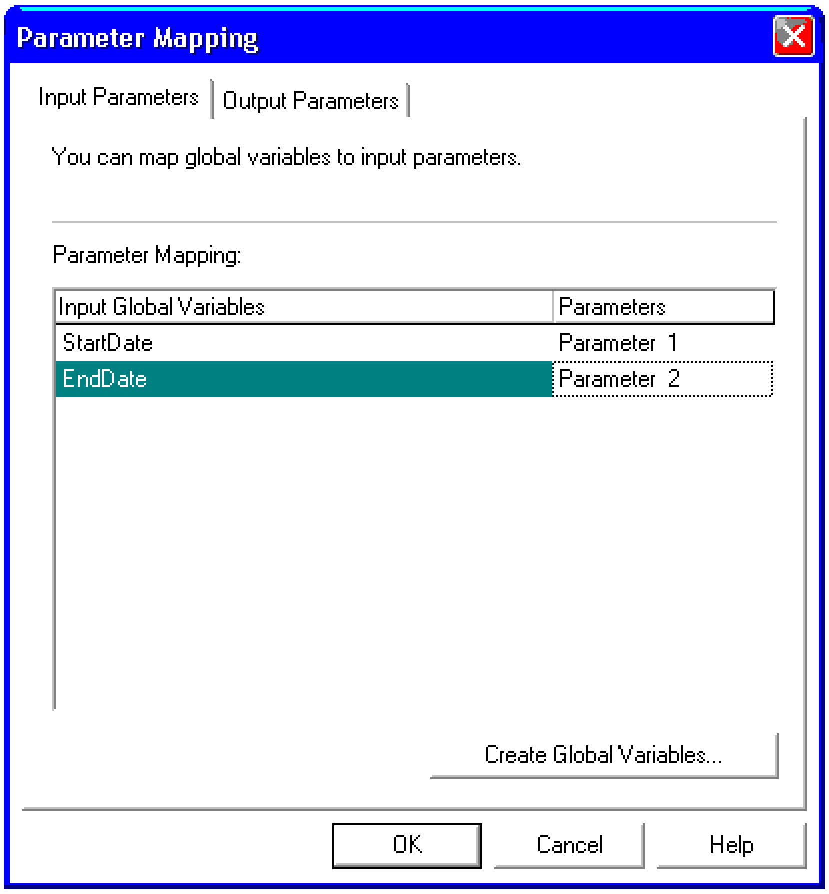
12. Type **StartDate** in the first row of the Name column. Choose **Date** from the drop-down list in the Type column. Type **01-JAN-1995** in the Value column.

13. Type **EndDate** in the second row of the Name column. Choose **Date** from the drop-down list in the Type column. Type **01-JAN-2005** in the Value column. Click the **OK** button.

14. Next to **Parameter 1**, select **StartDate** from the drop-down list in the Input Global Variables column.

15. Next to **Parameter 2**, select **EndDate** from the drop-down list in the Input Global Variables column.

16. Verify the mappings. Your screen should look like Figure 23. Click the **OK** button.



You can map global variables to input parameters.

Parameter Mapping:

Input Global Variables	Parameters
StartDate	Parameter 1
EndDate	Parameter 2

Figure 23. The Parameter Mapping dialog box.

17. Click the **OK** button to exit the Execute SQL Task Properties dialog box.

18. Click the **Save** button on the Designer toolbar. Name the package **time_dim**, set the location drop-down list to **SQL Server**, and click the **OK** button.

19. Click the **Execute** button on the Designer toolbar. Click the **OK** button in the Results message box.

20. If any errors show up in the Executing Package dialog box, be sure to double-click on them to find the error message. When finished, click the **Done** button and close the package.

21. Verify the results by using Query Analyzer. Type in and execute the following statement:

```
SELECT * FROM whistler.dbo.time_dim
```

There should be 3653 rows in the table.

