

DTS Programming

Programming DTS Applications

The Data Transformation Services (DTS) object model includes the objects and collections, as well as their associated properties, methods and events, that are used to write applications that manipulate the DTS object model.

Although any programming language that supports COM can be used to implement DTS applications, this section focuses on implementing DTS applications in Microsoft® Visual Basic® and Microsoft Visual C++®.

Topic	Description
DTS Object Model Diagram	Provides a diagram of the hierarchy of DTS objects and collections and offers an explanation of extended objects.
Creating DTS Packages with the DTS Object Model	Describes how to implement DTS package applications not specific to a particular programming environment.
Creating DTS Packages in Visual Basic	Describes how to implement DTS package applications in Visual Basic.
Building a DTS Custom Task	Describes how to implement and register a DTS custom task.
Building a DTS Custom Transformation	Describes how to implement and register a DTS custom transformation.

See Also

[Creating a DTS Package](#)

[DTS Basics](#)

[DTS Programming Reference](#)

DTS Programming

Extended DTS Objects

Several Data Transformation Services (DTS) objects that enhance objects from Microsoft® SQL Server™ version 7.0 have been added to Microsoft SQL Server 2000. These have been named by appending a **2** to the name of the existing object. For example, the **DataPumpTask2** object enhances the **DataPumpTask** object through the addition of the **RowsComplete** and **RowsInError** properties. In this section, **DataPumpTask2** refers to both the extended and original object.

Each extended object extends the functionality of the SQL Server 7.0 object and inherits the properties and methods of that object. However, none is compatible with SQL Server 7.0 or earlier. The SQL Server 7.0 objects still are available and should be used where interoperability with earlier versions of SQL Server is required.

Using the Extended Objects

When you run a DTS application that includes an extended object on SQL Server 7.0, a "type mismatch" or similar error will occur. Therefore, if you want a DTS application to run on both an instance of SQL Server 2000 and SQL Server 7.0, it is recommended that you do not use the extended objects.

A potential for failure under SQL Server 7.0 exists even when the extended objects are not used. If you run a DTS application on SQL Server 2000, you can access the new properties of the extended objects through the **Properties** collection of the existing objects. For example, you can access the **LogServerName** property through the **Properties** collection of the **Package** object, even though **LogServerName** is new for **Package2**.

The following Microsoft Visual Basic® code shows how you can access the new **LogServerName** property through the **Properties** collection of the **Package** object:

```
Dim objPackage As DTS.Package
Set objPackage = New DTS.Package
objPackage.Properties("LogServerName") = "(local)"
```

This code works when run on an instance of SQL Server 2000. However, if you run the application on SQL Server 7.0, the same code will fail with an error message similar to "property 'LogServerName' was not found."

DTS Programming

DTS Object Model Diagram

The diagrams in the topics that follow illustrate the hierarchy of Data Transformation Services (DTS) objects and collections. Objects are represented by a single blue rectangle. Collections, and the properties that return a reference to these collections, are represented by several overlaid yellow rectangles. The red-bordered hexagonals represent placeholders that summarize parts of the DTS object model. View the summarized objects, collections, and properties by clicking on the placeholders. A parent element (for example, an object or collection) is placed above and to the left of its child element, with a line connecting them.

An element that is the child of an object is created when the parent object is created, and a reference to the child can be obtained from a property of the parent. An object that is the child of a collection is created using a collection method.

For more information about using the DTS object model, see [Creating DTS Packages with the DTS Object Model](#).

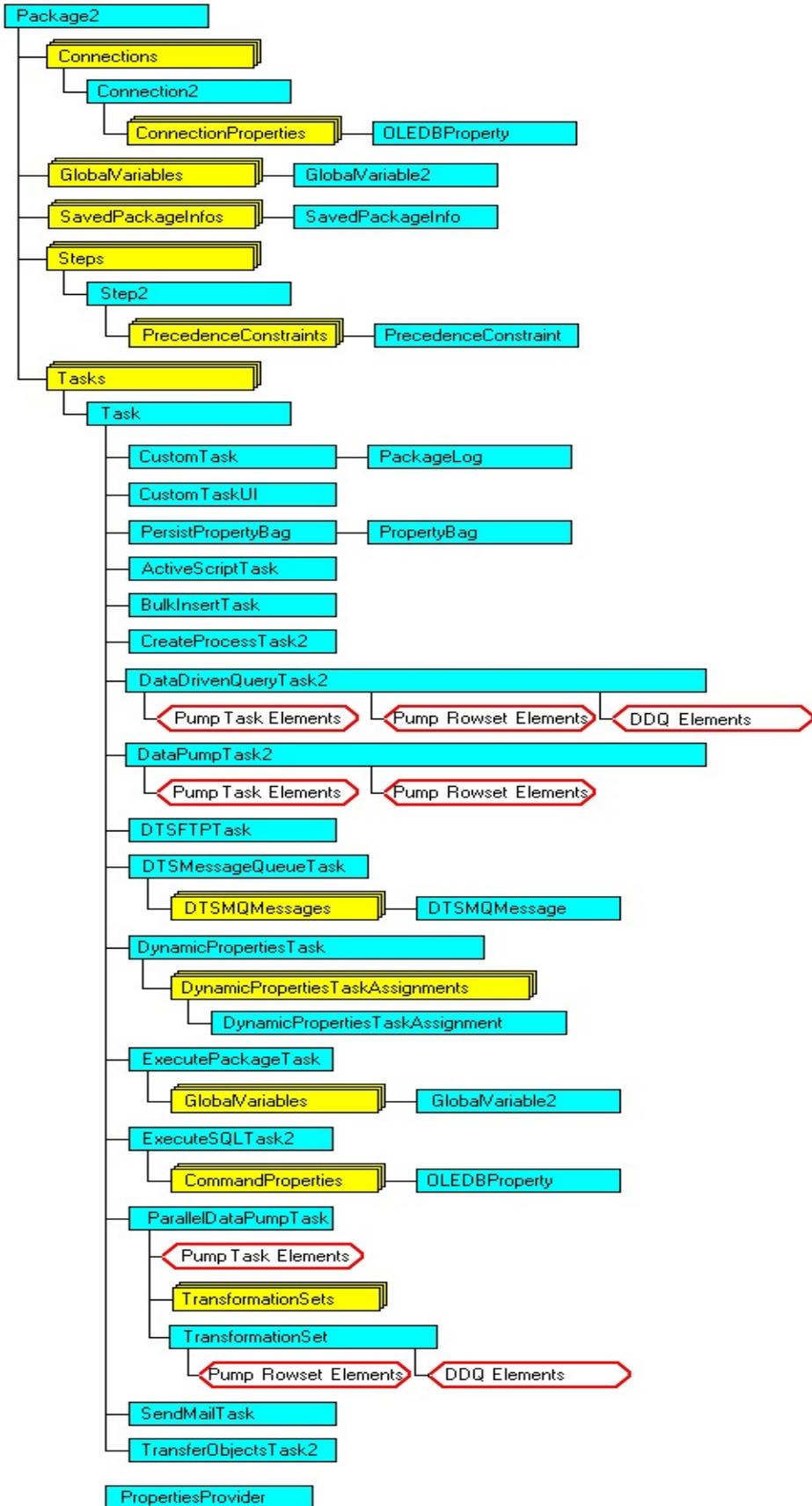
These topics contain object model diagrams and information about extended DTS objects.

Topic	Description
DTS Package2 Hierarchy	Describes the structure of the DTS Package2 hierarchy.
Pump Task Elements	Describes the object model in tasks that use the DTS data pump.
Pump Rowset Elements	Describes the object model associated with rowset processing in tasks using the DTS data pump.
Data Driven Query Elements	Describes the object model associated with the Data Driven Query task.
DTS Application Hierarchy	Describes the structure of the DTS Application hierarchy.

DTS Programming

DTS Package2 Hierarchy

This diagram illustrates the structure of the objects and collections of the Package2 hierarchy.



DTS Programming

Pump Task Elements

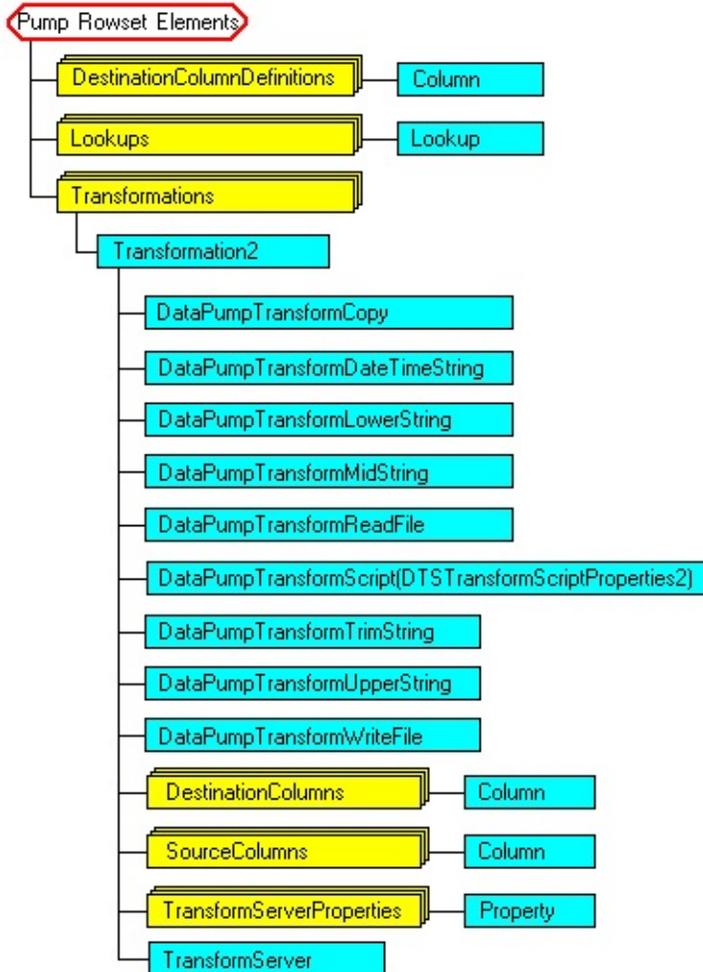
This diagram illustrates the structure of the objects and collections associated with tasks using the Data Transformation Services (DTS) data pump.



DTS Programming

Pump Rowset Elements

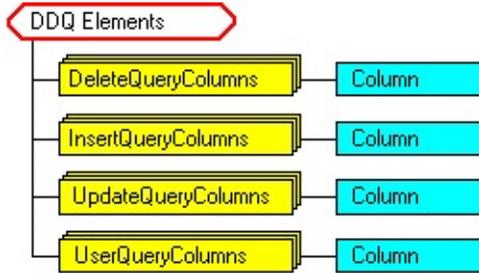
This diagram illustrates the structure of the objects and collections associated with rowset processing in tasks using the Data Transformation Services (DTS) data pump.



DTS Programming

Data Driven Query Elements

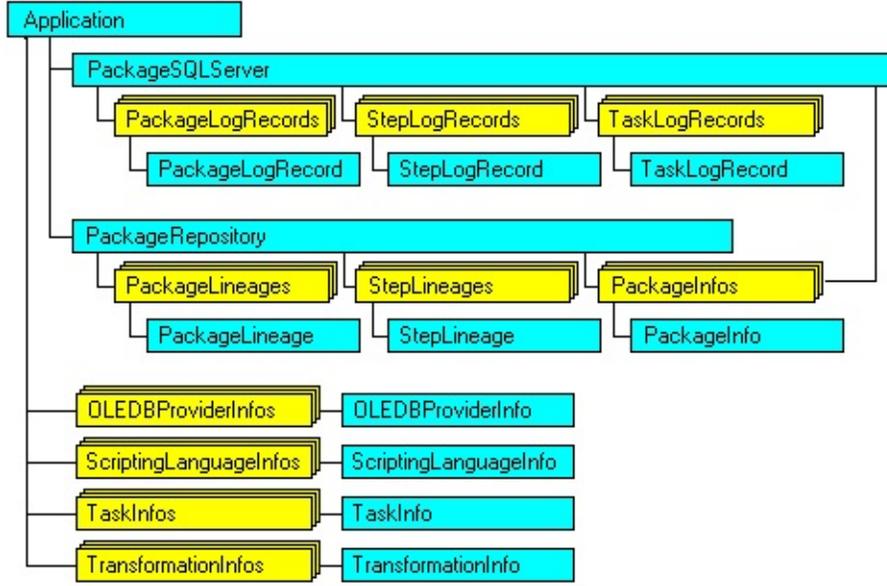
This diagram illustrates the structure of the objects and collections associated with the Data Driven Query task.



DTS Programming

DTS Application Hierarchy

This diagram illustrates the structure of the objects and collections of the Data Transformation Services (DTS) Application hierarchy.



DTS Programming

Creating DTS Packages with the DTS Object Model

To create a Data Transformation Services (DTS) package using a programming language that supports COM, you need to create a hierarchy of objects headed by a **Package2** object from the DTS object model. After setting the properties of this hierarchy, you can then invoke methods of a **Package2** object to run the package. You also can save the package to Microsoft® SQL Server™, a COM-structured storage file, a Microsoft Visual Basic® file, or to SQL Server 2000 Meta Data Services.

Note An **object** whose name ends with the digit **2** is an extended version of a DTS object from SQL Server version 7.0. If you want to run a package on SQL Server 7.0, it is recommended that you use the corresponding object whose name does not have a **2** appended. For more information, see [Extended DTS Objects](#).

This section summarizes building DTS packages and using DTS objects and features without regard to the programming language you are using for implementation. For more information about configuring a particular development environment and using it to implement DTS packages, see [Creating DTS Packages in Visual Basic](#).

The following table summarizes the topics describing the DTS objects and features you use to implement a package. Most objects and features are optional for a particular package. You are only required to create at least one step and at least one task. You do not need to follow the order provided here except where the DTS object hierarchy dictates. For example, you must create the **Package2** object first to gain access to the methods used to create other DTS objects.

Topic	Description
Creating DTS Package Objects and Connections	Create and configure a Package2 object. Create Connection2 objects to access data sources.
Creating DTS Package Workflow and Tasks	Create Step2 objects for the operations the package is to perform. Create PrecedenceConstraint objects to

	<p>define workflow among the steps.</p> <p>Create a Task object of the type needed for each step.</p>
Adding DTS Transformations	Create Transformation2 objects if needed for tasks that move data between connections.
Adding DTS Column Objects	Assign source and destination Column objects to the transformations if necessary.
Adding DTS Lookups and Global Variables	<p>Create a Lookup object when you need a transformation to look up data in another query rowset.</p> <p>Use GlobalVariable objects to pass data between steps and packages.</p>
Adding DTS ActiveX Scripts	Add a Microsoft ActiveX® script to a step or to a task or transformation that uses scripts.
Adding DTS Query Strings	Add query strings to an object that issues database queries.
Handling DTS Events and Errors	<p>Add handlers for the events of the Package2 object.</p> <p>Add an error handler to a DTS package program.</p>
Managing DTS Package Programs	<p>Execute a DTS package program.</p> <p>Save a DTS package in one of several formats, and load a package from these formats.</p>
Retrieving DTS System, Package, and Log Data	Retrieve information about registered components and DTS packages, and retrieve log data.

Creating DTS Package Objects and Connections

The **Package2** object is at the top of the Data Transformation Services (DTS) object hierarchy and is the first created. You then add **Connection2** objects to access databases and other data sources.

Creating Package Objects

The first step in implementing a DTS package is creating the **Package2** object. The way you create the **Package2** object depends on your programming environment. For more information about configuring the **Package2** object, see [DTS Packages in Visual Basic](#).

Some **Package2** properties and features you can use are:

- **AutoCommitTransaction**, **TransactionIsolationLevel** and **UseTransaction**. These control whether **Package2** components run under a package-level transaction, and how that transaction is used.
- **FailOnError** and **FailPackageOnLogFailure**. These determine whether component failures cause package execution to fail.
- **CreationDate**, **CreatorComputerName**, **CreatorName**, **PackageID** and **VersionID**. These provide creation and identification information.

Adding Connections

Usually, you create a **Connection2** object for each data source you want to access, although **Connection2** objects can be reused. You need an OLE DB provider for the data source you wish to access. The following table describes the typical data sources for which OLE DB providers are supplied with Microsoft® SQL Server™ 2000.

Data Source Type	Data Source
Databases	SQL Server

	Microsoft Access 2000 Oracle Paradox DB2 DBase 5 Other ODBC-compliant database
Other data sources	Microsoft Excel 2000 worksheet HTML file Text file

For more information about the OLE DB providers available on a computer system, see [OLEDBProviderInfos Collection](#). For more information about the interfaces and schema that user-implemented OLE DB providers used with DTS must support, see [ProviderID Property](#). For more information about the interfaces and schema that user-implemented OLE DB providers used with DTS must support, see [ProviderID Property](#).

To create a **Connection2** object, use the **New** method of the **Connections** collection of the **Package2** object. Set the properties as needed. The typical properties you use are:

- **Catalog.** This is typically a database name.
- **DataSource.** This is a server name or a data source file specification.
- **ID.** This is a numeric identifier for the connection.
- **Password, UserID, or UseTrustedConnection.** These contain user authentication information.

Then, **Add** each **Connection2** object to the **Connections** collection of the **Package2** object.

You can reference properties unique to specific OLE DB providers through the **ConnectionProperties** collection of the **Connection2** object.

For more information about configuring the **Connection2** object, see [DTS Connections in Visual Basic](#).

See Also

[Handling DTS Events and Errors](#)

[Managing DTS Package Programs](#)

[Retrieving DTS System, Package, and Log Data](#)

Creating DTS Package Workflow and Tasks

Workflow in Data Transformation Services (DTS) packages is implemented by creating steps, which are the units of functionality, and precedence relationships between steps, which determine the sequencing of the steps. Tasks are the components the steps use to perform their functions.

Creating the Step Object

You need to create a **Step2** object for each operation the package is to perform. For each step, you create a **Task** object of the appropriate type. The **Task** object performs the operation for the step.

To create a **Step2** object, use the **New** method of the **Steps** collection of the **Package2** object. Set the **TaskName** property of the **Step** object to the name of the associated task. Other properties you can use are:

- **ActiveXScript**, **FunctionName**, and **ScriptLanguage**. These properties specify the Microsoft® ActiveX® script to run before the task.
- **CommitSuccess**, **JoinTransactionIfPresent**, **RollbackFailure**. These properties determine whether the step uses the package transactions.
- **ExecuteInMainThread**. This property runs the step in the package main thread rather than in a worker thread.
- **FailPackageOnError**. This property fails the package if the step fails.

Then, **Add** the **Step2** object to the **Steps** collection of the package.

Creating the Precedence Constraint Object

When a package is executed, DTS attempts to execute steps in parallel up to the limit established by the **MaxConcurrentSteps** property of the **Package2** object.

However, you can order the steps by using precedence constraints. A **Precedence Constraint** object inhibits the step with which it is associated from starting execution until an event by another named step occurs. As a result, the step only begins execution when all of its precedence constraints have been satisfied.

To create the **PrecedenceConstraint** object, use the **New** method of the **PrecedenceConstraints** collection of the **Step** object. Set its **StepName** property to the name of the preceding task and set the **PrecedenceBasis** and **Value** properties to specify the type of event. Then, **Add** the **PrecedenceConstraint** to the **PrecedenceConstraints** collection of the associated **Step** object.

