DTS Overview

Microsoft® SQL Server[™] 2000 Data Transformation Services (DTS) is a set of graphical tools and programmable objects that lets you extract, transform, and consolidate data from disparate sources into single or multiple destinations.

The following sections provide essential information on DTS concepts, tools, and features.

Торіс	Description
DTS Basics	Describes the capabilities of DTS and summarizes the business problems it addresses.
<u>DTS Tools</u>	Describes the graphical tools supplied with SQL Server 2000 that are used to build DTS packages.
<u>DTS Package Elements</u>	Describes the main components of a package, such as connections, DTS tasks, and DTS transformations.
Managing a DTS Package	Explains the basics of package management, including creating, editing, saving, deleting, and executing packages.
<u>Adding Functionality to a DTS</u> <u>Package</u>	Explains how to build complex packages by using Microsoft ActiveX® scripts to customize a package. Describes how to join package steps to a transaction and expose package data to external sources.
<u>Sharing Meta Data</u>	Explains how to use SQL Server 2000 Meta Data Services with DTS to track package data in a data mart or data warehouse.
<u>Usage Considerations in DTS</u>	Describes specific data conversion and data transformation issues that can arise when using DTS.

For more information about programming with the DTS object model, see

Programming DTS Applications.

DTS Basics

Many organizations need to centralize data to improve corporate decisionmaking. However, their data may be stored in a variety of formats and in different locations. Data Transformation Services (DTS) addresses this vital business need by providing a set of tools that lets you extract, transform, and consolidate data from disparate sources into single or multiple destinations supported by DTS connectivity. By using DTS tools to graphically build DTS packages or by programming a package with the DTS object model, you can create custom data movement solutions tailored to the specialized business needs of your organization.

DTS Packages

A DTS package is an organized collection of connections, DTS tasks, DTS transformations, and workflow constraints assembled either with a DTS tool or programmatically and saved to Microsoft® SQL Server[™], SQL Server 2000 Meta Data Services, a structured storage file, or a Microsoft Visual Basic® file.

Each package contains one or more steps that are executed sequentially or in parallel when the package is run. When executed, the package connects to the correct data sources, copies data and database objects, transforms data, and notifies other users or processes of events. Packages can be edited, password protected, scheduled for execution, and retrieved by version.

For more information, see <u>Creating a DTS Package</u>.

DTS Tasks

A DTS task is a discrete set of functionality, executed as a single step in a package. Each task defines a work item to be performed as part of the data movement and data transformation process, or as a job to be executed.

DTS supplies a number of tasks that are part of the DTS object model and can be accessed graphically, through DTS Designer, or programmatically. These tasks, which can be configured individually, cover a wide variety of data copying, data transformation, and notification situations. For example: • Importing and exporting data.

DTS can import data from a text file or an OLE DB data source (for example, a Microsoft Access 2000 database) into SQL Server. Alternatively, data can be exported from SQL Server to an OLE DB data destination (for example, a Microsoft Excel 2000 spreadsheet). DTS also allows high-speed data loading from text files into SQL Server tables.

• Transforming data.

DTS Designer includes a Transform Data task that allows you to select data from a data source connection, map the columns of data to a set of transformations, and send the transformed data to a destination connection. DTS Designer also includes a Data Driven Query task that allows you to map data to parameterized queries.

• Copying database objects.

With DTS, you can transfer indexes, views, logins, stored procedures, triggers, rules, defaults, constraints, and user-defined data types in addition to the data. In addition, you can generate the scripts to copy the database objects.

Note There are restrictions on this capability. For more information, see <u>Copy SQL Server Objects Task</u>.

• Sending and receiving messages to and from other users and packages.

DTS includes a Send Mail task that allows you to send an e-mail if a package step succeeds or fails. DTS also includes an Execute Package task that allows one package to run another as a package step, and a Message Queue task that allows you to use Message Queuing to send and receive messages between packages.

• Executing a set of Transact-SQL statements or Microsoft ActiveX® scripts against a data source.

The Execute SQL and ActiveX Script tasks allow you to write your own SQL statements and scripting code and execute them as a step in a package workflow.

Because DTS is based on an extensible COM model, you can create your own custom tasks. You can integrate custom tasks into the user interface of DTS Designer and save them as part of the DTS object model.

For more information, see <u>DTS Tasks</u>.

DTS Transformations

A DTS transformation is one or more functions or operations applied against a piece of data before the data arrives at the destination. The source data is not changed. For example, you can extract a substring from a column of source data and copy it to a destination table. The particular substring function is the transformation mapped onto the source column. You also can search for rows with certain characteristics (for example, specific data values in columns) and apply functions only against the data in those rows. Transformations make it easy to implement complex data validation, data scrubbing, and conversions during the import and export process. Against column data, you can:

• Manipulate column data.

For example, you can change the type, size, scale, precision, or nullability of a column.

• Apply functions written as ActiveX scripts.

These functions can apply specialized transformations or include conditional logic. For example, you can write a function in a scripting language that examines the data in a column for values over 1000. Whenever such a value is found, a value of -1 is substituted in the destination table. For rows with column values under 1000, the value is copied to the destination table.

• Choose from among a number of transformations supplied with DTS.

An example would be a function that reformats input data using string and date formatting, various string conversion functions, and a function that copies the contents of a file specified by a source column to a destination column.

• Write your own transformations as COM objects and apply those transformations against column data.

For more information, see <u>DTS Transformations</u>.

DTS Package Workflow

You can define the sequence of step execution in a package with:

- Precedence constraints that allow you to link two tasks together based on whether the first task executes, executes successfully, or executes unsuccessfully. You can use precedence constraints to build conditional branches in a workflow. Steps without constraints are executed immediately, and several steps can execute in parallel.
- ActiveX scripts that modify workflow. For more information, see <u>Using</u> <u>ActiveX Scripts in DTS</u>.

For more information, see <u>DTS Package Workflow</u>.

Connectivity

DTS is based on an OLE DB architecture that allows you to copy and transform data from a variety of data sources. For example:

- SQL Server and Oracle directly, using native OLE DB providers.
- ODBC sources, using the Microsoft OLE DB Provider for ODBC.
- Access 2000, Excel 2000, Microsoft Visual FoxPro®, dBase, Paradox, HTML, and additional file data sources.
- Text files, using the built-in DTS flat file OLE DB provider.
- Microsoft Exchange Server, Microsoft Active Directory[™] and other nonrelational data sources.
- Other data sources provided by third-party vendors.

DTS functionality may be limited by the capabilities of specific databases, ODBC drivers, or OLE DB providers. For more information, see <u>Data</u> <u>Conversion and Transformation Considerations</u>.

For more information, see <u>DTS Connections</u>.

DTS Tools

DTS includes several tools that simplify package creation, execution, and management:

- The DTS Import/Export Wizard, which is used to build packages to import, export, and transform data, or to copy database objects.
- DTS Designer, a graphical application that lets you construct packages containing complex workflows, multiple connections to heterogeneous data sources, and event-driven logic.
- The **Data Transformation Services** node in the SQL Server Enterprise Manager console tree, which is used to view, create, load, and execute DTS packages, to control DTS Designer settings, and to manage execution logs.
- Package execution utilities:
 - The **dtswiz** utility starts the DTS Import/Export Wizard by using command prompt options.
 - The **dtsrun** utility runs a package from a command prompt.
 - The DTS Run utility (**dtsrunui**) allows you to run a package using dialog boxes.
- DTS Query Designer, a visual database tool that makes it easy to build queries in DTS Designer.

For more information, see <u>DTS Tools</u>.

Meta Data

DTS includes features for saving package meta data and data lineage information to Meta Data Services and linking those types of information. You can store catalog meta data for databases referenced in a package and accounting information about the history of a particular row of data for your data mart or data warehouse.

For more information, see <u>Sharing Meta Data</u>.

See Also

Programming DTS Applications

DTS Tools

Data Transformation Services (DTS) includes the following set of tools for creating, scheduling, and executing DTS packages.

Tool	Description
DTS Import/Export Wizard	Wizard used to copy data to and from an instance of Microsoft® SQL Server [™] and to map transformations on the data.
<u>DTS Designer</u>	Graphical tool used to build complex packages with workflows and event-driven logic. You also can use DTS Designer to edit and customize packages created with the DTS Import/Export Wizard.
DTS and SQL Server Enterprise Manager	Options available for manipulating packages and accessing package information from SQL Server Enterprise Manager.
DTS Package Execution Utilities	 Includes the following: The DTS Run utility, a set of dialog boxes used to schedule and run packages. The dtsrun utility, a command prompt utility used to run packages. For more
DTS Query Designer	information, see <u>dtsrun Utility</u> . A graphical tool used to build queries in DTS.

DTS Import/Export Wizard

Of all the Data Transformation Services (DTS) tools, the DTS Import/Export Wizard provides the simplest method of copying data between OLE DB data sources.

After connecting to the source and destination, you can select the data to import or export and apply transformations to the data being copied (for example, by selecting columns or using Microsoft® ActiveX® scripts). In many cases, you can automatically copy primary and foreign key constraints along with the source data.

Note You can copy data that results from an SQL query. SQL queries can include joins of multiple tables from the same database or distributed queries. As part of the process, the DTS Import/Export Wizard creates the destination table for you automatically if none exists.

Available Data Sources

With the DTS Import/Export Wizard, you can connect to the following data sources:

- Most OLE DB and ODBC data sources, as well as user-specified OLE DB data sources.
- Text files.
- Other connections to one or more instances of Microsoft SQL ServerTM.
- Oracle and Informix databases.

You must have the Oracle or Informix client software installed.

- Microsoft Excel spreadsheets.
- Microsoft Access and Microsoft FoxPro® databases.

• dBase or Paradox databases.

For more information, see <u>DTS Connections</u>.

Transforming Data

In addition to copying data, you can transform column-level data with an ActiveX scripting language such as Microsoft Visual Basic® Scripting Edition (VBScript) or Microsoft JScript®. For more information, see <u>DTS</u> <u>Transformations, Transform Data Task</u>, and <u>Using ActiveX Scripts in DTS</u>.

Copying Database Objects

With the DTS Import/Export Wizard, you can transfer database objects such as indexes, views, roles, stored procedures, and referential integrity constraints. For more information, see <u>Copy SQL Server Objects Task</u>.

Saving DTS Packages

After you complete the DTS Import/Export Wizard, you can save the connections, transformations, and scheduling information as a DTS package. The package can be saved:

- To the SQL Server **msdb** database.
- To SQL Server 2000 Meta Data Services.
- As a structured storage file (.dts file).
- As a Visual Basic file.

You can run the package immediately or schedule it for later execution.

For more information, see <u>Saving a DTS Package</u> and <u>Scheduling a DTS</u> <u>Package for Execution</u>.

Editing Packages

If you create a package with the DTS Import/Export Wizard and then save it, you can edit it in DTS Designer. Using DTS Designer, you can customize the basic package you created in the DTS Import/Export Wizard, adding steps, tasks, transformations, event-driven logic, and configuring workflow. For more information, see <u>DTS Designer</u>.

See Also

dtswiz Utility

Creating a DTS Package with the DTS Import/Export Wizard

The Data Transformation Services (DTS) Import/Export Wizard offers the simplest method of building a DTS package, interactively guiding you through the process of copying and transforming data. Following are the basic steps for creating a package with the DTS Import/Export Wizard:

1. Specify whether you are importing or exporting data.

You need to specify whether you are exporting data from an instance of Microsoft® SQL Server[™] to another data source (for example, a second instance of SQL Server 2000) or importing data from another data source to an instance of SQL Server. Both choices are available in SQL Server Enterprise Manager, through the **Data Transformation Services** node of the console tree, and as command switches through the **dtswiz** command prompt utility.

When accessing the DTS Import/Export Wizard from the **Start** menu or the command prompt, you do not need to specify whether you are importing or exporting data.

2. Choose a data source and data destination.

You can select from a list of OLE DB data sources, which includes providers for both databases and nondatabase sources (for example, text files). You also must specify any required login, security, or file location information. If you are importing data from a text file, you must specify the format and delimiters of the text file.

When you import data, the active server connection is specified as the default destination server. When you export data, the active server connection is specified as the default source server.

- 3. Choose whether to copy a table or view, copy query results, or transfer objects and data.
 - If you choose to copy data, you need to decide which columns or views to copy and whether to transform the data. If your

source data is a view, the DTS Import/Export Wizard automatically converts the view to a table in the destination.

- If you choose to query the source data and copy the results, you need to construct an SQL query, which also can be a heterogeneous or distributed query. You can enter the SQL query manually or graphically. After you have completed the query, you can decide whether to add transformations to the query results. For more information about graphically entering an SQL query, see <u>DTS Query Designer</u>.
- If you choose to transfer database objects between instances of SQL Server, you need to: select which objects to transfer (for example, views, stored procedures, indexes, and rules); choose whether to drop existing database objects first; replace or append existing data; and include dependent objects.

For more information, see <u>Copy SQL Server Objects Task</u>.

4. Optionally select columns, add transformations, or copy constraints.

If you copy the data or the results of a query, you can customize the data being copied to the destination. You can:

- Select which source or destination columns to copy.
- Select which source or destination columns to ignore.
- Change the data type where valid.
- Define how the data is to be converted between source and destination.
- Map transformations onto column or row data using Microsoft ActiveX® scripts.

5. Save, run, or schedule a package.

After you have created the package, you need to decide:

• The format in which to save the DTS package.

You can save the package to the SQL Server **msdb** database, to SQL Server 2000 Meta Data Services, as a structured storage file (.dts file), or as a Microsoft Visual Basic® file.

• When you want to run the package.

You can run the package after the DTS Import/Export Wizard completes, or you can schedule the package to execute on a regular basis using SQL Server Agent.

You can use the Create Publication Wizard to publish the data. For more information, see <u>Replication Wizards</u>.

IMPORTANT Do not open a Microsoft Excel file that is being used as a source or destination during the wizard creation or execution, because a "file in use" error will occur.

To create a DTS package using the DTS Import/Export Wizard

DTS Designer

Data Transformation Services (DTS) Designer is a tool you can use to import, export, and transform heterogeneous data between one or more databases and an instance of Microsoft® SQL Server[™]. DTS Designer graphically implements the DTS object model, allowing you to create DTS packages with a wide range of functionality. You can use DTS Designer to:

- Create a simple package (for example, a package that copies data between databases).
- Create a package that includes complex workflows (for example, a package that contains branches, multiple steps, multiple connections, complicated logic, and event-driven code).
- Edit an existing package (for example, a package you have created with the DTS Import/Export Wizard).

Accessing DTS Designer

You access DTS Designer through SQL Server Enterprise Manager, through the **Data Transformation Services** node of the console tree.

DTS Designer User Interface

The DTS Designer graphical user interface allows you to build and configure packages by using drag-and-drop methods and by completing property sheets on the various DTS objects composing the package. The user interface includes:

- The DTS Designer main panel, which consists of the following parts:
 - A design sheet upon which you create workflows by dragging graphical objects that represent DTS tasks, DTS transformations, and precedence constraints.
 - A menu bar containing selections for package operations, edit

operations, data sources, tasks, and workflow items.

- A toolbar containing buttons for: creating, saving, and executing a package; printing a workflow; cutting, copying, and pasting graphical objects in a workflow; annotating a workflow; and changing both the workflow layout and the size of a workflow on the design sheet.
- A **Connection** toolbar containing connections for data sources.
- A **Task** toolbar containing DTS tasks.

You can dock and undock the **Connection** and **Task** toolbars by using their shortcut menus.

• Shortcut menus for configuring and editing package components and workflow (for example, connections, tasks, and workflow objects).

Creating a Package with DTS Designer

The following example shows you how to build, configure, execute, and save a Data Transformation Services (DTS) package.

To create a DTS package using DTS Designer

DTS Designer Example: A Completed DTS Package

The following diagram shows a completed Data Transformation Services (DTS) package on the DTS Designer design sheet. The graphical objects on the design sheet represent connections, tasks, and precedence constraints.



The following is a description of the graphical objects:

- A connection to a text file containing source data (the **Text File (Source)** icon) and a second connection to a destination, the **Northwind** database (the **Northwind** icon).
- A Transform Data task (the gray arrow) that defines the data being copied and transformed.
- Two precedence constraints (striped arrows) that further designate workflow:
 - If the package is run and the data is copied successfully from the text file to an instance of Microsoft® SQL ServerTM, an index is created on the table to which the data is copied (the striped arrow from the **Northwind** icon to the **Create Index** icon).
 - If the package is run and the data copy fails, an e-mail is sent to a database administrator (the striped arrow from the

Northwind data icon to the Send Mail task icon).

• A text annotation (the label "Customer Update, Chicago").

DTS Designer Example: Copying Northwind Data

This example demonstrates how to use Data Transformation Services (DTS) Designer to copy a **Northwind** database table from the source to the destination. The basic steps are:

- 1. Make two connections, one to the source and a second to the destination.
- 2. Add a Transform Data task that defines a source and destination table and the copying operation to be performed.
- 3. Add an Execute SQL task that checks for the existence of the destination table prior to copying the data. If the table does not exist, it is created. If the table exists, it is dropped and re-created.
- 4. Configure the workflow so the DTS package steps execute in the correct sequence.
- 5. Run the package.
- 6. Save the package so that it can be reused.

Connecting to the Source

Begin by creating a connection to the **Northwind** database on your local server.

To create a connection to Northwind in DTS Designer

DTS Package Templates

Data Transformation Services (DTS) package templates are partially configured packages built around typical usage situations. Each template contains tasks and workflow items geared toward a specific task or set of tasks (for example, copying data between instances of Microsoft® SQL Server[™] 2000). Copy and complete these templates to configure tasks, connections, and workflow easily and quickly.

After you configure the template, save a copy as a package to a new location. Then, you can reuse the template, customize copies, and save the information in any DTS format. Because DTS templates are read-only files (with a .dtt extension), you cannot accidentally overwrite a template.

By default, package templates are saved in the \\Tools\Templates\DTS folder. However, you can save and use package templates from any location.

Creating a Template

To create a package template, build a package and use Disconnected Edit to stub out the properties you want template users to configure. For example, instead of entering a specific user name in the **Connection Properties** dialog box, use Disconnected Edit to assign text such as "Enter your login name here" to the property for user name. You also can add instructions and labels to the package template. After you build the package template, save it as a structured storage file with a .dtt extension.

To create and save a package template

DTS and SQL Server Enterprise Manager

The Data Transformation Services (DTS) node of the SQL Server Enterprise Manager console tree provides facilities for accessing DTS tools, manipulating DTS packages, and accessing package information. You can use these facilities to:

• Open a new package in the DTS Import/Export Wizard or DTS Designer. In DTS Designer, you can select and edit an existing package saved to SQL Server, SQL Server 2000 Meta Data Services, or to a structured storage file.

For more information, see <u>Creating a DTS Package</u> and <u>Editing a DTS</u> <u>Package</u>.

• Connect to and import meta data from a data source, and display the meta data in the **Meta Data** node of SQL Server Enterprise Manager.

For more information, see <u>Viewing Meta Data in DTS</u>.

• Open a package template in DTS Designer.

For more information, see <u>DTS Package Templates</u>.

• Display the version history of a package, edit a specific package version in DTS Designer, and delete package versions.

For more information, see <u>Saving a DTS Package to Meta Data</u> <u>Services</u>.

• Display and manipulate package log information.

For more information, see <u>Using DTS Package Logs</u>.

- Set the properties of DTS Designer by right-clicking the **Data Transformation Services** node and clicking **Properties**.
- Execute a package.

For more information, see **Executing a DTS Package**.

• Schedule a package.

For more information, see <u>Scheduling a DTS Package for Execution</u>.

DTS Package Execution Utilities

Data Transformation Services (DTS) packages can be run from either of two package execution utilities included with Microsoft® SQL Server[™] 2000: the DTS Run utility and the **dtsrun** command prompt utility. By using these utilities, you do not need to open a package to run it.

The DTS Run utility allows you to create a command prompt and run a package or schedule a package outside of the Microsoft Management Console (MMC). With **dtsrun**, you can run a package from the command prompt or a batch file.

DTS Run Utility

With the DTS Run utility, you can execute an existing package from a set of dialog boxes that you call from the command prompt.

Using the DTS Run utility, you can set:

• Connection settings.

You can specify the server name or file name, identify how the package was saved, and provide login information.

• Scheduling options.

You can specify regular package execution through SQL Server Agent.

• Logging options.

You can identify and enable an event log.

• Global variable settings.

You can add new global variables and change the properties of existing global variables. Modifications to package global variables are in effect only for the duration of a DTS Run utility session. When the session is closed, changes to package global variables are not saved.

• Encryption options.

You can encrypt the command prompt options to be executed by the DTS Run utility, allowing you to create an encrypted **dtsrun** command

for later use.

You also can use the DTS Run utility to generate the text of a command prompt for **dtsrun**, with or without encrypted arguments. You can save this text, copy it to a file, and reuse it later to execute the same operation from a command prompt.

You access the DTS Run utility by executing **dtsrunui** from a command prompt without any command switches.

To execute a DTS package using the DTS Run utility

DTS Query Designer

Data Transformation Services (DTS) Query Designer uses a graphical user interface for creating SQL queries. Use DTS Query Designer to:

• Work visually or with SQL commands.

DTS Query Designer includes graphical panes that display your query visually and a text pane that displays the SQL text of your query. You can work in either the graphical or text panes. DTS Query Designer synchronizes the views so they are always current.

• Join related tables.

If you add more than one table to your query, DTS Query Designer automatically determines how the tables are related and constructs the appropriate join command.

• Query or update databases.

You can use DTS Query Designer to return data using Transact-SQL SELECT statements and to create queries that update, add, or delete records in a database.

• View and edit results immediately.

You can execute your query and work with a record set in a grid that allows you to scroll through and edit records in the database.

See Also

Data Transformation Services Query DesignerDiagram PaneGrid PaneNavigating in the Query DesignerResults PaneSQL Pane

DTS Package Elements

This section describes the main elements of a Data Transformation Services (DTS) package.

Торіс	Description
DTS Tasks	Describes the DTS tasks and their typical uses.
DTS Transformations	Describes the column-level transformations
	available through the DTS tasks that transform
	data.
DTS Connections	Explains the different types of connections
	available in a DTS package.
DTS Package Workflow	Discusses how workflow is ordered in a package
	and how package steps execute.

DTS Tasks

Usually, a Data Transformation Services (DTS) package includes one or more DTS tasks. Each task defines a work item to be performed as part of the data movement and data transformation process.

Microsoft® SQL Server[™] 2000 supplies several DTS tasks that are part of the DTS object model. These tasks can be accessed through DTS Designer (except for the Parallel Data Pump task, which can only be accessed programmatically). You can use them to:

• Transform data. For example, you can use the Transform Data task to copy data, map a wide variety of transformations onto the data, and customize the transformations with a Microsoft ActiveX® script.

For more information, see <u>Data Driven Query Task</u>, <u>Transform Data</u> <u>Task</u>, and <u>ParallelDataPump Task Object</u>.

• Copy and manage data. For example, you can drop a table, re-create and repopulate the table, and execute a series of queries against the table. Also, you can generate a disconnected Microsoft ActiveX Data Objects (ADO) recordset, which you can then manipulate and access from other steps in the package.

For more information, see <u>Copy SQL Server Objects Task</u>, <u>Execute</u> <u>SQL Task</u>, <u>Bulk Insert Task</u>, and <u>Transfer Database Objects Tasks</u>.

Run tasks as jobs from within a package. For example, you can use an Execute Process task to run a custom Microsoft Visual Basic® application that collects and aggregates data on a daily basis. Then, you can use an Execute Package task to run a second package that imports and transforms the data generated by the Visual Basic application. You also can use the Send Mail task to send an e-mail to a system administrator if a package step succeeds or fails.

For more information, see <u>ActiveX Script Task</u>, <u>Dynamic Properties</u> <u>Task</u>, <u>Execute Package Task</u>, <u>File Transfer Protocol Task</u>, <u>Execute</u> <u>Process Task</u>, and <u>Send Mail Task</u>. Additionally, you can build your own custom task in a programming language that supports COM (for example, Visual Basic). You can create a user interface for the custom task, including its own icon, if you want to access the custom task in DTS Designer. For more information, see <u>DTS Custom Task</u>.

Adding and Configuring Tasks

A DTS package can contain a single task (for example, an ActiveX Script task that displays a message box when the package is run). However, a package often contains several tasks, connections, and workflow constraints, with each task set to run in the context of an ordered package workflow. You can include multiple tasks of the same type in a package (for example, six Execute SQL tasks), with each task configured differently.

You can add tasks to a package and set their properties in the following ways:

- Graphically, using DTS Designer.
- Programmatically, using the DTS object model to build a package in Visual Basic or Microsoft Visual C++®. For more information, see <u>Programming DTS Applications</u>.

To add a DTS task to a DTS package

Tasks That Transform Data

In Data Transformation Services (DTS), tasks that transform data are based on an architectural component called the DTS data pump.

Task Description **Transform Data Task** Allows point-to-point copying and transforming of data between a broad range of OLE DB-compliant data sources. Data Driven Query Task Allows you to perform flexible, Transact-SQL based operations on data, including stored procedures and INSERT, UPDATE or **DELETE** statements. Allows copying and transforming of data ParallelDataPump Task containing OLE DB hierarchical rowsets. <u>Object</u> This task is only accessible programmatically, through the DTS object model.

The following DTS tasks implement the DTS data pump.

The DTS data pump, an OLE DB service provider, is a COM object that provides a set of data movement interfaces.

The data pump architecture supports:

- High-speed batch copying of transformed or non-transformed data.
- Use of scripting code to define transformations.
- A variety of supplied transformations for converting string data.
- Custom transformations, written in Microsoft® Visual C++® and compiled as COM objects, that you can access programmatically or through the DTS Designer user interface.

The DTS data pump also allows users to add programs or Microsoft ActiveX® script functions that can access specific phases of a data pump operation. For example, you can add a function that instructs the data pump to write header information to a file before the source data is copied and transformed. For more information, see <u>Multiphase Data Pump Functionality</u>.

Configuring a Task that Transforms Data

To configure a task that transforms data, follow these steps:

1. Establish a connection. You need to connect, at minimum, to the source and destination data sources.

Before configuring these tasks in DTS Designer, you must create two live connections to data sources. If you define additional connections in an ActiveX Script transformation, those connections can be made at package run time.