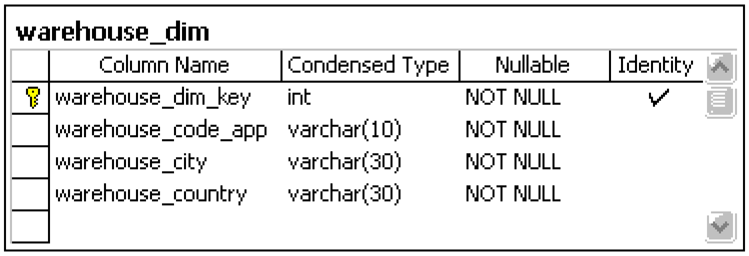
Load Warehouse Dimension

Objective

In this exercise, you'll load the warehouse dimension in the Whistler data mart with data from a Microsoft Excel file. The data in the Excel spreadsheet is not quite what you want to see in the dimension table. The code in the spreadsheet has too much information, the city is not the right case, and the country is not the right case and has too much white space. Specifically, the code that the data mart is using is the 4th, 5th, and 6th characters from the code in the spreadsheet. The warehouse dimension is a table called warehouse_dim in the SQL Server whistler database. The Excel file is warehouse.xls, found in this lab's folder. Error and exception logging are to be turned on for this data transfer.

Things to Consider

- The Transfer SQL Task is best suited for this data load. Create a DTS package named warehouse_dim, saved to SQL Server.
- The spreadsheet has columns for Code, City, and Country.
- The destination table is shown in Figure 24. Notice that the warehouse_dim_key column is an identity column.



warehouse_dim

Column Name	Condensed Type	Nullable	Identity
warehouse_dim_key	int	NOT NULL	✓
warehouse_code_app	varchar(10)	NOT NULL	
warehouse_city	varchar(30)	NOT NULL	
warehouse_country	varchar(30)	NOT NULL	

Figure 24. The warehouse_dim table from the whister database is the destination table.

- Create a task to delete data from the destination table before loading the data from the Excel file.
- The exercise assumes that you are using Windows authentication, and that you have administrative privileges on your SQL Server. If you need to use SQL Server authentication, then enter the appropriate user name and password.

Step-by-Step Instructions

1. In the Enterprise Manager, right-click the Data Transformation Services folder, and choose **New Package** from the context menu.

2. Click the **Microsoft Excel 97-2000** button on the Connection toolbar.

3. Name the connection **Warehouse Source** in the text box next to the New Connection option. Then set the file name to **warehouse.xls**, from this lab’s folder. Click the **OK** button.

4. Create a Microsoft Data Link connection named **Warehouse Target** that uses the **whistler.udl** file from this lab’s folder. Don’t forget to select the **Always read properties from UDL file** check box.

5. Click the **Transform Data Task** button on the Task toolbar.

6. Click the **Warehouse Source** connection icon, and then click the **Warehouse Target** connection icon.

7. Double-click the **Transform Data Task** arrow that joins the icons.

8. This should open the Transform Data Task Properties dialog box to the Source tab. Type **load warehouse_dim** in the Description text box, and choose **warehouse$** from the Table/View drop-down list.