For more information about configuring the **Step** and **PrecedenceConstraint** objects and the **PrecedenceConstraints** collection, see [DTS Package Workflow in Visual Basic](#).

Creating the Task Object

To implement a DTS task, you need a generic **Task** object and a task object specific to the task class being created (for example, a **DataDrivenQueryTask2** object or a **BulkInsertTask** object). To create both of these, use the **New** method of the **Tasks** collection of the **Package2** object.

Configure the properties of these objects as appropriate for the processing you want to perform. While the elements of the generic **Task** object manipulate information generic to all tasks, those of the class-specific task object manipulate information unique to the class. The **CustomTask** property of the **Task** object returns a reference to the class-specific task object. The properties of the class-specific task object also can be referenced through the **Properties** collection of the (generic) **Task** object.

Add each **Task** object to the **Tasks** collection of the **Package2** object.

For more information about the task classes supplied with Microsoft SQL Server™ 2000, see [Task Objects](#).

For more information about configuring the **Task** object and the class-specific task objects, see [DTS Tasks in Visual Basic](#).

Adding DTS Transformations

Data Transformation Services (DTS) transformations are used by the data pump to perform various operations that you specify. The data pump is the engine for the **DataPumpTask2**, **DataDrivenQueryTask2**, and **ParallelDataPumpTask** objects. Transformations can be viewed as callbacks from the data pump. The other task classes supplied with Microsoft® SQL Server™ 2000, which do not host the data pump, do not use transformations.

The data pump fetches data rows from a source connection and writes data rows to a destination connection. The table below describes the phases of the data pump operations for which transformations can be specified. They are listed in the order in which they are invoked by the data pump.

Phase	Description	Possible Uses
PreSourceData	Occurs before first row is fetched from source connection.	Writing header records to the destination. Initializing objects, connections or memory for use in later phases.
Transform	Occurs after each source row is fetched, before the destination row is written.	Converting data types. Validating.
OnTransformFailure	Occurs after a failure in the Transform phase, indicated by the return of DTSTransformStat_Error or DTSTransformStat_ExceptionRow . Typically caused by conversion errors.	Handling custom data based on the Transform failure.

OnInsertSuccess	Occurs after each data row is written successfully to the destination connection.	Maintaining aggregation when this function cannot be done by a Transform phase transformation.
OnInsertFailure	Occurs after each attempt to write a data row to the destination connection failed (for example, by attempting to write a duplicate value to a primary key field, or a null to a NOT NULL field).	Handling custom data based on the Insert failure (for example, writing the data to an error table).
OnBatchComplete	Occurs in DataPumpTask2 when using FastLoad option after each batch is written, successful or failed.	Recording the current position within the source rowset, which could then be used as the starting point if the task needed to be restarted.
PostSourceData	Occurs after the last row is written to the destination connection.	Writing trailer records to the destination or freeing up resources or committing data held in global variables.
OnPumpComplete	Occurs at the end of the task execution.	Freeing up resources or

		committing data held in global variables.
--	--	---

In the case of the **ParallelDataPumpTask**, the **PreSourceData** and **PostSourceData** phases occur at the beginning and end, respectively, of each constituent rowset of the hierarchical rowset. The **OnPumpComplete** phase occurs once.

The **DTSTransformScriptProperties2** transformation can support multiple phases. You provide a script function for each supported phase. Transformations for phases other than **Transform** must be **DTSTransformScriptProperties2** transformations or custom transformations.

Creating Transformation Objects

To implement a transformation, you need a generic **Transformation2** object and a transform server object, which is an object specific to the transformation class (for example, **DataPumpTransformCopy** or **DataPumpTransformDateTimeString**). To create both of these, use the **New** method of the **Transformations** collection of the **DataPumpTask2**, **DataDrivenQueryTask2**, or the **TransformationSet** object of the **ParallelDataPumpTask**.

To access the transform server object, use the **TransformServer** property of **Transformation2** object to return a reference to the object. The properties of the transform server object also can be referenced through the **TransformServerProperties** collection of the generic **Transformation2** object.

If the transformation is to support any phase other than the **Transform** phase, set the **TransformPhases** property of the **Transformation2** object to the sum of the codes from **DTSTransformPhaseEnum** for the phases it is to support. Add the **Transformation2** object to the **Transformations** collection.

At least one transformation is required for the **Transform** phase. Transformations for the other phases are optional. When multiple transformations are supplied for a phase, they are all executed at the time appropriate for the phase, in the order the **Transformation2** objects were added to the **Transformations** collection.

For more information about the transformation classes supplied with SQL Server

2000, see [Transformation Objects](#).

For more information about configuring the **Transformation2** object and the transform server objects, see [DTS Transformations in Visual Basic](#).

Adding DTS Column Objects

You create **Column** objects to specify the source and destination columns referenced by a transformation and to specify the destination column parameters for the queries of the **DataDrivenQueryTask2** object.

Typically, transformations reference columns of the source and destination connections. In most cases, source and destination columns need to be assigned to a transformation to define these columns.

To create a **Column** object, use the **New** method of the **SourceColumns** or **DestinationColumns** collections of the **Transformation2** object. Set properties as appropriate, then use the **Add** method of the appropriate collection. Or, use the **AddColumn** method to create and add the column in a single step.

You do not need to define **Column** objects if:

- Only a single **Transformation2** object has been defined for the task.
- The number of columns in the data source and the data destinations is the same.
- The source and destination column ordering correctly match up.

Note You can use a Select query on the data source, specified with the **DataDrivenQueryTask2**, **DataPumpTask2**, or **ParallelDataPumpTask** object **SourceSQLStatement** property, to control the number and ordering of source columns.

Whether a transformation can access the source or destination connections is determined by the phase in which it runs. The following table specifies the permitted access.

Phase	Source column access	Destination column access
PreSourceData	Read access to meta data	Write access to columns
Transform	Read access to columns	Write access to columns

OnTransformFailure	Read access to columns	Write access to columns
OnInsertSuccess	Read access to columns	No access to columns
OnInsertFailure	Read access to columns	No access to columns
OnBatchComplete	Read access to meta data	Write access to columns
PostSourceData	Read access to meta data	Write access to columns
OnPumpComplete	No access to columns	No access to columns

For more information about configuring the **Column** object and the **SourceColumns** and **DestinationColumns** collections, see [DTS Column Objects in Visual Basic](#).

To assign destination column parameters to the **DataDrivenQueryTask2** and **ParallelDataPumpTask** object queries, use the procedure specified above to create **Column** objects and add them to the **DeleteQueryColumns**, **InsertQueryColumns**, **UpdateQueryColumns**, or **UserQueryColumns** collections, as appropriate. For more information about parameterized queries for these objects, see [Adding DTS Query Strings](#).

Adding DTS Lookups and Global Variables

Use a **Lookup** object when you need a transformation to look up data in another query rowset through a separate connection.

Global variables provide a means for tasks within a package to exchange data. When using the **ExecutePackageTask** object, global variables provide a means for tasks to exchange data between packages.

Creating Lookups

To create a **Lookup** object, use the **New** method of the **Lookups** collection of a class-specific task object that uses transformations. Set the appropriate properties. Typically, you use the following properties:

- **ConnectionID.** This specifies the connection through which the lookup rowset is queried.
- **MaxCacheRows.** This determines the number of queried rows cached for reuse.
- **Query.** This specifies the SQL statement that generates the rowset.

Then, **Add** the **Lookup** object to the **Lookups** collection.

Alternatively, you can create and add the **Lookup** object to the **Lookups** collection with the **AddLookup** method.

You access the lookup in the script of a **DataPumpTransformScript** or **DTSTransformScriptProperties2** object through the **Execute** method of a **DTSLookup** object, which is the Data Transformation Services (DTS) scripting object model counterpart of the **Lookup** object. Usually, you refer to the lookup by name from the **DTSLookups** collection.

For more information about the **Lookup** object and the **Lookups** collection, see [DTS Lookups in Visual Basic](#).

Creating and Using Global Variables

To create a global variable in a DTS package prior to package execution, use the **New** method of the **Package2** object **GlobalVariables** collection. Set the **Value** property, and then **Add** the object to the collection.

Alternatively, you can create and add the **GlobalVariable** object to the **GlobalVariables** collection with the **AddGlobalVariable** method.

You need to create **GlobalVariable** objects before package execution if the **ExplicitGlobalVariables** property of the **Package2** object is set to TRUE. However, if **ExplicitGlobalVariables** is set to FALSE, you do not need to create **GlobalVariable** objects. The package automatically creates global variables that do not exist at first reference.

Setting global variables with the ExecuteSQLTask2 object

You can create and assign values to global variables in the **ExecuteSQLTask2** object. Specify a list of global variable names with the **OutputGlobalVariableNames** property. Values from the first row of the rowset generated by the **ExecuteSQLTask2** query (specified with the **SQLStatement** property) are stored in the named global variables. Set the **OutputAsRecordset** property to store the entire rowset as a disconnected Microsoft® ActiveX® Data Objects (ADO) recordset in the global variable named first in the list.

Using global variables as input parameters

You can use global variables as input parameters for the queries of the **DataDrivenQueryTask2**, **DataPumpTask2**, **ExecuteSQLTask2** and **ParallelDataPumpTask** objects. Specify a list of global variable names with the **InputGlobalVariableNames** property. For more information, see [Adding DTS Query Strings](#).

Exporting global variables to a DTS package

Create and add global variables, as described above, to the **GlobalVariables** collection of the **ExecutePackageTask** object to export these global variables to the target package. These global variables are independent of the global variables in the **GlobalVariables** collection of the calling package. Use the

InputGlobalVariableNames property of **ExecutePackageTask** to specify global variables from the collection of the package that are to be exported.

Referencing global variables in ActiveX scripts

Reference global variables in ActiveX scripts as members of the **DTSGlobalVariables** collection. For example, in Microsoft Visual Basic® Scripting Edition (VBScript):

```
DTSGlobalVariables( "GV1" ).Value
```

If you assign a value to the above expression and GV1 does not exist, and if the package **ExplicitGlobalVariables** property is not set, GV1 is created.

For more information about the **GlobalVariable** object and the **GlobalVariables** collection, see [DTS Global Variables in Visual Basic](#).

Adding DTS ActiveX Scripts

Microsoft® ActiveX® scripts can be used to add functionality to Data Transformation Services (DTS) packages. Typical supported scripts are Microsoft Visual Basic® Scripting Edition (VBScript), Microsoft JScript®, PerlScript and XMLScript. The following DTS object types require or can use ActiveX scripts:

- The **Step2** object can use an ActiveX script. The script runs before the step's task executes.
- The **ActiveScriptTask** object requires an ActiveX script that performs the functionality of the task.
- The **DataPumpTransformScript** and **DTSTransformScriptProperties2** transformations require ActiveX scripts that perform the transformations.

You assign ActiveX scripts to a property of objects that use scripts as a single text string, which can include embedded carriage return/line feed pairs. Each scripted object also has properties for the script language and the script function name. For example:

- For the **Step2** and **ActiveScriptTask** objects, the script text is assigned to the **ActiveXScript** property. The **ScriptLanguage** and **FunctionName** properties are used to specify the scripting language and function entry point.
- For the **DataPumpTransformScript** transformation, the script text is assigned to the **Text** property. The **Language** and **FunctionEntry** properties are used for the script language and function name, respectively.

The **DTSTransformScriptProperties2** object extends the functionality of

DataPumpTransformScript by providing multiple transformation phases. The script specified by the **Text** property must have a function for each supported phase.

The following table describes the property you use to specify the entry point for each phase the **DTSTransformScriptProperties2** object supports.

Phase	Entry Point Property
PreSourceData	PreSourceDataFunctionEntry
Transform	FunctionEntry
OnTransformFailure	TransformFailureFunctionEntry
OnInsertSuccess	InsertSuccessFunctionEntry
OnInsertFailure	InsertFailureFunctionEntry
OnBatchComplete	BatchCompleteFunctionEntry
PostSourceData	PostSourceDataFunctionEntry
OnPumpComplete	PumpCompleteFunctionEntry

For more information about including ActiveX scripts in DTS programs, see [DTS ActiveX Scripts in Visual Basic](#).

Adding DTS Query Strings

Many Data Transformation Services (DTS) tasks and objects require queries to access or store database information. You assign queries to a property of the object that uses the query as a text string. You also can include carriage return/line feed pairs. Depending on the parent object, you can define query sequences and parameter placeholders. The following table defines the query types that are supported for the objects that use queries.

Objects Using Queries	Query Type	Query String Attributes
DataPumpTask2 , DataDrivenQueryTask2 and ParallelDataPumpTask objects	Source query	Single Select or stored procedure query that returns a rowset. Can use ? placeholder for global variable parameters, specified by InputGlobalVariableNames property.
DataPumpTask2 , DataDrivenQueryTask2 and ParallelDataPumpTask objects	Destination query	Single Select or stored procedure query that returns a rowset. Parameters are not supported.
DynamicPropertiesTaskAssignment object of DynamicPropertiesTask	Property value query	Single Select or stored procedure query that returns a rowset. Rowset has single row and one column.
ExecuteSQLTask2 object	Executed query	Sequence of one or more SQL statements or stored procedure queries, which can contain the Transact-SQL GO statement. Can use ? placeholder for global variable parameters,

		specified by the InputGlobalVariableNames property.
DataDrivenQueryTask2 and TransformationSet objects of ParallelDataPumpTask	Action queries	Sequence of one or more SQL statements or stored procedure queries, which cannot contain the Transact-SQL GO statement. Can use ? placeholder for destination columns.
Lookup object	Lookup query	Single Select or stored procedure query that returns a rowset. Can use ? placeholder for parameters, specified with the Execute method of DTSLookups scripting collection.

For more information about including query strings in DTS programs, see [DTS Query Strings in Visual Basic](#).

Handling DTS Events and Errors

The **Package2** object raises events that report package status during execution. The **Package2 Execute** method can raise errors that must be handled by the calling application.

Package Events

The following table specifies the events the **Package2** object raises and the information that is returned when the specified condition occurs.

Event	Condition	Information returned
OnError	A Data Transformation Services (DTS) error occurred during package execution.	Step name, error code and description, help file and context, interface ID.
OnFinish	A step has completed.	Step name.
OnProgress	This event occurs periodically during step execution.	Step name, progress count (typically rowcount), percent complete, description.
OnQueryCancel	This event gives application a chance to cancel a step.	Step name.
OnStart	A step has started.	Step name.

For more information about these individual events, see [Events](#).

If you implement handlers for any of the events, you must supply handlers for all the events. Unneeded event handlers can consist of a single statement (for example, **Exit Sub** or **return;**).

For more information about implementing event handlers in DTS programs, see [DTS Package Events in Visual Basic](#).

Package Error Handler

During the phase of your program where you are creating DTS objects and setting their properties, you can implement error handling that is typical for your programming environment.

Errors that occur during the **Execute** method of the **Package2** object are not propagated back to the caller unless you set the **Package2** object **FailOnError** property or the **Step2** object **FailPackageOnError** property to TRUE.

FailPackageOnError causes an error in the referenced step to fail the package, while **FailOnError** causes an error in any step to fail the package.

You must retrieve errors that occur within individual steps with the **GetExecutionErrorInfo** method of the **Step2** object. **GetExecutionErrorInfo** provides information only about errors that cause a step to fail. For more detailed information, you must implement handlers for the **OnError** and **OnProgress** events. You must provide at least stub handlers for the other events. The **OnError** event will describe the error, while the **OnProgress** event will indicate the step and the row being processed. The **ProgressRowCount** property of the **DataPumpTask2**, **DataDrivenQueryTask2**, and **TransformationSet** objects can be used to specify how frequently **OnProgress** is raised.

For more information about implementing package error handlers in DTS programs, see [DTS Error Handlers in Visual Basic](#).

DTS Programming

Managing DTS Package Programs

You can either run the configured **Package2** object or save it in several formats. You also can load a saved Data Transformation Services (DTS) package into a **Package2** object.

Executing Packages

After you have created the hierarchy of DTS objects and set their properties, as needed, use the **Execute** method of the **Package2** object to execute the package. DTS may raise errors from the Execute method. For more information about handling these errors, see [Handling DTS Events and Errors](#).

If you plan to do anything further with the **Package2** object, release all references to other DTS objects, then use the **UnInitialize** method.

For more information, see [Executing DTS Packages in Visual Basic](#).

Saving and Loading Packages

You can save the package as a project in your current development environment. You can also save it in the formats in which DTS tools save packages. To do the latter, use one of the following methods of the **Package2** object.

Methods	Description
SaveToRepository , SaveToSQLServer , SaveToStorageFile	Save the package to a specified storage type.
SaveToRepositoryAs , SaveToSQLServerAs , SaveToStorageFileAs	Assign a new name and package ID to the package, then save it to a specified storage type.
SaveAs	Assign a new name and package ID to the Package2 object, but do not save it to storage.

To load a **Package2** object with the state of a previously saved package, use the **LoadFromSQLServer**, **LoadFromRepository**, or **LoadFromStorageFile** methods. You can delete saved packages by using the **RemoveFromSQLServer**

and **RemoveFromRepository** methods.

For more information about saving and loading DTS packages, see [Saving DTS Packages in Visual Basic](#).

DTS Programming

Retrieving DTS System, Package, and Log Data

Data Transformation Services (DTS) provides features for requesting information about registered components and saved packages and for retrieving the contents of log records.

Registered Components

The **Application** object provides access to the system, package, and log data. You create it independently of a DTS package.

Use the collections of the **Application** object to obtain information about several different types of registered components used by DTS. The following table describes the collections, the objects they contain, and the type of component for which information is available.

Collection	Object	Component
OLEDBProviderInfos	OLEDBProviderInfo	OLE DB providers
ScriptingLanguageInfos	ScriptingLanguageInfo	Microsoft® ActiveX® scripting languages
TaskInfos	TaskInfo	DTS task classes
TransformationInfos	TransformationInfo	DTS transformation classes

The DTS task and transformation classes include those supplied with Microsoft SQL Server™ and custom tasks and transformations implemented by other vendors and users.

Normally DTS must scan all the registered classes in the operating system registry to determine the membership of each of these collections, which can take a significant amount of time. DTS maintains a cache, also in the operating system registry, of each component type. Use the **Refresh** method of these collections to update the cache for that component from a full-registry scan. Set the **UseCache** property before iterating through the collection to make it scan the cache rather than the system registry.

Meta Data Services

DTS packages can be saved to an instance of SQL Server 2000 Meta Data Services. Lineage information is saved for such packages, if the **LineageOptions** property of the package specifies this be done. A package lineage record is written each time a package is executed and a step lineage record is generated for the execution of each step.

Use the **GetPackageRepository** method, specifying server, database and login information, to return a **PackageRepository** object that provides access to an instance of Meta Data Services. The following methods of the **PackageRepository** object return package and lineage information:

- Use the **EnumPackageInfos** method to return a **PackageInfos** collection with information about all or a subset of the packages saved in the Meta Data Services instance.
- Use the **EnumPackageLineages** method to return a **PackageLineages** collection with lineage data for a particular package version.
- Use the **EnumStepLineages** method to return a **StepLineages** collection with step lineage data for a particular package lineage (each step execution associated with a single execution of a particular package).
- Use the **RemovePackageLineages** method to remove some or all of the lineage data for a package version.

SQL Server Storage and Logging

All DTS packages can log to an instance of SQL Server. Log records are written to the **msdb** database on the server specified by the package **LogServerName** property each time a DTS package is executed, if the package **LogToSQLServer** property has been set.