For more information, see <u>DTS Connections</u> and <u>Configuring a Simple</u> <u>Lookup Query</u>.

2. Perform any required data type conversions and transformations onto the data copied from the source connection.

If you intend to transform the data, you need to map column-level transformations. You can use one of the following DTS tools:

• The DTS Import/Export Wizard. By default, the wizard creates one or more transformation tasks (depending on the number of source tables selected for copying). To transform column or row data in the wizard, you must write a transformation script using ActiveX scripting code. To edit the transformation script, you must open the package in DTS Designer, open the properties dialog box for the task, and edit the script.

For more information, see <u>Using ActiveX Scripts in DTS</u>.

• DTS Designer. You can graphically map source columns to destination columns, select a transformation type, and apply one to a mapping. Alternatively, you can map transformations

using selection boxes.

Note For most situations, you will only map columns with the Transform Data task. It is not recommended you change the default column mappings for a Data Driven Query task unless you are an advanced user. For more information, see <u>Building a Data Driven Query</u>.

Detecting Row-Level Errors

Tasks that transform data use exception files to record information about failed rows. Exception files can contain:

- Package information, such as package name, description, and version.
- Step execution information, including the name of the package step associated with the data pump operation and step execution times.
- Error information, including the source of the error (for example, the data pump or a connection) and a description of the error (for example, an insert error that occurred on **EmployeeData** column, row 2007).

The tasks that transform data are able to detect row-level errors before the row is submitted to the database. For example, suppose an input row contains missing or incorrectly formatted data. When these tasks encounter such a row, they fail the row and do not pass it to the destination. This error counts as one failure toward the maximum error count.

Some errors, such as duplicate keys or referential integrity violations, cannot be detected at row level by these tasks. Such rows fail only after being passed to the destination. The failure is noted in the exception file, but the actual rows that failed are not logged. Thus, complete error information is not always available in the exception logs.

If you configure an exception log for a task that transforms data, step execution information is appended to the exception file you specify each time the package is run. If you specify an exception file that does not currently exist, the file will be created at package execution time. If the step associated with the task does not run, no exception file data is generated.

You also can create additional log files to capture source and destination rows that failed when a task that transforms data is executing by using the Microsoft SQL Server[™] 2000 exception file options. You can use these files to examine failed rows and troubleshoot problems with the data. The source row and destination row log files have the same name as the exception file, but with the extensions ".Source" and ".Destination", respectively, appended to the name. These files are only created if source row errors or destination row errors occur during execution of the transformation task.

To configure the data pump exception files

Transform Data Task

You use the Transform Data task to copy data between a source and destination and to optionally apply column-level transformations to the data. The Transform Data task is the most basic implementation of the data pump engine in Data Transformation Services (DTS).

The Transform Data task is optimized for insert-based copying and transforming of column-level data between commercial databases, spreadsheets, and text files. You can use the task to copy and transform data between any supported OLE DB connections. Because the task handles such a wide variety of data sources and transformation scenarios, you will frequently use one or more instances of it when creating packages that consolidate data from disparate sources.

Note If you need to bulk insert text files into Microsoft® SQL ServerTM and are concerned with performance, use the Bulk Insert task. However, you cannot transform data with the Bulk Insert task. For more information, see <u>Bulk Insert Task</u>.

Batching Data

If the destination connection for a Transform Data task is the Microsoft OLE DB Provider for SQL Server, you can use the fast load option, which is available through the OLE DB **IRowsetFastLoad** interface. The fast load option supports high-performance bulk-copy processing. When the fast load option is enabled, the data pump can accept batches of transformed data.

When you batch data, the data pump writes the transformed rows to a destination buffer but does not submit them to SQL Server until either the specified batch size or the end of the source data is reached.

You can customize batch processing in a Transform Data task by writing your own functions. For more information, see <u>Multiphase Data Pump Functionality</u>.

You cannot use fast load with a Data Driven Query task.

IMPORTANT Also, when using the fast load option in a Transform Data task with a lookup connection, make sure your lookup connection is not the same as the

source or destination connections.

Applying Transformations to Batches

Transformations are applied prior to the bulk copying process. Therefore, you can use the same column-level transformations with the fast load option that you can use without the option (for example, a Microsoft ActiveX® Script transformation).

Handling Errors and Batched Data

If a row contains errors and the Transform Data task does not detect them, the row causes the entire batch to fail on submission. Valid rows in such a batch are neither inserted nor noted as an error in the data pump exception files. The failure of an entire batch counts as only one failure toward the maximum error count.

You can control both the size of a batch and the way the data will be committed if a failure occurs. Before setting the batch size, consider the following:

- Error descriptions that occur in a batch will be recorded in a log file, but the error rows may not be available by the time the batch rolls back. As a result, the error rows may not be logged.
- By default, the batch size is set to 0, which means that all the rows copied from the source are placed in a single batch before being submitted to SQL Server.

For example, if your source contains a million rows of data, and the batch size is set to the default, the data will not commit until the one-millionth row is processed. In cases such as these, you may want to commit the data in batches of one thousand, or ten thousand, rather than in a single batch of one million.

- A batch size of one means that each row of data processed is treated as a batch.
- Input rows containing errors detected by the Transform Data task are

neither added to the current batch nor counted as rows in the batch. For example, a batch size of 10 might be filled from source rows 1 thru 12, where rows 3 and 4 contain incorrectly formatted dates and are not copied to the destination. You cannot in this case assume that the third batch of data contains source rows 21 thru 30.

Before configuring the error options for batching data, consider the following:

- When you enter a maximum error count value for a Transform Data task, you specify the sum of row-level errors detected by the Transform Data task and batch failures. When the **Max error count** value is exceeded, task execution is terminated.
- Because rows containing errors detected by the Transform Data task are discarded before batch submission, these errors do not trigger nor count as batch failures. Errors caught at the destination will fail the batch and add one to the error count regardless of how many rows are in the batch.
- You can set an option that will trigger the submission of the current batch when the maximum error count value is exceeded. This option is useful when the only types of errors you anticipate are row-level errors that will be detected by the Transform Data task.

Batches and Package Transactions

Batches succeed and fail independently of the package transaction. For more information, see <u>Configuring Properties for DTS Transactions</u>.

If the Transform Data task takes part in the package transaction, any successfully added batches are submitted only after the package transaction is committed.

Configuring a Transform Data Task

In DTS Designer, configure a Transform Data task by following these steps:

1. Configure connections. You must first establish a source and destination connection.

To configure the connections for a Transform Data task

Data Driven Query Task

The Data Driven Query task allows you to perform flexible, Transact-SQL based operations on data, including stored procedures and INSERT, UPDATE or DELETE statements. For each row in a source rowset, the Data Driven Query task selects, customizes, and executes one of several SQL statements. You select which statement to execute via a constant return value set in a Microsoft® ActiveX® script transformation. Based on the return constant you use in the script, one of four different parameterized SQL statements that you create may be executed for each source row.

When designing a Data Driven Query task, you need to decide whether the task should include a single query or multiple queries. For example, if you only want the task to delete data, you use a single query (a Delete query). If you want to update some rows and delete others, you need to use two queries (an Update and Delete query). If you use multiple queries, you need to provide scripting code that supplies conditional logic. That logic determines when each query type is applied to the data.

Using the Data Driven Query task, you can:

- Run any large collection of updates that are not necessarily inserts. For example, you can purge expired historical data from a database, given a list of keys generated by another database.
- Perform traditional file maintenance. For example, you can optionally insert or update records depending on their previous existence or some other external factor.
- Customize Microsoft SQL Server[™] transformable subscriptions, which are used to distribute and transform incremental changes during replication. For example, you can tailor the response of each subscriber to insert, delete, or update requests.

Use the Data Driven Query task when, for each of many source rows, you must

either:

• Choose among more than one edit operation.

-or-

• Perform a non-insert edit (for example, perform an update or delete operation, or execute a stored procedure).

The Transform Data task and the Bulk Insert task are optimized for insert operations. Choose the Data Driven Query task for insert operations only if these tasks do not meet the requirements of your application.

You can use either DTS Designer or the DTS object model to create and manage Data Driven Query tasks. For more information about programming for the Data Driven Query task, see <u>DataDrivenQueryTask2 Object</u> and <u>Creating a</u> <u>Transformable Subscription Using Visual Basic</u>.

See Also

Transform Data Task
<u>Using ActiveX Scripts in DTS</u>
<u>Using Parameterized Queries in DTS</u>

Data Flow in a Data Driven Query Task

When you use a Data Driven Query task, the data flows as follows:

- 1. Rows are selected from the source.
- 2. Each row is transformed by the Microsoft® ActiveX® script (and possibly additional transformations). The return value of the ActiveX script determines which query will be selected later.
- 3. The results of the transformation are mapped to the binding table. Nothing is written to the destination at this time. The binding table is used to determine the attributes of the columns (for example, data type, scale, precision) that will be used by the parameterized queries in the next step.
- 4. For each row, the values in the binding table are mapped into the selected query, and the query is executed.
- 5. The query execution writes the row to the destination.

See Also

Using ActiveX Scripts in DTS Using Parameterized Queries in DTS Using Return Codes in DTS

Building a Data Driven Query

Although you can use Data Driven Query tasks to build data driven queries that vary in complexity, you follow the same basic steps for building a data driven query in all cases.

The following examples show how to build data driven queries, one simple and one complex, using the Data Driven Query task in Data Transformation Services (DTS) Designer. In the first example, every source row leads to the update of a destination row. Only one query type, an Update query, is used. In the second example, multiple query types, both an Update and a Delete query, are used.

To update a destination row using a single query type

1. Define the source data.

The data can be a table or the results of a SELECT statement, parameterized SELECT statement, or stored procedure. You choose the data source on the **Source** tab of the **Data Driven Query Task Properties** dialog box.

2. Define a binding table.

This binding table allows you to transform source data before it appears in your SQL statements. The Data Driven Query task makes no actual changes to the destination unless you specifically request them in your queries. The binding table must be an existing table. This table is used to define the schema for binding the parameters. The columns of this binding table must match those of the parameterized queries that will be written against it (Step 5) in type. You declare the binding table on the **Bindings** tab of the **Data Driven Query Task Properties** dialog box.

3. Provide the decision-making logic by using a Microsoft® ActiveX® script.

You create ActiveX script code in the text box of the **ActiveX Script Transformation Properties** dialog box to determine which query will be executed. You display that dialog box by double-clicking the column mapping line for the transformation in the **Transformations** tab of the **Data Driven Query Task Properties** dialog box.

Note By default, the Data Driven Query task maps all source columns to all columns in the binding table as a single transformation. The mapping is displayed on the **Transformations** tab of the **Data Driven Query Task Properties** dialog box. It is recommended you do not use more then one ActiveX script for a Data Driven Query task, for that would reduce performance. For more information, see Enhancing Performance of DTS Packages.

4. Edit the ActiveX script so that the return code matches **DTSTransformStat_UpdateQuery**.

The default return code for any query type is **DTSTransformStat_InsertQuery**. However, the query type labels are suggestions only. Although you can use the default Insert query to perform an Update query, for readability, you may want to change the return code to the appropriate one for an update.

The ActiveX script contains references to the **DTSDestination** columns collection, which refers to the binding table (Step 2). The parameters of the SQL operation will be applied to this table.

5. Create the Update query.

Click **Update** from the **Query type** list on the **Queries** tab of the **Data Driven Query Task Properties** dialog box. Build the parameterized query, either by typing the query into the edit box or by using DTS Query Designer. Use a question mark (?) as the placeholder for the parameters that will be filled in by the processed data.

The attributes of a binding table column must match that of any parameter it is assigned to.

On the **Queries** tab, a grid displays one row for every parameter (?) entered in the selected query (Update). By default these rows will map to binding table columns ordinally, from the transformation previously defined. For more information, see <u>Using Parameterized Queries in DTS</u>.

To update or delete a destination row using multiple query types

- 1. Define the source data and binding table as you did in the previous example (Steps 1 and 2).
- 2. Provide the decision logic to apply to each query type.

Your script, based on the conditional logic you provide, returns one of four return codes. The return code tells DTS which query to execute for the current source row. You edit the default ActiveX script in the text box of the **ActiveX Script Transformation Properties** dialog box to include the logic and the return code.

The decision logic you use to determine the query type typically consists of branching statements (IF-THEN-ELSE or CASE).

In this example, the following two return codes are used:

DTSTransformStat_UpdateQuery

DTSTransformStat_DeleteQuery

For more information, see <u>Using ActiveX Scripts in DTS</u> and <u>Using</u> <u>Return Codes in DTS</u>.

3. Create the queries.

For each query, select a query type from the **Query type** list on the **Queries** tab of the **Data Driven Query Task Properties** dialog box. Build the parameterized query, either by typing the query into the text edit box or by using DTS Query Designer.

The queries do not have to have the same number of parameters or map to the same columns in the binding table. The queries and the mappings are edited separately.

See Also

Using ActiveX Scripts in DTS

Using Return Codes in DTS

Data Driven Query Example: Changing Customer Accounts

The following example details an appropriate situation in which to use a Data Driven Query task: A source row triggers one of three different edits, two of which are not insert operations.

You have an **Account** table, with columns for **CustomerID** and **CompanyName**. **CustomerID** serves as a key:

CREATE TABLE Account (CustomerID nchar (5) NOT NULL, CompanyName nvarchar (40) NOT NULL)

Required **Account** table changes accumulate in the **AccountChange** table. It contains **CustomerID** and **CompanyName** columns, as well as a **ChangeCode** column:

CREATE TABLE AccountChange (CustomerID nchar (5) NOT NULL, ChangeCode nchar (10) NOT NULL, CompanyName nvarchar (40) NULL)

Different values of **ChangeCode** are used to request different **Account** table modifications.

ChangeCode	Required Action
New	Add a new customer to the Account table.
Change	Change the CompanyName for an existing customer.
Delete	Remove a customer row from the Account table.

The **AccountChange** table serves as the source. Each **AccountChange** row triggers one of three actions. These actions will be represented by INSERT, UPDATE, and DELETE queries.

Note In this example, all the changes were made to a single table. However, that is not a requirement of the Data Driven Query task. It requires only that the affected data all reside on the same connection.

Specifying Query Statements

After identifying an appropriate problem for the Data Driven Query task, you first specify up to four parameterized SQL statements to carry out required edit operations.

Each action requires an SQL query or a stored procedure invocation. These statements are parameterized by replacing, with a question mark, any expressions that vary from source row to source row. For example, two change actions might trigger the following commands:

UPDATE Account SET CompanyName = 'Big Pizza' WHERE Custom UPDATE Account SET CompanyName = 'Tasty Gyro' WHERE Custo

Parameterized, the UPDATE query reads:

UPDATE Account SET CompanyName = ? WHERE CustomerID = ?

At run time, the question marks will be replaced by values drawn from, or based on, source column data.

Stored procedure calls are parameterized like queries, with question marks replacing arguments:

sp_updatebalance ?, 'Credit', ?

Parameterized queries for the new and delete actions are as follows:

INSERT INTO Account (CustomerID, CompanyName) VALUES (?, ? DELETE FROM Account WHERE (CustomerID = ?)

Assigning Query Types to Statements

In order to refer to your SQL statements, you assign each statement a name, called a query type. A query type, returned by your Microsoft® ActiveX® script code, is used to select one of your SQL statements to execute. Data

Transformation Services (DTS) provides the following four names:

- Insert
- Update
- Delete
- User

These query types should be viewed only as unique identifiers assigned to each statement. It is in fact possible to perform any SQL operation supported by the connection. It would be possible for example, to perform four different updates, four different inserts or any mix of these or stored procedures.

The example is one of those applications in which the available query types match the parameterized SQL statements. Therefore, the assignments are as follows.

Query type	Parameterized Query
Insert	INSERT INTO Account (CustomerID, CompanyName) VALUE
Update	UPDATE Account SET CompanyName = ? WHERE Customer
Delete	DELETE FROM Account WHERE (CustomerID = ?)

Specifying the Binding Table

After you assign query types to your SQL statements, you need to specify a binding table whose columns match the parameters in your parameterized query. This binding table allows you to transform source data before it appears in your SQL statements. For example:

- A customer name can be uppercase characters.
- An address line can be constructed by concatenating several fields.
- City information can be looked up, given a postal code in the source data.

The Data Driven Query task makes no actual changes to the destination unless you specifically request them in your queries.

To specify the binding table, list all the parameters required by your queries. Then, review your existing tables to see if any contain all the columns in your list. If you find such a table, it can serve as the binding table. If no existing table contains all of the required parameters, create a new table that does.

Note The Data Driven Query task requires that the source table and binding table use different connections.

For example, the preceding queries use two parameters:

- CustomerID
- CompanyName

A review of the database yields two tables that contain both columns: **Account** and **AccountChange**. Because **AccountChange** is likely to serve as the source, **Account** is the better choice for the binding table. No new table is necessary.

In this example, the binding table is the same table that the queries update. This frequently happens but is not required by the Data Driven Query task, as the binding table exists only to map meta data (size, scale, precision, and nullability) for the queries. It is not actually written to. Only the query operation affects the data

Specifying the Source Rowset

After specifying the binding table, you either must choose an existing table as a source or specify a new source rowset. Each source row must contain enough

information to:

- Determine the appropriate query to execute.
- Fill any parameters required by the chosen query.

If this information is available in a single table, it can serve as the source. If not, you can create an SQL query to collect required information in a single source rowset.

If necessary, custom ActiveX script code, perhaps referencing DTS lookup queries, can be used to help determine the proper query to execute. Source data can be copied immediately into binding column parameters, or the data may undergo intermediate processing through ActiveX code or DTS custom transformations.

In the example, the **AccountChange** table will serve as the source. It fulfills both requirements:

- The **ChangeCode** column determines the choice of query.
- The **CustomerID** and **CompanyName** columns are sufficient to fill all required parameters.

Specifying the ActiveX Transformation

To choose which query to execute, you must code a single ActiveX transformation. This script returns one of four values, which is then used to select the query to execute. Additionally, you may choose to include ActiveX code to populate destination parameters.

The return values and their associated query types are as follows.

Return value	Executes Query Type
DTSTransformstat_InsertQuery	Insert
DTSTransformstat_UpdateQuery	Update
DTSTransformstat_DeleteQuery	Delete
DTSTransformstat_UserQuery	User

These query types should be viewed only as identifiers for one of your queries. You can assign, for example, the Insert type to a DELETE query. If your script returns **DTSTransformstat_InsertQuery**, the DELETE query will be triggered.

Usually your code takes the form of a nested IF or SELECT CASE structure. For example, to choose among three queries based on the value of **ChangeCode**, use the following code:

```
Select Case Trim(DTSSource("ChangeCode"))
Case "New"
Main = DTSTransformStat_InsertQuery
Case "Change"
Main = DTSTransformStat_UpdateQuery
Case "Delete"
Main = DTSTransformStat_DeleteQuery
Case Else
Main = DTSTransformStat_SkipRow
End Select
```

The above code responds to erroneous **ChangeCode** values by returning **DTSTransformStat_SkipRow**. No query is triggered for the source row.

You can use a Copy Column transformation or other column-level transformations to populate binding table columns, or you can fill them through additional code in your ActiveX transformation:

```
DTSDestination("CustomerID") = DTSSource("CustomerID")
DTSDestination("CompanyName") = DTSSource("CompanyName")
```

You are not required to fill every destination column, only those required by the chosen query.

See Also

Lookup Queries

DTS Transformations
Data Driven Query Task

Data Driven Query Example: File Maintenance

The following example presents a file maintenance problem and then examines the steps necessary to prepare a Data Driven Query task solution.

For more information about the basics of the Data Driven Query task, see <u>Data</u> <u>Driven Query Task</u>.

Identifying a Data Driven Query Problem

You have an **Account** table, with columns for **CustomerID** and **CompanyName**. **CustomerID** serves as a key:

CREATE TABLE Account (CustomerID nchar (5) NOT NULL, CompanyName nvarchar (40) NOT NULL)

You also have an **AccountJournal** table. One row is written to this table each time there is a change in the customer balance:

CREATE TABLE AccountJournal (UpdateID int NOT NULL, CustomerID char (5) NOT NULL, JournalAmount money NOT NULL)

You want to use the **AccountUpdate** table as a source table:

CREATE TABLE AccountUpdate (UpdateID int IDENTITY (1, 1) NOT NULL, UpdateCode char (10) NOT NULL, CustomerID char (5) NOT NULL, CompanyName char (30) NULL, UpdateAmount money NULL)

Different values of UpdateCode are used to request different Account and

AccountJournal modifications.

UpdateCode	Required Action
Purchase	Write a new AccountJournal row with JournalAmount set
	equal to UpdateAmount .
Payment	Write a new AccountJournal row with JournalAmount set
	equal to UpdateAmount * -1.
New	If a customer does not exist, add one. If the customer exists,
	change the old CompanyName to the new one.

A Data Driven Query task is appropriate for this problem. Each entry in the **AccountUpdate** table triggers one of three different queries.

Specifying Query Statements

Three parameterized SQL statements are required to solve this problem:

INSERT AccountJournal (UpdateID, CustomerID, JournalAmount)
VALUES (?, ?, ?)
INSERT INTO Account (CustomerID, CompanyName) VALUES (?, ?
UPDATE Account SET CompanyName = ? WHERE CustomerID = ?

The first query adds a row to the **AccountJournal** table in the case of a "Purchase" or a "Payment"; the second query adds a new account if the customer was not previously on file; and the last updates the **CompanyName** for an existing customer.

Assigning Query Types to Statements

There are two INSERT queries and one UPDATE query. This means that query types are not going to match the actual query content.

Query	
type	Parameterized Query
Insert	INSERT AccountJournal (UpdateID, CustomerID, JournalAmo
	VALUES (?, ?, ?)

Update	UPDATE Account SET CompanyName = ? WHERE Customer
User	INSERT INTO Account (CustomerID, CompanyName) VALUE

The second INSERT query is assigned arbitrarily to the User query type. The Delete type would work just as well.

Specifying the Binding Table

The binding table provides names and data types for your SQL parameters. Your queries use the following parameters:

- UpdateID
- CustomerID
- JournalAmount
- CompanyName

There is no table in your database that contains all four of these columns. Therefore, you must create a new binding table:

CREATE TABLE AccountDestination (UpdateID int NOT NULL, CustomerID char (5) NOT NULL, CompanyName char (30) NULL, JournalAmount money NULL)

No rows will ever be written to this table. Its only function is to provide an empty row to serve as a staging area for the SQL statement parameters.

Specifying the Source Rowset

The **AccountUpdate** table is not ready to serve as a source table:

- An **UpdateCode** of "New" triggers an UPDATE or an INSERT, depending on whether the customer is on file or not. However, customer-on-file status is not present in the **AccountUpdate** table.
- **JournalAmount**, a binding column, is not found in **AccountUpdate**. For purchases, it is equal to **UpdateAmount**, but for payments, it must be calculated by multiplying **UpdateAmount** by -1.

Incorporating the customer-on-file status into the source rowset, by using a SELECT statement, solves the first of these two problems. The new source SQL statement initializes **OnFile** with a subquery:

```
SELECT UpdateID, UpdateCode, CustomerID, CompanyName, Upda
OnFile = (SELECT COUNT(*)FROM Account
WHERE CustomerID = AccountUpdate.CustomerID)
FROM AccountUpdate
```

The second problem is solved with Microsoft® ActiveX® code.

Specifying the ActiveX Transformation

In this example, Data Transformation Services (DTS) transformations perform the following three jobs:

- Correctly choose the query to execute.
- Compute the right value for **JournalAmount**.
- Populate the required binding table columns.

A single ActiveX Script transformation is sufficient to carry out these requirements:

```
Function Main()
DTSDestination("UpdateID") = DTSSource("UpdateID")
```

```
DTSDestination("CustomerID") = DTSSource("CustomerID")
 DTSDestination("CompanyName") = DTSSource("CompanyName
 Select Case Trim(DTSSource("UpdateCode"))
   Case "Purchase"
    DTSDestination("JournalAmount") = DTSSource("UpdateAmo
    Main = DTSTransformstat_InsertQuery
   Case "Payment"
    DTSDestination("JournalAmount") = -1 * DTSSource("Update<sub>1</sub>
    Main = DTSTransformstat InsertQuery
   Case "New"
    If DTSSource("OnFile") = 1 Then
      Main = DTSTransformstat UpdateQuery
     Else
      Main = DTSTransformstat_UserQuery
    End If
 End Select
End Function
```

The script first initializes three binding table columns, and then, in a SELECT CASE statement, fills the remaining parameter and sets the return value. Both the "Purchase" and "Payment" cases result in a newly inserted **AccountJournal** record. The two cases differ only in how the **JournalAmount** parameter is calculated. In the "New" case, the source **OnFile** value is used to determine whether to update an existing customer or insert a new customer. Neither of the two possible queries requires **JournalAmount**, so it is not initialized.

Multiphase Data Pump Functionality

Data Transformation Services (DTS) provides advanced users the capability to add programs that customize the data pump at various phases of its operation. By customizing the operation of the data pump, you can add a wide range of functionality to a package. For example:

- Row-level restartability, or the ability to restart the data pump without having to reload large numbers of rows that were already processed. You can add functions to save processed row data, batches, or partial batches, writing that data back to the source or a status table for later use.
- Individual handling of types of insert or transformation errors. For example, you could add special error handlers to customize problems handling NULL data or constraint violations.
- Customizing data pump initialization or termination steps. For example, on data pump initialization you could write out a schema header to a file prior to writing XML (Extensible Markup Language) data to the file.