9. Click the Destination tab and select the **warehouse_dim** table from the Table name drop-down list.

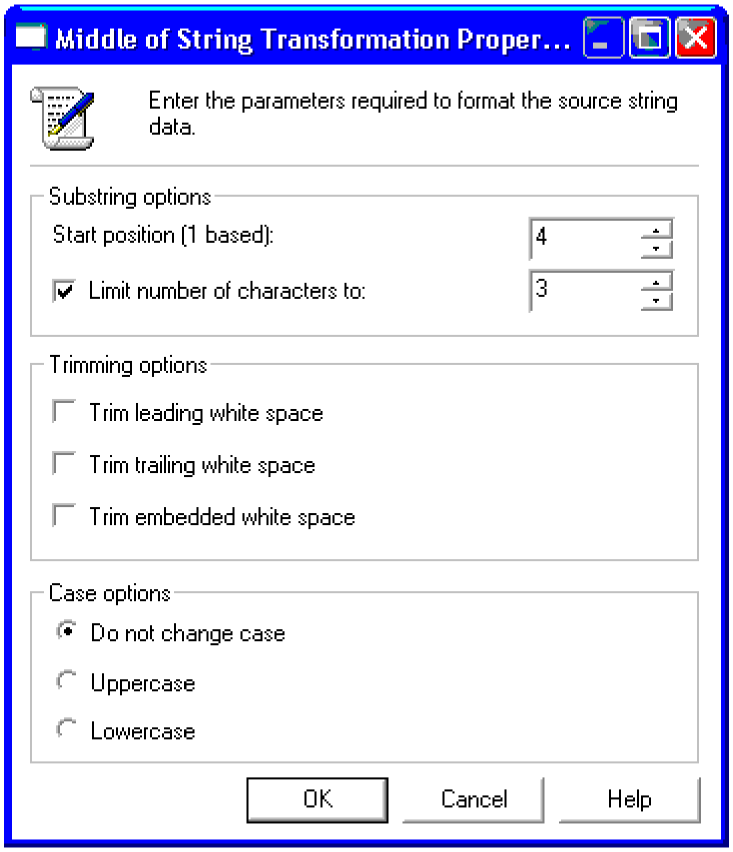
10. Click the Transformations tab and click the **Delete All** button to remove all of the default transformations.

11. Highlight the **Code** column from the Source list, highlight the **warehouse_code_app** column from the Destination list, and click the **New** button.

12. In the Create New Transformation dialog box, choose **Middle of String** from the list and click the **OK** button.

13. On the General tab of the Transformation Options dialog box, type **middle of string transformation** in the Name text box and click the **Properties** button.

14. In the Middle of String Transformation Properties dialog box, set the **Start position (1 based)** to **4,** set the **Limit number of characters to** check box, and set the value to **3**. Your screen should look like Figure 25. Click the **OK** button and then click **OK** again.



Enter the parameters required to format the source string data.

Substring options

Start position (1 based): 4

[✓] Limit number of characters to: 3

Trimming options

[ ] Trim leading white space

[ ] Trim trailing white space

[ ] Trim embedded white space

Case options

[● Do not change case]

○ Uppercase

○ Lowercase

Figure 25. Middle of String Transformation Properties dialog box.

15. Back on the Transformations tab, highlight the **City** column from the Source list, highlight the **warehouse_city** column from the Destination list, and click the **New** button.

16. In the Create New Transformation dialog box, choose **Uppercase String** from the list and click the **OK** button.

17. On the General tab of the Transformation Options dialog box, type **uppercase transformation** in the Name text box and click the **OK** button.

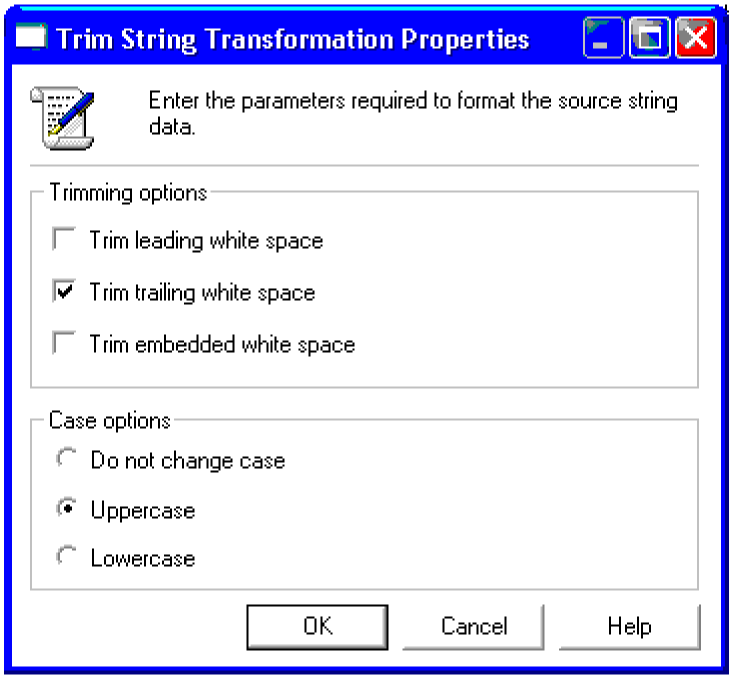
18. On the Transformations tab, highlight the **Country** column from the Source list, highlight the **warehouse_country** column from the Destination list, and click the **New** button.