How log data is written

A package log record is written by DTS for each package execution, and a step

log record is written for the execution of each step.

Use the **PackageLog** object methods when custom tasks and the **ActiveScriptTask** object are to write task log records. A reference to **PackageLog** is passed as a parameter of the task **Execute** method. In task ActiveX scripts, it is available as the **DTSPackageLog** scripting object.

Retrieving package and log data

Use the **GetPackageSQLServer** method, specifying server and login information, to return a **PackageSQLServer** object that provides access to the package and log data on the server.

Use the **PackageSQLServer** object **EnumPackageInfos** method to return a **PackageInfos** collection with information about all or a subset of the packages in SQL Server storage on that server.

Use a **PackageSQLServer** method from the table to return the corresponding collection that contains data for all or a subset of the log records of the indicated type on the server. The removal methods will selectively remove log records of the indicated type.

Method	Returned collection	Removal method
EnumPackageLogRecords	PackageLogRecords	RemovePackageLogRecords
EnumStepLogRecords	StepLogRecords	RemoveStepLogRecords
EnumTaskLogRecords	TaskLogRecords	RemoveTaskLogRecords

In addition, **RemoveAllLogRecords** removes all log data for all packages from the server.

Packages Saved as Files

You can retrieve information about the contents of a DTS package storage file, which can contain multiple packages, each with multiple versions. Create a **Package2** object and then use the **GetSavedPackageInfos** method to return a reference to a **SavedPackageInfos** collection with information about all the package versions contained in the file.

For more information about getting saved package information, see [Retrieving DTS Information in Visual Basic](#).

DTS Programming

Creating DTS Packages in Visual Basic

You can implement Data Transformation Services (DTS) packages in Microsoft® Visual Basic® by following these installation instructions:

- You need to install the Microsoft SQL Server™ client tools and Visual Basic version 5.0 Service Pack 3 or later on the computer on which the packages are to be developed.
- You need to install the SQL Server client tools on the computers on which the packages are to be run.

You do not need to install Visual Basic on the target computers. The necessary Visual Basic files are supplied by the installation kit produced with the Visual Basic Setup or Package and Deployment Wizards.

Configuring the Visual Basic Development Environment

To implement a DTS program in Visual Basic, open a new or existing project, as appropriate, in the Visual Basic development environment. You can use any of the following project types:

- Standard EXE
- ActiveX EXE
- ActiveX DLL
- ActiveX Document EXE
- ActiveX Document DLL

From the **Project/References** dialog box, select the references listed in the table below if you use any of the corresponding DTS features in your application. This

+will include the library file from the table in your Visual Basic project.

Reference	DTS Features	Library File
Microsoft DTSPackage Object Library	Any DTS object or feature	dtspkg.dll
Microsoft DTSPump Scripting Object Library	Any transformation supplied with SQL Server or any DTS scripting object	dtspump.dll
Microsoft DTS Custom Tasks Object Library	The Message Queue task, the File Transfer Protocol task or the Dynamic Properties task	custtask.dll

The library files are installed in C:\Program Files\Microsoft SQL Server\80\Tools\Binn\ unless overridden by the SQL Server installation.

Using DTS Packages Saved as Visual Basic Files

If you have used DTS Designer or the DTS Import/Export Wizard to save a DTS package as a Visual Basic file, these files can be used as templates, or starting points, for user-implemented Visual Basic packages.

The generated code sets all properties of all objects referenced in the package to the initial values they will have when package execution begins. This includes those that are set to their default values. Thus, many of the property assignments are redundant and can be removed. These redundant property settings do not appear in the Visual Basic code examples in this section.

For more information, see [Running a DTS Package Saved as a Visual Basic File](#).

See Also

[Saving DTS Packages in Visual Basic](#)

DTS Programming

Creating DTS Objects in Visual Basic

The topics that follow describe the creation and configuration of Data Transformation Services (DTS) objects that are used to implement a DTS package in Microsoft® Visual Basic®.

Topic	Description
DTS Packages in Visual Basic	Explains how to create and configure a Package or Package2 object.
DTS Connections in Visual Basic	Explains how to create and configure a Connection or Connection2 object for a database or other data source.
DTS Package Workflow in Visual Basic	Explains how to create and configure a Step object and how to create and add PrecedenceConstraint objects to implement workflow.
DTS Tasks in Visual Basic	Explains how to create a Task object and how to configure it and the class-specific task object.
DTS Transformations in Visual Basic	Explains how to create a Transformation object and how to configure it and the transform server object.
DTS Column Objects in Visual Basic	Explains how to create and add Column objects in order to define source and destination columns.
DTS Lookups in Visual Basic	Explains how to create and configure a Lookup object and access it from a Microsoft ActiveX® script.
DTS Global Variables in Visual Basic	Explains how to create and configure a GlobalVariable object and how to explicitly create a global variable prior to package execution.
DTS ActiveX Scripts in Visual Basic	Explains how to add ActiveX scripts to ActiveScriptTask and

	DataPumpTransformScript objects.
DTS Query Strings in Visual Basic	Explains how to add query strings to the objects that use them and describes the details and limitations of each object type.
DTS Package Events in Visual Basic	Explains how to implement event handlers for Package object events.

DTS Packages in Visual Basic

To create a **Package** object in Microsoft® Visual Basic®, you declare an object variable of the appropriate type and then create the object with the Visual Basic **New** operator.

The **Package2** class of Microsoft SQL Server™ 2000 extends the **Package** class of SQL Server 7.0. For more information, see [Extended DTS Objects](#).

However, **Package2** objects cannot be created, and a **Package2** object variable cannot be declared **WithEvents**. To create a **Package** object that is compatible with SQL Server 7.0, or one that does not use the new package features, use the following code example:

```
'Declare the object variable.  
Private WithEvents objPackage As DTS.Package  
...  
'Create the package object.  
Set objPackage = New DTS.Package
```

The **WithEvents** keyword must be omitted if package events are not to be handled. For more information about handling package events, see [DTS Package Events in Visual Basic](#).

Creating a Package2 Object

To create a **Package2** object that makes available the new DTS features, use the following code example:

```
'Declare the object variables.  
Private objPackage As DTS.Package2  
Private WithEvents objPkgEvents As DTS.Package  
...  
'Create the package object.  
Set objPackage = New DTS.Package
```

Set objPkgEvents = objPackage

The declaration of and assignment to **objPkgEvents** must be omitted i

When using late binding in Visual Basic, object variables are declared **As Object**. In the following example, the new package object is created the same way it was created in SQL Server 7.0:

'Declare the object variable.

Private objPackage As Object

...

'Create the package object.

Set objPackage = New DTS.Package

DTS Connections in Visual Basic

Add a **Connection** object for each database or other OLE DB store you want to access. ODBC data sources can be accessed through the Microsoft® OLE DB provider for ODBC (MSDASQL).

Here are the basic steps for adding a **Connection** object in Microsoft Visual Basic®:

1. Declare an object variable of the appropriate type.
Use the **Connection** class in the object variable declaration if the application must be compatible with Microsoft SQL Server™ version 7.0. Use **Connection2** if it is to run only with SQL Server 2000.
2. Create the object with the **New** method of the **Connections** collection of the **Package2** object. Pass the programmatic identifier (ProgID) of the appropriate OLE DB provider to **New** as an argument.
3. Set properties to identify the connection and data source.
For more information, see [Connection2 Object](#).
4. Use the **Add** method of the **Connections** collection of the **Package2** object to add the **Connection** object to the package.

Creating a Connection with the Microsoft OLE DB Provider for SQL Server

The following code example shows you how to create a connection using the Microsoft OLE DB Provider for SQL Server. The **New** method references a specific version of the SQLOLEDB provider. If you do not need a specific version, you should use the version-independent ProgID, in this case "SQLOLEDB" rather than "SQLOLEDB.1":

```
'Declare the object variable.  
Private objConnect As DTS.Connection2  
Private objPackage As DTS.Package2
```

...

```
'Create the connection object. The package is already created at this po
Set objConnect = objPackage.Connections.New("SQLOLEDB.1")
With objConnect
    .ID = 1
    .DataSource = "(local)"
    .UseTrustedConnection = True
End With
Set objPackage.Connections.Add = objConnect
```

Creating a Connection with the Microsoft OLE DB Provider for Jet

The following code example shows you how to create a connection using the Microsoft OLE DB Provider for Jet:

```
'Establish a connection to the Access database.
Set objConnect = objPackage.Connections.New("Microsoft.Jet.OLEDB.4.0")
objConnect.ID = 2
objConnect.DataSource = "D:\DTS_UE\Data\JetPubs.mdb"
Set objPackage.Connections.Add = objConnect
```

Creating a Connection with the Data Shaping Service for OLE DB

The following code example shows you how to create a connection using the Microsoft Data Shaping Service for OLE DB. The example also illustrates how you can use the **ConnectionProperties** collection to access the properties of the specific OLE DB provider:

```
Set objConnect = objPackage.Connections.New("MSDataShape")
With objConnect
    .ConnectionProperties("Data Provider") = "SQLOLEDB"
    .ID = 1
    .Catalog = "Northwind"
    .UserID = "sa"
End With
objPackage.Connections.Add objConnect
```

After the **Connection** object has been added to the **Connections** collection, the object variable is no longer needed and can be reused for another connection or set to **Nothing** to release its reference.

DTS Package Workflow in Visual Basic

You create workflow in Data Transformation Services (DTS) packages by assigning all the tasks to steps and defining precedence relationships between the steps. **Task** objects that are not assigned to steps can be included in the package, but they will not be executed.

Creating DTS Step Objects

Here are the basic steps for adding **Step** objects in Microsoft® Visual Basic®:

1. Create a **Step** object with the **New** method of the **Steps** collection of the **Package2** object.
2. Assign a unique step name to the **Name** property and assign the name of the associated task to the **TaskName** property.
3. Set other **Step** object properties, as appropriate.

If package event handlers coded in Visual Basic are being used, the **ExecuteInMainThread** property must be set TRUE. Visual Basic does not support free threading, which DTS uses.

4. Use the **Add** method of the **Step** collection to add the **Step** object to the collection.

Example

The following code example shows you how to create, include, and assign a task to a **Step** object:

```
'Declare the step and the generic and class-specific task.  
Dim objTask As DTS.Task  
Dim objStep As DTS.Step  
Dim objDataPump As DTS.DataPumpTask2
```

...

```
'Create the step and task, and then link the step to the task.  
Set objStep = objPackage.Steps.New  
Set objTask = objPackage.Tasks.New("DTSDDataPumpTask")  
Set objDataPump = objTask.CustomTask  
objDataPump.Name = "LowerCaseTask"  
With ObjStep  
    .Name = "LowerCaseStep"  
    .TaskName = objDataPump.Name  
    .ExecuteInMainThread = True  
End With  
objPackage.Steps.Add objStep
```

Creating Precedence Constraint Objects

Unless otherwise constrained, package steps run in parallel, up to the limit specified by the package **MaxConcurrentSteps** property. To serialize step execution, create and add **PrecedenceConstraint** objects to the **PrecedenceConstraints** collection of the appropriate **Step** objects.

When one task, the successor task, is not to start execution until some event associated with another task, the predecessor task, occurs, a **PrecedenceConstraint** object that names the predecessor task is added to the **PrecedenceConstraints** collection of the successor task. Typically, these events will be step completion, step successful completion, and step failure.

Here are the basic steps for configuring a **PrecedenceConstraint** object in Visual Basic:

1. Create the **PrecedenceConstraint** object with the **New** method of the **PrecedenceConstraints** collection of the successor task. Use the name of the predecessor task as the argument to **New**.
2. Set the **PrecedenceBasis** property to indicate whether the constraint is to use step status or step result, and set the **Value** property to the appropriate step status or result code.

For more information, see [PrecedenceBasis Property](#) and [Value Property](#).

3. **Add** the **PrecedenceConstraint** object to the **PrecedenceConstraints** collection of the successor task.

Example

In the following example, step **TransformData** does not run until step **ClearTables** completes. Step **GenerateDoc** does not run until **TransformData** completes successfully. Step **SendMail** runs only if **TransformData** fails and begins execution at that point:

```
'Declare the step and precedence constraint.
```

```
Dim objStep As DTS.Step
```

```
Dim objConstraint As DTS.PrecedenceConstraint
```

```
...
```

```
'TransformData only runs when ClearTables completes.
```

```
Set objStep = objPackage.Steps("TransformData")
```

```
Set objConstraint = objStep.PrecedenceConstraints.New("ClearTables'
```

```
objConstraint.PrecedenceBasis = DTSSStepPrecedenceBasis_ExecStatu
```

```
objConstraint.Value = DTSSStepExecStat_Completed
```

```
objStep.precedenceConstraints.Add objConstraint
```

```
'GenerateDoc only runs when TransformData is successful.
```

```
Set objStep = objPackage.Steps("GenerateDoc")
```

```
Set objConstraint = objStep.PrecedenceConstraints.New("TransformD
```

```
objConstraint.PrecedenceBasis = DTSSStepPrecedenceBasis_ExecResu
```

```
objConstraint.Value = DTSSStepExecResult_Success
```

```
objStep.precedenceConstraints.Add objConstraint
```

```
'SendMail only runs when TransformData fails.
```

```
Set objStep = objPackage.Steps("SendMail")
```

```
Set objConstraint = objStep.PrecedenceConstraints.New("TransformD
```

```
objConstraint.PrecedenceBasis = DTSSStepPrecedenceBasis_ExecResu
```

```
objConstraint.Value = DTSSStepExecResult_Failure  
objStep.precedenceConstraints.Add objConstraint
```

After the **Step** and **PrecedenceConstraint** objects have been added to the **Steps** and **PrecedenceConstraints** collections, respectively, the object variables are no longer needed and can be reused for other objects or set to **Nothing** to release their references.

DTS Tasks in Visual Basic

In Microsoft® Visual Basic®, a Data Transformation Services (DTS) task consists of a generic **Task** object and a class-specific task object, such as **DTSFTPTask**, **DataPumpTask2** or **DynamicPropertiesTask** object. For more information about the task classes supplied with Microsoft SQL Server™ 2000, see [Task Objects](#).

The basic steps for adding a DTS task in Visual Basic are as follows:

1. Declare an object variable of type **Task** and a class-specific task object variable.
2. Create the task with the **New** method of the **Tasks** collection of the **Package2** object. Pass the programmatic identifier (ProgID) of the task class to **New** as an argument.
3. Use the **CustomTask** property of the **Task** object to return a reference to the class-specific task object.

Do not declare the class-specific object variable as **DTS.CustomTask**. If you do, the class-specific properties and methods will not be accessible. However, you can use late binding and declare it as **Object**.

4. Assign a unique name to the **Name** property, either of the **Task** object or class-specific task object.

This name also must be assigned to the **TaskName** property of the **Step** object. For more information, see [DTS Package Workflow in Visual Basic](#).

5. Set other class-specific task object properties as necessary.

Most task classes require a reference to one or more **Connection** objects. Usually, you do this through a property such as **SourceConnectionID** of the class-specific task object.

6. Add the **Task** object to the package with the **Add** method of the **Tasks** collection.

Data Pump Task Example

The following code example shows one way to create a Data Pump task, link to **Connection** objects, and assign names to the source and destination tables. Use of fully qualified table names, as shown here, makes it unnecessary to set the **Catalog** property of the corresponding **Connection** object.

'Declare the generic and class-specific task variables.

```
Dim objTask As DTS.Task
```

```
Dim objDataPump As DTS.DataPumpTask2
```

```
...
```

'Create the task and then link the task to the connections.

```
Set objTask = objPackage.Tasks.New("DTSDataPumpTask")
```

```
Set objDataPump = objTask.CustomTask
```

```
objDataPump.Name = "LowerCaseTask"
```

```
With objDataPump
```

```
    .SourceConnectionID = 1
```

```
    .SourceObjectName = "pubs..authors"
```

```
    .DestinationConnectionID = 2
```

```
    .DestinationObjectName = "[DTS_UE].[dbo].[AuthNames]"
```

```
End With
```

```
...
```

```
objPackage.Tasks.Add objTask
```

File Transfer Protocol Task Example

The following code example shows one way to create a File Transfer Protocol (FTP) task and assign the files to be copied and the destination directory to the appropriate properties:

Note The **DTSFTPTask** object requires that Microsoft Internet Explorer 5 be installed on the computer on which the task is to run. Internet Explorer 5 is supplied with SQL Server 2000, but not with SQL Server 2000 Desktop Engine.

'Declare the generic and FTP task objects.

```
Dim objTask As DTS.Task
```

```
Dim objFTPTask As DTSCustTasks.DTSFTPTask
```

'Create the task. Specify the files, the source, and the destination direct

```
Set objTask = objPackage.Tasks.New("DTSFTPTask")
```

```
Set objFTPTask = objTask.CustomTask
```

```
objFTPTask.Name = "FTPSrcDirTask"
```

```
With objFTPTask
```

```
    .SourceLocation = DTSFTPSourceLocation_Directory
```

```
    .SourceSite = "I:\DTS\TestData"
```

```
    .SourceFilename = _
```

```
        "File3.dat';";'123';'NWProdWiz.XLS';";'458240';"
```

```
    .DestSite = "D:\DTS_UE\Dest"
```

```
End With
```

```
objPackage.Tasks.Add objTask
```

After the **Task** object has been added to the **Tasks** collection, the object variables are no longer needed and can be reused for another task or set to **Nothing** to release their references.

DTS Transformations in Visual Basic

A Data Transformation Services (DTS) transformation consists of a **Transformation** object and a class-specific transform server object, such as **DataPumpTransformScript** or **DataPumpTransformTrimString**. For more information about the transformations supplied with Microsoft® SQL Server™ 2000, see [Transformation Objects](#).

Here are the basic steps for adding a **Transformation** object to a Microsoft Visual Basic® file:

1. Declare an object variable of **Transformation** type and a transform server object variable of the appropriate type.
2. Create the transformation with the **New** method of the **Transformations** collection of the class-specific task object or **TransformationSet** object. Pass the programmatic identifier (ProgID) of the transformation class to **New** as an argument.
3. Use the **TransformServer** property of the **Transformation** object to return a reference to the transform server object.
4. Assign a name unique among the objects in the **Transformations** collection to the **Name** property.
5. Set other **Transformation** or transform server object properties as necessary.
6. Add the **Transformation** object to the task with the **Add** method of the **Transformations** collection.

Example

The following code example shows you how to create a Copy Column transformation that does not have **Column** objects added. This transformation copies all columns. The source and destination must have the same number of columns, and this must be the only **Transformation** object in the **Transformations** collection.

'Declare the class-specific task and the transformation.

```
Dim objTransform As DTS.Transformation2
```

```
Dim objPumpTask As DTS.DataPumpTask2
```

```
...
```

'Create and add the transformation.

```
Set objTransform = objPumpTask.Transformations.New( _  
    "DTS.DataPumpTransformCopy")
```

```
objTransform.Name = "Transform"
```

```
objTransform.TransformFlags = _  
    DTSTransformFlag_AllowLosslessConversion
```

```
objPumpTask.Transformations.Add objTransform
```

DTS Column Objects in Visual Basic

You must specify the columns to be transformed when:

- There are multiple **Transformation** objects in the **Transformations** collection.
- The number of source and destination columns is different (for example, if you are not transforming all columns).
- The order of the source and destination columns does not match.