Data Pump Process

The multiphase data pump option allows you to access the data pump at several points during its operation and add functionality. When copying a row of data from source to a destination, the data pump follows this basic process:

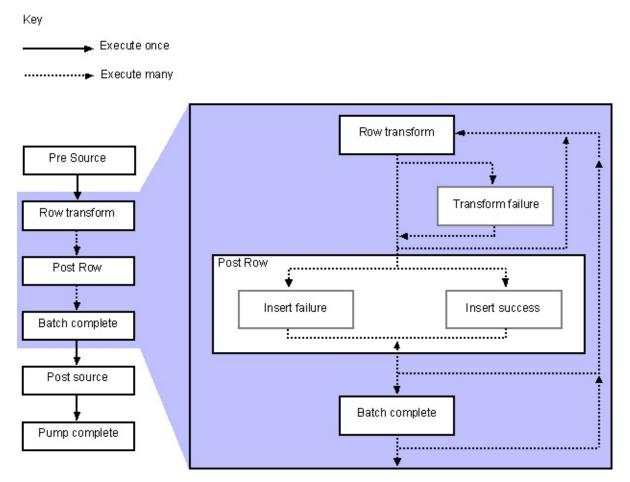
- Fetches a row of source data.
- Optionally applies transformations to the row.
- Attempts to insert the row of data to the destination buffer.

- Processes exceptions.
- Stores the results in a batch, if a batch is specified.
- Repeats the previous steps until the batch is filled, then commits the data stored in the buffer and starts the next batch, or rolls back the batch.

After the data pump processes the last row of data, the task is finished and the data pump operation terminates.

Data Pump Phases

The following figure shows the data pump phases and how they map to the data flow.



In the figure, the data flow (detailed in the expanded, gray area of the figure):

- Originates in the Row Transform phase, where a row of data is copied from the source and any transformations are applied.
- If the transformation is successful, the data for the row moves to the next phase. If the transformation is unsuccessful, a Transform Failure occurs, and the next row of data is fetched.
- Moves to the Post Row phase, where an attempt is made to copy the row to the destination buffer. There are two possible outcomes for this operation: Insert Success or Insert Failure.
- Ends in the Batch Complete phase, where the row data is stored in a batch and eventually inserted or not committed, depending on whether a batch was configured for the task, and the size of the batch.

Three additional data pump phases not directly tied to the processing of row data are shown in the figure: Pre Source, Post Source, and Pump Complete. Each of these phases covers events prior to or after the row-by-row copying of data, transformation, and commit (or rollback) process. For example, the Pre Source phase occurs before the first row of data is fetched. The Post Source phase occurs after the last row of data is processed, and the Pump Complete phase occurs at the end of the transformation task.

Accessing Data Pump Phases

To display the multiphase data pump options in DTS Designer, you must select an option in SQL Server Enterprise Manager. After you have selected the multiphase data pump option, the feature will remain accessible to any packages opened in DTS Designer, for any future sessions. By default, this option is not selected.

To activate the multiphase data pump feature

Tasks that Copy and Manage Data

Data Transformation Services (DTS) includes several tasks designed to copy and manage data and meta data.

Task	Description
<u>Bulk Insert Task</u>	Copies large amounts of data from a text file
	into a Microsoft® SQL Server TM table or view.
Execute SQL Task	Runs SQL statements during package execution
	and saves data that is the result of a query.
Copy SQL Server Objects	Copies or creates SQL Server objects such as
<u>Task</u>	tables, views, indexes, and constraints from one
	instance of SQL Server to another.
<u>Transfer Database</u>	Copies entire databases, jobs, error messages,
<u>Objects Tasks</u>	logins, and master database stored procedures.

Bulk Insert Task

The Bulk Insert task provides the quickest way to copy large amounts of data into a Microsoft® SQL Server[™] table or view. To ensure high-speed data movement, transformations cannot be performed on the data while it is moved from the source file to the table or view.

For example, suppose your company keeps your million-row product list on a mainframe system. Your e-commerce system uses SQL Server 2000 to populate Web pages. You need to update the product table nightly with the master product list from the mainframe. To do this, you save the product list in a tab-delimited format and use the Bulk Insert task to copy the data directly into the SQL Server table. The table is now refreshed with the updated product data.

Configuring the Bulk Insert Task

The Bulk Insert task encapsulates a Transact-SQL BULK INSERT statement that is run during task execution. In Data Transformation Services (DTS) Designer, you can set parameters for the BULK INSERT statement in the **Bulk Insert Properties** dialog box. Any BULK INSERT parameters not available graphically are set to their defaults. If you need to set parameters that are not available graphically, you can use the **BulkInsertTask** object or the **bcp** utility to set them programmatically.

Using the Bulk Insert Task with Transactions

If a batch size is not set, then an entire bulk copy operation is considered one transaction. If a batch size is set, then each batch constitutes a transaction that is committed when the batch finishes.

The behavior of the Bulk Insert task depends on whether the task is joined into the package transaction. If the Bulk Insert task does not join the package transaction, each error-free batch is committed, as a unit, before the next batch is attempted. If the Bulk Insert task joins the package transaction, error-free batches remain in the transaction at the conclusion of the task. These batches are subject to the commit or rollback operation of the step or package. A failure in the Bulk Insert task does not automatically roll back successfully loaded batches; task success does not automatically commit them. Commit and rollback operations happen only in response to package and workflow property settings. For more information, see <u>DTS Transaction Fundamentals</u>.

Usage Considerations

Before using the Bulk Insert task, consider the following:

- The Bulk Insert task can transfer data only from a text file into a SQL Server table or view. To transfer data from a database management system (DBMS), you need to export the data from the source program to a data file and then import the data from the data file into a SQL Server table or view.
- The data destination must be a table or view created by SQL Server. If the destination table or view contains data already, the new data will be appended when the Bulk Insert task runs.
- You can use a format file in the Bulk Insert task object. If you have a format file created by the **bcp** utility, you can specify its path in the Bulk Insert task. The path given must be with respect to the server. For more information about format files, see <u>Using Format Files</u>.

Specifying the Source and Destination

When specifying the path of the text source file, consider the following:

- When running the Bulk Insert task, only members of the **sysadmin** fixed server role can execute the package.
- Regardless of the location of the file, the server must have permissions to both the file and the destination database.
- The server will be running the Bulk Insert task. Therefore, the path given must be with respect to the server.

Optimizing Performance

To optimize performance, consider the following:

- If the text file is located on the same computer running the instance of SQL Server, the copy operation occurs at an even faster rate because the data is not moved across the network.
- The Bulk Insert task does not log error-causing rows. If you need to capture this information, another task, the Transform Data task, can capture error-causing rows to an exception file. However, writing errors to the log will slow down the data transfer. If speed is your priority, use the Bulk Insert task. If capturing errors is more important, use the Transform Data task.

To add the Bulk Insert task to a DTS package

Execute SQL Task

With the Execute SQL task, you can run SQL statements during package execution. The Execute SQL task also can save data that is the result of a query. Using the Execute SQL task, you can:

- Populate multiple global variables.
- Save the complete rowset returned from the query into one global variable.
- Drop a table.
- Re-create fact and dimension tables before loading them.
- Run stored procedures.

The task can contain either a single SQL statement or multiple SQL statements that execute sequentially. SQL statements can range from being a SELECT command to running a stored procedure.

The SQL statements must be written in the dialect of the source database management system (DBMS).

Sending Multiple Statements in a Batch for Execution

If multiple statements are contained in the task, they can be grouped and executed a batch at a time. To signal the end of a batch, use the GO command. All the SQL statements from one GO command to the next are sent in a batch to the OLE DB provider for execution.

Note There are restrictions on the kinds of SQL statements that can be grouped together in a batch. For more information, see <u>Batches</u>.

For example, suppose you have three tables: a table containing customer orders;

a table containing a daily order summary; and a table of year-to-date orders. After the customer order table is updated, you can use the Execute SQL task to run two stored procedures, one to create the new daily sales summary and the other to update the year-to-date order summary. The following code example shows you how to execute the two stored procedures:

```
Execute sp_UpdateDailySales
GO
Execute sp_UpdateYTDSales
GO
```

Running Parameterized Queries

The Execute SQL task can use global variables to populate input parameters in SQL commands, including queries and stored procedures when the source data provider supports parameters. You can write a parameterized query where the value in the SQL statement is filled in at run time by using a question mark as a parameter placeholder. Then, you can map a global variable to the parameter placeholder to specify which global variable will be used at run time in place of the question mark.

To execute a stored procedure with an input parameter

Copy SQL Server Objects Task

With the Copy SQL Server Objects task, you can copy Microsoft® SQL ServerTM objects from one instance of SQL Server to another. You can transfer objects such as data and tables, as well as the definitions of objects such as views and stored procedures. Additional objects include referential integrity constraints and indexes.

If you select a table, the Copy SQL Server Objects task will automatically copy any associated table and views that have a foreign key constraint on the selected table. For example, if you transfer the **employee** table from the **pubs** database, the **jobs** and **publishers** tables also will be transferred. The **jobs** table will be transferred because of the foreign key relationship between the **employee** table and the **jobs** table on the **jobs_id** field. The **publishers** table will be transferred because of the foreign key relationship between the **employee** table and the **jobs** table on the **jobs_id** field. The **publishers** table will be transferred because of the foreign key relationship between the **employee** table and the **publishers** table on the **pub_id** field.

If either the destination table or the view exists and contains data, you can specify whether to overwrite or append the incoming data or drop and re-create the table.

You can transfer objects only from:

- One instance of SQL Server version 7.0 to another.
- An instance of SQL Server 7.0 to an instance of SQL Server 2000.
- One instance of SQL Server 2000 to another.

When configuring the Copy SQL Server Objects task, you are not required to create separate source and destination connections. You set source and destination properties in the **Copy SQL Server Objects properties** dialog box.

Note DTS packages containing Copy SQL Server Objects tasks, where the source and destination are both SQL Server 7.0 databases, may fail if the source server is upgraded to SQL Server 2000 and the package is executed. If the source

server was upgraded, you need to refresh each Copy SQL Server Objects task by opening the package in SQL Server 2000, opening each Copy SQL Server Objects task with the above configuration, and clicking the **Destination** tab and then clicking **OK** for each of the tasks.

Transfer Database Objects Tasks

Usually, you use the Copy SQL Server Objects task to copy database objects from one database to another. However, you can use Transfer Database Objects tasks to copy server-wide information not necessarily found in individual databases. These tasks cover situations not addressed by the Copy SQL Server Objects task.

Note The Transfer Database Objects tasks are custom tasks used by the Copy Database Wizard. For more information, see <u>Using the Copy Database Wizard</u>.

Transfer Database Task

Use this task to move or copy a Microsoft® SQL Server[™] database from an instance of SQL Server version 7.0 or SQL Server 2000 to an instance of SQL Server 2000. When configuring this task, specify each database, indicate whether each database is to be moved or copied, and specify the location of the resulting data and log files.

Transfer Error Messages Task

Use this task to copy user-specified error messages, created by the **sp_addmessage** system stored procedure, from an instance of SQL Server 7.0 or SQL Server 2000 to an instance of SQL Server 2000. When configuring this task, specify the source and destination servers and the error messages to be transferred.

Transfer Logins Task

Use this task to copy logins from an instance of SQL Server 7.0 or SQL Server 2000 to an instance of SQL Server 2000. When configuring this task, specify the source and destination servers, the database to be moved or copied, and the logins to be transferred.

Transfer Jobs Task

Use this task to copy jobs from an instance of SQL Server 7.0 or SQL Server 2000 to an instance of SQL Server 2000. When configuring this task, specify the source and destination servers and the jobs to be copied.

Transfer Master Stored Procedures Task

Use this task to copy stored procedures from a **master** database on an instance of SQL Server 7.0 or SQL Server 2000 to the **master** database on an instance of SQL Server 2000. When configuring this task, specify the source and destination servers and the stored procedures to be copied.

Note If you unregister a Transfer Database Objects task and want to re-register the task, do the following: from the **Task** menu, click **Register Custom Task**, click the browse (...) button, and then open Cdwtasks.dll.

See Also

Copy SQL Server Objects Task

<u>sp_addmessage</u>

Tasks That Function as Jobs

Data Transformation Services (DTS) includes a number of tasks that function as jobs, performing operations external to the packages containing the tasks.

Task	Description
<u>ActiveX Script Task</u>	Uses scripting code to perform functions that are not available in the other tasks in DTS Designer.
<u>Dynamic Properties Task</u>	Retrieves values from sources outside a DTS package at package run time and assigns those values to selected package properties.
<u>Execute Package Task</u>	Runs other DTS packages as part of a workflow.
<u>Execute Process Task</u>	Runs an executable program or batch file as part of a package.
<u>File Transfer Protocol Task</u>	Downloads data files from a remote server or an Internet location as part of a package workflow.
<u>Message Queue Task</u>	Uses Message Queuing to send and receive messages between DTS packages.
Send Mail Task	Sends an e-mail message as a task.

ActiveX Script Task

With the Microsoft® ActiveX® Script task, you can write code to perform functions that are not available in the other tasks in Data Transformation Services (DTS) Designer. For example:

- As a package executes, you can replace a two-digit state code in the source data with the legal abbreviation of the state in the destination data.
- If the destination data is a table from which mailing labels are created, you can set the "title" column to "Mr.", "Mrs.", or a default of spaces.
- You can validate important columns in the source data and skip records that contain invalid data to prevent them from being copied to the destination.

For more information, see <u>Using ActiveX Scripts in DTS</u>.

To execute a ActiveX script, you must have the scripting language library installed on the computers that will be running the package (for example, both the development and production computers). The languages that can be used to write your script include Microsoft Visual Basic® Scripting Edition (VBScript) and Microsoft JScript®.

IMPORTANT An ActiveX script can affect the execution speed of a DTS package as it executes on each row of the source data. Therefore, if performance is a priority, use scripting carefully when building a package. For more information, see <u>Enhancing Performance of DTS Packages</u>.

Dynamic Properties Task

The Dynamic Properties task works by retrieving values from sources outside a Data Transformation Services (DTS) package at package run time and assigning those values to selected package properties. Typically, the external values assigned by the task are unknown until package run time. Therefore, a package may need to:

- Get data from a backup file, where the file name changes according to the date.
- Connect to an available server, where the name of the server is not known until package run time.
- Run a scheduled query on a set of data for a particular date range. The package must update the date range for each execution and change the source SQL statement accordingly.

Available Source Types

The Dynamic Properties task can assign external data or information to a package property from one of the following sources:

- An initialization (*.ini) file, such as Win.ini, or any initialization file that you want to create. This selection only supports property values a single line in length.
- A data file containing a property value that can be read and assigned. Unlike the initialization file selection, the data file selection supports property values greater than one line in length.
- A query. When you assign the results of a query to a DTS package property, the Dynamic Properties task uses only the results of the first

column of the first row. For this reason, consider designing your queries so they generate a single result (for example, a COUNT, SUM or SELECT statement for a particular name).

- A DTS package global variable. For more information, see <u>Using</u> <u>Global Variables with DTS Packages</u>.
- An environment variable, which can encompass any available user or system variable (for example, COMPUTERNAME, LOGONSERVER, and so on).
- A constant, commonly used to assign a default value to a property in the event a previous assignment fails.

Execute Package Task

The Execute Package task extends the enterprise capabilities of Data Transformation Services (DTS) by letting you run other DTS packages as part of a workflow. This capability is useful when:

- Package workflow is complex and can be broken down into two or more modular packages. For example, if you are loading a star schema, you can build a set of packages so that different packages are associated with filling each dimension table and the fact table. Such a strategy increases package readability and simplifies debugging because the individual packages are smaller and each package workflow is more focused. It also provides a higher level of security because authors may not require access to all packages.
- Units of work can be encapsulated into separate packages and joined as transactional components to the workflow of a master package. The master package runs the accessory packages and, based on the success or failure of the accessory packages, either commits or rolls back the transaction.
- Parts of a package workflow can be reused by other packages. For example, you can build a data extraction module that can be called from different packages. Each of the packages calling the extraction module performs different data scrubbing, filtering, or aggregation operations.

Global Variables and the Execute Package Task

You can use the Execute Package task to dynamically assign the values of global variables from a parent package to a child package. The child can process the global variable data passed to it in any Microsoft® ActiveX® script in its executable workflow.

Using global variables to pass information from one package to another is useful

when parts of a larger workflow are assigned to different packages. For example, one package could download data on a nightly basis, summarize the data, assign summary data values to global variables, and pass the values to another package for further processing of the data.

Global variable values passed to the child package are handled according to their scope definitions described in the child package. Global variables defined as static in the child package retain their last assigned values, and global variables not defined as static return to their initial values after the script runs.

Transactions and the Execute Package Task

Joining Execute Package tasks to package transactions lets you join the execution of entire packages and other workflow steps into units of work that can be committed or rolled back. Before joining an Execute Package task to a transaction, consider the following:

- Only one transaction can be run at a time from a package.
- The Microsoft Distributed Transaction Coordinator (MS DTC) must be operational and running on the server in order for transactions to work. Also, the MS DTC client must be running on each computer on which a package runs for transaction joining to work.

The transactional context within which an Execute Package task runs can range from the execution context of the entire master package to specific parts of the workflow, and to the Execute Package task itself, if the task is not joined to any transactions.

When joining the Execute Package task to a transaction in DTS Designer, use:

- The **Advanced** tab in the **DTS Package Properties** dialog box to enable the use of transactions in a package and set general transaction characteristics, such as isolation level.
- The **Options** tab in the **Workflow Properties** dialog box to join the step associated with the **Execute Package** task to a transaction.

• The **Insert Commit Size** check box in the Advanced tab of the **Data Transformation Properties** dialog box to control the number of rows of data moved prior to committing a transaction. This option is available only when the **Use fast load** check box is selected).

CAUTION Creating workflows or ActiveX scripts in which an Execute Package task is used to call its own package can generate a stack overflow and cause Microsoft Management Console (MMC) to shut down. Generally, recursive operations are not recommended with the Execute Package task. If you need to use the Execute Package task recursively, make sure you set a global variable in an ActiveX script to monitor the nesting depth (for example, in a step ActiveX script or in an ActiveX Script task that determines precedence), and terminate the recursion in the script code before a specified depth is reached. The global variable must be passed to the subpackage through its **GlobalVariable** collection.

Execute Process Task

With the Execute Process task, you can run an executable program or batch file as part of a Data Transformation Services (DTS) package. Although you can use the Execute Process task to open any standard application such as Microsoft® Excel or Microsoft Word, more often you use it to run business applications or batch files that work against a data source.

For example, you can use the Execute Process task to run a custom Microsoft Visual Basic® application that generates a daily sales total report. Other steps or tasks in the package can use the sales total report. Then, you can attach the report to a Send Mail task and forward the report to a management distribution list.

File Transfer Protocol Task

The File Transfer Protocol (FTP) task lets you download data files from a remote server or an Internet location as part of a Data Transformation Services (DTS) package workflow. By including an FTP task, your DTS package can:

- Copy directories and data files from one server directory location to another, before or after performing data movement and transformations.
- Log in to a source FTP location and copy files or packages to a destination directory. The transformed data or package becomes available on an FTP site for download over the Internet, as part of the process of shipping the information to a data warehouse.
- Schedule the download of different files on a regular basis. By using the FTP task in conjunction with a Dynamic Properties task, you can change the names of files to be downloaded, according to properties you specify (for example, a date or the value of a run-time global or environment variable).

Note If you experience a problem using the FTP task, use ftp.exe in a command prompt to help identify the problem, as both the FTP task and ftp.exe use the same connection method.

Message Queue Task

The Message Queue task allows you to use Message Queuing to send and receive messages between Data Transformation Services (DTS) packages. These messages can take the form of simple text, files, or global variables and their values. Messages can be sent when the destination is unavailable or busy, for example, from or to the laptop of a sales representative. The Message Queue task therefore offers you the ability to asynchronously coordinate operations throughout your enterprise. You can:

- Delay task execution until other packages have checked in. For example, at each of your retail sites, after nightly maintenance, a message queue task sends a message to your corporate computer. A package running on this computer contains message queue tasks, each waiting for the message from one retail site. When the message arrives, a task is triggered to upload site data. Only after all sites have checked in does the package proceed to compute summary totals.
- Send data files to the computer responsible for processing them. For example, restaurant cash register output can be sent in a data file message to the corporate payroll system, for waiter tip data extraction.
- Distribute files throughout your enterprise. For example, one package can use a Message Queue task to send a package file to another computer. A package running on the destination computer can then use a message queue task to retrieve and save the package locally.
- Split a large job into component parts and parcel them out to several computers, using string or global variable messages to coordinate operations. For example, you can send a global variables message containing the name of a package. On another computer a package uses a Message Queue task, a Dynamic Properties task, and an Execute Package task to execute the requested package.

Transactions and the Message Queue Task

In Microsoft® SQL Server[™] 2000, a Message Queue task cannot take part in the package transaction.

There are two types of message queues: transactional and non-transactional. The transactional queue gives you the assurance that each message is delivered exactly once. This feature makes the transactional queue more likely to satisfy current and long-term requirements than the non-transactional queue when you use the Message Queue task. The transactional status of a queue is set at the time the table is created and cannot be changed.

Installing and Configuring Message Queuing

To use the Message Queue task, you must:

- Install Message Queuing server software on your network.
- Install Message Queuing client software on your computer.
- Configure a queue for your messages.

Two versions of Message Queuing are supported:

- MSMQ. This version is provided in the Microsoft® Windows NT® 4.0 Option Pack.
- Message Queuing for Windows® 2000.

You can install Message Queuing server software on a Windows 2000 domain controller running Active Directory[™], the directory service included in Windows 2000, as well as on queues on a local computer.

Message Queuing client software cannot be installed unless Message Queuing server software is installed on a domain controller in your organization. Message Queuing client software is not installed as part of Windows 2000 Setup.

For more information about Message Queuing, see the Windows 2000 Server documentation.

Note Installation and configuration of MSMQ and Message Queuing differ significantly.

Message Types

A Message Queue task sends and receives the following types of messages:

- String message
- Data file message
- Global variables message

String Messages

A string message contains a text string. To send a string message, you must specify the message text.

When you configure a Message Queue task to receive a string message, you can specify filtering criteria. You can accept any string message, or you can type a compare string and proceed only if a queued message:

- Matches the compare string exactly.
- Matches the compare string (ignoring case).
- Contains the compare string.

Data File Messages

A data file message contains a Data Transformation Services (DTS) package ID, a version ID, and the name and contents of a data file. To send a data file message, you must specify the path of the file to be sent.

When you configure a Message Queue task to receive a data file message, you must answer the following questions:

• Is the file to be saved under its original name or with a name you

```
specify?
```

- Do you want the Message Queue task to overwrite an existing file?
- Will you accept files from any source or limit them to only those sent from a particular package or package version?

Global Variables Messages

A global variables message contains a DTS package ID, a version ID, and the name, type, and value of one or more variables. To send a global variables message, you must add the variables to your package, and then specify them when you configure the Message Queue task.

When you configure a Message Queue task to receive a global variables message, you can specify filtering criteria. You can accept any global variables message, or you can proceed only if a queued message comes from a particular package or package version.

Sending Messages with the Message Queue Task

Use the Message Queue task to place one or more messages on a queue for later delivery to DTS packages running on this or other computers.

The Message Queue task will fail at run time if:

- The queue name is entered incorrectly or the named queue is not on your network.
- A data file message references an unavailable or nonexistent data file.
- A message contains more than 4 megabytes (MB).

Data files and variables must use slightly less than 4 megabytes in order to accommodate any included package or version IDs, file names, or variable names and types.

IMPORTANT A Message Queue task is not allowed to take part in the package transaction. As a result, your package will fail at run time if, in the **Workflow Properties** dialog box for a Message Queue task, you select the **Join transaction if present** check box.

To send a message with the Message Queue Task

Receiving Messages with the Message Queue Task

Use the Message Queue task to retrieve and process a single message after it has been delivered to a queue.

When executed, the Message Queue task scans the queue for the first message that meets the specified filter criteria. If such a message is found, the Message Queue task terminates successfully after processing the message. If no such message is found, the task waits for the first of the following events:

- An acceptable message arrives.
- An optionally specified timeout interval expires.
- Package execution is canceled manually.

If the wait yields an acceptable message, the Message Queue task terminates successfully after processing the message. If not, it fails.

Messages can be removed from the queue on receipt.

IMPORTANT If a message, once read, is not removed from the queue, the next time the task executes, the same message will be returned. Any other acceptable messages on the queue remain inaccessible until the first message is removed. If you do not remove a message on receipt, then you must elsewhere create another Message Queue task that does.

IMPORTANT A Message Queue task is not allowed to take part in the package transaction. As a result, your package will fail at run time if, in the **Workflow Properties** dialog box for a Message Queue task, you select the **Join transaction if present** check box.

Receiving String Messages

Use string messages to respond to notifications of external events. You can proceed on receipt of any string message, or you can use filter criteria to narrow the range of acceptable messages.

To receive a string message with the Message Queue Task

Receiving Data File Messages

Use data file messages to transfer data files from computer to computer. A data file message contains a Data Transformation Services (DTS) package ID, a version ID, and the name and contents of a data file.

On receipt of an acceptable message, the transmitted data file is saved and the Message Queue task terminates successfully. If no acceptable message is initially on the queue, the Message Queue task waits until one arrives. If the specified timeout interval expires first, the task fails.

You can save the transmitted data file to a file or a directory location. If the save path specifies an existing file, you need to overwrite the file or the step fails.

IMPORTANT Any change in the package sending the data file message will result in a new version ID. If you click **Filter by version** on the **Message Queue Task Properties** dialog box, messages from the modified package are ignored.

To receive a data file message with the Message Queue Task

Receiving Global Variables Messages

Use global variables messages to transfer variables and their values from one package to another. A global variables message contains the name, type, and value of zero or more variables.

On successful receipt, transmitted variables, with their values, are added to the DTSGLobalVariables collection and the Message Queue task terminates successfully. Previously existing variables with the same name are replaced. You can access newly transmitted variables from Microsoft® Visual Basic® Scripting Edition (VBScript) with DTSGlobalVariables("variable name").

If no acceptable message is initially on the queue, the Message Queue task either waits until one arrives, or fails after the specified timeout interval has passed.

IMPORTANT Any change in the package sending the global variables message will result in a new version ID. If you click **Filter by version** in the **Message Queue Task Properties** dialog box, messages from the modified package are ignored.

To receive a global variables message with the Message Queue task

Message Queue Task Examples

The following package examples illustrate the use of the Message Queue task with each message type:

- The Prepare Facts and Update Warehouse packages use string messages to coordinate operations on two computers.
- The Add New Employees package uses global variable messages to retrieve new employee information from a queue and save it in a database.
- The Load Expenses package uses data file messages to take Microsoft® Excel worksheets from a queue, extract expense data, and save it in a database.