19. In the Create New Transformation dialog box, choose **Trim String** from the list and click the **OK** button.

20. On the General page of the Transformation Options dialog box type **trim string transformation** in the Name text box and click the **Properties** button.

21. In the Trim String Transformation Properties dialog box, under Trimming options, set the **Trim trailing white space** option.

22. Under Case options, set the **Uppercase** option. The dialog box should look like Figure 26. Click the **OK** button and then click **OK** again.



Trim String Transformation Properties

Enter the parameters required to format the source string data.

Trimming options
- [ ] Trim leading white space
- [x] Trim trailing white space
- [ ] Trim embedded white space

Case options
- ○ Do not change case
- ● Uppercase
- ○ Lowercase

Figure 26. The Trim String Transformation Properties dialog box.

23. Back in the Transform Data Task Properties dialog box, click the Options tab and type **warehouse_error.txt** with the path to the folder for this module in the Name text box.

24. Under File type, clear the **7.0 format** check box, and set the **Error text**, **Source error rows**, **Dest error rows** options, and click the **OK** button.

25. To create the delete rows task, first create the connection. From the Connection toolbar, select the **Microsoft Data Link** button.

26. Type the name **Delete** in the New Connection text box. Set the file name to the **whistler.udl** file in the Link Files folder. Set the **Always read properties from UDL file** check box and then click **OK**.

27. In the menu, choose the **Task|Execute SQL Task** option. This brings up the Execute SQL Task Properties dialog. Type **Delete** in the Description text box, select **Delete** from the Connection drop-down list, and in the SQL Statement box, type the following code:

```
DELETE FROM warehouse_dim
```

28. Click the **OK** button.

29. Click the **Delete** task icon, press the **CTRL** key, click the **Warehouse Source** connection icon, and choose **Workflow|On Success** from the menu. The design sheet of your package should look like Figure 27, depending on where you placed the icons and arrows.

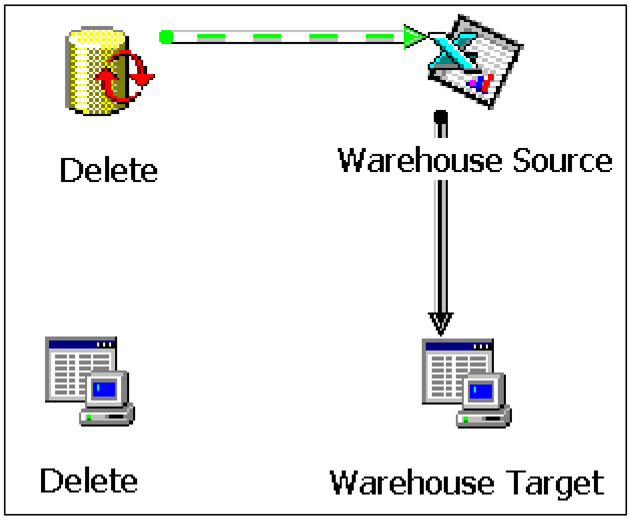


Figure 27. The completed warehouse_dim DTS package.

30. Click the **Save** button on the Designer toolbar. Name the package **warehouse_dim**, set the location drop-down list to **SQL Server**, and click the **OK** button.

31. Click the **Execute** button on the Designer toolbar. Click the **OK** button in the Results message box.

32. If any errors show up in the Executing Package dialog box, be sure to double-click on them to find the error message. When finished, click the **Done** button and close the package.

33. Verify the results by using Query Analyzer. Type in and execute the following statement:

```
SELECT * FROM whistler.dbo.warehouse_dim
```

There should be five rows in the table. All the character data should be uppercase, and the code should be two uppercase characters followed by the digit one.