Here are the basic steps for adding **Column** objects to a transformation in Microsoft® Visual Basic®:

1. Create each **Column** object with the **New** method of the **SourceColumns** or **DestinationColumns** collections of the **Transformation** object.
2. Specify the column name and ordinal position as arguments to **New**.
3. Set the properties of the **Column** object as appropriate.
4. Use the **Add** method of the **SourceColumns** or **DestinationColumns** collection to add the **Column** object to the appropriate collection.

Example

The following example shows you how to create and include one source and one destination **Column** object and to use the transform server object to set class-specific properties:

'Declare the class-specific task, the transformation, the column, and the

```

Dim objDataPump As DTS.DataPumpTask
Dim objTransform As DTS.Transformation
Dim objColumn As DTS.Column
Dim objMidString As DTSPump.DataPumpTransformMidString

...
'Create an area code transformation.
' create columns, define start and width
Set objTransform = objDataPump.Transformations. _
    New("DTSPump.DataPumpTransformMidString")
With objTransform
    .Name = "AreaCodeTransform"
    Set objColumn = .SourceColumns.New("phone", 1)
    .SourceColumns.Add objColumn
    Set objColumn = .DestinationColumns.New("AreaCode", 1)
    .DestinationColumns.Add objColumn
End With
Set objMidString = objTransform.TransformServer
objMidString.CharacterStart = 1
objMidString.CharacterCount = 3
objDataPump.Transformations.Add objTransform

```

Using the AddColumn Method

If it is not necessary to set **Column** object properties, you can use the **AddColumn** method of the **SourceColumns** and **DestinationColumns** collections to create the column object and add it to the appropriate collection in a single step. However, **AddColumn** does not return a reference to the **Column** object. Usually, it is not necessary to set **Column** object properties like **DataType**, **Nullable**, **Precision** and **Size**, because the defaults are set from the corresponding column in the data source or destination.

Example

The following code example shows you how to use **AddColumn** to create and

add the **Column** objects. The declarations from the previous example apply to this one, as well.

```
'Create a transformation for the local phone number. Create the columnr
Set objTransform = objDataPump.Transformations. _
    New("DTSPump.DataPumpTransformMidString")
With objTransform
    .Name = "LocalNumTransform"
    .SourceColumns.AddColumn "phone", 1
    .DestinationColumns.AddColumn "LocalNumber", 1
End With
Set objMidString = objTransform.TransformServer
objMidString.CharacterStart = 5
objMidString.CharacterCount = 8
objDataPump.Transformations.Add objTransform
```

If you need to reference a column property when you do not have a reference to the **Column** object, you can do it through the collection that contains it. The following code example shows you how to set the **Nullable** property of the **phone** column from the previous example:

```
objTransform.SourceColumns("phone").Nullable = True
```

After the **Transformation** object has been added to the **Transformations** collection, the object variables are no longer needed and can be reused for other objects or set to **Nothing** to release their references.

DTS Lookups in Visual Basic

You can create and configure a **Lookup** object to look up data in another query rowset through a separate connection.

Here are the basic steps for configuring a **Lookup** object:

1. Use the **New** method of the **Lookups** collection of a **DataDrivenQueryTask(2)**, **DataPumpTask(2)** or **TransformationSet** object of a **ParallelDataPumpTask** object to create the **Lookup** object.
2. Use the **ConnectionID** property to specify the connection, which should be different from the source and destination connections for the task.
3. Specify the text of a query that returns a rowset (for example, a Select query or stored procedure) with the **Query** property.

The query should have one or more parameters denoted by the "?" placeholder. The query should be designed to return a rowset with a single row. Additional rows in the rowset are ignored.
4. Use the **Add** method of the **Lookups** collection to add the **Lookup** object to the collection.

Example

The following code example shows you how to create and configure a **Lookup** object:

```
'Declare a lookup object and a class-specific task.  
Dim objLookup As DTS.Lookup  
Dim objDataPump As DTS.DataPumpTask2
```

...

```
'Define the lookup object.  
Set objLookup = objDataPump.Lookups.New("JobDesc")  
With objLookup  
    .ConnectionID = 2  
    .Query = "SELECT job_desc FROM jobs " & vbCrLf  
    .Query = .Query & "WHERE job_id = ?"  
    .MaxCacheRows = 60  
End With  
objDataPump.Lookups.Add objLookup
```

In the Microsoft® ActiveX® script, the lookup must be referenced with the **Execute** method of an element of the **DTSLookups** collection, as illustrated in the following code example:

```
DTSDestination( "JobDesc" ) = _  
    DTSLookups( "JobDesc" ).Execute( DTSSource( "job_id" ) )
```

If the lookup rowset has more than one column, the **Execute** method returns a Variant array. The script may need to iterate through the array to use multiple values.

After the **Lookup** object has been added to the **Lookups** collection, the object variable is no longer needed and can be reused for other objects or set to **Nothing** to release its reference.

DTS Global Variables in Visual Basic

Global variables that do not exist when first referenced during Data Transformation Services (DTS) package execution are created at that time. Prior to package execution, you can create global variables explicitly by adding a **GlobalVariable** object to the package.

Here are the basic steps for creating a global variable in a DTS package prior to package execution:

1. Use the **New** method of the **GlobalVariables** collection of the **Package2** object.
2. Set the **Value** property of the created **GlobalVariable** object to the initial value of the global variable.
3. Add the object to the package with the **Add** method of the **GlobalVariables** collection.

Example

The following code example shows you how to create a global variable named **ALuckyName** initialized with the string "SevenSevenSeven":

'Declare the package and global variable objects.

```
Dim objPackage As DTS.Package2
```

```
Dim objGlobal As DTS.GlobalVariable
```

```
...
```

'Define the global variable.

```
Set objGlobal = objPackage.GlobalVariables.New("ALuckyName")
```

```
objGlobal.Value = "SevenSevenSeven"
```

```
objPackage.GlobalVariables.Add objGlobal
```

Alternatively, the **AddGlobalVariable** method of the **GlobalVariables**

collection creates the **GlobalVariable** object and adds it to the collection in a single step. However, it does not return a reference to the object. The following code sample shows you how to create global variable **ALuckyName** using **AddGlobalVariable**:

```
'Define the global variable.  
objPackage.GlobalVariables.AddGlobalVariable _  
    "ALuckyName", "SevenSevenSeven"
```

After the **GlobalVariable** object has been added to the **GlobalVariables** collection, the object variable is no longer needed and can be reused for other objects or set to **Nothing** to release its reference.

DTS ActiveX Scripts in Visual Basic

You assign Microsoft® ActiveX® scripts to a property of objects that require scripts as a single text string. That text string can include embedded carriage return/line feed pairs. If the script string constant contains the " character, represent it as double quotations "", as you would in any Microsoft Visual Basic® string constant. Each scripted object also has properties for the script language and the script function name.

For the **ActiveScriptTask**, the script text is assigned to the **ActiveXScript** property. The **ScriptLanguage** and **FunctionName** properties are used to specify the scripting language and function entry point.

For the **DataPumpTransformScript** transformation, the script text is assigned to the **Text** property. The **Language** and **FunctionEntry** properties are used for the script language and function name, respectively.

Example

The following code example assigns a Microsoft Visual Basic Scripting Edition (VBScript) function to a **DataPumpTransformScript** transformation used by a **DataPumpTask2** object:

```
'Declare the class-specific task, the transformation, and the transform s
Dim objDataPump      As DTS.DataPumpTask
Dim objTransformation As DTS.Transformation
Dim objTransScript   As DTSPump.DataPumpTransformScript

'Create the transformation and the transform server object. Then assign
'  objDataPump already exists at this point
Set objTransformation = objDataPump.Transformations.New( _
    "DTSPump.DataPumpTransformScript")
Set objTransScript = objTransformation.TransformServer
With objTransScript
    .Text = "Function Main()" & vbCrLf
```

```
.Text = .Text & "  DTSDestination( ""emp_id"" ) = _  
    DTSSource( ""emp_id"" )" & vbCrLf  
.Text = .Text & "  DTSDestination( ""Name"" ) = _  
    DTSSource( ""lname"" ) & ", "" & DTSSource( ""fname"" )" &  
.Text = .Text & "  DTSDestination( ""JobDesc"" ) = _  
    DTSLookups( ""JobDesc"" ).Execute( DTSSource( ""job_id"" ) )  
    & vbCrLf  
.Text = .Text & "  DTSDestination( ""PubName"" ) = _  
    DTSLookups( ""PubName"" ).Execute( DTSSource( ""pub_id"" )  
    & vbCrLf  
.Text = .Text & "  Main = DTSTransformStat_OK" & vbCrLf  
.Text = .Text & "End Function"  
.Language = "VBScript"  
.FunctionEntry = "Main"  
End With  
...  
objDataPump.Transformations.Add objTransScript
```

DTS Query Strings in Visual Basic

You can assign queries to a property of the object using a query as a text string. The property name differs for each object that uses a query. The string can include carriage return/line feed pairs. When the query string is a constant and contains the " character, represent it as double quotations "", as in all Microsoft® Visual Basic® string constants.

If you are supplying source or destination queries for the **DataPumpTask(2)**, **DataDrivenQueryTask(2)** or **ParallelDataPumpTask**, you must assign a single Select or stored procedure query to the **SourceSQLStatement** or **DestinationSQLStatement** properties. The destination query defines a rowset into which destination rows are inserted. This feature is not accessible to packages created in Data Transformation Services (DTS) Designer.

In the **DynamicPropertiesTaskAssignment** object of the **DynamicPropertiesTask**, assign a single Select or stored procedure query that returns a rowset to the **SourceQuerySQL** property. This query returns a single row containing a single column. Additional fields in the rowset are ignored.

Example

The following code example shows one way to assign the source and destination queries for the **ParallelDataPumpTask** object:

```
'Declare the generic and class-specific task object variables.
```

```
Dim objTask As DTS.Task
```

```
Dim objParallelPumpTask As DTS.ParallelDataPumpTask
```

```
'Create the ParallelDataPumpTask and assign the connections and quer
```

```
Set objTask = objPackage.Tasks.New("DTSParallelDataPumpTask")
```

```
Set objParallelPumpTask = objTask.CustomTask
```

```
With objParallelPumpTask
```

```
    .TransformationSetOptions = DTSTranSetOpt_DataDrivenQueries
```

```
    .SourceConnectionID = 1
```

```

.SourceSQLStatement = _
    "SHAPE {SELECT au_id, au_lname, au_fname FROM authors}
    "APPEND ({SELECT au_id, title FROM titleauthor TA, titles TS
        "WHERE TA.title_id = TS.title_id} " & _
        "AS title_chap RELATE au_id TO au_id)"
.DestinationConnectionID = 2
.DestinationSQLStatement = _
    "SHAPE {SELECT * FROM AuthNames} " & _
    "APPEND ({SELECT * FROM TitleNames} " & _
        "AS TitleChap RELATE AuthID TO AuthID)"
End With

```

Defining Execute SQL Task Queries

With the **ExecuteSQLTask** object, you must assign a sequence of one or more queries to the **SQLStatement** property. You can insert the Transact-SQL **GO** statement to break the query sequence into batches.

Example

The following code example shows you how to use the **SQLStatement** property with a batched sequence of SQL statements:

'Declare the generic and class-specific task object variables.

```
Dim objTask As DTS.Task
```

```
Dim objExecSQL As DTS.ExecuteSQLTask
```

'Create the ParallelDataPumpTask and assign the connections and quer

```
Set objTask = objPackage.Tasks.New("DTSExecuteSQLTask")
```

```
Set objExecSQL = objTask.CustomTask
```

With objExecSQL

```
.Name = "PubsEmplClearOut"
```

```
.SQLStatement = "USE DTSTest" & vbCrLf & _
```

```
"TRUNCATE TABLE EmployeeHires" & vbCrLf & "GO" & vbC
```

```
"UPDATE MissingStuff SET MissingMI = 0" & vbCrLf & _
```

```
"UPDATE StartYearCounts SET NumberStarts = 0" & vbCrLf &  
.ConnectionID = 1  
End With  
objPackage.Tasks.Add objTask
```

Defining Data Driven Queries

With the **DataDrivenQueryTask(2)** and the **TransformationSet** object of the **ParallelDataPumpTask** (when in **DTSTranSetOpt_DataDrivenQueries** mode), you can define up to four queries by assigning query strings to the **InsertQuery**, **UpdateQuery**, **DeleteQuery** and **UserQuery** properties.

Each query string consists of a sequence of one or more SQL statements or stored procedure references, although the Transact-SQL GO statement cannot be included. Use the parameter placeholder ? to indicate where you have substituted a reference to a destination column. These columns are defined by creating **Column** objects and adding them to one of the **InsertQueryColumns**, **UpdateQueryColumns**, **DeleteQueryColumns**, **UserQueryColumns** collections. Use the **New** method of the appropriate collection, followed by the **Add** method. Alternatively, you can create and add the **Column** object in a single step with the **AddColumn** method.

Example

The following code example shows how you could define the Insert query, the Update query, and the associated destination columns for a **DataDrivenQueryTask2** object:

```
'Declare the generic and class-specific task object variables.
```

```
Dim objTask As DTS.Task
```

```
Dim objDDQTask As DTS.DataDrivenQueryTask
```

```
'Create the DataDrivenQueryTask and assign the connections and quer
```

```
Set objTask = objPackage.Tasks.New("DTSDataDrivenQueryTask")
```

```
Set objDDQTask = objTask.CustomTask
```

```
With objDDQTask
```

```
    .Name = "PubsDDQ"
```

```

.SourceConnectionID = 2
.SourceObjectName = "[pubs].[dbo].[employee]"
.DestinationConnectionID = 1
.DestinationObjectName = "[DTSTest].[dbo].[EmployeeHires]"
.InsertQuery = "UPDATE StartYearCounts" & vbCrLf & _
    "SET NumberStarts = NumberStarts + 1 " & vbCrLf & _
    "WHERE StartYear = datepart( yyyy, ? )" & vbCrLf & _
    "INSERT INTO EmployeeHires (FullName, StartDate) VALUES
.UpdateQuery = "UPDATE StartYearCounts" & vbCrLf & _
    "SET NumberStarts = NumberStarts + 1 " & vbCrLf & _
    "WHERE StartYear = datepart( yyyy, ? )" & vbCrLf & _
    "UPDATE MissingStuff SET MissingMI = MissingMI + 1" & vbCrLf & _
    "INSERT INTO EmployeeHires (FullName, StartDate) VALUES
With .InsertQueryColumns
    .AddColumn "StartDate", 1
    .AddColumn "FullName", 2
    .AddColumn "StartDate", 3
End With
With .UpdateQueryColumns
    .AddColumn "StartDate", 1
    .AddColumn "FullName", 2
    .AddColumn "StartDate", 3
End With
...
End With
objPackage.Tasks.Add objTask

```

Defining Lookup Queries

For the **Lookup** object, a single query that generates a rowset is assigned to the **Query** property. This query must contain one or more ? parameter placeholders. For more information, see [DTS Lookups in Visual Basic](#).

DTS Package Events in Visual Basic

To handle Data Transformation Services (DTS) package events in Microsoft® Visual Basic®, declare the **Package** object variable **WithEvents**. A **Package2** object variable cannot be declared **WithEvents**. If you want to use the new features of the **Package2** object and also handle package events, create a **Package** object and assign it to a **Package2** object variable. You can use this object to access the package properties and methods. Also assign the **Package2** object variable to a **Package** object variable that has been declared **WithEvents**.

After you have declared a **Package** object variable **WithEvents**, you must provide event handlers for all the package events. If you fail to do so, you will typically receive an access violation error at the time the unhandled event is raised.

Because Visual Basic objects do not support multiple threads, you also must set the **ExecuteInMainThread** property to TRUE for each **Step** object in the package.

Example

The following code example is a private class that handles package events after its **PackageObj** property has been set.

Note The **OnQueryCancel** and **OnStart** event handlers consist of only a comment. This comment satisfies the requirement that a handler be supplied for these events and causes Visual Basic to compile the Subs. You also can use the single statement **Exit Sub**.

Option Explicit

```
Private WithEvents objPackage As DTS.Package
```

```
...
```

```
Private Sub objPackage_OnError(ByVal EventSource As String, _  
    ByVal ErrorCode As Long, ByVal Source As String, _  
    ByVal Description As String, ByVal HelpFile As String, _  
    ByVal HelpContext As Long, _
```

```
    ByVal IDofInterfaceWithError As String, pbCancel As Boolean  
Dim sMsg As String
```

```
sMsg = "EventSource: " & EventSource & vbCrLf & _  
      "ErrorCode: " & (ErrorCode) & vbCrLf & _  
      "Source: " & Source & vbCrLf & _  
      "Description: " & Description & vbCrLf & _  
      "HelpFile: " & HelpFile & vbCrLf & _  
      "IDofIFWErr: " & IDofInterfaceWithError  
MsgBox sMsg, vbExclamation, "OnError"
```

```
End Sub
```

```
Private Sub objPackage_OnFinish(ByVal EventSource As String)  
    MsgBox EventSource, vbInformation, "OnFinish"  
End Sub
```

```
Private Sub objPackage_OnProgress(ByVal EventSource As String, _  
    ByVal ProgressDescription As String, _  
    ByVal PercentComplete As Long, _  
    ByVal ProgressCountLow As Long, _  
    ByVal ProgressCountHigh As Long)  
Dim sMsg As String
```

```
sMsg = "EventSource: " & EventSource & vbCrLf & _  
      "ProgressDescr: " & ProgressDescription & vbCrLf & _  
      "PercentComplete: " & (PercentComplete) & vbCrLf & _  
      "ProgressCountLow: " & (ProgressCountLow) & vbCrLf & _  
      "ProgressCountHigh: " & (ProgressCountHigh)  
MsgBox sMsg, vbExclamation, "OnProgress"
```

```
End Sub
```

```
Private Sub objPackage_OnQueryCancel(ByVal EventSource As String,
    pbCancel As Boolean)
    MsgBox EventSource, vbInformation, "OnQueryCancel"
End Sub
```

```
Private Sub objPackage_OnStart(ByVal EventSource As String)
    MsgBox EventSource, vbInformation, "OnStart"
End Sub
```

```
Public Property Get PackageObj() As DTS.Package2
    Set PackageObj = objPackage
End Property
```

```
Public Property Set PackageObj(ByVal oNewPack As DTS.Package2)
    Set objPackage = oNewPack
End Property
```

DTS Programming

Managing DTS Package Programs in Visual Basic

The topics that follow describe the implementation of DTS functions in Microsoft® Visual Basic®. These functions use the Data Transformation Services (DTS) object model.

Topic	Description
Executing DTS Packages in Visual Basic	Explains how to use the Execute method of the Package2 object and describes what you need to consider before reusing Package2 .
Handling DTS Errors in Visual Basic	Explains how to detect errors that occur during DTS package execution.
Saving DTS Packages in Visual Basic	Explains how a Package2 object can save the package to storage and how to load a DTS package into a Package2 object.
Running a Package Saved as a Visual Basic File	Explains how to incorporate a DTS package saved as a Visual Basic file by a DTS tool into a Visual Basic project.
Retrieving DTS Information in Visual Basic	Explains how to obtain package and version information from a DTS package storage file.

Executing DTS Packages in Visual Basic

After you have created the necessary Data Transformation Services (DTS) objects, set their properties and added them to the appropriate collections, use the **Execute** method of the **Package2** object to run the package. For more information about handling errors raised by the **Execute** method, see [DTS Error Handlers in Visual Basic](#).