Using String Messages to Trigger Tasks

You can use string messages to trigger a task on another computer or to wait for a signal from that computer before proceeding.

For example, you have an online transaction processing system running on one computer (OLTP). Your data warehouse resides on another computer (Warehouse). Every night, shipment facts are summarized on OLTP and transferred to Warehouse. Before the transfer can take place though, Warehouse dimension tables must be updated.

Two packages are used to manage this process: Update Warehouse and Prepare Facts.

Update Warehouse runs on the Warehouse computer. When Update Warehouse starts, the Ask for Facts task sends a string message telling the OLTP computer to start summarizing shipment data. While shipment data is being summarized on the OLTP computer, the Update Dimensions task updates Warehouse dimension data. When this task is complete, the Wait for Facts task waits for a string message from the OLTP computer that says shipment data is ready. Only after this message is received does a Transform Data task move the shipment data to the Warehouse computer.

Prepare Facts runs on the OLTP computer. The Wait For Trigger task initially waits for the string message from the Ask for Facts task. On receipt, the Shipment Summary task prepares the data for transfer. When the data is ready, the Alert Warehouse task sends the message which, when received by the Wait For Facts task, tells Update Warehouse to start the transfer.

Ask For Facts and Wait For Trigger form a matched pair of Message Queue tasks. Ask For Facts sends the message and Wait For Trigger receives it. Alert Warehouse and Wait For Facts form another such matched pair. Their configuration differs from the first only in the text of the message.

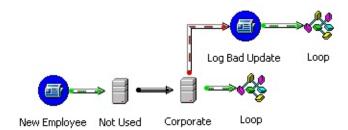
Configuring the Ask For Facts Task

Ask For Facts sends the message "Summarize shipments" to a queue, where it can later be read by the Wait For Trigger task.

To configure the Ask For Facts task

Using Global Variable Messages to Queue Database Updates

The Add New Employees package uses global variable messages to take an employee ID and name from a queue and add the employee to the corporate database. It continues to add employees until the queue is emptied, at which point it stops.



Follow these instructions to create and run this package.

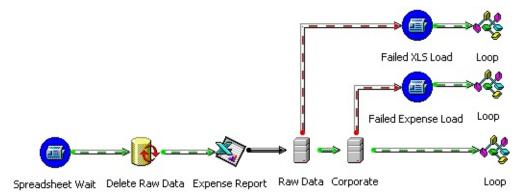
Creating and Configuring the Add New Employees Package

The new package must be created and named. In addition, package properties must be set so that a single failure does not prevent the rest of the new employees from being added.

To create and configure the Add New Employees package

Using Data File Messages to Collect Data

The Load Expenses package uses a data file message to take a spreadsheet from a queue, extract expense data, and load it into a database.



Your traveling sales force uses a standard Microsoft® Excel template to enter expenses on their laptops. Every week, a Data Transformation Services (DTS) package on the laptop sends the completed worksheet in a data file message addressed to a queue at corporate. When the laptop synchs with the network, the message is delivered. The Load Expenses package processes these messages on arrival.

Follow these steps to create this package on your computer.

Creating and Configuring the Load Expenses Package

The new package must be created and named. In addition, package properties must be set so that a single failure does not shut down the service.

To create and configure the Load Expenses package

Send Mail Task

With the Send Mail task, you can send an e-mail message as a task. For example, if you want to notify a database administrator about the success or failure of a backup operation, you can link a Send Mail task to the preceding backup task. To use a Send Mail task, you need to install MAPI with a valid user profile on the instance of Microsoft® SQL ServerTM you are running.

A Send Mail task can include attached data files. You can point to a location for an attached file and send a dynamically updated file, rather than a static copy of the file fixed when you create the task. This feature is useful for sending attachments such as log files and exception files.

Note If an attachment file does not exist when the package is run, you will receive the message: "Error sending mail: Internal MAPI error: the address book has no directories that contain names." This message indicates that either the file is not available at the specified location or that access permissions are not granted for the file.

To add a DTS task to a DTS package

DTS Transformations

This section describes the types of column-level transformations available with Data Transformation Services (DTS) and explains how they work.

Before you use a DTS transformation, you need to know:

- How to map a column transformation. For more information, see <u>Mapping Column Transformations</u>.
- How to use the transformation type you want. For more information, see <u>Transformation Types</u>.

DTS also allows you to write custom transformations written as COM objects with DTS Designer. For more information, see <u>Building a DTS Custom</u> <u>Transformation</u>.

Mapping Column Transformations

In Data Transformation Services (DTS), you can transfer data from a source rowset to a destination table by using one of the available DTS transformation types or by supplying a custom transformation. For some transformation types, the data is copied. In other cases, data is modified as transformations are applied or copied according to the rules of the transformation type. You can perform a column transformation in the following ways:

• Graphically, in DTS Designer, with the Transform Data task and the Data Driven Query task. Each Transform Data task or Data Driven Query task you add to a package contains one or more column transformations.

To configure the transformations, use the **Transformations** tab of the **Transform Data Task Properties** and **Data Driven Query Properties** dialog boxes. This tab graphically displays all the column mappings between the source and destination that are used in the task. You use this tab as a starting point for configuring the relationships between source and destination columns and the specific transformations to use.

• By using the default Copy Column transformation or by writing a Microsoft® ActiveX® script in the DTS Import/Export Wizard. The DTS Import/Export Wizard is limited to these two types of transformation. For more information, see <u>Using ActiveX Scripts in DTS</u>.

To modify a transformation script written and saved to a package created in the DTS Import/Export Wizard, you need to open the package in DTS Designer and edit the task associated with the script.

• Programmatically, using a Transform Data task, a Data Driven Query task, or a Parallel Data Pump task.

For more information, see <u>DTS Transformations in Visual Basic</u> and <u>DTS Column Objects in Visual Basic</u>.

Mapping a Transformation in DTS Designer

You map a transformation in DTS Designer to establish the relationship between the source and destination columns. Mapping configurations can be of several types:

- One-to-one mappings, which contain a single source column and a single destination column.
- N-to-N mappings, which contain an equal number of multiple source and destination columns.
- Mappings with unequal numbers of source and destination columns.

DTS allows you to create your own custom transformations, with their own column requirements, by programming objects that implement the **IDTSDataPumpTransform** interface. The mappings for these transformations can fall in one of the above categories or can have different requirements.

For more information, see <u>Building a DTS Custom Transformation</u> and <u>IDTSDataPumpTransform (DTS)</u>.

One-to-One Column Mappings

You use one-to-one column mappings when the transformation requires one source and one destination column. By default, DTS Designer maps each source and destination column in a Transform Data task in this configuration, attempting to match each source and destination column by name (for example, **CategoryName** in the source would be mapped to **CategoryName** in the destination, and so forth). These are the types of transformations you use with one-to-one column mappings:

- Copy Column transformation
- Trim String transformation
- Date Time String transformation

- Middle of String transformation
- Read File transformation
- ActiveX Script transformation

Transformations that use one-to-one column mappings are displayed with a single connecting arrow indicating the flow of data. The following diagram, from the **Transformations** tab of the **Transform Data Task Properties** dialog box, shows four such transformations (the bold arrow indicates that one of the transformations is selected).

N-to-N Column Mappings

Transformations using N-to-N column mappings require a matching number of multiple source and destination columns. You use this mapping in situations where each source column must have a corresponding destination column, and it is more efficient to configure all the transformations together (as a single data pump operation) rather than as separate transformations called individually for each row. By default, the Data Driven Query task uses this type of mapping configuration.

N-to-N column mappings include the following types of transformations:

- Copy Column transformation
- Uppercase String transformation
- Lowercase String transformation
- ActiveX Script transformation

N-to-N column mappings are shown with a single arrow connecting an equal number of branches at each end. The following diagram shows a mapping for

this transformation that connects four source and four destination columns:

If you edit a Copy Column transformation so that the same source column is copied to multiple destination columns, the mapping will change to indicate the data flow, and the number of mapping lines touching the source and destination tables will be unequal. This type of mapping indicates a single source column is being copied to multiple destination columns.

Note A single many-to-many Copy Column transformation is faster then many one-to-one Copy Column transformations. For more information, see <u>Enhancing</u> <u>Performance of DTS Packages</u>.

Mappings with Unequal Numbers of Source and Destination Columns

A transformation mapping can include an unequal number of source and destination columns. For example:

- More source columns than destination columns
- More destination columns than source columns
- No source or destination columns

Following are several examples of these types of mappings.

One or More Source Columns and No Destination Columns

You can have an ActiveX Script transformation where only the values from a source table are processed. In the following example, written in Microsoft Visual Basic® Scripting Edition (VBScript), each row of the **CategoryName** column (from the **Categories** table of the **Northwind** sample database) is checked for the presence of a NULL value or a null string. If neither of those values is found for the row, a package global variable is assigned the value of **CategoryName**.

Following is the sample ActiveX transformation script for this type of mapping:

```
Function Main()
If Not IsNull (DTSSource("CategoryName")) Then
If LEN(DTSSource("CategoryName")) > 0 Then
DTSGlobalVariables("gv2") = DTSSource("CategoryName")
End If
End If
Main = DTSTransformStat_OK
End Function
```

Although this example uses only one source column, you also can create ActiveX Script transformations using multiple source columns and no destination columns.

Two Source Columns and No Destination Column

The Write File column transformation is an example of a transformation with a specialized mapping requirement. The transformation takes data from one source column and writes it to a file, the name of which it finds in a second source column. As a result, it requires two source columns (one column containing the data to be copied and a second column containing a list of file names) and zero destination columns.

In the following diagram, Write File column transformations originate from the source table and do not touch any destination columns.

One or More Destination Columns and No Source Columns

You can have an ActiveX Script transformation where only the values from a destination table are processed. In the example below, an incrementing counter value is appended to the value of two global variables, and the concatenated strings are assigned to the **CategoryName** and **Description** columns of the **Categories** table.

Following is the sample ActiveX transformation script for this type of mapping:

Dim N

Function Main()

```
If IsEmpty(N) Then
N = 0
End If
```

```
DTSDestination("CategoryName") = DTSGlobalVariables("gv1") &
DTSDestination("Description") = DTSGlobalVariables("gv2") & (N)
N = N + 1
Main = DTSTransformStat_OK
```

End Function

No Destination Columns and No Source Columns

There may be cases where the ActiveX Script transformation does not reference any source or destination columns. For example, the script may only involve the processing of global variables or lookup queries, or an action such as a notification.

Following is an example of a simple ActiveX transformation script for this type of mapping. In the following script, the value of a global variable is incremented for each row of data in the source:

Dim counter

```
Function Main()
Counter = counter + 1
DTSGlobalVariables("gv1").Value = DTSGlobalVariables("gv1") + 1
Main = DTSTransformStat_SkipInsert
End Function
```

The following script could be used to skip further inserts after the required items have been loaded in a transformation:

```
Function Main()

If DTSGlobalVariables("LoadComplete").Value = True Then

Main = DTSTransformStat_SkipRow

Else

Main = DTSTransformStat_OK

End if

End Function
```

See Also

Data Driven Query Task

ParallelDataPump Task Object

Transform Data Task

Using Global Variables With DTS Packages

Transformation Types

This section describes the individual column-level transformations available in Data Transformation Services (DTS).

Торіс	Description
Copy Column Transformation	Describes the transformation used to copy source data to the destination.
ActiveX Script Transformation	Explains how to use Microsoft ActiveX® scripts to define column-level transformations.
<u>Date Time String</u> <u>Transformation</u>	Describes the transformation used to convert a source date into a new destination format.
<u>Uppercase String</u> <u>Transformation</u>	Describes the transformation used to convert a string into uppercase characters.
Lowercase String Transformation	Describes the transformation used to convert a string into lowercase characters.
<u>Middle of String</u> <u>Transformation</u>	Describes the transformation used to extract a substring from a source and optionally change its case or trim white space before placing the result in the destination.
Trim String Transformation	Describes the transformation used to remove leading, trailing, or embedded white space from a source string and place the (optionally case-shifted) result in the destination.
Read File Transformation	Describes the transformation used to copy the contents of a file specified by a source column to a destination column.
Write File Transformation	Describes the transformation that creates a new data file for each file named in a source column and initializes the contents of each file from data in a second source

corumi.	column.
---------	---------

Copy Column Transformation

A Copy Column transformation copies data directly from source to destination columns, without Microsoft® ActiveX® scripts or any other transformations applied to the data. When copying data in this manner, you should consider the following:

- Specify columns. You can specify a Copy Column transformation from a single source column to a single destination column, or from multiple source columns to multiple destinations columns.
- Make conversions. You can use transformation flags to enforce the stringency with which data type conversions are allowed. These transformation flags are available when creating or editing a transformation in Data Transformation Services (DTS) Designer, or programmatically. By default, DTS allows all possible conversions between source and destination columns.

For more information, see <u>DTSTransformFlags</u>.

- Truncation issues. The copy column transformation truncates text without error or notification. Although you can set transformation flags to prevent the possibility of truncation, there is no way to have the transformation fail on the first row for which truncation is necessary.
- Change the columns in the copy. When using the Copy Column transformation to copy multiple source and destination columns, it is possible to copy the same column to multiple destination columns.

See Also

DataPumpTransformCopy Object (DTS)

ActiveX Script Transformation

You can use a Microsoft® ActiveX® script to modify data as it is moved from its source to its destination. You can code a transformation between one or more source and destination columns.

You can write ActiveX transformation scripts in several places in the Data Transformation Services (DTS) user interface. However, an ActiveX Script transformation applies to tasks that work on source data on a row-by-row basis. Use it with:

- The DTS Import/Export Wizard.
- A Data Driven Query task.
- A Transform Data task.

For these items, the transformation scripts are executed for each row of data coming in from the source.

Note Because ActiveX Script transformations are executed for each row of data, place the code for opening and closing connections in such a way that a new connection is not opened and closed each time a row of data is processed.

For more information about using ActiveX scripts or code examples, see <u>Using</u> <u>ActiveX Scripts in DTS</u>.

See Also

ActiveX Script Task

Transform Data Task

Data Driven Query Task

Date Time String Transformation

The Date Time String transformation converts a date or time in a source column to a different format in the destination column.

The transformation is carried out in two steps:

- 1. Each entry is converted in the source column to an OLE DB DBTYPE_TIMESTAMP data type. If the source column is a string type, a source date format string guides the conversion.
- 2. The OLE DB DBTYPE_TIMESTAMP data type is converted to the destination column format. If the destination column is a string type, the destination date format string is used to format the resulting text.

If the source or destination column is not a string type, the corresponding format string is ignored.

The following standard date format strings are available:

- *dd MMM yy*
- *dd MMMM yy HHmm*
- *dd MMMM yy HHmmss.ff*
- *dddd MM/dd/yy hh:mm tt*
- *dddd*, *MMMM dd*,*yyyy hh:mm:ss.ffff tt*
- hh:mmtt

If yours is not among the standard formats, create your own date format string

using any of the following tokens.

Value	Token(s)
Year	уууу, уу
Month	MMMM, MMM, MM, M
Day	dddd, ddd, dd, d
12 hour	hh, h
24 hour	НН, Н
Minute	mm, m
Second	SS, S
Fraction	f (may be repeated)
AM/PM	Tt

Additionally, you may specify:

- Long and short names for months.
- Long and short names for days of the week.
- The language in which dates appear.

Note All conversions use regional settings current on the instance of Microsoft® SQL Server[™] running the package.

If the formatted result is too large to fit in a string type destination column, it is truncated.

The Date Time String transformation fails if:

- The source or destination column is not a string or date type.
- An invalid date format string is specified, for example, "mm/dd/mm."
- A source string does not match the source date format string.

To convert the format of a Date Time String transformation

Lowercase String Transformation

The Lowercase String transformation converts a source column to lowercase characters and, if necessary, to the destination data type. Both the source and destination columns must be a string data type. Multiple source columns may be processed in a single transformation.

If the source data is too large to fit in the destination column, it is truncated. For more information, see <u>Copy Column Transformation</u>.

The Lowercase String transformation is an example of an N-to-N transformation mapping. Any number of source columns may be selected, as long as each is provided a matching destination column. For more information, see <u>Mapping</u> <u>Column Transformations</u>.

To convert a string to lowercase characters

Uppercase String Transformation

The Uppercase String transformation converts a source column to all uppercase characters and, if necessary, to the destination data type. Both the source and destination columns must be a string data type. Multiple source columns may be processed in a single transformation.

If the source data is too large to fit in the destination column, it is truncated.

The Uppercase String transformation is an example of an N-to-N transformation mapping. Any number of source columns can be selected, as long as each is matched to a destination column. For more information, see <u>Mapping Column</u> <u>Transformations</u>.

To convert a string to uppercase characters

Middle of String Transformation

The Middle of String transformation extracts a substring from the source column, transforms it, and copies the result to the destination column.

You specify the substring by providing a start position and a maximum number of characters to include. Consider the following:

- The first character in the string occupies position 1.
- If you specify no maximum number of characters, all characters that occupy positions greater than or equal to the start position are included in the substring.
- If the start position falls beyond the end of the string, an empty string results.

You can further process the resulting substring by using:

• Trimming options, which include deleting leading, trailing, or embedded white spaces.

White space consists of the following characters: tab, line feed, vertical tab, form feed, carriage return, and space (0x09 - 0x0D, 0x20), as well as their Unicode equivalents.

• Case options, which include converting the substring to either uppercase or lowercase characters.

If the final result is too large to fit in the destination column, it is truncated.

The Middle of String transformation fails if the source or destination columns are not of a string data type.

To perform a Middle of String transformation

Trim String Transformation

The Trim String transformation removes leading, trailing, and embedded white space from a string in the source column and copies the result to the destination column.

White space consists of the following characters: tab, line feed, vertical tab, form feed, carriage return, and space (0x09 - 0x0D, 0x20), as well as their Unicode equivalents.

You may optionally convert the trimmed string to either uppercase or lowercase characters.

If the trimmed source data is too large to fit in the destination column, it is truncated.

The Trim String transformation fails if the source or destination columns are not of a string data type.

To perform a Trim String transformation

Read File Transformation

The Read File transformation locates and opens the contents of a file, whose name is specified in a source column, and copies the contents into a destination column.

When you configure the transformation, you define the path containing the files listed in the source column. All the files must be in the same directory path.

Note The directory name can start with a disk drive or a Universal Naming Convention (UNC).

If a file is not found in the path, you can either:

- Null the destination column. Or
- Fail the task.

A Read File transformation fails if:

- The **Error if file not found** check box is selected and the path matches no existing file.
- The source column is not a string type.
- The destination column is not a string or binary type.
- The process does not have permission to read the file.
- Invalid file name characters appear in the source column.
- The read path specifies a location in a nonexistent folder.

Translating File Content

If the destination column is a string type, file contents are interpreted according to their file type (for example, ANSI, OEM, or Unicode). Translation is performed on the instance of Microsoft® SQL Server[™] running the package, using current code pages. If the destination column is of binary or image type, no translation occurs.

To perform a Read File transformation

Write File Transformation

The Write File transformation copies the contents of a source column (data column) to a file whose path is specified by a second source column (file name column).

When you configure the transformation, you define the path containing the files listed in the file name column. All the files must be in the same directory path. If the file is not found in the path, one is created and initialized with the contents of the data column.

Note The directory name can start with a disk drive or a Universal Naming Convention (UNC).

If a file with the same name already exists, the transformation does one of the following:

- Replaces the existing file.
- Appends the contents of the data column to the existing file.
- Fails step execution.

Before configuring a Write File transformation, consider the following:

- A destination connection must be specified, although the content may or may not be copied to a destination.
- If the contents of the data column are Null, the transformation deletes the file specified by the save path. However, if you select the **Append if file exists** check box, the file is not deleted.

The Write File transformation fails if:

• The save path matches an existing file and you do not select the **Overwrite if file exists** check box.

- The value in the **File name column** check box is not a string type.
- The data column is not a string or binary type.
- The process does not have permission to write to the specified directory or overwrite an existing file.
- The contents of the **File name column** check box are NULL or zero-length.
- Invalid file name characters appear in the **File name column** check box.
- The save path specifies a location in a nonexistent folder.

To perform a Write File transformation

DTS Connections

To successfully execute Data Transformation Services (DTS) tasks that copy and transform data, a DTS package must establish valid connections to its source and destination data and to any additional data sources (for example, lookup tables).

Because of its OLE DB architecture, DTS allows connections to data stored in a wide variety of OLE DB-compliant formats. In addition, DTS packages usually can connect to data in custom or nonstandard formats if OLE DB providers are available for those data sources and if you use Microsoft® Data Link files to configure those connections.

DTS allows the following varieties of connections:

• A data source connection.

These are connections to: standard databases such as Microsoft SQL Server[™] 2000, Microsoft Access 2000, Oracle, dBase, Paradox; OLE DB connections to ODBC data sources; Microsoft Excel 2000 spreadsheet data; HTML sources; and other OLE DB providers.

• A file connection.

DTS provides additional support for text files. When specifying a text file connection, you specify the format of the file. For example:

- Whether a text file is in delimited or fixed field format.
- Whether the text file is in a Unicode or an ANSI format.
- The row delimiter and column delimiter if the text file is in fixed field format.
- The text qualifier.
- Whether the first row contains column names.

• A data link connection.

These are connections in which an intermediate file outside of SQL Server stores the connection string.

Configuring a Connection

When creating a package in the DTS Import/Export Wizard, in DTS Designer, or programmatically, you configure connections by selecting a connection type from a list of available OLE DB providers. The properties you configure for each connection vary depending on the individual provider for the data source.

You can configure a new connection or use an existing one. You can use the same connection multiple times in a package.

Before configuring a connection, consider the following:

- Each connection can be used by only one DTS task at a time because the connections are single-threaded. When designing a package that requires multiple task connections, consider opening up several connections and balancing the load to improve performance.
- If two tasks use the same connection, they are compelled to execute serially, rather than in parallel. If two tasks use different connections, they may execute in parallel. If two tasks use separate connections that refer to the same instance of SQL Server, they will execute in parallel. If both of these tasks have joined the package transaction, the package fails.

For more information, see <u>DTS Transaction Fundamentals</u>.

• If you plan to run a package on different servers, you may need to edit the direct connections made in a package (for example, if the original data sources will be unavailable, or you will be connecting to different data sources). To simplify editing, consider using a data link file, where the connection string is saved in a separate text file. Alternately, consider using the Dynamic Properties task to change the connection information at run time. • When scheduling a package, consider the security information you have provided. If you used Windows Authentication when configuring a connection, the SQL Server Agent authorization information is used to make the connection rather than the account information you used when designing the package. For more information, see <u>Handling Package</u> <u>Security in DTS</u>.

To create a connection

Data Link Connection

In Microsoft® SQL Server[™] 2000, Data Transformation Services (DTS) packages can use Microsoft Data Link (.udl) files to create OLE DB connections and resolve the connections at run time. This feature lets you encapsulate the connection properties from a DTS package into a separate file. In situations where connection information such as the server name, login, or even the OLE DB provider may change, you can edit the connection string in a data link file instead of the connection properties in a DTS package.

In Microsoft SQL Server version 7.0, you can use a data link file, but the connections are not resolved at run time and can be modified only by editing the DTS package. This choice is useful in situations where an OLE DB provider has special connection requirements that can be addressed only through the data link dialog boxes.

If you are specifying a data link connection, you first must specify whether to load an existing data link file or create a data link to save with DTS Designer.

If you want to use a data link file rather than saving the data link with DTS Designer, you can create one either from Windows Explorer or during the data link configuration process in DTS Designer.

To create a data link file with run-time resolution

DTS Package Workflow

Data Transformation Services (DTS) steps and precedence constraints order work items in a DTS package. You can design DTS package workflow graphically, through DTS Designer, or programmatically. For more information, see <u>Creating DTS Package Workflow and Tasks</u>.

You also can use a Microsoft® ActiveX® script to customize step execution. For more information, see <u>Using ActiveX Scripts in a DTS Workflow</u>.

DTS Package Steps

Steps control the order in which tasks are executed in a DTS package. Steps represent the execution units in the DTS object model, and they define which tasks execute in what sequence when the package is run.

In DTS Designer, you do not manipulate steps directly. Instead, you manipulate tasks on the DTS Designer design sheet and use precedence constraints to control the sequence in which the tasks execute. When you place a task on the design sheet, a step is automatically added to the package, for a step references a task.

When creating a package programmatically, you can control the relationship between a step and a task more precisely. You can create multiple steps for different package operations and associate the execution of those steps with a single task. For example, suppose you write a package in Microsoft Visual Basic® and specify in several parts of the package that errors can be generated. By linking the steps associated with those errors, you can make the different types of errors execute the same Send Mail task. That Send Mail task can send an e-mail notifying the database administrator (DBA) that the package failed.

DTS Designer allows you to execute an individual package step. This action is useful for testing and troubleshooting individual steps without having to run the entire package.

To execute a single package step in DTS Designer

Using ActiveX Scripts in a DTS Workflow

You can use Microsoft® ActiveX® scripts to customize the execution of steps in a Data Transformation Services (DTS) package. Because the code is run before the steps executes, you can use an ActiveX script in a workflow to:

- Restart a workflow.
- Turn off a step under certain conditions.
- Initiate retries of connections and other operations.
- Implement loop conditions.

You can also use a step ActiveX script to initialize or reference global variables. For more information, see <u>Using Global Variables with DTS Packages</u>.

To add ActiveX workflow scripts in DTS Designer

Managing a DTS Package

You can manage Data Transformation Services (DTS) packages from SQL Server Enterprise Manager and from within DTS tools. The following topics describe the various aspects of package management.

Торіс	Description	
Creating a DTS Package	Explains different ways to create a DTS	
	package.	
Editing a DTS Package	Describes how to modify an existing package.	
Deleting a DTS Package	Describes different ways to delete a package	
	and package versions.	
Executing a DTS Package	Explains different ways to run a package.	
Saving a DTS Package	Describes the different formats in which you	
	can save a package.	
Using DTS Package Logs	Describes the information contained in a	
	package log.	
Managing DTS Package	Describes different ways to view, configure, and	
<u>Properties</u>	edit package properties.	
Handling Package	Discusses security issues surrounding packages	
Security in DTS	and ways to increase package security.	

You also can save and execute packages programmatically. For more information, see <u>Managing DTS Package Programs in Visual Basic</u>.

Creating a DTS Package

You create Data Transformation Services (DTS) packages either by using tools provided with Microsoft® SQL Server[™] 2000 or by programming the DTS object model.

Package Construction Method	Description	Recommended Usage
<u>DTS</u> Import/Export <u>Wizard</u>	An easy-to-use tool that guides you, a step at a time, through the process of creating a DTS package.	For simple data transformation or data movement solutions (for example, importing tabular data into a SQL Server 2000 database).
DTS Designer	An application that uses graphical objects to help you build packages containing complex workflows. DTS Designer includes a set of model <u>DTS Package</u> <u>Templates</u> , each designed for a specific solution that you can copy and customize for your own installation.	For sophisticated data transformation solutions requiring multiple connections, complex workflows, and event-driven logic. DTS package templates are geared toward new users who are learning about DTS Designer or more experienced users who want assistance setting up specific DTS functionalities (for example, data driven queries).
Programming DTS Applications	Programming applications that you can use to write and compile a DTS package either in Microsoft	For developers who want to access the DTS object model directly and exert a fine degree of control over

Visual Basic® or Microso Visual C++®.	oft package operations. Packages created programmatically can be opened and further customized in DTS Designer. In addition, packages created in the DTS Import/Export Wizard or DTS Designer can be saved as a Visual Basic program and then opened and further customized in a development environment such as Microsoft Visual Studio®.
--	---

DTS packages created on an instance of SQL Server 2000 cannot be loaded or run on an instance of SQL Server version 7.0 or earlier. For more information, see <u>SQL Server Backward Compatibility Details</u>.

To create a DTS package using the DTS Import/Export Wizard

Editing a DTS Package

When you edit a Data Transformation Services (DTS) package, you modify or further customize a previously saved package. For example, you can create a package that copies data from an Oracle server to an instance of Microsoft® SQL ServerTM 2000, and later you can add a task that sends an e-mail notification when the copy operation completes.

You edit a package by:

• Using DTS Designer.

Note You cannot use the DTS Import/Export Wizard to edit a package.

- Using the Microsoft Visual Basic® development environment, if a package has been created with DTS Designer or the DTS Import/Export Wizard and saved as a Visual Basic file.
- Using the Microsoft Visual Studio® development environment, if a package has been created in Microsoft Visual Basic or Microsoft Visual C++® modules and saved as a Visual Basic or Visual C++ project. In most cases, you can open these types of packages for editing in DTS Designer after saving them.

To edit a package, you must have authorization to open the package. If a package is saved with an owner password and you do not have access to that password, you cannot edit the package. If a user password is set, and you have access to that password, you can execute but not edit the package.

You cannot set DTS package password protection if you save a package to SQL Server 2000 Meta Data Services. In that case, you need to handle security through Windows Authentication or SQL Server Authentication. For more information, see <u>Handling Package Security in DTS</u>.

To edit a DTS package saved to a structured storage file

Deleting a DTS Package

You can delete any Data Transformation Services (DTS) package. The method you use depends on the format in which you saved the package. For example:

- If the package was saved to Microsoft® SQL Server[™] or SQL Server 2000 Meta Data Services, delete the package through SQL Server Enterprise Manager.
- You can only delete a package saved to SQL Server or Meta Data Services if you are the package creator or a member of the **sysadmin** fixed server role. If the package was saved to a structured storage file or to a Microsoft Visual Basic® file, delete the package through a file manager.

Deleting Package Versions

To delete package versions, you need to consider the format in which you saved the package. For example:

- If you save a package to SQL Server, you can delete any package version.
- If you save a package to Meta Data Services, you can delete only the most recent package version.
- If you save a package to a structured storage file, you must delete the entire file. You cannot delete individual packages or package versions saved to the file.

Packages saved to a Visual Basic file do not contain version information.

To delete a DTS package

Executing a DTS Package

When you run a Data Transformation Services (DTS) package, all of its connections, tasks, transformations, and scripting code are executed in the sequence described by the package workflow.

You can execute a package from:

- Within a DTS tool.
- SQL Server Enterprise Manager.
- Package execution utilities.

Executing a Package from a DTS Tool

You can execute a package in DTS Designer after creating or editing a package or in the DTS Import/Export Wizard after creating a package.

When you run a package in DTS Designer or the DTS Import/Export Wizard, a summary status of the execution progress and execution status of all the steps in the package is displayed. You can check the status of each step and gather information on steps that failed to execute successfully.

Note You also can view the status of step execution when you use the DTS Run utility.

To execute a DTS package from DTS Designer

Scheduling a DTS Package for Execution

You can schedule a saved Data Transformation Services (DTS) package to execute at specific times, either once or at recurring intervals. For example:

- Daily at 12:00 midnight.
- Weekly on Sunday at 6:00 A.M.
- The first or last day of the month.

A scheduled DTS package is executed by SQL Server Agent as a job. Because SQL Server Agent controls the underlying automation for scheduling, it must be running for any scheduled packages to execute. To schedule a DTS package for execution, do one of the following:

• In SQL Server Enterprise Manager, right-click a DTS package, and then click **Schedule package**.

This option is the easiest way to schedule packages created in the DTS Designer for execution. However, the package needs to have been saved to either the Microsoft® SQL ServerTM **msdb** database or SQL Server 2000 Meta Data Services and needs to exist on the local server.

To schedule a DTS package using the Schedule Package option

Saving a DTS Package

When you save a Data Transformation Services (DTS) package, you save all DTS connections, DTS tasks, DTS transformations, and workflow steps and preserve the graphical layout of these objects on the DTS Designer design sheet.

You can save a package to:

• Microsoft[®] SQL ServerTM.

With this default save option, you can store a package as a SQL Server **msdb** table, allowing you to: store packages on any instances of SQL Server on your network; keep a convenient inventory of saved packages in SQL Server Enterprise Manager; and create, delete, and branch multiple package versions during the package development process.

To save a DTS package to SQL Server

Saving a DTS Package to SQL Server

Save your Data Transformation Services (DTS) package to Microsoft® SQL Server[™] if you want to store packages on any instance of SQL Server on your network, keep a convenient inventory of those packages, and add and delete package versions during the package development process. This option saves a DTS package in the **sysdtspackages** table in the SQL Server **msdb** database as BLOB (binary large object) data.

You can save and delete versions of a SQL Server package. If a package has multiple versions, you can display a version history in SQL Server Enterprise Manager and open the version you want. Otherwise, the latest package version is opened.

To save a DTS package to SQL Server

Saving a DTS Package to Meta Data Services

Save your Data Transformation Services (DTS) package to Microsoft® SQL Server[™] 2000 Meta Data Services if you plan to track package version, meta data, and data lineage information.

You can save versions of a package to Meta Data Services. If a package has multiple versions, you can display a version history in SQL Server Enterprise Manager and open the version you want. Otherwise, the latest package version is opened. You can also view version information for packages saved to Meta Data Services with Meta Data Services viewing tools supplied with SQL Server Enterprise Manager.

When you save a package to Meta Data Services, the DTS package protection options are not available. If package security is important, consider saving the package to SQL Server or as a structured storage file instead.

If you create a package outside of DTS Designer (for example, in Microsoft Visual Basic®), you can specify the repository database to which you want to save the package data (for example, Microsoft Access).

Versioning

DTS Designer maintains version information about each saved package, and this information can be stored in Meta Data Services. Version information for each package includes:

- A package GUID, a globally unique identifier (GUID) that identifies the package.
- A version GUID, a GUID that identifies the package version.

When a package is first created, the package GUID and version GUID are the same, and there is only one version. If changes made to a package are saved, the package is versioned, and the new version is assigned a different version GUID than that of the previously saved version. Rather than the last version overwriting the previous one (as in a typical save operation), each package

version is preserved.

When you save a package to Meta Data Services, version information can be linked to saved meta data by using the scanning options. In addition, data lineage information can be saved for a package. These features let you track:

- Changes to the package meta data, such as changes to table columns and keys (displayed in the **Meta Data Browser** in SQL Server Enterprise Manager) across package versions.
- Which package version produced a particular set of transformations. To do this, use the data lineage lookup feature in SQL Server Enterprise Manager.

To save a DTS package to Meta Data Services

Saving a DTS Package to a Structured Storage File

Save a Data Transformation Services (DTS) package to a structured storage file if you want to copy, move, and send a package across the network without having to store the package in a Microsoft® SQL ServerTM database.

With the structured storage format, you can maintain multiple packages and multiple package versions in a single file. If you want to save multiple packages to the same structured storage file, use different package names but the same file name when saving. However, every package and package version you save under the structured storage file name persists for the lifetime of that file. Although you can edit individual packages or package versions saved to a structured storage file, you cannot delete them. Therefore, you need to manage package versions differently when saving to structured storage files than when saving to SQL Server.

When saving a package to SQL Server, you can delete any package version you want. With a structured storage file, you can delete only the entire file. If you want to retain or branch a specific package version saved in a structured storage file, save the package version under a new file name. It is recommended that you do this infrequently, because saving multiple packages with multiple versions can generate a sizeable number of large files.

Note When you save a DTS package to a structured storage file, you create a file with the extension .dts.

To save a DTS package to a structured storage file

Saving a DTS Package to a Visual Basic File

You can save a Data Transformation Services (DTS) package that has been created by DTS Designer or the DTS Import/Export Wizard to a Microsoft® Visual Basic® file. Packages saved in this way can be incorporated into Visual Basic programs or can be used as prototypes by Visual Basic developers who need to reference the components of the DTS object model.

To save a DTS package to a Visual Basic file

Using DTS Package Logs

Use the Data Transformation Services (DTS) package log to troubleshoot problems that occurred during the execution of a DTS package. The DTS package log, unlike the Microsoft® SQL Server[™] error log and the DTS exception log, contains information about the success or failure of each step in a package and can help determine the step at which a package failure occurred. Each time a package executes, execution information is appended to the package log, which is stored in **msdb** tables in SQL Server or in SQL Server Meta Data Services. You can save package logs on any server running an instance of SQL Server 2000. If a package log does not exist, the log will be created when a package is run.

An executing package writes information to the package log about all steps in the package, whether or not an individual step runs. If a step runs, it will retain start and end times, and the step execution time. For steps that do not run, the log lists the steps and notes that the step was not executed.

Package logging is only available on servers running an instance of SQL Server 2000.

Along with DTS package logs, DTS exception files provide helpful troubleshooting information. The Data Driven Query task and Transform Data task use exception files to save error information about rows of data that were not copied to the destination and to store the actual source and destination rows that failed. For more information, see <u>Tasks That Transform Data</u>.

To view DTS package logs

Managing DTS Package Properties

In Data Transformation Services (DTS), you can retrieve or set package properties graphically, with DTS Designer, or programmatically. For more information on managing package properties programmatically, see <u>Creating DTS Packages with the DTS Object Model</u>.

With DTS Designer, you can view or monitor properties associated with:

- Package identification.
- Errors.
- Microsoft Windows® events.
- Global variables.
- Transactions.

To view or modify DTS package properties

Viewing and Modifying DTS Package Properties

In Data Transformation Services (DTS) Designer, you can view or modify properties associated with:

• Package identification.

Most of these properties are read-only. The information includes package name, package and version globally unique identifiers (GUID), creator name, creation date, computer name, and a text description for the package.

• Error information.

You can specify information about the log file, which contains package run-time errors. You also can specify whether package execution is terminated after the first error, which reflects a step failure.

• Microsoft® Windows® events.

You can specify whether the package execution status is written to the Windows event log. You also can define the process priority for Windows events and the maximum number of tasks that can execute concurrently with the package.

• Package global variables.

You can view information about global variables, which can be referenced by any Microsoft ActiveX® script in the package. You can create new global variables or edit existing global variables.

• Data lineage.

You can track the source of any piece of data and the transformations applied to that data when saving it to Microsoft SQL Server[™] 2000 Meta Data Services.

• Meta Data Services scanning.

You can relate objects referenced by the DTS package to catalog meta data in Meta Data Services.

• Transactions.

You can assign steps in a workflow to a transaction, and commit and roll back individual steps based on the success or failure of the transactional unit. For transactions to work, the Microsoft Distributed Transaction Coordinator (MS DTC) must be running on the computer executing the package.

To view or modify DTS package properties

Editing DTS Package Properties with Disconnected Edit

Data Transformation Services (DTS) Designer includes a Disconnected Edit feature that allows you to view or modify the value of any property associated with a package. For example:

- General package properties (for example, package name, description, creator name, package priority class).
- Connection properties (for example, server name, user name, and password).
- Task properties (for example, custom tasks, tasks that transform data, and data driven queries).
- Step properties, including precedence constraints.
- Global variables.

Use the Disconnected Edit feature to:

 Modify a package when the source or destination connection is unavailable. Usually, when you build a DTS package, connectivity is required as a precaution against setting properties or including components that do not work. Disconnected Edit allows you to edit a package without establishing a live connection. For example, you can modify a package created on a test system so that it works on a production system at a different site. You can use Disconnected Edit to change the connection properties of the source and destination to those of the production system, without having to actually connect to the production system. • View and modify properties that are not exposed through the DTS Designer user interface, such as task names, step names, and connection names.

When changing a task name with Disconnected Edit, change the **TaskName** property of the step associated with the task.

CAUTION Because Disconnected Edit does not validate changes to property values, entering invalid data can result in package failure and unwanted effects on your system. Disconnected Edit is similar to Registry Editor. Both are powerful tools for editing properties directly, and it is recommended that both be used only by advanced users when there are no alternative methods of modifying values.

To use Disconnected Edit to modify DTS package properties

Handling Package Security in DTS

To view, edit, protect, schedule, and run Data Transformation Services (DTS) packages on your network, you need to understand issues that affect package access, permissions, and connections.

DTS Package Passwords

When you save a package to Microsoft® SQL Server[™] or as a structured storage file, you can use DTS package passwords. You use DTS passwords in addition to the Windows Authentication or SQL Server Authentication passwords you use to connect to an instance of SQL Server. The following types of DTS package passwords are available:

- If you set an owner password, the package user needs the password to edit or run the package.
- If you set a user password, you also must set an owner password. Package users with access only to the user password can run the package. However, they can neither open nor edit the package unless they have access to the owner password.

It is strongly recommended you use DTS package passwords for all packages to ensure both package and database security. At a minimum, always use DTS package passwords when connection information to a data source is saved and Windows Authentication is not used.

To set a DTS package password

Adding Functionality to a DTS Package

Microsoft® SQL Server[™] 2000 Data Transformation Services (DTS) includes capabilities for extending the functionality of a DTS package. The following sections provide information on these capabilities and explains how to use them.

Торіс	Description
Using ActiveX Scripts in DTS	Explains how to write scripting code that
	executes as a task, workflow step, or
	transformation.
Incorporating Transactions in a	Describes how to bind multiple package
DTS Package	steps, including disparate operations on
	multiple platforms, into a single
	transactional unit.
Lookup Queries	Explains how to use lookup queries, which
	allow you to run queries and stored
	procedures against other connections
	besides the source and destination.
Using Global Variables with	Explains how to use global variables to
DTS Packages	pass data between different package steps
	and tasks, and dynamically assign values.
Using Parameterized Queries in	Describes how parameterized queries can
DTS	be used with several DTS tasks.
Querying a DTS Package from	Explains how to make DTS package data
External Sources	available to an external source, such as
	SQL Query Analyzer, and how to join
	package data in a distributed query.

Using ActiveX Scripts in DTS

In Data Transformation Services (DTS), you can extend the capabilities of your DTS package by using Microsoft® ActiveX® scripts that implement the objects, properties, methods, and collections of the DTS object model. Using ActiveX scripts, you can:

- Format and transform the data as it is copied from its source to its destination.
- Write functions that use conditional logic to manage package workflow or that process data on a row-by-row basis.
- Create, use, and modify the values stored in DTS global variables.
- Manipulate COM objects built for data access and utility functions.
- Create and use Microsoft ActiveX Data Objects (ADO) connections, commands, recordsets, and other objects to access and manipulate data.

Microsoft Visual Basic[®] Scripting Edition (VBScript) and Microsoft JScript[®] are available with an installation of Microsoft SQL Server[™]. If you plan to write ActiveX scripts in a language other than VBScript and JScript, be sure the language library for the scripting language you use is installed. For more information, search on "VBScript" and "JScript" in the MSDN® Library at <u>Microsoft Web site</u>.

Writing ActiveX Scripts in DTS

In DTS, you can write the following types of scripts:

• Transformation scripts that are applied to data on a row-by-row basis. The script executes each time a row of data is read from the source.

You add these scripts only to DTS tasks that transform data: the

Transform Data task; the Data Driven Query task; and the Parallel Data Pump task (available only programmatically). These tasks use the data pump to transform the data.

• ActiveX scripts that function as tasks. The script is run once each time the task is called by the package.

You can use an ActiveX script task to perform functions that are not available in the other tasks in DTS Designer. For more information, see <u>ActiveX Script Task</u>.

• Workflow scripts that are applied to a package step prior to step execution. The script is run once each time the task is called by the package.

You can use an ActiveX workflow step script to customize step execution. For example, you can use certain return codes in an ActiveX workflow step script to prevent other tasks in a package from executing, or you can allow a task to execute or mark the task as waiting to be run. For more information about writing scripts that control workflow, see <u>DTS Package Workflow</u>.

Generally, you do not use an ActiveX script task or ActiveX workflow script to operate on data on a row-by-row basis because it is less efficient than using a transformation script. However, it is possible to do so. For example, you could use an ActiveX Script task to create one or more ADO connections and populate a set of text files with data from an ADO recordset.

IMPORTANT An ActiveX script in a transformation can affect the execution speed of a DTS package. Therefore, if performance is a priority, use scripting carefully when building a package. For more information, see <u>Enhancing Performance of DTS Packages</u>.

Adding ActiveX Scripts to a DTS Package

You can add scripts to a package in DTS Designer, in the DTS Import/Export Wizard, or programmatically. For more information about adding scripts programmatically, see <u>Adding DTS ActiveX Scripts</u>.

To add transformation scripts:

- In DTS Designer, add transformation scripts in the **ActiveX Script Transformation Properties** dialog box to define an ActiveX Script transformation for a Transform Data task or a Data Driven Query task.
- In the DTS Import/Export Wizard, add transformation scripts in the **Transformation** tab of the **Column Mappings and Transformations** dialog box. For more information, see <u>Creating a DTS Package with the DTS Import/Export Wizard</u>.

To add ActiveX Script Tasks:

• In DTS Designer, drag an ActiveX Script Task onto the design sheet and add the script when configuring the task.

To add ActiveX workflow scripts:

• In DTS Designer, access the **Workflow Properties** dialog box associated with a package step.

Scripting Capabilities

The range of functionality you can access from ActiveX scripts covers:

- The SQL Server environment. You can use Transact-SQL statements in your scripts and access SQL Server tables.
- Data access interfaces. SQL Server installations include familiar data access interfaces such as ADO, Data Access Objects (DAO), and Remote Data Objects (RDO), which you can use in your scripts to make connections, create recordsets, and execute SQL commands.
- Custom COM objects. You can access custom COM objects you develop in your scripts, if the objects are available on the server running the package.
- The scripting language. You can use any function of the scripting

language you code with, provided the scripting engine for the language is installed on your server. Most scripting languages allow you to:

- Use looping and conditional logic.
- Write functions that control row selection, determine workflow, control the success or failure of an operation, or throw exceptions. These determinations are made by specialized return code constants specified by the DTS object model.

For example, a scripting language such as VBScript allows you to use intrinsic functions supplied by the language, such as Trim, Len, and CInt, and validate data in a field with functions such as IsNumeric or IsDate. If you require functionality beyond what can be achieved using a scripting language, you may want to program your own DTS applications or custom tasks. For more information, see <u>Creating DTS Packages in Visual Basic</u>.

Scripting Examples

The following DTS ActiveX script examples show:

- ActiveX Script transformations that transform date data, concatenate columns, and validate data.
- ActiveX Script tasks that connect and use ADO objects, and populate global variables.

Simple Column Transformation

You have a daily sales table sent to your Accounting department. However, your accounting month is not based on a calendar month. Sales before the 15th day of the month are considered sales for that month. Any sales that take place on the 15th or after are considered sales for the following accounting month.

The following ActiveX Script transformation, written in VBScript, copies all the columns from the source to destination table, for each row of source data, except for the **AccountingMonth** column. For that column, the **Sale_Date** is checked

to see if the sale took place before or after the 15th, and the **AccountingMonth** adjusted accordingly.

```
Function Main()
 DTSDestination("DeptName") = DTSSource("DeptName")
 DTSDestination("Sales") = DTSSource("Sales")
 DTSDestination("Sale_Date") = DTSSource("Sale_Date")
' break the day out of the Sale_Date
 theDay = Day(DTSSource("Sale Date"))
' if the day field is before the 15th, the Accounting Month is the same a
' the current Sale Date month
 If the Day < 15 then
   DTSDestination("AccountingMonth") = DTSSource("AccountingN
 Else
' if the day is the 15th or later, the sales dollars belong in the next
' Accounting Month
   theMonth = DTSSource("AccountingMonth") + 1
   ' check if we were in December when we added 1 to the month, and
   If theMonth > 12 then
     the Month = 1
   End if
   DTSDestination("AccountingMonth") = theMonth
 End If
 Main = DTSTransformStat OK
```

End Function

Concatenating Columns

In the following example, an ActiveX Script transformation, written in VBScript, consolidates the **Sales_Month**, **Sales_Day**, and **Sales_Year** columns from the source table into a single **Sales_Date** column in the destination table.

This script is run on each row in the source data, and can be used in those tasks that operate on a row-by-row basis.

```
Function Main()
DTSDestination("CustomerID") = DTSSource("CustomerID")
DTSDestination("Sales_Date") = Trim(DTSSource("Sales_Month"))
+ "/" + Trim(DTSSource("Sales_Day")) + "/" + _
Trim(DTSSource("Sales_Year"))
Main = DTSTransformStat_OK
End Function
```

Transforming Date Data

When importing data from a file to an OLE DB destination table, you can use the VBScript CDate function to convert date data if the date format is in a text or character field and is not in the format required by OLE DB, which is *yyyy-mmdd hh:mm:ss:sss*. CDate is useful when the source data is in more than one format. If the source data is in a single format, then consider using the Date Time transformation, which is faster.

Function Main()

DTSDestination("Total Sales") = DTSSource("Total Sales ") DTSDestination("DestColumnDate") = CDATE(DTSSource("Source

Main = DTSTransformStat_OK

End Function

Reading Values from a Text File Using FileSystemObject

In the following VBScript example, the input text file, Start_End_Dates.txt, contains the start and end dates to be read into global variables. The text file is stored on the C:\ drive. The start date is the first line of text and contains "01/01/00" and the second line contains the end date, which is "01/31/00". After the package executes and the script runs, two message boxes are displayed. The first message box shows "The Start Date is: 01/01/00", and the second message

box shows "The End Date is: 01/31/00".

'Read start and end dates from a flat file and'store the values in dynamically generated global variables'Function Main()

dim oFSO dim x

- ' instantiate the Scripting Object set oFSO = CreateObject("Scripting.FileSystemObject")
- ' Open the file
 set x = oFSO.OpenTextFile("C:\Start_End_Dates.txt")
- ' store the first line, which is the Start Date, in a global variable DTSGlobalVariables("StartDate").value = x.Readline MsgBox "The Start Date is: " & DTSGlobalVariables("StartDate").v
- store the second line, which is the End Date, in a global variable DTSGlobalVariables("EndDate").value = x.Readline MsgBox "The End Date is: " & DTSGlobalVariables("EndDate").va

x.Close

Main = DTSTaskExecResult_Success

End Function

Using an ADO Connection and Recordset to Check Records

The following ActiveX script creates a connection to the **Northwind** database and the **employee** table and counts the number of employee records. If employee records are found in the table, the script displays the number of employees and sends a success flag back to the package. Otherwise, the script sends a failure flag. Those flags can be used to trigger other tasks. For example, the success flag can signal that the table has records and then execute a Bulk Insert task. You can use the failure flag to execute a Send Mail task informing a database administrator (DBA) that a potential problem exists.

dim myConn dim myRecordset dim iRowCount

```
' instantiate the ADO objects
set myConn = CreateObject("ADODB.Connection")
set myRecordset = CreateObject("ADODB.Recordset")
```

```
' set the connection properties to point to the Northwind database,
' using the Customers table
myConn.Open = "Provider=SQLOLEDB.1;Data Source=(local); _
Initial Catalog=Northwind;user id = 'sa';password=""
```

```
mySQLCmdText = "Select 'rowcount' = Count(*) from Customers"
```

```
myRecordset.Open mySQLCmdText, myConn
```

```
set Flds = myRecordset.Fields
set iRowCount = Flds("rowcount")
```

```
If iRowCount.Value = 0 then

Main = DTSTaskExecResult_Failure

Else

MsgBox "The number of customers is: " & iRowCount.Value

Main = DTSTaskExecResult_Success

End If
```

Inserting Rows into a Table using an ADO Recordset

The following example contains an ActiveX script written in VBScript that

shows how to connect to a source using ADO and how to insert the rows into the destination table. The example uses tables from the **Northwind** database, and one that you must create, a **NewEmployeeTerritory** table that contains the **EmployeeID** and the new **TerritoryID** that the employee is assigned to. These new territory assignments need to be entered into the **EmployeeTerritory** table.