If the **Package2** object is to be used again (for example, for saving or running), or if the application is to perform significant processing outside of DTS after the DTS package is run, it is recommended that you call the **Package2 UnInitialize** method. **UnInitialize** performs various clean-ups, for example, re-initializing global variables, closing user-opened connections, closing the log, releasing threads, and terminating event connection points.

Before calling **UnInitialize**, it is strongly recommended that you release references to all DTS objects, except the **Package2** object, through which you are going to run **Uninitialize**. This includes additional **Package** or **Package2** object variables that you have declared **WithEvents** in order to handle package events.

You can release references by either setting the appropriate object variables to **Nothing** or arranging your code so that they go out of scope. If you fail to do this, resources such as computer memory will not be released by **Uninitialize**, giving the appearance of a memory leak.

Example

The following code example shows a DTS package using an **ExecutePackageTask**, through the **Execute** and **UnInitialize** methods:

```
Private WithEvents mobjPkgEvents As DTS.Package
...
Private Sub RunPackage()
'Run the package stored in file C:\DTS_UE\TestPkg\VarPubsFields.dts
Dim objPackage As DTS.Package2
```

```
Dim objStep      As DTS.Step
Dim objTask      As DTS.Task
Dim objExecPkg   As DTS.ExecutePackageTask
```

```
On Error GoTo PackageError
Set objPackage = New DTS.Package
Set mobjPkgEvents = objPackage
objPackage.FailOnError = True
```

'Create the step and task. Specify the package to be run, and link the st

```
Set objStep = objPackage.Steps.New
Set objTask = objPackage.Tasks.New("DTSExecutePackageTask")
Set objExecPkg = objTask.CustomTask
With objExecPkg
```

```
    .PackagePassword = "user"
    .FileName = "C:\DTS_UE\TestPkg\VarPubsFields.dts"
    .Name = "ExecPkgTask"
```

```
End With
```

```
With objStep
```

```
    .TaskName = objExecPkg.Name
    .Name = "ExecPkgStep"
    .ExecuteInMainThread = True
```

```
End With
```

```
objPackage.Steps.Add objStep
objPackage.Tasks.Add objTask
```

'Run the package and release references.

```
objPackage.Execute
```

```
Set objExecPkg = Nothing
Set objTask = Nothing
Set objStep = Nothing
Set mobjPkgEvents = Nothing
```

```
objPackage.UnInitialize  
End Sub
```

Handling DTS Errors in Visual Basic

You need to take the steps described below when handling errors in Data Transformation Services (DTS) applications implemented in Microsoft® Visual Basic®.

DTS applications typically consist of two phases:

- In the first phase, the applications create DTS objects, set their properties, and add them to collections of parent objects.

Errors that occur during the object creation/property definition phase can be handled by a typical Visual Basic error handler.

- In the second phase, the **Execute** method of the **Package2** object is invoked.

Errors that occur during the **Execute** will not be propagated back to the caller unless the **FailOnError** property of the **Package2** object is set to **TRUE**.

When **FailOnError** is **TRUE**, the description of the returned error will often tell you only that the package failed because a (named) step failed. To determine why a step failed, the **GetExecutionErrorInfo** method of the **Step** object will return the properties of a Visual Basic error object that describe the error.

Troubleshooting Package Execution

To determine the step(s) that raised errors, the **ExecutionStatus** property of the **Step** object should have the value **DTSSStepExecStat_Completed** (in enum **DTS.DTSSStepExecStatus**) and the **ExecutionResult** property should have the value **DTSSStepExecResult_Failure** (in enum **DTS.DTSSStepExecResult**). If **FailOnError** is **TRUE**, there will only be one such step. If not, there may be multiple failed steps, depending on the package workflow. The error handler should iterate through all the objects in the **Steps** collection and not stop when it finds an error.

Error Handler Example

The following code example is a typical error handler that could be used while a package is being developed, and **FailOnError** is set to TRUE. If failing the package on the first error is undesirable, the **sAccumStepErrors** function could still be used, but it would need to be called following a normal return from **objPackage.Execute**, as well as from the error handler.

```
Private Sub RunDTSPackage( )
    Dim objPackage      As New DTS.Package
    ...
    On Error GoTo PackageError
    ...
    objPackage.FailOnError = True
    objPackage.Execute
Exit Sub
```

PackageError:

```
    Dim sMsg  As String
    sMsg = "Package failed, error: " & sErrorNumConv(Err.Number) &
        vbCrLf & Err.Description & vbCrLf & sAccumStepErrors(objPac
    MsgBox sMsg, vbExclamation, objPackage.Name
    Exit Function
End Sub
```

```
Private Function sAccumStepErrors( _
    ByVal objPackage As DTS.Package) As String
'Accumulate the step error info into the error message.
    Dim oStep      As DTS.Step
    Dim sMessage   As String
    Dim lErrNum    As Long
    Dim sDescr     As String
    Dim sSource    As String
```

'Look for steps that completed and failed.

```

For Each oStep In objPackage.Steps
    If oStep.ExecutionStatus = DTSSStepExecStat_Completed Then
        If oStep.ExecutionResult = DTSSStepExecResult_Failure Then

            'Get the step error information and append it to the message.
            oStep.GetExecutionErrorInfo lErrNum, sSource, sDescr
            sMessage = sMessage & vbCrLf & _
                "Step " & oStep.Name & " failed, error: " & _
                sErrorNumConv(lErrNum) & vbCrLf & sDescr & vbCr

        End If
    End If
Next
sAccumStepErrors = sMessage
End Function

```

```

Private Function sErrorNumConv(ByVal lErrNum As Long) As String
'Convert the error number into readable forms, both hexadecimal and d

```

```

    If lErrNum < 65536 And lErrNum > -65536 Then
        sErrorNumConv = "x" & Hex(lErrNum) & ", " & CStr(lErrNum)
    Else
        sErrorNumConv = "x" & Hex(lErrNum) & ", x" & _
            Hex(lErrNum And -65536) & " + " & CStr(lErrNum And 65
    End If
End Function

```

Error Message Example

The following code example is the message generated by the above handler when a package with a connection that references a non-existent database is run:

```

Package failed, error: x80040428, x80040000 + 1064
Package failed because Step 'ParallelDPStep' failed.

```

Step ParallelDPStep failed, error: x80074005, x80070000 + 16389
Data provider could not be initialized. (Microsoft OLE DB Provider
for SQL Server (80004005): Cannot open database requested in login
'DTSFest'. Login fails.)

See Also

[DTSSStepExecResult](#)

[DTSSStepExecStatus](#)

[Execute \(Package\) Method](#)

[ExecutionResult Property](#)

[ExecutionStatus Property](#)

[FailOnError Property](#)

[GetExecutionErrorInfo Method](#)

Saving DTS Packages in Visual Basic

When you use the **Package2** object methods, you can save or load a package in the formats available to you through Data Transformation Services (DTS) Designer and the DTS Import/Export Wizard. You can save packages to Microsoft® SQL Server™ 2000, to SQL Server 2000 Meta Data Services and to a COM-structured storage file.

To save a package, use one of the following **Package2** methods:

- **SaveToSQLServer**
- **SaveToSQLServerAs**

- **SaveToRepository**
- **SaveToRepositoryAs**

- **SaveToStorageFile**

- **SaveToStorageFileAs**

If the package is run before being saved, call the **UnInitialize** method first.

To load a package, first create the **Package2** object and then invoke one of the following **Package2** object methods:

- **LoadFromSQLServer**

- **LoadFromRepository**

- **LoadFromStorageFile**

To delete a package from SQL Server or Meta Data Services, use the **Package2**

object **RemoveFromSQLServer** or **RemoveFromRepository** methods.

Example

The following code example shows a function loading a package in one format and saving it in another:

```
Enum eDTSPkgFormat
    REPOSITORY
    SQL_SERVER
    STORAGE_FILE
End Enum
```

```
Public Function blnCopyDTSPackage( _
    ByVal strReposServerName As String, ByVal strReposDBName As
    ByVal strReposUserName As String, ByVal strReposPassword As S
    ByVal blnReposNTAuth As Boolean, ByVal strSQLServerName As
    ByVal strSQLSvUserName As String, ByVal strSQLSvPassword As
    ByVal blnSQLSvNTAuth As Boolean, ByVal strPackageID As Strin
    ByVal strPackageVerID As String, ByVal strPackageName As String
    ByVal strPkgOwnerPwd As String, ByVal strPkgUserPwd As String
    ByVal strPkgUNCPath As String, ByVal dpfPkgSource As eDTSPkg
    ByVal dpfPkgDestination As eDTSPkgFormat) As Boolean
'Copy the DTS package source to the destination format.
Dim objPackage    As DTS.Package2
Dim rsfFlags      As DTS.DTSRepositoryStorageFlags
Dim ssfFlags      As DTS.DTSSQLServerStorageFlags
Dim strPhase      As String    'load/save phase for error msg
```

```
On Error GoTo ErrorHandler
```

```
'Copying the source to the destination in the same format is not support
If dpfPkgSource = dpfPkgDestination Then
    MsgBox "Same format for source and destination not supported", _
```

```

        vbExclamation
    Exit Function
End If

'Create the package object and calculate the storage flags.
Set objPackage = New DTS.Package
rsfFlags = IIf(blnReposNTAuth, DTSReposFlag_UseTrustedConnectio
                DTSReposFlag_Default)
ssfFlags = IIf(blnSQLSvNTAuth, DTSSQLStgFlag_UseTrustedConne
                DTSSQLStgFlag_Default)

'Load the package from the specified storage type.
strPhase = "loading"
Select Case dpfPkgSource
    Case REPOSITORY
        objPackage.LoadFromRepository _
            strReposServerName, strReposDBName, strReposUserName, _
            strReposPassword, strPackageID, strPackageVerID, _
            strPackageName, rsfFlags

    Case SQL_SERVER
        objPackage.LoadFromSQLServer _
            strSQLServerName, strSQLSvUserName, strSQLSvPassword,
            ssfFlags, strPkgOwnerPwd, strPackageID, _
            strPackageVerID, strPackageName

    Case STORAGE_FILE
        objPackage.LoadFromStorageFile _
            strPkgUNCPath, strPkgOwnerPwd, strPackageID, _
            strPackageVerID, strPackageName
End Select

'Save the package to the specified storage type.

```

```
strPhase = "saving"
Select Case dpfPkgDestination
  Case REPOSITORY
    objPackage.SaveToRepository _
      strReposServerName, strReposDBName, strReposUserName, _
      strReposPassword, rsfFlags

  Case SQL_SERVER
    objPackage.SaveToSQLServer _
      strSQLServerName, strSQLSvUserName, strSQLSvPassword,
      ssfFlags, strPkgOwnerPwd, strPkgUserPwd

  Case STORAGE_FILE
    objPackage.SaveToStorageFile _
      strPkgUNCPath, strPkgOwnerPwd, strPkgUserPwd
End Select
```

```
bInCopyDTSPackage = True
Exit Function
```

```
ErrorHandler:
MsgBox "Error " & strPhase & " DTS package: 0x" & Hex(Err.Number) &
vbCrLf & Err.Description, vbExclamation
Exit Function
End Function
```

Running a DTS Package Saved as a Visual Basic File

You can run a Data Transformation Services (DTS) package that has been saved by one of the DTS tools as a Microsoft® Visual Basic® file. The saved module, a Visual Basic .bas file, consists of declarations and a Sub Main and may contain other Subs called by Sub Main. The Subs contain all the logic of the DTS package.

Here are the basic steps for incorporating a Visual Basic module file into a Visual Basic project and executing it on a computer running the Microsoft SQL Server™ client tools:

1. In Visual Basic, create a new **Standard EXE** project.
2. On the **Project** menu, click **References**, and then select the **Microsoft DTSPackage Object Library**, **Microsoft DTS Custom Tasks Object Library**, and **Microsoft DTS Data Pump Scripting Object Library** check boxes.

Not all DTS programs will require all three of these libraries.

3. On the **Project** menu, click **Add File**, and then add the Visual Basic file produced by the DTS Import/Export Wizard or DTS Designer.
4. In the Project Explorer, select **Form1** and then on the **Project** menu, click **Remove Form1** to remove the blank form from the Project.
5. Run the project.

No indication of completion will be given other than the Visual Basic Development Environment will go back to design mode.

You may want to add completion notification and error handling and controls to allow the user to invoke the transformation.

Using the Visual Basic File to Save to SQL Server

The Visual Basic project you created from the generated Visual Basic file can be used to save the DTS package to SQL Server.

Here are the basic steps for saving Visual Basic files to SQL Server:

1. Go to the end of the Sub Main and uncomment the line
`'objPackage.SaveToSQLServer ...`
2. Comment out the following line
`objPackage.Execute`
3. Run the project.

When the Visual Basic Development Environment goes back to design mode, the package is saved to SQL Server.

The package can now be edited, maintained, and run from DTS Designer. It can be saved again as a Visual Basic file from DTS Designer.

See Also

[Executing DTS Packages in Visual Basic](#)

[Saving DTS Packages in Visual Basic](#)

[Saving a DTS Package to a Visual Basic File](#)

Retrieving DTS Information in Visual Basic

Data Transformation Services (DTS) provides features for requesting information about registered components and saved packages and for retrieving the contents of log records.

Registered Components

The **Application** object provides access to the system, package, and log data. You create it independently of a DTS package.

Use the **OLEDBProviderInfos**, **ScriptingLanguageInfos**, **TaskInfos**, and **TransformationInfos** collections of the **Application** object to obtain information about:

- OLE DB providers.
- Microsoft® ActiveX® scripting languages.
- DTS task classes and DTS transformation classes that are registered on the computer and can be used by DTS.

The DTS task and transformation classes include those supplied with Microsoft SQL Server™ and custom tasks and transformations implemented by other vendors and users.

Example

The following example creates a DTS Application object, then iterates through the collections named above to retrieve information about the registered components available to DTS.

To register components

1. In a Microsoft Visual Basic® development environment, create a new **Standard EXE** project.

2. On the **Project** menu, click **References**, and then select the **Microsoft DTSPackage Object Library** check box.
3. Place a textbox on **Form1**, and then accept the default name **Text1**.
4. Set the **MultiLine** property to **TRUE** and set the **ScrollBars** property to **3 - Both**.
5. Copy the following code into the code window for **Form1**, and then run the project:

Note Be sure to include the **Form_Resize** sub. It allows you to drag the borders of **Form1** to view the information.

```
Private Sub Form_Load()  
    Dim objDTSAppl As DTS.Application  
    Dim colScripInfo As DTS.ScriptingLanguageInfos  
    Dim objScripInfo As DTS.ScriptingLanguageInfo  
    Dim colOLEDBInfo As DTS.OLEDBProviderInfos  
    Dim objOLEDBInfo As DTS.OLEDBProviderInfo  
    Dim colTaskInfo As DTS.TaskInfos  
    Dim objTaskInfo As DTS.TaskInfo  
    Dim colTransInfo As DTS.TransformationInfos  
    Dim objTransInfo As DTS.TransformationInfo  
    Dim strMsg As String  
  
    Set objDTSAppl = New DTS.Application  
  
    strMsg = "OLEDB Provider Information" & vbCrLf & "=="  
    Set colOLEDBInfo = objDTSAppl.OLEDBProviderInfos  
    For Each objOLEDBInfo In colOLEDBInfo  
        strMsg = strMsg & vbCrLf & _
```

```
vbTab & "ClassID:" & vbTab & objOLEDBInfo.ClassID  
vbTab & "Descr:" & vbTab & objOLEDBInfo.Description  
vbTab & "File:" & vbTab & objOLEDBInfo.Implementat  
vbTab & "Version:" & vbTab & objOLEDBInfo.Implementat  
vbTab & "Name:" & vbTab & objOLEDBInfo.Name & vbTab & _  
vbTab & "Parse:" & vbTab & objOLEDBInfo.ParseName  
Next
```

```
strMsg = strMsg & vbCrLf & "Scripting Language Information"  
strMsg = strMsg & vbCrLf & "=====  
Set colScripInfo = objDTSApp1.ScriptingLanguageInfos  
For Each objScripInfo In colScripInfo  
strMsg = strMsg & vbCrLf & _  
vbTab & "ClassID:" & vbTab & objScripInfo.ClassID  
vbTab & "Descr:" & vbTab & objScripInfo.Description  
vbTab & "File:" & vbTab & objScripInfo.Implementat  
vbTab & "Version:" & vbTab & objScripInfo.Implementat  
vbTab & "Name:" & vbTab & objScripInfo.Name & vbTab & _  
Next
```

```
strMsg = strMsg & vbCrLf & "Registered DTS Task Information"  
strMsg = strMsg & vbCrLf & "=====  
Set colTaskInfo = objDTSApp1.TaskInfos  
For Each objTaskInfo In colTaskInfo  
strMsg = strMsg & vbCrLf & _  
vbTab & "ClassID:" & vbTab & objTaskInfo.ClassID  
vbTab & "Descr:" & vbTab & objTaskInfo.Description  
vbTab & "File:" & vbTab & objTaskInfo.Implementat  
vbTab & "Version:" & vbTab & objTaskInfo.Implementat  
vbTab & "Icon:" & vbTab & objTaskInfo.IconFile & vbTab & _  
vbTab & "Index:" & vbTab & objTaskInfo.IconIndex & vbTab & _  
vbTab & "Name:" & vbTab & objTaskInfo.Name & vbTab & _  
Next
```

```

strMsg = strMsg & vbCrLf & "Registered DTS Transforma
"===== " & vb
Set colTransInfo = objDTSAppI.TransformationInfos
For Each objTransInfo In colTransInfo
    strMsg = strMsg & vbCrLf & _
        vbTab & "ClassID:" & vbTab & objTransInfo.ClassID
        vbTab & "Descr:" & vbTab & objTransInfo.Descriptio
        vbTab & "File:" & vbTab & objTransInfo.Implementat
        vbTab & "Version:" & vbTab & objTransInfo.Impleme
        vbTab & "Name:" & vbTab & objTransInfo.Name & v
Next

Text1.Text = strMsg
End Sub

Private Sub Form_Resize()
    Text1.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
End Sub

```

Meta Data Services

Lineage information is saved for DTS packages saved to SQL Server 2000 Meta Data Services if the **LineageOptions** property of the package specifies that this be done.

Use the **GetPackageRepository** method of the **Application** object to return a **PackageRepository** object. The methods of the PackageRepository object provide access to the package and lineage information.

- Use the **EnumPackageInfos** method to return information about the DTS packages in Meta Data Services.
- Use the **EnumPackageLineages** method to return lineage data for a particular package version.

- Use the **EnumStepLineages** method to return step lineage data for a particular package lineage.
- Use **RemovePackageLineages** to purge the lineage data for a package version.

Example

The following example uses the **GetPackageRepository** method to access the Meta Data Services instance in the **msdb** database (the default instance) on the local server. Then it accesses and displays the package information and the lineage data for those packages.

Note This example displays all the lineage data in the Meta Data Services instance. On a production server, this may be an unmanageably large amount of data which will exceed the capacity of the textbox. Use a test computer that has a few packages stored in the repository that have been configured to write lineage data. Run each package only a few times since the lineage data was last purged.

To run this example, follow the procedure used to run the registered components and use the following code:

```
Private Sub Form_Load()
    Dim objDTSApp1 As DTS.Application
    Dim objPkgRepository As DTS.PackageRepository
    Dim colPkgInfo As DTS.PackageInfos
    Dim objPkgInfo As DTS.PackageInfo
    Dim strMsg As String

    Set objDTSApp1 = New DTS.Application

    Set objPkgRepository = objDTSApp1.GetPackageRepository( _
        "(local)", "msdb", "", "", DTSReposFlag_UseTrustedConnection)

    Set colPkgInfo = objPkgRepository.EnumPackageInfos("", False, "")
```

```

strMsg = "DTS Package Information" & vbCrLf & "====="
Set objPkgInfo = colPkgInfo.Next
Do Until colPkgInfo.EOF
    With objPkgInfo
        strMsg = strMsg & vbCrLf & _
            "Name:" & vbTab & .Name & vbCrLf & _
            "Descr:" & vbTab & .Description & vbCrLf & _
            "Date:" & vbTab & .CreationDate & vbCrLf & _
            "PkgID:" & vbTab & .PackageID & vbCrLf & _
            "VerID:" & vbTab & .VersionID & vbCrLf & _
            "Owner:" & vbTab & .Owner & vbCrLf & _
            "Size:" & vbTab & .PackageDataSize & _
            vbTab & "Type:" & vbTab & .PackageType & _
            vbTab & "IsOwner:" & vbTab & .IsOwner & vbCrLf

        strMsg = strMsg & strPackageLineages( _
            .VersionID, objPkgRepository)
    End With
    Set objPkgInfo = colPkgInfo.Next
Loop

Text1.Text = strMsg
End Sub

```

```

Private Function strPackageLineages(ByVal strPkgVerID As String, _
    ByVal objPkgRepository As DTS.PackageRepository) As String
    Dim colPkgLineage As DTS.PackageLineages
    Dim objPkgLineage As DTS.PackageLineage
    Dim strMsg As String

    Set colPkgLineage = objPkgRepository.EnumPackageLineages(strPkg

```

```

strMsg = vbCrLf & vbTab & "Package Lineage: " & strPkgVerID &
vbTab & "=====

Set objPkgLineage = colPkgLineage.Next
Do Until colPkgLineage.EOF
  With objPkgLineage
    strMsg = strMsg & vbCrLf & _
      vbTab & "Name:" & vbTab & .Name & vbCrLf & _
      vbTab & "Cmptr:" & vbTab & .Computer & vbCrLf & _
      vbTab & "Date:" & vbTab & .ExecutionDate & vbCrLf & _
      vbTab & "PkgID:" & vbTab & .PackageID & vbCrLf & _
      vbTab & "VerID:" & vbTab & .VersionID & vbCrLf & _
      vbTab & "Oprtr:" & vbTab & .Operator & vbCrLf & _
      vbTab & "FullID:" & vbTab & .LineageFullID & vbCrLf & _
      vbTab & "ShortID:" & vbTab & .LineageShortID & vbCrLf

    strMsg = strMsg & strStepLineages( _
      .LineageFullID, objPkgRepository)
  End With
  Set objPkgLineage = colPkgLineage.Next
Loop
strPackageLineages = strMsg
End Function

```

```

Private Function strStepLineages(ByVal strFullID As String, _
  ByVal objPkgRepository As DTS.PackageRepository) As String
  Dim colStepLineage As DTS.StepLineages
  Dim objStepLineage As DTS.StepLineage
  Dim strMsg As String

  Set colStepLineage = objPkgRepository.EnumStepLineages(strFullID

  strMsg = vbCrLf & vbTab & vbTab & "Step Lineage: " & strFullID

```

```
vbTab & vbTab & "=====
```

```
Set objStepLineage = colStepLineage.Next
Do Until colStepLineage.EOF
  With objStepLineage
    strMsg = strMsg & vbCrLf & _
      vbTab & vbTab & "Name:" & vbTab & .Name & vbCrLf & _
      vbTab & vbTab & "Start:" & vbTab & .StartTime & vbCrLf & _
      vbTab & vbTab & "Finish:" & vbTab & .FinishTime & vbCrLf & _
      vbTab & vbTab & "Elapse:" & vbTab & .ExecutionTime & vbCrLf & _
      vbTab & vbTab & "Result:" & vbTab & .StepExecutionResult & vbCrLf & _
      vbTab & vbTab & "Status:" & vbTab & .StepExecutionStatus & vbCrLf & _
    If .ErrorCode <> 0 Or Len(.ErrorDescription) > 0 Or Len(.ErrorSource) > 0 Then
      strMsg = strMsg & _
        vbTab & vbTab & "Code:" & vbTab & .ErrorCode & vbCrLf & _
        vbTab & vbTab & "Descr:" & vbTab & .ErrorDescription & vbCrLf & _
        vbTab & vbTab & "Source:" & vbTab & .ErrorSource & vbCrLf & _
    End If
  End With
  Set objStepLineage = colStepLineage.Next
Loop
strStepLineages = strMsg
End Function
```

```
Private Sub Form_Resize()
  Text1.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
End Sub
```

Logging to SQL Server

Logging to SQL Server is available to all DTS packages, regardless of where they are stored. Log records are written to the **msdb** database on the server specified by the package **LogServerName** property if the package **LogToSQLServer** property has been set.

Use the **GetPackageSQLServer** method, specifying server and login information, to return a **PackageSQLServer** object. The methods of that object provide access to the package and log data on the server.

- Use the **EnumPackageInfos** method to return information about the packages in SQL Server storage on that server.
- Use the **EnumPackageLogRecords**, **EnumStepLogRecords** and **EnumTaskLogRecords** methods to return log data of the indicated type on the server.
- Use the **RemovePackageLogRecords**, **RemoveStepLogRecords**, **RemoveTaskLogRecords** methods to purge log records of the indicated type. In addition, **RemoveAllLogRecords** removes all log data for all packages from the server.

Example

The following example uses the **GetPackageSQLServer** method to gain access to the stored packages and the log data on the local server. Then the example shows how the methods of the **PackageSQLServer** object access and display information about the packages in SQL Server storage and the package and step log data on that server.

Note This example displays all the log data on the local server. On a production server, this may be an unmanageably large amount of data which will exceed the capacity of the textbox. Use a test computer on which a few packages have been run only a few times each since the log data was last purged.

To run this example, follow the procedure used to run the registered components, and use the following code in step 5.

```
Private Sub Form_Load()  
    Dim objDTSApp As DTS.Application  
    Dim objPkgSQLServer As DTS.PackageSQLServer  
    Dim colPkgInfo As DTS.PackageInfos  
    Dim objPkgInfo As DTS.PackageInfo
```

```

Dim strMsg      As String

Set objDTSApp = New DTS.Application

Set objPkgSQLServer = objDTSApp.GetPackageSQLServer( _
    "(local)", "sa", "", DTSSQLStgFlag_Default)

Set colPkgInfo = objPkgSQLServer.EnumPackageInfos("", False, ""

strMsg = "DTS Package in SQL Server Storage" & vbCrLf & _
    "===== " & vbCrLf
Set objPkgInfo = colPkgInfo.Next
Do Until colPkgInfo.EOF
    With objPkgInfo
        strMsg = strMsg & vbCrLf & _
            vbTab & "Name:" & vbTab & .Name & vbCrLf & _
            vbTab & "Descr:" & vbTab & .Description & vbCrLf & _
            vbTab & "Date:" & vbTab & .CreationDate & vbCrLf & _
            vbTab & "PkgID:" & vbTab & .PackageID & vbCrLf & _
            vbTab & "VerID:" & vbTab & .VersionID & vbCrLf & _
            vbTab & "Owner:" & vbTab & .Owner & vbCrLf & _
            vbTab & "Size:" & vbTab & .PackageDataSize & _
            vbTab & "Type:" & vbTab & .PackageType & _
            vbTab & "IsOwner:" & vbTab & .IsOwner & vbCrLf
    End With
    Set objPkgInfo = colPkgInfo.Next
Loop

strMsg = strMsg & strPackageLogRecords( _
    "", objPkgSQLServer)

strMsg = strMsg & strStepLogRecords( _
    "", objPkgSQLServer)

```

```
Text1.Text = strMsg  
End Sub
```

```
Private Function strPackageLogRecords(ByVal strPkgVerID As String  
    ByVal objPkgSQLServer As DTS.PackageSQLServer) As String  
    Dim colPkgLogRec As DTS.PackageLogRecords  
    Dim objPkgLogRec As DTS.PackageLogRecord  
    Dim strMsg As String
```

```
    Set colPkgLogRec = objPkgSQLServer.EnumPackageLogRecords(  
        "", False, "", strPkgVerID, "")
```

```
    strMsg = vbCrLf & "DTS Package Log Records " & strPkgVerID &  
        "=====" & vbCrLf
```

```
    Set objPkgLogRec = colPkgLogRec.Next
```

```
    Do Until colPkgLogRec.EOF
```

```
        With objPkgLogRec
```

```
            strMsg = strMsg & vbCrLf & _  
                vbTab & "Name:" & vbTab & .Name & vbCrLf & _  
                vbTab & "Descr:" & vbTab & .Description & vbCrLf & _  
                vbTab & "Cmptr:" & vbTab & .Computer & vbCrLf & _  
                vbTab & "Logged:" & vbTab & .LogDate & vbCrLf & _  
                vbTab & "Start:" & vbTab & .StartTime & vbCrLf & _  
                vbTab & "Finish:" & vbTab & .FinishTime & vbCrLf & _  
                vbTab & "Elapse:" & vbTab & .ExecutionTime & vbCrLf & _  
                vbTab & "PkgID:" & vbTab & .PackageID & vbCrLf & _  
                vbTab & "VerID:" & vbTab & .VersionID & vbCrLf & _  
                vbTab & "Oprtr:" & vbTab & .Operator & vbCrLf & _  
                vbTab & "FullID:" & vbTab & .LineageFullID & vbCrLf & _  
                vbTab & "ShortID:" & vbTab & .LineageShortID & vbTab & .  
            If .ErrorCode <> 0 Or Len(.ErrorDescription) > 0 Then
```

```

        strMsg = strMsg & _
            vbTab & vbTab & "ECode:" & vbTab & .ErrorCode & vbTab & vbTab & "EDesc:" & vbTab & .ErrorDescription
    End If
End With
Set objPkgLogRec = colPkgLogRec.Next
Loop
strPackageLogRecords = strMsg
End Function

```

```

Private Function strStepLogRecords(ByVal strFullID As String, _
    ByVal objPkgSQLServer As DTS.PackageSQLServer) As String
    Dim colStepLogRec As DTS.StepLogRecords
    Dim objStepLogRec As DTS.StepLogRecord
    Dim strMsg As String

    Set colStepLogRec = objPkgSQLServer.EnumStepLogRecords(strFullID)

    strMsg = vbCrLf & "DTS Step Log Records: " & strFullID & vbCrLf
        "===== " & vbCrLf

    Set objStepLogRec = colStepLogRec.Next
    Do Until colStepLogRec.EOF
        With objStepLogRec
            strMsg = strMsg & vbCrLf & _
                vbTab & "Name:" & vbTab & .Name & vbCrLf & _
                vbTab & "Start:" & vbTab & .StartTime & vbCrLf & _
                vbTab & "Finish:" & vbTab & .FinishTime & vbCrLf & _
                vbTab & "Elapse:" & vbTab & .ExecutionTime & vbCrLf & _
                vbTab & "FullID:" & vbTab & .LineageFullID & vbCrLf & _
                vbTab & "ExecID:" & vbTab & .StepExecutionID & vbCrLf & _
                vbTab & "ProgCt:" & vbTab & .ProgressCount & vbCrLf & _
                vbTab & "Result:" & vbTab & .StepExecutionResult & _

```

```

        vbTab & "Status:" & vbTab & .StepExecutionStatus & vbCrLf
    If .ErrorCode <> 0 Or Len(.ErrorDescription) > 0 Then
        strMsg = strMsg & _
            vbTab & "Code:" & vbTab & .ErrorCode & vbTab & "x" & vbCrLf
            vbTab & "Descr:" & vbTab & .ErrorDescription & vbCrLf
    End If
End With
Set objStepLogRec = colStepLogRec.Next
Loop
strStepLogRecords = strMsg
End Function

```

```

Private Sub Form_Resize()
    Text1.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
End Sub

```

Packages Saved as Files

The DTS storage file can contain multiple packages, each with multiple versions. To determine programmatically what packages and versions a storage file contains, the **Package2** object **GetSavedPackageInfos** method returns a reference to the **SavedPackageInfos** collection. The program can examine the details of each saved package version by iterating through this collection.

Example

The following code example shows the **strShowDTSPkgComps** function formatting a text string, with the information on each package version in the specified storage file:

```

Private Function strShowDTSPkgComps( _
    ByVal strUNCPath As String) As String
Dim objPackage    As DTS.Package2
Dim objPkgInfos   As DTS.SavedPackageInfos
Dim objPkgInfo    As DTS.SavedPackageInfo

```

```
Dim strPackageName As String
Dim strMsg As String
'Display the packages and versions in a DTS storage file.
```

```
On Error GoTo ErrorHandler
```

```
'Create the package object and get the package information collection.
Set objPackage = New DTS.Package
Set objPkgInfos = objPackage.GetSavedPackageInfos(strUNCPath)
```

```
'Iterate thru the package information collection.
strPackageName = ""
For Each objPkgInfo In objPkgInfos
```

```
    'If this is a different package than the last one, format full informatio
    If strPackageName <> objPkgInfo.PackageName Then
        strMsg = strMsg & vbCrLf & objPkgInfo.PackageName & vbCrLf
        "PackageID: " & objPkgInfo.PackageID & vbCrLf & _
        "Pkg Create Date: " & objPkgInfo.PackageCreationDate & vbCrLf
        strPackageName = objPkgInfo.PackageName
    End If
```

```
'Format version information.
strMsg = strMsg & vbCrLf & vbTab & "VersionID: " & _
    objPkgInfo.VersionID & vbCrLf & vbTab & _
    "Version Save Date: " & objPkgInfo.VersionSaveDate & _
    vbCrLf & vbTab & "Encrypted: " & _
    (objPkgInfo.IsVersionEncrypted) & vbCrLf
Next objPkgInfo
```

```
strShowDTSPkgComps = strMsg
Exit Function
```

ErrorHandler:

```
MsgBox "Error retrieving package information: 0x" & Hex$(Err.Number) & vbCrLf & Err.Description, vbExclamation
```

```
strShowDTSPkgComps = ""
```

```
Exit Function
```

```
End Function
```

DTS Programming

Building a DTS Custom Task

In Data Transformation Services (DTS) packages, you can include custom tasks, which are DTS tasks implemented by your or third party vendors. Custom tasks can be included in packages created by applications, or they can be registered so that they are referenced from DTS Designer. There are additional constraints a custom task must satisfy if it is to be used in DTS Designer. For more information, see [DTS Example: Running Concurrent Operations in Visual Basic](#).

To implement a custom task, you must:

- Use a programming language that supports COM, such as Microsoft® Visual C++® or Microsoft Visual Basic®.
- Implement the **CustomTask** interface and optionally implement other DTS custom task interfaces. Microsoft SQL Server™ 2000 supplies objects that define these interfaces.
- Add registration code to your custom task if you want to avoid registering the task from DTS Designer.

This section explains how to implement custom tasks and provides examples.

Topic	Description
DTS Custom Task Fundamentals	Describes the CustomTask interface that all custom tasks must implement.
Including a DTS Custom Task User Interface	Describes the CustomTaskUI interface, which custom tasks that have a custom property page must implement.
Registering a DTS Custom Task	Describes how to register custom tasks from DTS Designer and how to support registration from the command prompt.
Additional DTS Custom Task Features	Explains how to raise package events, write to log tables and files, and use the DTS properties provider from a custom

	task.
DTS Custom Task Examples in Visual Basic	Shows how to implement a basic DTS custom task and how to add functionality to it in Visual Basic.
Implementing DTS Custom Tasks in Visual C++	Explains how to use the Active Template Library (ATL) to implement DTS custom tasks in Visual C++.
DTS Custom Task Examples in Visual C++	Provides examples of DTS custom task examples implemented in Visual C++.

DTS Programming

DTS Custom Task Fundamentals

A Data Transformation Services (DTS) custom task is implemented as an in-process COM component. To be used in DTS Designer, the custom task must be an in-process DLL. When used programmatically, the custom task can be an out-of-process executable.

All custom tasks must implement the **CustomTask** interface. If the custom task has a property sheet, the task also must implement the **CustomTaskUI** interface. For more information, see [Including a DTS Custom Task User Interface](#).

CustomTask Interface

In Microsoft® Visual Basic®, the **CustomTask** interface is defined by the **CustomTask** object from the Microsoft DTSPackage Object Library. In Microsoft Visual C++®, it is defined by **IDTSCustomTask** in the include file dtspkg.h.

The DTS CustomTask interface includes the following elements.

Element	Description
Description property	A textual description that identifies the task in DTS Designer or a programming environment.
Name property	A unique identifier used by DTS to reference the task.
Properties collection	A reference to a collection of Property objects that defines each property of the custom task.
Execute method	A subprogram that performs the function of the custom task.

As required by COM, all elements must be present, but they can be placeholders.

Description Property

DTS Designer uses the **Description** property to label the icon for the custom task. To implement **Description**, you save the value to which the property is set and return that value when the property is read. If you provide a placeholder for **Description**, the label disappears when you close the task property page or the

Custom Task Properties dialog box.

In Visual Basic, if you plan to use the default properties grid, you must provide an additional **Description** property for the class-specific task object. Tie **Description** and **CustomTask_Description** together so that setting either the class-specific **Description** property or **CustomTask_Description** causes the values of both to be updated.

This step is necessary because the **CustomTask_Description** Get and Let functions implement the **Task.Description** property. However, **Description** implements *CustTask.Description*, where *CustTask* is the name you gave to your custom task. Implementing *CustTask.Description* also causes **Description** to be included in the **Properties** collection. The properties grid uses the **Properties** collection to read and update custom task properties.

If you do not plan to use the custom task in DTS Designer and you do not plan to use the **Description** property, you can provide a placeholder for **CustomTask_Description**.

Name Property

The **Name** property identifies the **Task** objects in the package. Thus, it always must be implemented. To implement **Name**, you save the value to which the property is set and return that value when the property is read.

It is recommended that you do not expose **Name**, especially in a read/write mode. DTS Designer assigns a unique name to the task when the task icon is placed on the design sheet. If you change the value of **Name**, DTS Designer will look for the task using the old name and fail when it cannot find it.

In a DTS application, you can set or change **Name** before adding the **Task** object to the **Tasks** collection. However, you will need logic to detect when the user enters a name already used by another task. It is recommended that you have the application specify task names and guarantee they are unique.

Properties Collection

The **Properties** collection contains **Property** objects that identify the properties of the custom task. You always must implement **Properties**, but you can use a default properties provider supplied with DTS to do so. Invoke the default by

returning either **NULL** or **Nothing**, as appropriate for the programming environment, from **Properties**.

The default property grid displayed by DTS Designer uses the **Properties** collection to read and update the custom task properties. In Visual Basic, the properties of the **CustomTask** interface are not included in the default **Properties** collection. It may be necessary to add a duplicate property and tie it to the related **CustomTask** property, as was the case for **Description**.

Execute Method

The **Execute** method provides the functionality of the custom task. Use its parameters in the following ways.