To run this example, do the following:

- 1. Create a table named **NewEmployeeTerritory** in the **Northwind** database that has a schema identical to the **EmployeeTerritory** table.
- 2. Insert the following four records into the **NewEmployeeTerritory** table:

EmployeeID TerritoryID

1	03801
1	07960
3	40222
9	11747

- 3. Create a new DTS package in DTS Designer.
- 4. Drag an ActiveX Script task onto the design sheet.
- 5. In the ActiveX script box, place the following code between the FUNCTION MAIN() and END FUNCTION statements:
 'These values were copied from the ADOVBS.INC file.
 '---- CursorTypeEnum Values ----Const adOpenForwardOnly = 0
 Const adOpenKeyset = 1
 Const adOpenDynamic = 2
 Const adOpenStatic = 3

'---- CommandTypeEnum Values ----Const adCmdUnknown = &H0008 Const adCmdText = &H0001 Const adCmdTable = &H0002 Const adCmdStoredProc = &H0004

dim countr

' Instantiate the ADO objects.
set mySourceConn = CreateObject("ADODB.Connection")
set mySourceRecordset = CreateObject("ADODB.Recordset")

'Set the connection properties to point to Northwind. 'Use the NewEmployeeTerritories table. mySourceConn.Open = "Provider=SQLOLEDB.1;Data Sourc Initial Catalog=Northwind;user id = 'sa';password=""

mySQLCmdText = "Select * from NewEmployeeTerritories"

'Execute the mySQLCmdText, and put the data into the myRe mySourceRecordset.Open mySQLCmdText, mySourceConn,

If mySourceRecordset.RecordCount < 1 Then

MsgBox " There are no records found. Return a Failure cod Main = DTSTaskExecResult_Failure

Else

' Since we have records to insert into the EmployeeTerritory ta

' a Connection object and do the INSERT.

dim EmpID, TerrID, myDestSQL set myDestConn = CreateObject("ADODB.Connection") myDestConn.Open = "Provider=SQLOLEDB.1;Data Sourc Initial Catalog=Northwind;user id = 'sa'''

for countr = 1 to mySourceRecordset.RecordCount EmpID = mySourceRecordset.Fields("EmployeeID").val

```
TerrID = mySourceRecordset.Fields("TerritoryID").value
' Put single quotes around the TerrID since it is a varchar
TerrID = """ & Terrid & """
```

```
myDestSQL = "INSERT INTO EmployeeTerritories _
VALUES ( " & EmpID & "," & Terrid & ")"
myDestConn.Execute myDestSQL
```

```
mySourceRecordset.MoveNext
Next
```

Main = DTSTaskExecResult_Success

End If

Validating Data

The following ActiveX script, written in VBScript, modifies data on a row-byrow basis. Using the **Customers** table of the **Northwind** database as a source, the script moves the data into a new destination table in **Northwind**. The script validates several columns in the source data and transforms some column data before the row is inserted into the destination. The transformations change the **Company Name** to uppercase characters, trim leading and trailing spaces from the first name and last name, and fill the **Region** field with the string "unknown" if it is empty.

' Verify that there is a CompanyName. If there is, process the record. If ' there is not, skip the record.

If DTSSource("CompanyName") <> "" Then

DTSDestination("CustomerID") = DTSSource("CustomerID")

' Uppercase the Company Name

DTSDestination("CompanyName") = Ucase(DTSSource("Company

' Trim leading and trailing spaces from the Name

DTSDestination("ContactName") = Trim(DTSSource("ContactNam DTSDestination("ContactTitle") = DTSSource("ContactTitle")

```
DTSDestination("Address") = DTSSource("Address")
DTSDestination("City") = DTSSource("City")
```

' Check to see if the region is empty. If it is, fill it with string ' of "unknown".

```
If IsNull(DTSSource("Region").value then
DTSDestination("Region") = "unknown"
```

Else

```
DTSDestination("Region") = DTSSource("Region")
End if
```

```
DTSDestination("PostalCode") = DTSSource("PostalCode")
DTSDestination("Country") = DTSSource("Country")
DTSDestination("Phone") = DTSSource("Phone")
DTSDestination("Fax") = DTSSource("Fax")
```

```
' This was a successful row. Send an OK status back for this row.
Main = DTSTransformStat OK
```

Else

```
' This row contained data that could not be processed.
```

```
' Skip it and get another row.
```

```
Main = DTSTransformStat_SkipRow
```

End If

Using a Global Variable that Contains Columns of Data

This example uses an Execute SQL task to select data from a table, and populate global variables with the data from the first row returned. Each column is stored in its own global variable. The second half of this sample uses ActiveX script to display the data stored in the global variables.

To save row values into global variables

Using Return Codes in DTS

In Data Transformation Services (DTS), you can use Microsoft® ActiveX® script return codes to:

- Process row data in a transformation, including the handling of inserts, errors, and exceptions.
- Set up conditions in a transformation script for skipping rows, reusing rows, choosing whether to write the row to the destination, and terminating row processing based on an error.
- Control package workflow or step execution.
- Execute the correct query type for a data driven query.

DTS provides a number of return code constants. Depending on where you use the ActiveX script, these return codes have different effects on the package or rows:

- Return codes used inside an ActiveX Script task are sent to the package, where they control the flow of steps through the use of precedence constraints. The success or failure of one task activates any On Success or On Failure precedence constraints on any tasks that follow it.
- Return codes used in a transformation script or a script that operates on a row-by-row basis apply to the row being processed.

Note Use the return code constants rather than the decimal or hexadecimal values of the transformation return code in the script.

• A return code in a Data Driven Query task defines which SQL query to execute on each row processed.

Using Multiple Return Codes

Some situations require multiple return codes. For example, if you want to transform a row multiple times but do not want to insert it into the destination multiple times, you can use **DTSTransformStat_SkipFetch** to transform the row again. Or, you can use **DTSTransformStat_SkipInsert** to prevent it from being put into the destination. However, no return code can do both. In that situation, you can use an OR operator to combine actions. For example, Main = **DTSTransformStat_SkipFetch** prevents another row from coming into the transformation, and **DTSTransformStat_SkipInsert** keeps the row from being inserted into the destination.

See Also

DTSTransformStatus

Debugging ActiveX Scripts

If you have Microsoft® Windows® 2000, Microsoft Visual InterDev® 6.0 or the Microsoft Windows NT® 4.0 Option Pack installed, you can use the script debugger supplied with those products to troubleshoot your Microsoft ActiveX® scripts. Three types of events cause errors:

- A forced break in the script execution.
- Incorrect script syntax or object.
- Objects that contain incorrect references.

Your ability to debug these errors depends on whether or not you have selected the **Turn on just-in-time debugging** check box.

To select the Turn on just-in-time debugging option

Incorporating Transactions in a DTS Package

You use database transactions to bind multiple updates into a single atomic unit. In this way, you help to ensure that your data remains in a consistent state. Distributed transactions carry this concept a step further, allowing you to bind disparate operations on multiple platforms into a single transaction.

Data Transformation Services (DTS) uses functions offered by the Microsoft® Distributed Transaction Coordinator (MS DTC) to extend the benefits of distributed transactions to the DTS package developer. For transactions to work, MS DTC must be running on the computer executing the package. Use the SQL Server Service Manager to start MS DTC or to verify that it is running.

With DTS transactions, you can:

- Gather the results of several tasks into a single transaction and so ensure consistent updates. For example, orders and line items can be uploaded by two tasks, which succeed or fail together.
- Perform consistent updates on multiple database servers. For example, a customer address can be changed in two different online transaction processing (OLTP) systems, all in the context of one transaction.
- Combine database modifications and message queue operations in a single transaction to provide guaranteed updates in an asynchronous environment. For example, a package might employ a Message Queue task to read and delete a message bearing the name of a file to upload. If the task that uploads the file fails, the subsequent rollback both reverses the database changes and puts the message back on the queue. The package can be restarted with no user intervention.
- Carry out multiple transactions under the control of a single package. For example, using Execute Package tasks, you can simultaneously run an end-of-day sequence of transactions on each of three different servers. For more information, see <u>Execute Package Task</u>.

You can use DTS Designer to manage package transactions, or you can access the same functionality programmatically. For more information, see <u>DTS</u> <u>Designer</u> and <u>DTS Programming Reference</u>.

Configuring Properties for DTS Transactions

Several Data Transformation Services (DTS) package properties and workflow step properties are used to control transaction initiation, step participation in the transaction, and the final commit or rollback operation. These properties can be set from within DTS Designer or programmatically, through the package and step objects of the DTS object model.

DTS transaction settings fall into two groups. Package properties are global settings that affect transaction behavior across the entire package. Step properties operate at the level of the individual task. In the following list of DTS transaction settings, package properties are listed before step properties. For each transaction option, the corresponding DTS object model property is listed.

In the DTS Package Properties dialog box, on the General tab, you can find:

• Fail package on first error.

If this check box is selected, the first step failure triggers package failure, terminating all tasks. If cleared, the package continues to run after the first and subsequent step failures, always completing successfully, no matter how many errors occur.

In the DTS object model, set the **FailOnError** property of the package object. For more information, see <u>FailOnError Property</u>.

In the **DTS Package Properties** dialog box, on the **Advanced** tab, you can find:

• Use transactions.

If this check box is selected, transactions are enabled. If cleared, no package transaction is created, and requests by steps to join the transaction are ignored.

In the DTS object model, set the **UseTransaction** property of the package object. For more information, see <u>UseTransaction</u> (DTSMQMessage) Property.

• Commit on successful package completion.

If this check box is selected, updates pending in an open package

transaction are committed when a package finishes executing and one or both of the following is true:

- No steps failed.
- The **Fail package on first error** check box is cleared.

If the **Commit on successful package completion** check box is cleared, an open transaction is rolled back on package completion and pending updates are lost.

In the DTS object model, set the **AutoCommitTransaction** property of the package object. For more information, see <u>AutoCommitTransaction</u> <u>Property (DTS)</u>.

• Transaction isolation level.

In this check box, you can select the level of locking used within transactions to protect the user from dirty reads, nonrepeatable reads, and phantom data. In order of increasing protection, available isolation levels are: **Chaos, Read uncommitted, Read committed, Repeatable read,** and **Serializable**. For more information, see <u>Isolation Levels</u>.

In the **Workflow Properties** dialog box of a step, on the **Options** tab, you can find:

• Join transaction if present.

If this check box is selected (and transactions are enabled), the step joins the package transaction. Updates accumulate until commit or rollback. If cleared, updates are carried out one at a time, as they are requested.

In the DTS object model, set the **JoinTransactionIfPresent** property of the step object. For more information, see <u>JoinTransactionIfPresent</u> <u>Property (DTS)</u>.

• Rollback transaction on failure.

If this check box is selected, step failure triggers a rollback of the package transaction. Pending updates are discarded. If cleared, any

updates remain in the transaction until a later commit or rollback.

In the DTS object model, set the **RollbackFailure** property of the step object. For more information, see <u>RollbackFailure Property (DTS)</u>.

• Commit transaction on successful completion of this step.

If this check box is selected, successful step completion triggers a transaction commit. Pending updates are made permanent. If cleared, any updates remain in the transaction until a later commit or rollback.

In the DTS object model, set the **CommitSuccess** property of the step object. For more information, see <u>CommitSuccess Property (DTS)</u>.

See Also

Package Object (DTS)

Package2 Object (DTS)

Step Object (DTS)

DTS Transaction Fundamentals

Over the course of the execution of a single package, transactions are initiated, joined, and then committed or rolled back. After a commit or rollback operation, the cycle may repeat.

If the Data Transformation Services (DTS) package includes Execute Package tasks, transaction behavior can differ from that described in this topic. For more information about transactions and the Execute Package task, see <u>Inherited Transactions</u>.

Initiating a New Package Transaction

No package transaction exists until a step attempts to join it. At this point, a new transaction is created for the package, and the step proceeds with its attempt. If other steps attempt to join the package transaction before the first transaction has committed or rolled back, they are enlisted in the first transaction. Although a package may initiate several transactions, only one package transaction can be active at a time.

Joining the Package Transaction

The attempt to join the package transaction takes place only after any workflow script has been processed. If a step joins the package transaction, any updates made by the step accumulate in the package transaction. If a step does not join the package transaction, database changes are committed in autocommit mode: one at a time, as they are requested.

In order to join a transaction successfully, the package step must:

- Be one of several supported task types. For example, the Execute SQL task is supported, but the Send Mail task is not.
- Use supported connection types on outputs. For example, an instance of Microsoft® SQL Server[™] 2000 is supported, but a connection to a Microsoft Excel 2000 worksheet is not.

If the preceding conditions are not met, the attempt to join the package transaction fails, and the package halts at runtime. For more information about supported task and connection types, see <u>Supported Task Types</u> and <u>Supported Connection Types</u>.

In DTS Designer, a step attempts to join the package transaction if you:

- Select the **Use transactions** check box in the **DTS Package Properties** dialog box.
- Select the **Join transaction if present** check box in the **Workflow Properties** dialog box of a step.

Note When a step joins the package transaction, each connection used by the step is enlisted in the distributed transaction. All updates for such a connection accumulate in the package transaction, even if they originate in a step that did not explicitly join the package transaction. Therefore, to make transactional and non-transactional updates to the same database from one package, you must use two connections.

Committing and Rolling Back Package Transactions

When a package transaction is committed, any accumulated updates are made permanent. When a package transaction is rolled back, any accumulated updates are reversed.

A package transaction is committed when either of the following events occurs:

- A step completes successfully and the **Commit transaction on successful completion of this step** check box is selected.
- The package completes successfully and the **Commit on successful package completion** check box is selected.

The current package transaction is rolled back when any of the following events occur:

• The package fails.

- The package finishes and the **Commit on successful package completion** check box is cleared for the package.
- A step fails and the **Rollback transaction on failure** check box is selected for the step.

Note Some operations can leave the current transaction in an invalid state (for example, failure during a commit or rollback or a rollback in a subpackage). Attempts to join or commit an invalid transaction fail the package. To terminate the invalid transaction and so allow a new package transaction to start, trigger a rollback in the controlling package.

Inherited Transactions

The Execute Package task allows you to execute a Data Transformation Services (DTS) package as one step of a parent package. This subpackage may create its own package transactions, or it may inherit the parent package transaction.

A package inherits the parent package transaction if both of the following are true:

- The package is invoked by an Execute Package task.
- The Execute Package task that invoked the package also joined the parent package transaction.

In the following diagram, there are six packages that all use transactions. Each package contains numerous tasks. Only the Execute Package tasks are shown. Package A executes packages B and C, which in turn execute packages D, E, and F. The Execute Package tasks that join the package transactions are indicated with an underline.

Packages A, B, and D execute in one package transaction. Packages C and F execute in a second transaction. Package E gets its own separate package transaction.

Packages A, C, and E control their own transactions. Packages B, D, and F inherit their transactions.

If a package runs with an inherited transaction, transaction behavior differs considerably:

- No new package transaction is initiated. Steps join the inherited parent transaction.
- No commit takes place. In particular:
 - If the **Commit transaction on successful completion of this step** check box is selected, it is ignored.

- If the **Commit on successful package completion** check box is selected, it is ignored.
- No rollback takes place on package completion, even if the package fails. However, individual steps may roll back the parent package transaction if you select the **Rollback transaction on failure** check box in the **Workflow Properties** dialog box.

Note If a subpackage fails, its parent Execute Package task fails. If a subpackage completes successfully, its parent Execute Package task completes successfully. If a subpackage experiences many errors or rolls back the package transaction, but its **Fail package on first error** property is cleared, it will complete successfully. Its parent Execute Package task also will complete successfully.

See Also

Execute Package Task ExecutePackageTask Object

Supported Connection Types

In order to take part in a Data Transformation Services (DTS) package transaction, the data source of a connection must support distributed transactions. A step that attempts to join the package transaction with an unsupported connection, (for example, a Microsoft® Access 2000 table) fails at runtime. The following connection types, available in DTS Designer, can provide this support:

- Microsoft OLE DB Provider for SQL Server
- ODBC data source

The ODBC driver must support the connection attribute SQL_ATT_ENLIST_IN_DTC and this attribute must be set. For more information, see the ODBC documentation.

• Microsoft Data Link

Microsoft Data Link is used to access any installed OLE DB provider. An OLE DB provider must implement the **ITransactionJoin** interface if it is to join a distributed transaction. For more information, see the OLE DB documentation.

If your application requires tasks that access an unsupported connection, those tasks cannot join the DTS package transaction. However, failure in such a task still can be used to roll back an open transaction.

Supported Task Types

There are differences in how the custom tasks available through Data Transformation Services (DTS) participate in transactions. For example:

- Some tasks perform operations that cannot take part in a transaction. For example, the File Transfer Protocol (FTP) task writes files directly to disk. No rollback is possible. Other such tasks are:
 - The Dynamic Properties task.
 - The Send Mail task.
 - The Copy SQL Server Objects task.
- Some tasks allow the user to create their own scripts or programs and run them from DTS. Although these tasks can create and independently manage their own local or distributed transactions, they have no access to the DTS package transaction. These tasks are:
 - The Microsoft® ActiveX® Script task.
 - The Execute Process task.
- Some supported tasks can join the package transaction if the right connections are supported. These tasks are:
 - The Bulk Insert task.
 - The Data Driven Query task.
 - The Transform Data task.
 - The Execute Package task.

- The Execute SQL task.
- The Message Queue task.

All of the above three groups of tasks may commit or roll back the current transaction.

See Also

Bulk Insert Task Data Driven Query Task Execute SQL Task Inherited Transactions Message Queue Task Transform Data Task

Designing DTS Transactions

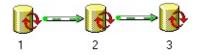
It is strongly recommended that you follow a few general design principles when you design Data Transformation Services (DTS) transactions to greatly reduce the chance of anomalous results:

- If possible, organize tasks sequentially.
- When steps execute in parallel, use DTS package failure to roll back the transaction.
- Use checkpoint tasks to commit intermediate results within a package.
- Use the Execute Package task and package failure to branch on transaction failure.

Sequential Execution

For the simplest Data Transformation Services (DTS) package transaction configuration, organize steps sequentially. If you do this, you can roll back transactions at the step level.

The following diagram shows a sequential ABC package.



Three Execute SQL tasks are arranged in order with precedence relationships. Only on the success of one step is the following task started. If any one of the tasks fails, no more steps execute. The sequential organization of the package allows you to roll back the transaction immediately on failure.

All three tasks join the package transaction. If any task fails, the transaction rolls back, and the package halts. If task C completes successfully, the entire transaction is committed.

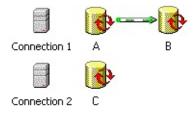
To run all three tasks as part of a single package transaction, do the following:

- Select the **Use transactions** check box for the package.
- Select the **Join transaction if present** check box for each step.
- Select the **Rollback transaction on failure** check box for each step.
- Select the **Commit transaction on successful completion of this step** check box for task C.

Parallel Execution

Time or resource constraints may require that Data Transformation Services (DTS) tasks execute in parallel. As a result, transaction configuration becomes more complex. To avoid anomalous results, use DTS package failure to roll back the transaction when steps execute in parallel.

The following diagram shows a parallel ABC package.



All three tasks still join the same transaction, but now, tasks A and C are supposed to start simultaneously. In this situation, DTS behavior differs, depending on the connections used by these tasks:

- If tasks A, B, and C all use the same connection, DTS will serialize their execution in spite of the parallel construction. Precedence relationships are enforced, but otherwise, order of execution is undefined.
- If tasks A and B use Connection 1 and task C uses Connection 2, then:
 - If Connection 1 and Connection 2 are on the same instance of Microsoft[®] SQL Server[™] 2000, the package fails when the second task attempts to join the transaction.

Therefore, you must use precedence relationships or the **Execute on main package thread** workflow property to ensure that no two tasks execute simultaneously as part of the package transaction.

• If Connection 1 and Connection 2 are not on the same instance of SQL Server 2000, tasks A and C execute in parallel, as expected.

Rollback and the Package Transaction

If you do not attend carefully to package configuration, your package may produce unanticipated results. For example, in the preceding diagram, you want all three tasks to join the same transaction. However, incorrect settings for transaction properties could result in tasks A and C rolling back, while changes made by task B are committed. For example, consider what happens if tasks A and C start simultaneously and then task C fails and rolls back before A completes:

- Task A is not canceled in mid-execution but continues to its normal conclusion.
- Any changes made by task A or C are rolled back after task A completes.
- The rollback has no effect on the success or failure status of task A. If task A encounters no problems, it completes successfully as usual.
- If the package is not configured to fail on the first error, task B will commence as usual on the successful completion of task A. Because there is no active transaction when task B starts, a new package transaction is created. If the **Commit on successful package completion** check box is selected, changes made by task B will be committed in spite of the earlier failed transaction.

Enforcing a Single Package Transaction

If a transaction includes several tasks executing in parallel, when one task fails, any changes must roll back and execution must stop. New tasks must not commence. To enforce a single package transaction when multiple tasks may be active, fail the package and then roll back the transaction on package failure.

Configure the Parallel ABC package as follows:

• Select the **Fail package on first error** check box for the package.

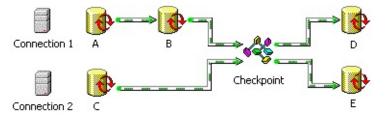
- Select the **Use transactions** check box for the package.
- Select the **Commit on successful package completion** check box for the package.
- Select the **Join transaction if present** check box for each step.
- Clear the **Commit transaction on successful completion of this step** check box for each step.
- Clear the **Rollback transaction on failure** check box for each step.

Selecting the **Fail package on first error** check box triggers an unsuccessful package completion as of the first step failure. As a consequence, no more tasks are started, and any updates in the existing package transaction are rolled back.

Checkpointing Package Transactions

Sometimes, work naturally falls into two or more transactions. You can use checkpoint tasks to commit intermediate results within a Data Transformation Services (DTS) package.

In the following diagram of a package, Checkpoint is a Microsoft® ActiveX® Script task. It functions as a placeholder, ensuring only that the package transactions operate properly. It is necessary in this case because there is no other place to commit the transaction without introducing potential problems. If there were a job that followed tasks B and C and took part in the same transaction, that task could carry out the checkpoint function.



Tasks A, B, and C execute as part of a single transaction. If no tasks fail, the Checkpoint task commits the new updates after both task B and task C finish. Tasks D and E then join a second package transaction, to be committed on successful package completion.

Configure Checkpoint workflow properties as follows:

- Clear the **Join transaction if present** check box.
- Select the **Commit transaction on successful completion of this step** check box.

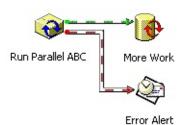
For more information about configuring the package and the remaining tasks, see <u>Parallel Execution</u>.

Branching on Transaction Failure

When tasks execute in parallel, you often have to fail the Data Transformation Services (DTS) package in order to prevent anomalous results. This poses a problem if you require an action on failure (for example, the dispatch of an e-mail message). Within a package, you cannot simultaneously fail the package and run extra steps.

Use the Execute Package task and package failure to conditionally run extra steps after transaction failure.

The following diagram shows a package executing the Run Parallel ABC package. Depending on the outcome, the package either continues processing or sends an error message. Although the Run Parallel ABC task appears in this example, the same procedure can be used to branch on the failure of any package, even those that do not use transactions.



The first step in the package executes the Parallel ABC package as a subpackage. The failure of step A, B, or C triggers subpackage failure, which in turn fails the Run Parallel ABC step. On failure of this step, Error Alert sends the e-mail message. If the subpackage completes successfully, however, Run Parallel ABC succeeds and More Work is commenced.

For the example, configure package properties as follows:

- Select the **Use transactions** check box.
- Clear the **Fail package on first error** check box.

Configure More Work workflow properties as follows:

• Select the **Join transaction if present** check box.

- Select the **Commit transaction on successful completion of this step** check box.
- Select the **Rollback transaction on failure** check box.

If Run Parallel ABC and More Work are to join the same transaction, configure Run Parallel ABC workflow properties as follows:

- Select the **Join transaction if present** check box.
- Clear the **Commit transaction on successful completion of this step** check box.
- Select the **Rollback transaction on failure** check box.

If Run Parallel ABC and More Work are to run in separate transactions, with changes committed after the first, you can delegate transaction processing to the subtask. To do this, configure Run Parallel ABC workflow properties as follows:

- Clear the **Join transaction if present** check box.
- Clear the **Commit transaction on successful completion of this step** check box.
- Clear the **Rollback transaction on failure** check box.

Lookup Queries

A feature of the Transform Data and Data Driven Query tasks, lookup queries allow you to run queries and stored procedures against other connections besides the source and destination. For example, by using a lookup query, you can make a separate connection during a query and include data from that connection in the destination table.

Lookup queries allow you to customize and execute an SQL statement from within a Microsoft® ActiveX® script transformation. The statement may be a stored procedure invocation, or a SELECT, INSERT, UPDATE, or DELETE statement. You customize these statements for each source row through the use of parameters, blanks left in the statement to be filled in before execution. When you execute a lookup query, your script provides values to be substituted for each parameter. Your results can be loaded into destination columns or can serve as input for further script processing.

You can use lookup queries to:

- Look up tabular information.
- Perform parallel updates on two database systems.
- Validate input data before loading it.
- Invoke stored procedures in response to input conditions.
- Use global variable values as query parameters.

You can use either Data Transformation Services (DTS) Designer or the DTS object model to create and manage lookup queries. For more information about programming with lookup queries, see <u>Adding DTS Lookups and Global</u> <u>Variables</u> and <u>Lookup Object (DTS)</u>.

See Also

Data Driven Query Task

Transform Data Task

Configuring a Simple Lookup Query

To configure a simple lookup query, complete the following steps:

1. Create a new connection.

Although a lookup query can share a connection with the source or destination under certain conditions, the best performance occurs when it is given its own connection.

2. Create and name your parameterized query.

The parameterized query can be a stored procedure invocation or a SELECT, INSERT, DELETE, or UPDATE statement. Mark parameters by including question marks in place of expressions that will be set at runtime. Parameter values typically come from source data but may be supplied by global variables or any other terms accessible to Microsoft® ActiveX® script code.

3. Create an ActiveX Script transformation with code to execute your query.

The query is executed with the following statement:

return value = DTSLookups("query name").Execute(argument list)

where *return value* is a variant that receives the result of the query, *query name* is the name you provided the query in step two, and *argument list* is a comma-separated list of parameter values, one for each question mark in the query.

Before you configure a lookup query, consider the following:

- Tasks that avoid lookup queries run much more quickly than those that use them. For example:
 - A task that joins two tables in a source SQL query runs faster than one that looks up the information from the second table.
 - Two separate Transform Data tasks usually can be run in less

time than one that inserts data in a second table with a lookup query.

- In general you should use a lookup query only in situations where there is no alternative. For example:
 - A source join is impossible because data resides in non-SQL Server[™] databases, or the volume of data generated by a source join would exceed system capacity.
 - A stored procedure must be called, or a DELETE or UPDATE query must be run.
 - The need for clarity outweighs any performance issues.