Parameter	Usage
<i>pPackage</i>	Use this reference to the Package2 object to access other objects in the DTS hierarchy. Do not save any reference obtained through <i>pPackage</i> after the return from Execute .
<i>pPackageEvents</i>	Use <i>pPackageEvents</i> to raise package events. Check <i>pPackageEvents</i> for NULL/Nothing before using.
<i>pPackageLog</i>	Use <i>pPackageLog</i> , a reference to the PackageLog object, to write records to the server log table or to the log file. Check <i>pPackageLog</i> for NULL/Nothing before using.
<i>pTaskResult</i>	Set <i>pTaskResult</i> to a code from the DTSTaskExecResult constants before returning from Execute in order to indicate success, retry or failure.

In a DTS application, you do not need to call **Execute** from the application. DTS will call it at the appropriate time. When **Execute** returns, task execution is complete.

Basic Custom Task

For more information about building a basic custom task, see [DTS Example: Basic Custom Task in Visual Basic](#), [DTS Example: Adding Properties and Icons in Visual Basic](#) and [DTS Example: Adding Properties and Icons in Visual C++](#).

See Also

[CustomTask Object](#)

[CustomTaskUI Object](#)

[DTSTaskExecResult](#)

[Execute Method](#)

[Package2 Object](#)

[Properties Collection](#)

[Property Object](#)

[Task Object](#)

[Tasks Collection](#)

DTS Programming

Including a DTS Custom Task User Interface

To provide a user interface for setting properties, a custom task must implement the **CustomTaskUI** interface. This user interface, sometimes called a property page, is displayed:

- In Data Transformation Services (DTS) Designer when the DTS package is being implemented.
- In a DTS application when the object hierarchy is being created.

A user interface that is active when the custom task is executing is not controlled through the **CustomTaskUI** interface.

If a custom task does not implement **CustomTaskUI**, DTS Designer displays a default property grid in place of the custom task user interface.

For more information about building a custom task with a user interface, see [DTS Example: Including a User Interface in Visual Basic](#) and [DTS Example: Including a User Interface in Visual C++](#).

CustomTaskUI Interface

In DTS Designer, a custom task calls the methods of **CustomTaskUI** to perform functions related to the display of user interface elements. When a custom task is part of a DTS application, the application calls the **CustomTaskUI** methods to perform these functions.

In Microsoft® Visual Basic®, this user interface is defined by the **CustomTaskUI** object from the Microsoft DTSPackage Object Library. In Microsoft Visual C++® it is defined by **IDTSCustomTaskUI** in the include file dtspkg.h.

The **DTS CustomTaskUI** interface includes the following elements.

Element	Description
Initialize method	Called before other CustomTaskUI methods to allow the custom task to perform initializations.

New method	Called when a custom task is created.
Edit method	Called when the user interface is to be displayed for an existing custom task.
Delete method	Called when a custom task is to be removed from its package.
Help method	Called when Help for the custom task is to be displayed.
GetUIInfo method	Called when the parent application is to display a ToolTip, to determine if the custom task is to generate the ToolTip window.
CreateCustomToolTip method	Creates a custom ToolTip window and draws the ToolTip, when custom ToolTips are supported.

As required by COM, all elements must be present, but they can be placeholders.

CAUTION It is strongly recommended that you check the parameters of all **CustomTaskUI** methods for validity before you use them. The caller may be a DTS application and you may not know how extensively the application has been tested.

Initialize Method

DTS Designer calls **Initialize** before **New**, **Edit**, **Delete** and **Help**. It is recommended that DTS applications follow this sequence so that the task can be used both within and outside of DTS Designer. The custom task can perform any initialization. The parameter, a reference to the **Task** object, can be saved for use by the subsequent method (for example, to access custom task properties).

New Method

DTS Designer calls **New** when the custom task icon is dragged to the design sheet. It is recommended that DTS applications call **New** after the custom task has been created with the **New** method of the **Tasks** collection.

Typically, the custom task displays a property page with default values. The parameter is the window handle of the DTS design sheet or of a window in the parent application.

If you do not implement **CustomTaskUI**, you see a default property grid in DTS Designer. If you implement **CustomTaskUI** but provide a placeholder for **New**, no user interface is displayed.

Edit Method

DTS Designer calls **Edit** when you right-click the custom task icon and click **Properties**. It is recommended that if DTS applications call **Edit**, they do so after the custom task has been created and values for custom task properties have been set.

Typically, the custom task displays a property page with current values. The parameter is the window handle of the DTS design sheet or of a window in the parent application.

If you do not implement **CustomTaskUI**, you see a default property grid. If you implement **CustomTaskUI** but provide a placeholder for **Edit**, no user interface is displayed.

Delete Method

DTS Designer calls **Delete** when you delete the custom task icon from the design sheet. The custom task can perform any cleanup. The parameter is the window handle of the DTS design sheet or of a window in the parent application.

Help Method

DTS Designer calls **Help** when you right-click the custom task icon and click **Help**.

Typically, the custom task displays a help topic (for example, by calling winhlp32.exe with a help file specification and topic ID). The parameter is the window handle of the DTS design sheet or of a window in the parent application.

If you do not implement **CustomTaskUI**, you see a generic help topic for custom tasks. If you implement **CustomTaskUI** but provide a placeholder for **Help**, no topic is displayed.

GetUIInfo Method

GetUIInfo is not implemented in DTS Designer. A DTS application can use this method to query the custom task for its tooltip text and description (for example, if the application used the custom task icon in its user interface). **GetUIInfo** also returns a value indicating whether the custom task generates a custom tooltip. **GetUIInfo** has the following parameters.

Parameter	Description
<i>pbstrToolTip</i>	Returns the tooltip text.
<i>pbstrDescription</i>	Returns the tooltip description.
<i>plVersion</i>	Returns the custom task version number.
<i>pFlags</i>	Returns a value from DTSCustomTaskUIFlags that indicates whether the task generates a custom tooltip.

CreateCustomToolTip Method

CreateCustomToolTip is not implemented in DTS Designer. A DTS application can call this method so that the custom task generates its custom tooltip after the **GetUIInfo** method has indicated the task can do so. **CreateCustomToolTip** has the following parameters.

Parameter	Description
<i>hwndParent</i>	The handle of the window in the parent application where the ToolTip is to be drawn.
<i>x, y</i>	The co-ordinates where the ToolTip window is to be drawn.
<i>plTipWindow</i>	The parameter through which the ToolTip window handle is returned.

The parent application is responsible for releasing the resources associated with the tooltip window.

See Also

[CreateCustomToolTip Method](#)

[CustomTaskUI Object](#)

[Delete Method](#)

[DTSCustomTaskUIFlags](#)

[Edit Method](#)

[GetUIInfo Method](#)

[Help Method](#)

[Initialize Method](#)

[New \(CustomTaskUI\) Method](#)

[Task Object](#)

[Tasks Collection](#)

DTS Programming

Registering a DTS Custom Task

Data Transformation Services (DTS) custom tasks require entries in their class registration to identify them as DTS tasks. You can add these entries by registering the task from DTS Designer, or you can add code or script to the custom task so that it can create these entries.

In Microsoft® ActiveX® components built with Microsoft Visual Basic®, the registration code is supplied by the build process and is inaccessible to the developer. Therefore, a custom task built with Visual Basic cannot be detected by DTS until it has been registered from within DTS Designer. To create the additional registry entries, you can add code or script to custom tasks built using the Active Template Library (ATL) COM wizards in Microsoft Visual C++®.

DTS Custom Task Registration Entries

In Visual Basic, an ActiveX DLL component containing a public class *Component.CTaskClass* creates a set of registry keys under `\HKEY_CLASSES_ROOT\CLSID\` with this structure:

- {*Class ID for Component.CTaskClass*}
(Default) *Component.CTaskClass*
 - Implemented Categories
 - {*Automation Objects component category GUID*}
 - InprocServer32
(Default) *path\Component.dll*
Threading Model Apartment
 - ProgID
(Default) *Component.CTaskClass*
 - Programmable
 - TypeLib

(Default) {*Component type library GUID*}

- Version

(Default) *version number*

A component with class *Component.CTaskClass* generated by ATL COM AppWizard in Visual C++ creates a similar set of registry keys under \HKEY_CLASSES_ROOT\CLSID\:

- {*Class ID for Component.CTaskClass*}

(Default) *CTaskClass* Class

- InprocServer32

(Default) *path\Component.dll*

Threading Model Both

- ProgID

(Default) *Component.CTaskClass.ver*

- Programmable

- TypeLib

(Default) {*Component type library GUID*}

- VersionIndependentProgID

(Default) *Component.CTaskClass*

DTS defines a component category for DTS tasks, which is a GUID that is added to the system registry when the Microsoft SQL Server™ client tools are installed on your system. To make a custom task registration visible to DTS Designer, you must add the Implemented Categories key, if it is not already there. Then, add a subkey to Implemented Categories that contains this component category. Optionally, you can add values to the {*Class ID for Component.CTaskClass*} key that specify the task icon and default description. The added key and values look like:

- {Class ID for Component.CTaskClass}
 (Default) CTaskClass Class
 DTSTaskIconFile path\iconfile.ext
 DTSTaskIconIndex index
 DTSTaskDescription description
- Implemented Categories
 - {GUID for DTS Tasks component category}

The added keys, values and data have the following descriptions.

Element	Description
<i>path\iconfile.ext</i>	File specification of the component that contains icons for the custom task. Typically, this is the component that contains the task, but it can be any file from which icons can be extracted.
<i>index</i>	Position of the icon in the list of icons for the custom task. The first icon has index = 0.
<i>description</i>	Installed description of the custom task. DTS Designer appends ": undefined" to <i>description</i> to generate the default description when the task icon is dragged to the design sheet.
<i>GUID for DTS Tasks component category</i>	GUID that identifies the component as a DTS custom task. The value is defined by the symbol CATID_DTSCustomTask in include file dtspkg.h.

Registry Script File

If you are using the ATL COM wizards to implement a custom task, the simplest way to add these extra keys and values is through the registry script (.rgs) file that ATL Object Wizard generates.

Example

The following registry script was created with the wizard for a custom task class **CustTaskOne** in a component named **DTSSimple**. The script in normal font

was generated by the wizard. The additional script, in bold, supports the DTS custom task features.

This custom task uses the second icon (offset 1) in the component DLL and the default description is "Simple Custom Task".

IMPORTANT Do not use the GUID shown in the example for the DTS tasks component category until you have verified from the dtspkg.h include file on your system that it is correct.

HKCR

```
{
  DTSSimple.CustTaskOne.1 = s 'CustTaskOne Class'
  {
    CLSID = s '{196617B8-5CE1-4529-B36F-3D8AF026E085}'
  }
  DTSSimple.CustTaskOne = s 'CustTaskOne Class'
  {
    CLSID = s '{196617B8-5CE1-4529-B36F-3D8AF026E085}'
    CurVer = s 'DTSSimple.CustTaskOne.1'
  }
  NoRemove CLSID
  {
    ForceRemove {196617B8-5CE1-4529-B36F-3D8AF026E085} = s
    {
      ProgID = s 'DTSSimple.CustTaskOne.1'
      VersionIndependentProgID = s 'DTSSimple.CustTaskOne'
      ForceRemove 'Programmable'
      InprocServer32 = s '%MODULE%'
      {
        val ThreadingModel = s 'Both'
      }
      'TypeLib' = s '{7852210C-8748-487F-80A7-0FAAB76F0154}'
      'Implemented Categories'
      {
```

```
    '{10020200-EB1C-11CF-AE6E-00AA004A34D5}'  
  }  
  val DTSTaskIconFile = s '%MODULE%'  
  val DTSTaskIconIndex = d 1  
  val DTSTaskDescription = s 'Simple Custom Task'  
}  
}  
}
```

After you add the script lines for the DTS task, rebuild the project and refresh the DTS cache, if necessary. The custom task appears in DTS Designer without being registered from within DTS Designer.

If you unregister the custom task from within DTS Designer, the unregistration function in DTS Designer will remove the DTS task component category subkey from the registration. To restore the subkey, you need to rebuild the project or reregister the task from the command prompt:

```
regsvr32 component.dll
```

DTS Programming

Additional DTS Custom Task Features

The following topics describe additional features supported by Data Transformation Services (DTS) custom tasks.

Topic	Description
Raising Events from a DTS Custom Task	Describes how to raise DTS package events from a custom task.
Writing Log Data from a DTS Custom Task	Explains how to write records to the Microsoft® SQL Server™ task log table and to the exception file from a custom task.
Using the DTS Custom Task Properties Provider	Describes how to invoke the DTS properties provider.

Raising Events from a DTS Custom Task

A custom task raises package events that are handled by the parent application. Typically, it raises the following events.

Event	Purpose	Frequency
OnError	To notify the parent application that an error has occurred, especially non-fatal errors.	When an error occurs.
OnProgress	To notify the parent application of progress in task processing.	Every time a few units (for example, rows) process, or every few seconds.
OnQueryCancel	To give the parent application the opportunity to terminate the custom task. In Data Transformation Services (DTS) Designer, click Cancel to handle this event.	Every few seconds.

Note In DTS, you do not need to raise **OnStart** or **OnFinish** because each DTS step raises **OnStart** when it starts and **OnFinish** when it finishes.

OnQueryCancel and **OnError** have a *pbCancel* parameter. If the handling application sets *pbCancel*, the custom task should terminate execution by returning from the **CustomTask_Execute** method.

One of the parameters of **CustomTask_Execute** is a reference through which package events can be raised. Check for **NULL** or **Nothing** (depending on programming language) before using it.

Example

The following Microsoft® Visual Basic® code raises **OnProgress** and **OnQueryCancel** and then terminates the task if requested:

```
Private Sub CustomTask_Execute(ByVal pPackage As Object, _
```

```

        ByVal pPackageEvents As Object, ByVal pPackageLog As Obj
        pTaskResult As DTS.DTSTaskExecResult)
Dim lngRowCount    As Long
Dim blnCancel     As Boolean

...

'Make sure package events object is valid.
If Not pPackageEvents Is Nothing Then

    'Raise OnProgress and OnQueryCancel, and then exit if response say
    pPackageEvents.OnProgress Me.Description, "Row Count", _
        0, lngRowCount, 0
    pPackageEvents.OnQueryCancel Me.Description, blnCancel
    If blnCancel Then
        pTaskResult = DTSTaskExecResult_Failure
        Exit Sub
    End If
End If

...

pTaskResult = DTSTaskExecResult_Success
End Sub

```

See Also

[Execute Method](#)

[OnError Event](#)

[OnProgress Event](#)

[OnQueryCancel Event](#)

Writing Log Data from a DTS Custom Task

Custom tasks can write log records to the Microsoft® SQL Server™ task log table and to the Data Transformation Services (DTS) package log file.

You write log records through a reference to a **PackageLog** object, which is one of the parameters of **CustomTask_Execute**. Check the reference for **NULL** or **Nothing** before using it.

Example

The following Microsoft Visual Basic® code writes a log file record specifying the number of rows processed upon successful completion. If an error occurs within **CustomTask_Execute**, a task log record is written. Then the error is propagated back to the caller:

```
Private Sub CustomTask_Execute(ByVal pPackage As Object, _  
    ByVal pPackageEvents As Object, ByVal pPackageLog As Obj  
    pTaskResult As DTS.DTSTaskExecResult)  
Dim lngRowCount As Long
```

```
On Error GoTo ErrorHandler
```

```
...
```

```
'Write rows processed message to log, if log object valid.
```

```
If Not pPackageLog Is Nothing Then
```

```
    pPackageLog.WriteStringToLog _
```

```
        Me.Description & ": Rows processed = " & (lngRowCount)
```

```
End If
```

```
pTaskResult = DTSTaskExecResult_Success
```

```
Exit Sub
```

```
ErrorHandler:
```

```

Dim lngErrorCode As Long
'Write error information to task log, if log object valid.
If Not pPackageLog Is Nothing Then
    pPackageLog.WriteTaskRecord Err.Number, _
        Me.Description & ": " & Err.Description
End If
pTaskResult = DTSTaskExecResult_Failure

'Extend error code to 32 bits if necessary, then propagate error.
lngErrorCode = Err.Number
If lngErrorCode >= 0 And lngErrorCode < 65536 Then
    lngErrorCode = lngErrorCode + vbObjectError
End If
Err.Raise lngErrorCode, Me.Description & "/" & Err.Source, Err.Descri
End Sub

```

Log file strings are written only if the package **LogFileName** property has been set to the log file specification. Log file strings and task log records also can be written through the **DTSPackageLog** scripting object from scripts within the **ActiveScriptTask** object.

See Also

[ActiveScriptTask Object](#)

[LogFileName Property](#)

[PackageLog Object](#)

Using the DTS Custom Task Properties Provider

A custom task can invoke the Data Transformation Services (DTS) properties provider explicitly and access the collection it returns.

When a custom task implements a placeholder for the **CustomTask_Properties** property or returns **NULL** or **Nothing**, the default DTS properties provider generates a **Properties** collection and returns a reference to it. However, the custom task is not able to access or modify the collection.

Example

The following Microsoft® Visual Basic® code shows how to invoke the DTS properties provider explicitly (for example, in order to change the default value of a custom task property). Create the **PropertiesProvider** object and invoke the **GetPropertiesForObject** method:

```
Private Property Get CustomTask_Properties() As DTS.Properties
'Use DTS properties provider to generate the collection.
    Dim objPropsProvider As New DTS.PropertiesProvider
    Dim colProperties As DTS.Properties

    Set colProperties = objPropsProvider.GetPropertiesForObject(Me)
    Set objPropsProvider = Nothing

    'Access the properties collection through colProperties here.

    Set CustomTask_Properties = colProperties
End Property
```

See Also

[GetPropertiesForObject Method](#)

[Properties Collection](#)

PropertiesProvider Object

DTS Programming

DTS Custom Task Examples in Visual Basic

This section provides examples of Data Transformation Services (DTS) custom tasks.

Topic	Description
DTS Example: Basic Custom Task in Visual Basic	Provides an example of a basic custom task and explains how to build and register it.
DTS Example: Adding Properties and Icons in Visual Basic	Provides an example of adding properties and icons to a custom task.
DTS Example: Including a User Interface in Visual Basic	Provides an example of a custom task with a property page user interface.
DTS Example: Running Concurrent Operations in Visual Basic	Provides an example of a custom task, implemented in Microsoft® Visual Basic®, that runs concurrently with other steps and uses other custom task features.

DTS Example: Basic Custom Task in Visual Basic

The following code example implements a basic Data Transformation Services (DTS) custom task in Microsoft® Visual Basic®.

When executed, the application displays a fixed message in a message box. The **Execute** method displays the message box. The **Name** property returns the value to which it was set. The **Description** property and **Properties** collection are placeholders.