Lookup Query Example

In this example, you have source data that includes a postal code but no city. A **Mail Codes** table contains a row for each postal code and a column for city name. The procedure for including this city name in your destination rows is as follows:

- 1. Create and name a connection to the database containing the **Mail Codes** table.
- 2. Write your query statement, leaving a question mark in place of the postal code value:SELECT City FROM MailCodes WHERE PostalCode = ?

When you configure the lookup query, you are required to provide a connection name and a query name. Use the connection you created in step one, and name your query GetCity.

3. In an ActiveX Script transformation, include the following code to execute your query and place the resulting city name in the destination row:

DTSDestination("City") = DTSLookups("GetCity").Execi

The postal code is drawn from the source row. Its value replaces the question mark each time the query is executed.

Using More Than One Argument

Sometimes a lookup query takes more than one argument. For example, when:

- Required information has a multiple column key.
- An INSERT or UPDATE statement must fill multiple columns.

To configure a lookup query with multiple arguments, you must:

- Include multiple parameters in your query statement.
- Provide values for each parameter when you execute the query.

In this example, you need to retrieve a city name, given the postal code and country. The GetInternationalCity query has two parameters:

SELECT City FROM MailCodes WHERE PostalCode = ? AND Coun

In your ActiveX script, values are provided for the postal code and the country:

DTSDestination(City) = DTSLookups("GetInternationalCity").Execut (DTSSource("PostalCode"), DTSSource("Country"))

Looking Up More Than One Value

Sometimes you want to retrieve multiple values with a single lookup (for example, when you have a customer account number and need a name and address).

Data Transformation Services (DTS) handles multiple columns in query results by returning an array of variants. Each entry in the array holds one result value. The index of the first value is 0.

In this example, you need to retrieve a city and a region, given the postal code. The GetCityAndRegion query selects both required columns:

SELECT City, Region FROM MailCodes WHERE PostalCode = ?

The returned values are accessed through the following ActiveX script code:

```
dim varArray
varArray = DTSLookups("GetCityAndRegion").Execute(DTSSource(
DTSDestination("City") = varArray(0)
DTSDestination("Region") = varArray(1)
```

See Also

Data Driven Query Task DTS Connections Transform Data Task Using ActiveX Scripts in DTS Using Parameterized Queries in DTS

Managing Zero or Multiple Result Rows in Lookup Queries

When you run a lookup query, Data Transformation Services (DTS) always returns the first row in the result set. Although you are not given access to succeeding rows, you can find out how many rows were returned. This can be useful when a query returns zero or multiple rows.

Result Sets With Zero Rows

Lookup queries sometimes fail to return any rows. For example, if you are tracking the number of vacation days your employees have taken, you might find that some have not taken any vacation days.

When a lookup query retrieves zero rows, DTS returns an empty variant. In Microsoft® ActiveX® code, you can test for this condition with the IsEmpty() function.

For example, the VacationDays query returns zero rows if the given employee has taken zero vacation days:

SELECT EmployeeID FROM VacationDay WHERE EmployeeID = ?

Using this query, you can skip over employees who have not taken any vacation days by using the following ActiveX script code:

Dim LookupResults

```
LookupResults = DTSLookups("VacationDays").Execute(DTSSource
If IsEmpty(LookupResults) Then
```

```
Main = DTSTransformStat_SkipRow
```

Else

```
Main = DTSTransformStat_OK
End If
```

Note If all data resides on a computer running an instance of Microsoft SQL Server[™], performance can be improved by using a source query, instead of a

lookup query, to filter out unwanted rows.

Result Sets With Multiple Rows

Lookup queries sometimes return many rows. For example, an employee may have taken many vacation days.

When a lookup query retrieves multiple rows, DTS discards all but the first row. If this is acceptable, you can use the ORDER BY phrase in your query to bring the most important row to the top of the results. In any case, the number of rows returned is accessible through the **LastRowCount** property of the lookup query.

For example, suppose you want to prepare a vacation day summary. The RecentVacationDays query lists vacation days for one employee with the most recent vacation day in the first row:

SELECT VacationDate FROM VacationDay WHERE EmployeeID = ? ORDER BY VacationDate DESC

Then, the most recent vacation day date and the total number of vacation days per employee are accessed with the following ActiveX script code:

DTSDestination("LastVacationDate") = DTSLookups("RecentVacationDate") = DTSLookups("RecentVacationDays") = DTSLookups("Re

If an employee has taken zero vacation days, the above code nulls the LastVacationDate and sets NumberOfVacationDays to zero.

See Also

Using ActiveX Scripts in DTS

Using Lookup Queries to Modify Data

Lookup queries are not limited solely to SELECT statements. INSERT, DELETE, and UPDATE statements, as well as stored procedure invocations, all can appear in lookup queries.

The UpdateEmployee query updates a value in the Employee table when you provide it an EmployeeID:

UPDATE Employee SET HasTakenVacation = 0 WHERE EmployeeII

Execute this query for every row in the source rowset with the following Microsoft® ActiveX® script code:

DTSLookups("UpdateEmployee").Execute(DTSSource("EmployeeII

Note An INSERT query can be used in conjunction with a Transform Data task to split a source rowset, sending rows to two different tables. However, when both tables reside in Microsoft SQL Server[™] databases, this practice results in longer execution times than sequentially running two Transform Data tasks with fast load enabled.

See Also

Using ActiveX Scripts in DTS

Using Multiple Lookup Queries

Data Transformation Services (DTS) places no limitations on the number of lookup queries that can appear in a single Microsoft® ActiveX® Script transformation. Therefore, you can use multiple lookup queries to:

- Add, change, or delete an account, depending on values in a source transaction table.
- Carry out an update only after a query returns successfully.
- Look up dimension table keys, given corresponding values from an online transaction processing (OLTP) system.

Carrying Out an Update

Two queries, VacationDays and UpdateEmployee, can be combined to update only those employees who have taken no vacation days:

Dim LookupResults

```
LookupResults = DTSLookups("VacationDays").Execute(DTSSource
If Not IsEmpty(LookupResults) Then
```

DTSLookups("UpdateEmployee").Execute(DTSSource("Employee End If

Looking Up Dimension Table Keys

Data warehouse dimension tables are often keyed with identity fields. These fields do not necessarily occur in the OLTP system that provides facts for the data warehouse. For example, in an OLTP system, the product might be keyed by the **SKU** field, whereas in the data warehouse, it is keyed by the automatically generated **ProductID**. Before a new **SalesFact** can be inserted, the **SKU** field must be used to look up the corresponding value of **ProductID** in the data warehouse. In the following example code, the GetProductID query retrieves a **ProductID**, given an **SKU** passed in as a parameter:

SELECT ProductID FROM Product WHERE SKU = ?

In the same way, a **CustomerID** can be retrieved given an account number. Here is the GetCustomerID query:

SELECT CustomerID FROM Customer WHERE AccountNumber = ?

Fill the **ProductID** and **CustomerID** columns in the **SalesFact** table with the following ActiveX script code:

DTSDestination("ProductID") = DTSLookups("GetProductID").Execute(DTSSource("SKU")) DTSDestination("CustomerID") = DTSLookups("GetCustomerID").Execute(DTSSource("AccountNu

To look up additional values, (for example, the StoreID), add another query and another line of script code.

For more information about queries that appear in this example, see <u>Managing</u> <u>Zero or Multiple Result Rows in Lookup Queries</u> or <u>Using Lookup Queries to</u> <u>Modify Data</u>.

Using Global Variables with DTS Packages

When you execute a Data Transformation Services (DTS) package, you can save data or a value from a DTS step to pass on to subsequent steps. For example, you can use the saved data to change the way a subsequent step executes or to dynamically modify a SELECT statement.

When you use global variables with DTS tasks, you can:

• Set a global variable to the accounting month-end date, according to the fiscal calendar of the accounting department. You can do this by using a Microsoft ActiveX® Script task to retrieve the values or by setting the global variable to the accounting month-end dates during design time and referencing during package execution. A step inside the package can use that date to determine the existence of a file with that date in its name. If it exists, the step inserts the records from that file into a table.

Alternatively, the global variable can be used in a WHERE clause to determine the specific records to be loaded (for example, only loading up records that were modified on that month-end date).

• You can populate multiple global variables with data in one pass with the Execute SQL task. Entire rowsets also can be saved in a single global variable and accessed as a disconnected ActiveX Data Objects (ADO) recordset. For more information, see <u>Execute SQL Task</u>.

You can then use the global variable as an in-memory lookup table. With the Execute SQL task, issue a SELECT statement against a **state** table and store the results in a global variable. Then, with the Transform Data task, for each source row, call a "StateLookup" function that iterates through the global variable recordset and matches the state code in the source field with the state code in the table. When there is a match, store the full name of the state in the destination column. For more information, see Lookup Queries.

Creating Global Variables

You can create global variables in DTS and assign them values in the following

ways:

- During design time by using:
 - The **DTS Package Properties** dialog box in DTS Designer.
 - The **Execute SQL Properties** dialog box. For more information, see <u>Execute SQL Task Properties</u>.
- Dynamically during package execution by:
 - Using an ActiveX script.
 - Issuing a **dtsrun** command prompt utility from the command prompt and using the /A command switch to allocate and initialize global variables.

Scope of Global Variables

Scope refers to the lifetime of the variable reference in memory. The scope depends on where the variables are declared or initialized. Whether a global variable is still accessible after a package has executed depends on how the global variable was created.

A global variable created during design time retains the value it had when the package finished execution, if the package is saved. For example, suppose you create the global variable, "*city*," and set it to the value of "*Boston*." During package execution, an ActiveX script changes the value of "*Boston*" to "*Philadelphia*." The next time you execute the package, the global variable will contain "*Philadelphia*," not "*Boston*." This is useful if you want to query the value of a package global variable after execution.

However, global variables created dynamically in an ActiveX script have two scopes. If they are created above the Function Main(), they are available to all functions in the script. This is the equivalent to module-level scope. If they are declared within a function, they are available only inside that function. This is equivalent to procedure-level scope.

Examples of Using Global Variables in DTS Packages

The following examples show you how to create, set, retrieve, and use the values of global variables in a DTS package by using an ActiveX script.

Creating a Global Variable Dynamically from an ActiveX Script

If a global variable does not exist when the package is run, you can create one dynamically by using an ActiveX script. To create a new global variable called "*city*" and assign it a value of "*Boston*", use the following Microsoft Visual Basic® Scripting Edition (VBScript) code:

```
DTSGlobalVariables("city").value = "Boston"
```

You can dynamically create a COM object and store it in a global variable from within an ActiveX script. In the following example, VBScript code is used to create an ADO connection, which can be used by scripts in the package to execute SQL commands and examine ADO recordsets. In this example, a year-to-date sales table containing a **Totals** field in the **pubs** database is created:

```
Function Main()
dim conn
set DTSGlobalVariables("MyConn").value = CreateObject("ADODE
```

```
set conn = DTSGlobalVariables("MyConn").value
conn.provider="sqloledb"
conn.open "(local)", "sa", ""
conn.DefaultDatabase = "pubs"
conn.execute("Create Table YTDSales (Totals int)")
Main = DTSTaskExecResult_Success
End Function
```

The following code example shows you how to create the year-to-date **sales** table using Microsoft JScript®:

```
function Main()
{
```

DTSGlobalVariables("MyConn").value = CreateObject("ADODB.Con conn = DTSGlobalVariables("MyConn").value;

```
conn.open = ("provider = sqloledb; data source = (local);user id = sa");
conn.DefaultDatabase = "pubs";
conn.execute("Create Table YTDSales (Totals int)");
```

```
return(DTSTaskExecResult_Success)
}
```

Setting the Value of a Global Variable Dynamically from an ActiveX Script

The following ActiveX script code, written in VBScript, sets the value of a global variable named count to 200:

DTSGlobalVariables("count").value = 200

Getting a Global Variable Dynamically from an ActiveX Script

The following ActiveX script code, written in VBScript, gets the value of a global variable named *count* and saves the value in a variable named *globalCount*:

globalCount = DTSGlobalVariables("count").value

Using a Global Variable Dynamically from an ActiveX Script

The following code concatenates the value of a column containing a file name with a global variable containing the Julian date, and stores the new results in the filename column in a destination table:

```
DTSDestination("FileName") = DTSSource("FileName") & DTSG
```

See Also

Adding DTS Lookups and Global Variables

dtsrun Utility

Using Parameterized Queries in DTS

Parameterized queries are SQL queries written for reusability. They contain parameter markers as placeholders for data that will change from execution to execution. In the Data Transformation Services (DTS) tasks that use parameterized queries, the placeholder syntax is a question mark. The following is an example of a parameterized query:

INSERT INTO Account (CustomerID, CompanyName) VALUES (?, ?

The following DTS tasks make use of parameterized queries:

- Execute SQL task
- Data Driven Query task
- Transform Data task

Lookup queries, which you can include in Microsoft® ActiveX® script transformations in a Data Driven Query or Transform Data task, make use of parameters to retrieve information from an additional connection. For more information, see <u>Lookup Queries</u>.

Input Parameters to DTS Tasks

All the tasks above can execute SQL queries written with parameters if the source that the query is running against supports it. You can map variables into the SQL parameters. The Data Driven Query task can bring in data from a text file, global variable, or the source data as input to its parameter set. The Transform Data task can use only global variables as input to source data queries. Lookup queries can use data from a text file, global variable, or other source data fields as input. However, the Execute SQL task can use only global variables as input to its parameterized queries. For more information, see Data Driven Query Task, Transform Data Task, and Lookup Queries.

Output Parameters to DTS Tasks

The Execute SQL task can save the results of a query to a global variable. You can use the task to save the data in several formats. For more information about these formats, see <u>Execute SQL Task</u>.

The Transform Data and Data Driven Query tasks can save query results into a table destination column, or a variable. You can also save data to an array when using a Lookup query.

Querying a DTS Package from External Sources

Data Transformation Services (DTS) package data can be made available to an external source, such as SQL Query Analyzer, by:

- Querying a package step associated with a transformation with the Transact-SQL OPENROWSET statement.
- Defining the package as a linked server and joining package data in a distributed query.

When querying package rowset data, the following conditions apply:

- The package supplying the data must be launched by the application getting the package data. You do not execute the package to send the data to the requesting application or process.
- You can only query a package step associated with a Transform Data task.
- The package must have destination columns to bind to; the destination cannot be a text file.

In DTS Designer, you make package data available by selecting the **DSO rowset provider** check box (on the **Options** tab of the **Workflow Properties** dialog box) for a package step associated with a Transform Data task. The data from that task then becomes available to an external data consumer.

Enable the **DSO rowset provider** check box only for packages that you intend to query. After the option is set, the package step where you set the option does not complete execution when the package is run normally. While the flag is set that pump task can only be accessed through OPENROWSET.

Note These methods are used typically to query packages from an external source; however, you can also query other packages from within a package by issuing OPENROWSET queries and distributed queries in an Execute SQL task

or as the source for another Transform Data task.

See Also

OPENROWSET

Transform Data Task

Using SQL Query Analyzer

Querying a Package with OPENROWSET

You can run queries against a Data Transformation Services (DTS) package by using a Transact-SQL OPENROWSET statement.

To prepare a package to serve as a data source, select the **DSO rowset provider** check box (on the **Options** tab of the **Workflow Properties** dialog box) for a package step.

Before querying a package with OPENROWSET, consider the following:

- DTS uses its own OLE DB provider, DTSPackageDSO. When you use OPENROWSET, you only specify its *provider_name*, *provider_string*, and *query* arguments:
 - Use 'DTSPackageDSO' for the *provider_name* argument.
 - Use any combination of **dtsrun** command switches necessary to describe the package for the *provider_string* argument. For more information about the **dtsrun** command switches, see <u>dtsrun Utility</u>.
 - Use either 'SELECT *', 'SELECT * FROM <package name>', or 'SELECT * FROM <package step name>' for the *query* argument.
- If you select the **DSO rowset provider** option for more than one step in a package, you need to specify the package step name in the OPENROWSET *query* argument. The package step name can be copied from the **Options** tab of the **Workflow Properties** dialog box for the step whose data you want to query.
- If you saved multiple versions of the package, you can specify the package version using the **dtsrun** command switch /vpackage_version_guid_string. If you do not specify a package version, the last-saved version is used.

For more information about using the Transact-SQL OPENROWSET statement, see <u>OPENROWSET</u>.

Querying a Package Saved to a File

Suppose you want to use SQL Query Analyzer to query a package saved to the package Dso.dts on your local server. Use the following query statement:

SELECT * FROM OPENROWSET('DTSPackageDSO', '/FC:\Dts\Dso

In the OPENROWSET statement:

- The *provider_name* argument is always DTSPackageDSO, an entry in the registry that functions as the package OLE DB provider.
- The *provider_string* argument contains the structured storage file name, preceded by the /**F** dtsrun command switch.
- The *query* argument is a SELECT * statement used to pass through the rowset data.

If you selected the **DSO rowset provider** check box for more than one Transform Data task in the above package, you can use the following code to query the second package step in Dso.dts:

SELECT * FROM OPENROWSET('DTSPackageDSO', '/FC:\Dts\Dso 'SELECT * FROM DTSStep_DTSDataPumpTask_2')

Querying a Package Saved to SQL Server

In the following example, the package, Sqlpackage, is saved to an instance of Microsoft® SQL Server[™] running on your local server. To query the package using SQL Query Analyzer, use the following query statement:

SELECT * FROM OPENROWSET('DTSPackageDSO', '/Usa /P /S /N

In the OPENROWSET statement:

• The *provider_name* argument is always DTSPackageDSO.

- The *provider_string* argument contains the following **dtsrun** command switches: /**U** for the user ID, /**P** for the password (blank here), /**S** for the network name of the server (if the server is local, *server_name* can be omitted, as shown here), /**N** for the name of the SQL Server package.
- The *query* argument is a SELECT * statement used to pass through the rowset data.

If you save multiple versions of this package and want to reference a specific version, you need to include the version globally unique identifier (GUID) string after the **/V dtsrun** command switch. The version GUID can be obtained from the **General** tab of the **DTS Package Properties** dialog box. Use the following code to querying a version of the package described above:

SELECT * FROM OPENROWSET('DTSPackageDSO', '/Usa /P /S /Nsqlpackage /V{3C904BA2-4E83-11D2-BB38-00C04 'Select *')

Issuing Distributed Queries Against Package Data

You can register a Data Transformation Services (DTS) package as a linked server and issue a distributed query against the package. This capability allows you to consolidate data from diverse sources (for example, from Oracle and DB2 data sources) in a single package, transform that data, and expose the results of the transformed distributed query to any outside data consumer.

To issue a distributed query against package data, you need to define the package as a linked server through the **sp_addlinkedserver** stored procedure. The following example code illustrates how to use **sp_addlinkedserver** against a DTS package:

sp_addlinkedserver 'DTSOLEDBPkg', 'PackageName', 'DTSPackageD

In the sp_addlinkedserver command:

- DTSOLEDBPkg is the name of the linked server you want to create.
- PackageName is the product name of the OLE DB data source; in this context, you can provide any name or a null string.
- DTSPackageDSO is the name of the DTS package OLE DB Provider.
- The last argument specifies the location of the file, DTS01.dts.

After you have defined the package as a linked server, you can execute distributed queries that include the package as a data source. Following is an example of a distributed query. It performs a join operation on the **Orders** table in the **Northwind** database with a package that gets data from a **Customer** table on an Oracle server. The query assumes the **Orders** and **Customers** tables have a common key, which is **CustomerID**.

SELECT a.OrderID, a.CustomerID, a.OrderDate, b.Companyname, b.I FROM Orders AS a, dtsLink...packageNameOracle AS b WHERE a.CustomerID = b.CustomerID In the above query, packageNameOracle is the DTS package name. However, you can use a package step name when multiple steps in a package serve as data sources.

See Also

Configuring Linked Servers

Configuring OLE DB Providers for Distributed Queries

sp_addlinkedserver

Sharing Meta Data

Data Transformation Services (DTS) Designer provides features for saving package meta data and data lineage information to Microsoft® SQL ServerTM 2000 Meta Data Services. You can store catalog meta data for databases referenced in a package and accounting information about the package version history of a particular row of data for your data mart or data warehouse.

DTS Designer uses its own information model, the DTS Information Model, for structuring package meta data and data lineage information and saving it to Meta Data Services.

To browse the data lineage and meta data information generated by DTS Designer, use the DTS Browser found in SQL Server Enterprise Manager (available through the console tree, under **Data Transformation Services**, in the **Meta Data** node). This tool allows you to explore the meta data and version history of a package and to look up the specific package version that generated a row of data.

For more information, search under Meta Data Services at Microsoft Web site.

See Also

Saving a DTS Package to Meta Data Services

DTS Information Model

Data Transformation Services (DTS) uses the DTS Information Model to persist data transformation meta data in Microsoft® SQL Server[™] 2000 Meta Data Services. The DTS Information Model is based on the Transformations package of the Open Information Model (OIM).

The DTS Information Model describes:

- The data transformations and how they are grouped into larger execution units.
- The types of data accessed.

Specifically, the DTS Information Model allows:

- The storage of data transformation meta data in one well-defined location. Storing this information in Meta Data Services allows existing transformations to be reused when a data warehouse or data mart is being rebuilt.
- The sharing of data transformation information across multiple tools. This allows the use of tools from different vendors during the building and maintenance of data warehouses.

For more information, search under Meta Data Services at Microsoft Web site.

Upgrading Meta Data Services Information Models

When you upgrade to SQL Server 2000, you must also upgrade the Meta Data Services Information Models in order to save and retrieve DTS package versions to and from Meta Data Services. Otherwise, you will receive an error when you save to Meta Data Services.

Upgrading the information models modifies the Meta Data Services table structure to support additional functionality and features provided by SQL Server 2000 Meta Data Services. In an upgrade, existing repository data is preserved in the new table structure.

Before you upgrade the information models, install SQL Server 2000. SQL Server 2000 contains the most recent DLL and EXE versions of **Insrepim**, the model installation program.

To upgrade information models, you need the following:

- The SQL Server 2000 CD or an equivalent installation directory that contains the information models required by DTS. To locate the information models, search for ***.rdm** files on the SQL Server 2000 CD.
- Insrepim.exe, the model installation program that creates or updates the Meta Data Services tables and installs information models. When you upgrade to SQL Server 2000, this program is installed on your computer.
- A batch file to install the models. This file must reside in the directory that contains the insrepim.exe file. After you create the batch file, run it from a command prompt.

Creating an Installation Batch File

Replace the *<placeholder>* values in the following batch file text with real values that apply to your system. For example:

- *<path>* must be the path to the CD or to the installation directory.
- *<servername>* must the name of the SQL Server.
- *<sa>* must be the SQL Server system administrator login. If you are using Windows Authentication, do not specify this parameter.
- *<password>* must be the SQL Server system administrator password. If you are using Windows Authentication, do not specify this parameter.

Information models must be installed in the exact order shown below. The lines headed with REM are comments:

```
REM
     Usage:
             InsRepIM.exe
REM
             /f[Model File] /r[Repository connect string] /u[User]
    Syntax:
REM
insrepim.exe /f<path>\uml.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\umx.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\gen.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\dtm.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\dbm.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\tfm.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\dts.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\sql.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\db2.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\ocl.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\ifx.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\olp.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\mds.rdm /rserver=<servername>;database=msdb /u
insrepim.exe /f<path>\sim.rdm /rserver=<servername>;database=msdb /u
```

See Also

OIM in Meta Data Services

Recording Data Lineage in DTS

Design a plan before implementing and using data lineage information. Base your plan on factors such as auditing needs, the amount of lineage data that can be managed, performance considerations, and whether you need to track row- or column-level lineage, or both.

If you save a Data Transformation Services (DTS) package to Microsoft® SQL Server[™] 2000 Meta Data Services, you can record and track two types of data lineage:

- Row-level data lineage, which reveals the source of any piece of data and the transformations applied to that data. This data lineage can be tracked at the package and row levels of a table and provides a complete audit trail of data transformation and DTS package execution information. In Meta Data Services, an object associated with this type of lineage contains lineage values (package execution globally unique identifiers), the user name and server name for package execution, and the time of execution.
- Column-level data lineage, which provides information about a package version and the database tables and columns the package uses as a source or destination (if any). You can browse packages and package versions that use a specific column as a source or destination, and check to see whether a specific column is used as a source or destination in any package saved to Meta Data Services.

Use the DTS Browser to display data lineage information. For more information, see <u>Viewing Meta Data in DTS</u>.

Implementing Row-Level Data Lineage

These are the steps for implementing row-level data lineage:

1. Connect to and import meta data from the database whose tables will be used by the package as a source or destination.

To import the meta data, use either the DTS Browser or right-click **Import Meta Data** on either the **Meta Data Services Packages** or **Meta Data** nodes in SQL Server Enterprise Manager.

- 2. Create the table columns in your data warehouse for receiving rowlevel data lineage information. DTS maintains the following lineage identifiers for each package execution:
 - A **short** (integer) value, which is the package version checksum.
 - A **long** (uniqueidentifier) value, which is the globally unique identifier (GUID) of the package version.

The short lineage value occupies less space in the destination table but is potentially less unique. Either one or both of these values need to be written to columns in your data mart or data warehouse.

3. Select the data lineage options in DTS Designer.

These options can be found on the **Advanced** tab of the **DTS Package Properties** dialog box. After you have turned on the data lineage options in a package, the two lineage values (GUIDs) are available when you configure a Transform Data task and Data Driven Query task.

4. Map the data lineage source columns to destination columns in DTS Designer.

You configure the lineage columns on the **Transformations** tab for these tasks and on the tabs of the **Transformation Options** dialog box.

- 5. Save the package containing the data lineage information to **Meta Data Services**.
- 6. Execute the package.

Implementing Column-Level Data Lineage

These are the steps for implementing column-level data lineage:

- 1. Create a DTS package containing at least one transformation task.
- 2. Save the package to Meta Data Services.
- 3. Scan the meta data. During the save operation, click **Scanning**, and in the **Scanning Options** dialog box, click **Resolve package references to scanned catalog metadata** to save the meta data, and link the package meta data to a repository database.

For more information, see <u>Importing and Saving Meta Data in DTS</u>.

Importing and Saving Meta Data in DTS

You can import meta data to Microsoft® SQL Server[™] 2000 Meta Data Services through SQL Server Enterprise Manager or by using the Data Transformation Services (DTS) scanning options. When you import meta data using SQL Server Enterprise Manager, you read the meta data from only one database. You can read the meta data from all referenced databases in a package by selecting the **Scan all referenced catalogs** check box in the **Scanning Options** dialog box.

Note For the import of meta data to work, the data provider specified must support OLE DB schema rowsets.

The OLE DB scanner for Meta Data Services is a utility that imports database schema from an OLE DB data source. The scanner is passed an OLE DB provider, examines the schema, and creates a set of corresponding instance objects in Meta Data Services.

If you save package information to Meta Data Services, you can save meta data about the databases referenced in the package, such as:

- Primary and foreign keys.
- Column type, size, precision, scale, and nullability information.
- Indexes.

You can view meta data information saved to Meta Data Services through the DTS Browser. For more information, see <u>Viewing Meta Data in DTS</u>.

See Also

Using OLE DB Scanner

Viewing Meta Data in DTS

Data Transformation Services (DTS) allows you to view meta data in SQL Server Enterprise Manager (through the console tree, under **Data Transformation Services**, in the **Meta Data** node). You can:

- Generate a hierarchical display of the meta data for any databases scanned to Microsoft® SQL Server[™] 2000 Meta Data Services, and jump to information on the DTS packages that reference the meta data. If you regularly scan changes to catalog meta data, this display provides useful historical information on changes to the meta data. You can display the properties of a package version associated with any item of meta data, and a version history of those packages.
- Retrieve the specific version and date of a package that populated a row of data. If you are tracking data lineage for a package, the DTS Browser lets you enter the lineage tracking number for a row of data to identify the package that created the row. For more information, see <u>Recording Data Lineage in DTS</u>.
- Browse through package versions saved to Meta Data Services and update packages. You can display a version history of each package, view the properties of a selected package version, and open and update a package version.

Usage Considerations in DTS

This section describes specific data conversion and data transformation issues that may arise when using Data Transformation Services (DTS) and supplies additional DTS error information.

Торіс	Description
Enhancing Performance of DTS	Describes ways to enhance the
Packages	performance of DTS packages.
Data Conversion and	Details the interactions between individual
Transformation Considerations	providers and DTS.
DTS Driver Support for	Lists the ODBC drivers and OLE DB
<u>Heterogeneous Data Types</u>	providers supplied by SQL Server to
	perform distributed operations against
	heterogeneous data sources.
Enhancing Data Driven Queries	Describes error handling issues, and issues
in DTS	regarding the use of transactions, lookups,
	and connections with a Data Driven Query
	task.
Enhancing Lookup Queries in	Lists issues and implementation details
DTS	you should consider when using lookup
	queries in a DTS package.

Enhancing Performance of DTS Packages

There are a number of factors that can affect the performance of Data Transformation Services (DTS) packages.

Using ActiveX Scripts

Depending on the type of transformation and your choice of scripting language, data pump operations that use Microsoft® ActiveX® transformation scripts can be up to two to four times slower than copy operations.

Using ordinal values to refer to columns in ActiveX transformations can be much faster then referring to columns by name. For example, use:

DTSSource(1)

instead of:

```
DTSSource("CustomerID").
```

This speed improvement is not significant when the number of columns is less then 20, but can become significant when the transformation contains many columns. For fewer columns, ignore this optimization to make the script easier to read.

In addition, scripts written in Microsoft Visual Basic® Scripting Edition (VBScript) run approximately 10 percent faster than scripts written in Microsoft JScript®, which in turn run approximately 10 percent faster than scripts written in PerlScript.

Enhancing Data Pump Performance

The data pump is the transformation component of the Transform Data task and the Data Driven Query task. When you use these tasks to transform data, you may be able to enhance performance when:

• There are large amounts of data.

- The transformations are numerous.
- The scripting code is complex.

To improve performance in these situations, use many-to-many mappings whenever possible. That way, you avoid mapping a separate transformation function for each column. The script engine is not invoked for each transformation. As a result, performance is faster. For more information about the column mappings used with transformations, see <u>Mapping Column</u> <u>Transformations</u>.

The DTS Import/Export Wizard creates packages with many-to-many column mappings. However, by default, the DTS Designer assigns one-to-one column mappings to transformations in order to improve readability. Therefore, when using DTS Designer, consider remapping as many transformations as possible to a many-to-many configuration. As with the use of ordinals in scripts, this optimization becomes more noticeable as the number of transformations increases. In general, if you have more than 20 transformations, you can see a noticeable performance decrease.

Data Driven Query Task vs. Transform Data Task

When Microsoft SQL Server[™] is a data destination of the Transform Data task, transformations, by default, use the **IRowsetFastLoad** interface. When SQL Server is not the destination, the Transform Data task uses the **IRowsetChange** interface (typically, sending INSERT statements).

Transformations in the Data Driven Query task use the **ICommand** interface on the destination using prepared insert operations. This might be faster than the **IRowsetChange** interface that transformations in the Transform Data task use, depending on how your destination OLE DB provider implements the interfaces. However, transformations using the **ICommand** interface or the **IRowsetChange** interface will not show better performance than transformations using the **IRowsetFastLoad** interface.

Using Bulk Insert and bcp

The Bulk Insert task creates and executes the Transact-SQL BULK INSERT

statement. BULK INSERT, supported by the Microsoft OLE DB Provider for SQL Server, is significantly faster than **bcp** or the data pump for performing text file import operations. Therefore, if transformations are not used, use the Bulk Insert task and achieve faster throughput.

The BULK INSERT statement is limited to file import operations.

When importing data, **bcp** and DTS copy operations are approximately the same speed. However, when exporting data, **bcp** operations can be approximately three to six times faster. Native **bcp**, which only applies to SQL Server data, is faster than the DTS data pump. BULK INSERT and native **bcp** are comparable in speed.

Using Connections

Use the **ExecuteOnMainThread** property only when necessary (for example, with drivers that are not thread-safe). Always avoid using **ExecuteOnMainThread** in ActiveX scripts to achieve concurrency unless the package or package step includes:

- Precedence constraints, which may affect the order of operation of a DTS package.
- Scripts that call COM objects written in Visual Basic.
- Custom tasks that are not free threaded.

For safety, only one task can use a connection at a time. To achieve parallel execution, you must set up different connections for each task. For example, a source (A) might connect to two destinations (C) and (D), but the operation occurs serially. The same source (A) could also be configured as (B), and parallelism could be achieved by (A) connecting to (C) and (B) connecting to (D).

However, if two tasks join the package transaction and then access the same instance of SQL Server in parallel, the package will fail. For more information, see <u>Incorporating Transactions in a DTS Package</u>.

By default, the maximum number of concurrent steps is four. Use the

MaxConcurrentSteps property to modify this setting.

Using Other SQL Server Solutions

A Transact-SQL query is the fastest method to move data without transformations or validations between tables. For multiple sources, consider running a distributed query such as a SELECT INTO statement.

In an environment using multiple packages, each DTS package must be run as a separate process using **dtsrun** in either a batch operation or using SQL Server Agent. This makes DTS a client process. If you must run a large number of packages (for example, 1,000 or more) as a server process, consider using SQL Server 2000 replication, which provides snapshot, transactional, and merge capabilities. Also, consider using SQL Server 2000 replication if your primary task is copying just the SQL Server data that has changed.

Improving Query Performance on Large DTS Packages Stored in a Repository

When you query a large DTS package that is stored in a SQL Server 2000 Meta Data Services repository, you can achieve better results if you increase the query time-out value. This ensures that your query has time to complete the roundtrip from the repository database, even if the DTS package is very large. By default, the query time-out value is 10 seconds. Adjust the query time-out value by creating the following registry key:

HKEY_LOCALMACHINE\SOFTWARE\Microsoft\Repository\Engine\ODBCQ For this key, set a value that is larger than the default (for example, 60 seconds or greater). The unit measurement is in seconds.

Data Conversion and Transformation Considerations

Before using Data Transformation Services (DTS) to convert or transform data between heterogeneous data and destinations, consider these variations in the way different programs, providers, and drivers support data types and SQL statements.

When using Microsoft® SQL ServerTM as a data source, consider the following:

- Transforming the **real** data type into the **int** data type may not return the exact value because SQL Server 2000 supports only six digits of precision for the **real** data type. For example, the **real** number 2147480000 may result in an **int** value of 2147480065.
- When transforming a **string** (DBTYPE_WSTR) into a **date** (DBTYPE_DATE) or **time** (DBTYPE_TIME) column from a text file, only one date or time format (*yyyy-mm-dd hh:mm:ss.fffffffff*) is accepted by the OLE DB data conversion service component. Use the Date Time String transformation, or code a Microsoft ActiveX® script transform using the **CDate** function to transform dates correctly.
- SQL Server 2000 does not support OLE DB types DBTYPE_DATE or DBTYPE_TIME. SQL Server 2000 only supports DBTYPE_DATETIME.
- To access data across multiple steps, avoid using a **temp** table during transformations. Instead, use a global **temp** table or create a permanent table in **tempdb**.
- Stored procedures that return rows from **temp** tables cannot be used as the source of a transformation. You can use stored procedures that return rows from a global **temp** table or table.
- When using temporary tables in the Transform Data task, the Data

Driven Query task, or the Execute SQL task in DTS Designer, be aware that you cannot use a Transact-SQL statement or stored procedure that calls a **temp** table as your source.

This limitation does not apply outside of DTS Designer. You can use source statements or stored procedures that access SQL Server **temp** tables programmatically.

DTS Import/Export Wizard and DTS Designer

When using the DTS Import/Export Wizard and DTS Designer to create packages, consider the following:

- The DTS user interface allows sharing existing connections among tasks but the same connection cannot be used for both the source and destination of a transformation.
- Using DTS Designer or the DTS Import/Export Wizard, it is possible to specify read-only or in-use status for some providers (for example, Microsoft Access and ODBC DSNs) that are to serve as data sources only. Click the **Advanced** tab in the **Connection Properties** dialog box, and in the **Advanced Connection Properties** dialog box, set the value of the mode property to 1.
- When creating a table using the DTS Import/Export Wizard or DTS Designer, the owner of a table created at the destination is the current user (generally the **dbo**), regardless of who the owner is at the source. This can result in a situation where the **dbo** attempts to create a table at the destination and the table name already exists, thus causing the attempt to fail.
- When defining a data-driven query using DTS Designer, a data destination must be able to support the OLE DB **ICommand** interface. Due to this restriction, destinations such as text files are not supported.
- The Copy SQL Server Objects task of DTS truncates fields of type **text**,

ntext, and **image** if they exceed 8388602 bytes in length. No error messages are displayed by either DTS Designer or the DTS Import/Export Wizard. These both indicate the task completed successfully.

The only indication of failure is a log message written to a log file named <server>.<database>.log, in the Script File Directory specified on the **Copy** tab of the **Copy SQL Server Objects Task Properties** dialog box. The log message specifies the table and column, but not the row, where the truncation occurred. No error records are written to the DTS error file or to the SQL Server log.

Microsoft SNA Server

When using Microsoft SNA Server as a data source, consider the following:

• The Microsoft OLE DB provider for AS/400 and VSAM does not support SQL statements that the DTS Import/Export Wizard uses to create or truncate a table.

Microsoft Access

When working with Access, consider the following:

• When exporting data from SQL Server 2000 to Microsoft Access 97 or earlier, the Microsoft OLE DB Provider for Access buffers all inserts in memory and only commits them when the DTS Import/Export Wizard completes operation. As a result, you can face a low memory situation when you export large tables. However, you can resolve this issue by constructing SELECT statements that send smaller numbers of rows in multiple passes.

Microsoft Visual FoxPro

Microsoft Visual FoxPro® supports only a precision of (15,9) for **numeric** data types. Data exported to Visual FoxPro that exceeds this precision is truncated and rounded.

• Visual FoxPro does not support the SELECT INTO statement.

- The DTS Query Designer supports the Visual FoxPro INSERT VALUE statement, but not the INSERT statement using a SELECT statement.
- The Microsoft OLE DB driver for ODBC is unable to write BLOBs to Visual FoxPro using the FoxPro ODBC driver because Visual FoxPro does not support dynamic cursors.

ODBC

When connecting to an ODBC data source, consider the following:

- The Microsoft OLE DB Provider for ODBC requires a unique key on all destination tables with a BLOB data column when performing export operations.
- When using the Microsoft OLE DB provider for ODBC with the SQL Server ODBC driver, all BLOB columns should be arranged after columns with other data types in a source rowset. You can use a SELECT statement to rearrange the BLOB columns to the end of the source rowset. The DTS Import/Export Wizard performs this operation automatically.

IMPORTANT When using the Microsoft OLE DB Provider for ODBC with the SQL Server ODBC driver, attempts to preview stored procedures fail with a connection busy error. This problem does not occur if you use the Microsoft OLE DB Provider for SQL Server.

- If a Microsoft ODBC Driver for SQL Server connection is being shared by multiple threads, the connection may fail, returning the error message "Connection is busy with results for another hstmt". In some cases, this affects packages built with the DTS Import/Export wizard. Use one of the following approaches to address this problem:
 - Set the **MaxConcurrentSteps** property to **1** to eliminate contending threads.

- Create additional ODBC connections to eliminate connection sharing.
- Use the Microsoft OLE DB Provider for SQL Server (SQLOLEDB) to connect to the database. If you need to connect to a SQL Server 6.5 database, run Instcatl.sql to enable access with the Microsoft OLE DB Provider for SQL Server.

Oracle

When using Oracle as a data source, consider the following:

- The Microsoft ODBC and OLEDB drivers for Oracle support the Oracle 7.3 BLOB data types, not Oracle 8.0 data types. For example, BLOB, CLOB, NCLOB, and BFILE are not supported.
- The Microsoft ODBC driver for Oracle does not support sending Unicode strings into an Oracle server. Oracle requires prefixing Unicode strings with the letter N.
- The Microsoft ODBC driver for Oracle does not support negative scaling for the Oracle **number** data type.
- The Microsoft ODBC driver for Oracle reports that an Oracle **number** data type without a specified precision has a size of 20 digits. When importing from Oracle (regardless of the destination), if there are more than 20 digits, you may have to manually increase the precision if the destination table does not already exist.
- Oracle supports only one LONG (BLOB) data column in a table.
- You cannot import or export Oracle columns that have mixed or lower case names. You also cannot transform or copy data using Oracle column names that contain spaces using the DTS Import/Export Wizard.

Oracle requires case-sensitive column names to be precisely specified and quoted.

- To perform distributed transactions between SQL Server 2000 and Oracle, you must use Oracle version 8.0.4.1 or later. For more information, see <u>Distributed Transactions</u>.
- Because the Microsoft OLE DB Provider for Oracle does not support **ICommandWithParameters**, it cannot be used as the destination of a Data Driven Query task. When using this provider in DTS Designer, the **Parameters** buttons on a Transform Data task, Data Driven Query task, and Execute SQL task will be disabled.

DB2 on the IBM AS/400

When connecting to a DB2 data source, consider the following:

- There is no Unicode or BLOB support on the AS/400 system.
- You cannot transform any table with a NULL column value to an AS/400 server because the AS/400 does not support NULL syntax in its CREATE TABLE statement. However, you can send NULL values if you edit the CREATE TABLE syntax to remove the references to NULL. The AS/400 does not support NOT NULL; NULL is assumed if not specified.

Using the Sybase ODBC Driver

When connecting to a Sybase ODBC data source, consider the following:

- When transforming data from SQL Server into Sybase version 11 using the DTS Import/Export Wizard:
 - The SQL Server **numeric** (3,0) data type maps to the Sybase **smallmoney** data type by default. Change this setting to avoid data loss.

- The SQL Server **numeric** (18,x or 19,x) data type maps to the Sybase **money** data type by default. Change this setting to avoid data loss.
- When moving data into a new Sybase table, if you click **OK** in the **Column Mappings and Transformations** dialog box, the wizard returns a "Table already exists" error message. You should ignore this message.
- You cannot drop and re-create a Sybase table using the DTS Import/Export Wizard. You must perform this action without using a wizard.
- The DTS Query Designer does not support the Sybase SQLAnywhere CREATE TABLE statement.
- The DTS Import/Export Wizard can only move one table at a time to a SQLAnywhere database due to a limitation in the SQLAnywhere driver. You can overcome this limitation using DTS Designer. However, you must set the **ExecuteInMainThread** property of the **Step** object to **True** for each table, as the SQLAnywhere driver is not thread safe.
- You cannot copy a table to a Sybase destination if it contains a BLOB column.
- If you programmatically copy a table containing an **image** data type from Sybase, changing the default BLOB settings can result in failure.

dBase and Paradox

When connecting to dBase and Paradox data sources, consider the following:

• Table names in dBase and Paradox are limited to eight characters. Column names in dBase are limited to 10 characters.

File Import or Export

When importing or exporting data from text files, consider the following:

- If you import into or export from **char** or **varchar** columns, some extended characters may not be copied correctly if your client OEM code page is different from the code page on the server. When you import into or export from **nchar** or **nvarchar** columns, all characters copy correctly.
- If you export BLOB (including SQL Server data types **text** and **ntext**) columns to a fixed length text field, the default length is set equal to the maximum BLOB field length (approximately two gigabytes). Prevent disk overflow by choosing a smaller but still adequate field length, or use a delimited format if possible.
- The OLE DB provider for text files used in DTS cannot process columns with BLOB data greater than two megabytes (MB).

Code Pages, Collation, and Non-Unicode Data Issues

When using DTS to copy data between SQL Server databases with different code pages and collations, data may be lost or incorrectly translated.

To avoid translation issues, store international data in Unicode. Once converted to Unicode, you can easily transfer data in any collation or code page without loss or incorrect translation to any Microsoft SQL Server 2000 or Microsoft SQL Server 7.0 database.

In Microsoft SQL Server 2000, collations are associated with particular code pages and are assigned to individual columns. (Microsoft SQL Server 7.0 uses a single default code page, and does not support column-level collations). If the code page used for a source and destination column match, no data loss will occur in non-Unicode columns. When data is copied between non-Unicode columns, and the source and destination code pages do not match, loss of data can result. In some cases, DTS will perform a best fit mapping, with data loss if the source contains characters that do not occur in the destination code page. In other cases, DTS will perform a copy without any intervening translation, resulting in the loss of any data not represented by the same binary value in both code pages. Following are problems and guidelines for using the Copy SQL Server Objects task and when copying data with the Copy Column transformation using different collations or code pages.

Copy SQL Server Objects Task

The following refers to how the Copy SQL Server Objects task handles non-Unicode data:

- When copying data from one instance of SQL Server 2000 to another instance of SQL Server 2000 there is no loss of data provided you set the **UseCollation** property of the Copy SQL Server Objects task.
- When copying data from an instance of SQL Server 2000 to SQL Server 7.0, a best fit mapping is used for columns that have collations that match the database default collation code page. Data stored in a column with a different code page is interpreted as being encoded in the default code page, with attendant losses on translation.
- When copying data from SQL Server 7.0 to an instance of SQL Server 2000, the **UseCollation** property is not available because SQL Server 7.0 is unable to determine which of several collations its default code page maps to. No collations are supported during Copy SQL Server Objects task execution, thus, non-Unicode destination columns will be assigned the default collation for the destination database. If the code page associated with the collation does not match that of the source database, DTS will perform a best fit mapping
- When copying data from SQL Server 7.0 to SQL Server 7.0, if the source and destination databases use different default code pages, DTS will perform a best fit mapping.

If you want to ensure that there is no data loss when copying non-Unicode data, you can use the SQL Server bulk copy feature to export data in Unicode format,

then use bulk copy or DTS to import it.

To disable the default scripting of collations, add code or use Disconnected Edit or the Dynamic Properties Task to add the value of **SQLDMOScript2_70Only** to the **ScriptOptionEx** property of the Copy SQL Server Objects Task.

Copy Column Transformation

The following refers to how the Copy Column Transformation handles non-Unicode data between different code pages:

- If the source column is Unicode and the destination column in non-Unicode, a best fit mapping is done, and an attempt is made to translate the data between source and destination.
- If the source column is non-Unicode and the destination column is Unicode, DTS interprets the source column as belonging to code page 1252 regardless of the actual code page used.
- If both the source and destination columns are non-Unicode, raw data will be copied without translation, and some loss of data will occur.

Data Transformation Services

DTS Driver Support for Heterogeneous Data Types

Data Transformation Services (DTS) uses the ODBC drivers and OLE DB providers supplied by Microsoft® SQL Server[™] 2000 to perform distributed operations against heterogeneous data sources.

The following table summarizes, for each of the major data sources, those supplied drivers or providers. Microsoft Product Support Services will help you resolve problems that you encounter when using these drivers and providers to perform distributed operations. If you are using another driver or provider, contact the vendor of that ODBC driver or OLE DB provider for support.

Data source	Driver or provider	Supported
Oracle 8	Microsoft ODBC for Oracle version	Yes
	2.573.3401.00	
Oracle 8	Microsoft OLE DB Provider for Oracle	Yes
	version 2.0010.3401.000	
Microsoft Jet	Microsoft OLE DB Provider for Jet version	Yes
version 4.0	4.0000.2115.0004	
Microsoft Excel	Microsoft OLE DB Provider for Jet version	Yes
spreadsheet	4.00.2115.15	
Microsoft Jet	Microsoft Access Driver version	No
version 3.51	4.00.3401.00	
IBM DB2/MVS	StarSQL 32 version 2.52.0501	Yes
IBM DB2/AS400	StarSQL 32 version 2.40.0805	Yes

DTS and Informix

- The Informix ODBC driver is not supported for use with DTS.
- The Merant Informix OLE DB provider is supported for DTS imports from Informix, but not DTS exports to Informix. This driver also cannot be used to import meta data.

- The Intersolv Informix ODBC driver is supported, but with the following restrictions:
 - BLOBs cannot be exported to Informix.
 - When creating new tables on Informix, the DTS Import/Export Wizard will incorrectly map the SQL Server 2000 **datetime** columns to the Informix 'Datetime year to fraction' data type. Manually change this to the Informix Date type.
 - The DTS meta data import will not import Informix catalog or table information.

Data Transformation Services

Enhancing Data Driven Queries in DTS

By changing properties in Data Transformation Services (DTS) Designer or via the DTS object model, you can configure the Data Driven Query task to:

- Fail on the first error, or continue, perhaps logging errors, until a maximum error count is reached.
- Commit modifications immediately, or join the package transaction to make all edits succeed or fail together.
- Use lookup queries to help populate destination columns or determine which query to execute.

Error Handling and the Data Driven Query Task

The Data Driven Query task offers a number of features to support error handling and restarts:

• To adjust the response of the Data Driven Query task to failures, you can either change the **Maximum errors** property in the **Data Driven Queries** dialog box or adjust the **MaximumErrorCount** property in the DTS object model.

When the number of failures exceeds this maximum value, the task halts and the step fails.

- You can log errors to a file you name, with formatting you provide.
- In order to support restarts, you can configure the Data Driven Query task to operate only on a numbered subrange of the source rowset.

Transactions and the Data Driven Query Task

The Data Driven Query task can join the package transaction. Before using the Data Driven Query task in transactions, consider the following:

- If the Data Driven Query task does not join the package transaction, updates are made one at a time, as they are requested.
- If the Data Driven Query task does join the package transaction, successfully run queries remain in the transaction at the conclusion of the task. These results are subject to commit or rollback in the current or following steps, or at package completion.
 - In order to join the package transaction, the binding table connection must support transactions.
 - Data Driven Query task failure does not automatically roll back successfully run queries; task success does not automatically commit them. Commit and rollback happen only in response to user package and workflow settings. For more information, see <u>Configuring Properties for DTS Transactions</u>.

Lookup Queries and the Data Driven Query Task

The Data Driven Query task can include lookup queries, which are additional parameterized queries that can be used to look up or modify data on local or distant connections.

Before adding lookup queries to the Data Driven Query task, consider the following:

- If the Data Driven Query task joins the package transaction, and the lookup connection supports transactions, any updates made by lookup queries also take part in the package transaction. For more information, see Lookup Queries.
- If the Data Driven Query task does not join the package transaction, or **Maximum errors** is not equal to zero, the possibility exists that a lookup might successfully execute before the corresponding data driven query fails. If the lookup modifies data, inconsistent updates might result.

Other Usage Considerations

Before using the Data Driven Query task, you also should consider the following:

- The data driven query task binding table must be able to support the OLE DB **ICommand** interface. Due to this restriction, binding tables such as text files are not supported.
- You should use unique connections for the source, binding table, and any lookup queries.

See Also

Data Driven Query Task

Data Transformation Services

Enhancing Lookup Queries in DTS

When implementing Data Transformation Services (DTS) lookup queries, consider the following:

- DTS lookups are best used when the input to the lookup is an external value (for example, a global variable).
- Lookups can be invoked from within a transformation function, allowing you to associate a query and a connection with the lookup. In a transformation function, you can then execute the query and have one or more values returned. You can achieve similar functionality using COM objects, VARIANTs, and the **DTS GlobalVariables** collection. However, the **Lookup** object uses an established connection and is optimized for quick data retrieval using caching.

If the transformation function can use an SQL statement instead of a lookup (for example, a SELECT statement with a join clause), performance can be greatly improved.

- You can configure a lookup query to cache results by specifying a cache size. If the cache size is larger than zero, then the results of the query are cached along with parameter values. If you provide the same parameter values to the query again, the results are returned from the cache with no additional database access. When the cache fills up, rows are removed in least recently used order.
- The lookup query connection must be able to support the OLE DB **ICommand** interface in order to accept SQL statements directly. Due to this restriction, connections such as text files cannot be used for lookups.
- If a task joins the package transaction and the lookup connection supports transactions, any updates made by lookup queries also take part in the package transaction. If the task joins the package transaction

and the lookup connection does not support transactions, on rollback, updates made by lookups will remain on file.

A lookup query may fail (for example, if it attempts to insert a row with a duplicate key). If a lookup fails, no further processing takes place for the source row. Lookup failure counts as one error against the **maximum errors** property of the containing task.

See Also

Lookup Queries