Implementing a Basic DTS Custom Task

Use the following Visual Basic code to implement a basic DTS custom task:

Implements DTS.CustomTask

```
Private mstrTaskName As String
```

```
Private Sub CustomTask_Execute(ByVal pPackage As Object, ByVal pPackageLog As Object, pTaskResult As DTS.DTSTaskResult)
    MsgBox "Minimum custom task!", vbExclamation
    pTaskResult = DTSTaskExecResult_Success
End Sub
```

```
Private Property Get CustomTask_Properties() As DTS.Properties
'CustomTask_Properties returns Nothing.
End Property
```

```
Private Property Get CustomTask_Description() As String
'Description returns empty string.
End Property
```

```
Private Property Let CustomTask_Description(ByVal RHS As String)
```

```
'Description set value is discarded.  
End Property
```

```
Private Property Get CustomTask_Name() As String  
'Implements Task.Name.  
    CustomTask_Name = mstrTaskName  
End Property
```

```
Private Property Let CustomTask_Name(ByVal strNewName As String)  
'Implements Task.Name.  
    mstrTaskName = strNewName  
End Property
```

To build this DTS custom task in Visual Basic

1. In the Visual Basic development environment, create a new Microsoft® ActiveX® DLL project.
2. On the **Project** menu, click **References**, and under **Available References**, select the check box for **Microsoft DTSPackage Object Library**. Then, on the **Project** menu, click **Properties**, and in the **Project name** box, change the project name from Project1 to something meaningful, such as **DTSBasic**.
3. Change the name of the class module from Class1 to something meaningful, such as **CustTask**.
4. Copy the Visual Basic code from the example and paste it into the class module you have just renamed.
5. Build the component by selecting **File/Make DTSBasic.dll**.

This procedure builds DTSBasic.dll and registers it in the operating system registry. The component is registered as a generic DLL. The registration does not

specify the component category for DTS tasks. The custom task can be used in DTS applications but must be registered in DTS Designer before being used there. For more information about the DTS task component category, see [Registering a DTS Custom Task](#).

To register the task in DTS Designer

1. Open SQL Server Enterprise Manager, right-click **Data Transformation Services**, and then click **New Package**.
2. On the **Task** menu, click **Register Custom Task**.
3. In the **Task description** box, enter an appropriate task description, and then in the **Task Location** box, enter the path to DTSMMinimum.dll. Click the browse (...) button to search for DTSMMinimum.dll.

On the **Task** menu, the custom task and icon appear. The default DTS task icon is displayed because there are no icons in DTSSBasic.dll.

To run the registered task

1. From the **Task** toolbar, drag the custom task icon to the design sheet.
The default property grid appears, but no properties are displayed.
2. Click **OK**.
The task icon description disappears. You must click **Execute** to display the task icon description.

See Also

[CustomTask Object](#)

[Execute Method](#)

[Properties Collection](#)

DTS Example: Adding Properties and Icons in Visual Basic

You can modify a Data Transformation Services (DTS) custom task so that users can:

- Enter and change the task description and update the icon label with that description.
- Enter and change the text of the displayed message.
- Add one or more icons to the task component.

Writing Task Description and Message Properties

To enter and save the task description, implement the **Description** property of the **CustomTask** interface so that it saves the value to which it is set and returns that value when the property is read. Also, you must add a **Description** property outside of the **CustomTask** interface and tie the properties together so that setting the value of either causes both to be changed. For more information, see [DTS Custom Task Fundamentals](#).

To enter and save the message text, add a property (called **Message** in the sample code below) and use that property value in the **MsgBox** function. Save the value to which **Message** is set and return that value when the property is read.

Example

This is the Microsoft® Visual Basic® code for adding these properties to the basic custom task:

Implements DTS.CustomTask

```
Private mstrTaskName As String
Private mstrDescription As String
Private mstrMessage As String
```

```
Private Sub CustomTask_Execute(ByVal pPackage As Object, ByVal p
    ByVal pPackageLog As Object, pTaskResult As DTS.DTSTask
    MsgBox mstrMessage, vbExclamation, mstrDescription
End Sub
```

```
Private Property Get CustomTask_Properties() As DTS.Properties
    'Set CustomTask_Properties = Nothing
End Property
```

```
Private Property Get CustomTask_Description() As String
'Implements Task.Description.
    CustomTask_Description = mstrDescription
End Property
```

```
Private Property Let CustomTask_Description(ByVal strNewDescr As
'Implements Task.Description.
    mstrDescription = strNewDescr
End Property
```

```
Private Property Get CustomTask_Name() As String
'Implements Task.Name.
    CustomTask_Name = mstrTaskName
End Property
```

```
Private Property Let CustomTask_Name(ByVal strNewName As String
'Implements Task.Name.
    mstrTaskName = strNewName
End Property
```

```
Public Property Get Message() As String
'Implements CustTask.Message.
    Message = mstrMessage
End Property
```

```
Public Property Let Message(ByVal strNewMsg As String)
'Implements CustTask.Message.
    mstrMessage = strNewMsg
End Property
```

```
Public Property Get Description() As String
'Implements CustTask.Description.
    Description = mstrDescription
End Property
```

```
Public Property Let Description(ByVal strNewDescr As String)
'Implements CustTask.Description.
    mstrDescription = strNewDescr
End Property
```

Adding Icons to a Custom Task Component

In a Visual Basic project, you typically assign an icon to the **Icon** property of each form in the project. Then you select one of the forms to supply the icon for the component. However, DTS Designer is not able to access an icon specified in this way.

For DTS custom tasks, you must add a resource file to the Visual Basic project and add one or more icons to the resource file. When you register a custom task, all the icons in the resource file will appear under **Select Icon** in the **Register Custom Task** dialog box in DTS Designer.

To add icons to the DTS custom task

1. On the **Task** menu, click **Unregister Custom Task**, and then select the task you registered.

2. In the Visual Basic development environment, replace the Visual Basic code in the **CustTask** class with the upgraded code from the example.
3. Add a resource file to the project and add one or more icons to the resource file.
4. On the **File** menu, click **Make DTSMinimum.dll** to build the component.

You may need to close SQL Server Enterprise Manager first to avoid a "permission denied" error.

5. Register the custom task in DTS Designer. The icons you added to the project resource file should appear in the **Register Custom Task** dialog box.

For more information, see [Registering a DTS Custom Task](#).

6. Drag a copy of the custom task onto the design sheet. Set values for **Description** and **Message**, and then close the properties grid. The icon label should change to the value of **Description**.
7. Execute the package.

In the message box, the value of **Message** is displayed.

If you select **Binary Compatibility** from the **Component** tab of the **Project/Properties** dialog box in Visual Basic, you do not have to complete Steps 1 and 5 of this procedure. However, selecting **Binary Compatibility** severely restricts the changes you can make to the public interface of the custom task.

Registration Problems in DTS Designer

If you rebuild a custom task component before unregistering it in DTS Designer, subsequent attempts to unregister the component will fail. DTS Designer will be

unable to find the component file.

To recover from registration problems in DTS Designer

1. From a DOS window, set the default device and directory to the folder containing the custom task component DLL. Unregister the component with this command:
`regsvr32 /u component.dll`
2. Close DTS Designer, right-click **Data Transformation Services**, and then click **Properties**. Do one of the following:
 - If the **Turn on Cache** check box is selected, click **Refresh Cache**.
 - If the **Turn on Cache** check box is not selected, the **Refresh Cache** button will be unavailable and you can skip this step.
3. Rebuild the custom task component DLL.
4. Reopen SQL Server Enterprise Manager and DTS Designer. The custom task should not appear on the **Task** menu or **Task** toolbar.
5. On the **Task** menu, click **Register Custom Task** and provide the information necessary to register the custom task in DTS Designer.

CAUTION Do not attempt to unregister components by deleting the registered file and removing the registry entries with a registry cleaning utility. Many utilities only partially remove the registry entries. You will then not be able to use regsvr32.exe because it calls the **DLLUnregisterServer** entry point in the registered component, which you have deleted.

See Also

[CustomTask Object](#)

[Execute Method](#)

Properties Collection

DTS Example: Including a User Interface in Visual Basic

The following Microsoft® Visual Basic® code example implements a property page for a Data Transformation Services (DTS) custom task. The task displays the value of a global variable and supports a timeout on the display. The task closes the display, if the user has not already done so, when the timeout occurs.

This Visual Basic project consists of a custom task class, a property page form, and a runtime display form.

Custom Task Class

The custom task class, called **FinalGlobal**, has these features:

- A **GVMonitor** property, which specifies the name of the global variable to be displayed.
- A **DisplayTime** property, which specifies the time after which the display is closed.
- **Description** and **Name** properties that tie CustomTask interface properties to the **FinalGlobal** class.

It is acceptable to use **Name** because the property page exposes **Name** as read-only. Thus, the user cannot cause an error by attempting to change it.

- A property page that is displayed when the CustomTaskUI **New** or **Edit** methods are invoked. These methods are called by DTS Designer when you either drag the task icon to the design sheet or right-click the icon and select **Properties**.
- A Help page that is displayed when the CustomTaskUI **Help** method is invoked.

Implementing the FinalGlobal Class

Use the following Visual Basic code to implement the **FinalGlobal** class:

Implements DTS.CustomTask

Implements DTS.CustomTaskUI

```
Const INVALID_PROP = "Invalid property value."
```

```
Private strDescription As String 'Task/FinalGlobal.Description pro
```

```
Private strTaskName As String 'Task/FinalGlobal.Name property
```

```
Private strGVMonitorName As String 'FinalGlobal.GVMonitor pro
```

```
Private sngDisplayTime As Single 'FinalGlobal.DisplayTime
```

```
Private frmShowGV As frmFinalGlobal
```

```
Private frmGVProperties As frmFinalGVProperties
```

```
Private objTask As DTS.Task
```

```
Private Sub CustomTask_Execute(ByVal pPackage As Object, _  
    ByVal pPackageEvents As Object, ByVal pPackageLog As Obj  
    pTaskResult As DTS.DTSTaskExecResult)
```

```
'Get reference to global variable, display its value.
```

```
Dim objPackage As DTS.Package2
```

```
Dim objMonitor As DTS.GlobalVariable
```

```
Dim blnCancel As Boolean
```

```
'Save reference to package, release parameter reference.
```

```
Set objPackage = pPackage
```

```
Set pPackage = Nothing
```

```
pTaskResult = DTSTaskExecResult_Success
```

```
'Get reference to global variable.
```

```
Set objMonitor = objPackage.GlobalVariables(strGVMonitorName)
```

```
'Create display form, pass GV name and value, and timeout.
```

```
Set frmShowGV = New frmFinalGlobal
frmShowGV.MonitorName = strGVMonitorName
frmShowGV.MonitorValue = objMonitor.Value
frmShowGV.DisplayTime = 1000 * sngDisplayTime
frmShowGV.Show vbModal
```

```
'Release display form after it closes.
Unload frmShowGV
Set frmShowGV = Nothing
```

```
End Sub
```

```
Private Property Get CustomTask_Properties() As DTS.Properties
'Use default Properties collection.
    Set CustomTask_Properties = Nothing
End Property
```

```
Private Property Let CustomTask_Description(ByVal strNewDescr As
'Implements Task.Description.
    strDescription = strNewDescr
End Property
```

```
Private Property Get CustomTask_Description() As String
'Implements Task.Description.
    CustomTask_Description = strDescription
End Property
```

```
Private Property Let CustomTask_Name(ByVal strNewName As String)
'Implements Task.Name.
    strTaskName = strNewName
End Property
```

```
Private Property Get CustomTask_Name() As String
```

'Implements Task.Name.

 CustomTask_Name = strTaskName

End Property

'-----

Private Sub DisplayPropertyPage()

'Validate task reference and display property page.

 If TypeOf objTask Is DTS.Task Then

 Set frmGVProperties = New frmFinalGVProperties

 Set frmGVProperties.TaskObject = objTask

 frmGVProperties.Show vbModal

 DoEvents

 Set frmGVProperties = Nothing

 Else

 MsgBox "Invalid task reference. Unable to display property page."
 vbExclamation, "FinalGlobal Task"

 End If

End Sub

Private Sub CustomTaskUI_CreateCustomToolTip(ByVal hwndParent

 ByVal x As Long, ByVal y As Long, plTipWindow As Long)

'CreateCustomToolTip not implemented.

End Sub

Private Sub CustomTaskUI_Delete(ByVal hwndParent As Long)

'Delete not implemented.

End Sub

Private Sub CustomTaskUI_Edit(ByVal hwndParent As Long)

'Display property page with current values.

 DisplayPropertyPage

End Sub

Private Sub CustomTaskUI_GetUIInfo(pbstrToolTip As String, _
 pbstrDescription As String, plVersion As Long, _
 pFlags As DTS.DTSCustomTaskUIFlags)

'GetUIInfo not implemented.

End Sub

Private Sub CustomTaskUI_Help(ByVal hwndParent As Long)

'Display Help screen.

 Dim strHelpText As String

 strHelpText = "Specify properties for FinalGlobal custom task. " &
 "Task should run as last step of package." & _
 vbCrLf & vbCrLf & _
 "Enter/change task description. " & _
 "It appears as task icon label on design surface." & _
 vbCrLf & vbCrLf & _
 "Enter name of global variable to be displayed." & _
 vbCrLf & vbCrLf & _
 "Enter display time in seconds. Display is removed after " & _
 "this time elapses, if not already closed by user. " & _
 "Enter 0 if display is not to be automatically removed."

 MsgBox strHelpText, vbInformation, "FinalGlobal Help"

End Sub

Private Sub CustomTaskUI_Initialize(ByVal pTask As DTS.Task)

'Initialize Description property if not already set, save task reference.

 If TypeOf pTask Is DTS.Task Then Set objTask = pTask

```
If Description = "" Then
    Description = "Final Global Variable Display"
End If
```

```
End Sub
```

```
Private Sub CustomTaskUI_New(ByVal hwndParent As Long)
'Display property page with default values.
    DisplayPropertyPage
End Sub
```

```
'-----
Public Property Get Name() As String
'Implements FinalGlobal.Name.
    Name = strTaskName
End Property
```

```
Public Property Let Name(ByVal strNewName As String)
'Implements FinalGlobal.Name.
    strTaskName = strNewName
End Property
```

```
Public Property Get Description() As String
'Implements FinalGlobal.Description.
    Description = strDescription
End Property
```

```
Public Property Let Description(ByVal strNewDescr As String)
'Implements FinalGlobal.Description and verifies that it is non-empty.

    If Len(strNewDescr) > 0 Then
        strDescription = strNewDescr
    Else
```

```
    Err.Raise 1001 + vbObjectError, Me.Name, INVALID_PROP  
End If
```

```
End Property
```

```
Public Property Get GVMonitor() As String
```

```
'Name of global variable to monitor.
```

```
    GVMonitor = strGVMonitorName
```

```
End Property
```

```
Public Property Let GVMonitor(ByVal strNewName As String)
```

```
'Name of global variable to monitor, verify non-empty.
```

```
    If Len(strNewName) > 0 Then
```

```
        strGVMonitorName = strNewName
```

```
    Else
```

```
        Err.Raise 1001 + vbObjectError, Me.Name, INVALID_PROP
```

```
    End If
```

```
End Property
```

```
Public Property Get DisplayTime() As Single
```

```
'Timeout for display form.
```

```
    DisplayTime = sngDisplayTime
```

```
End Property
```

```
Public Property Let DisplayTime(ByVal sngNewTime As Single)
```

```
'Timeout for display form.
```

```
'Validate non-negative, type check will validate numeric.
```

```
    If sngNewTime >= 0# Then
```

```
        sngDisplayTime = sngNewTime
```

```
    Else
```

```
Err.Raise 1001 + vbObjectError, Me.Name, INVALID_PROP
End If
```

```
End Property
```

Property Page Form

The property page form, named **frmFinalGVProperties**, supports the display and entry of the **Description**, **GVMonitor** and **DisplayTime** properties and the read-only display of the **Name** property. It handles errors raised by the Property Let functions of the **FinalGlobal** class. The form hosts the following controls.

Name	Type	Use to
TxtDescription	TextBox	Display and enter the Description property.
TxtTimeout	TextBox	Display and enter the DisplayTime property.
TxtGVName	TextBox	Display and enter the GVMonitor property.
CancelButton	CommandButton	Close the form without saving the properties.
OKButton	CommandButton	Validate and save properties, then close the form.
LblTaskName	Label	Display the task name.

Adding the Property Page Form

This is the Visual Basic code for **frmFinalGVProperties**:

```
Private objTask As DTS.Task
Private objFinalTask As FinalGlobal
```

```
Const MSG_TITLE = "FinalGlobal Properties"
```

```
Public Property Set TaskObject(ByVal objNewTask As DTS.Task)
```

'When ref'ce to task updated, fetch custom task properties.

Set objTask = objNewTask

Set objFinalTask = objTask.CustomTask

With objFinalTask

lblTaskName.Caption = "Task name: " & vbCrLf & .Name

txtDescription.Text = .Description

txtGVName.Text = .GVMonitor

txtTimeOut = (.DisplayTime)

End With

End Property

Private Sub CancelButton_Click()

'On Cancel button, exit without updating properties.

Unload Me

End Sub

Private Sub OKButton_Click()

'On OK button, validate and update properties.

With objFinalTask

On Error Resume Next

.Description = txtDescription.Text

If Err.Number <> 0 Then

MsgBox "Description must be non-empty.", _
vbExclamation, MSG_TITLE

Exit Sub

End If

On Error Resume Next

.GVMonitor = txtGVName.Text

If Err.Number <> 0 Then

```

    MsgBox "A global variable name must be entered.", _
        vbExclamation, MSG_TITLE
    Exit Sub
End If

On Error Resume Next
.DisplayTime = txtTimeOut.Text
If Err.Number <> 0 Then
    MsgBox "Invalid timeout value """" & txtTimeOut.Text & """"",
        vbExclamation, MSG_TITLE
    Exit Sub
End If
On Error GoTo 0

End With

Unload Me

End Sub

```

Runtime Display Form

The runtime display form, named **frmFinalGlobal**, supports the display of the global variable value and the implementation of the display timeout. The form caption displays the global variable name. **frmFinalGlobal** hosts the following controls.

Name	Type	Description
TxtMonitorValue	TextBox	Displays the global variable value.
TimDisplay	Timer	Implements the display timeout.

Adding the Runtime Display Form

This is the Visual Basic code for **frmFinalGlobal**:

```
Private blnUnloaded      As Boolean
```

```
Private Sub Form_Unload(Cancel As Integer)
'Turn off time and mark form unloaded for task.
    blnUnloaded = True
    timDisplay.Enabled = False
End Sub
```

```
Private Sub timDisplay_Timer()
'Timer has expired, unload the form.
    Unload Me
End Sub
```

```
Public Property Let MonitorValue(ByVal vntNewValue As Variant)
'Update global variable display
    txtMonitorValue.Text = CStr(vntNewValue)
    DoEvents
End Property
```

```
Public Property Let MonitorName(ByVal strNewName As String)
'Display name of global variable in form caption.
    Me.Caption = strNewName
    DoEvents
End Property
```

```
Public Property Get Unloaded() As Boolean
'Provide unloaded indication for task.
    Unloaded = blnUnloaded
End Property
```

```
Public Property Let DisplayTime(ByVal lngNewTime As Long)
'Set timeout for display form, start timer.
```

```
timDisplay.Interval = lngNewTime  
timDisplay.Enabled = True  
End Property
```

To build this DTS custom task

1. In the Visual Basic development environment, create a new Microsoft ActiveX® DLL project.
2. On the **Project** menu, click **References**, and under **Available References**, select the check box for **Microsoft DTSPackage Object Library**. Then, on the **Project** menu, click **Properties** and in the **Project name** box, change the project name from Project1 to something meaningful, like **DTSSampleUI**.
3. Copy the code for the **FinalGlobal** class in the preceding code example to the class module in the Visual Basic project. Change the name of the class module from Class1 to **FinalGlobal**.

If you use a different name, you need to change the references to **FinalGlobal** in the code to that name.
4. Add a form to the project for the property page. Change its name to **frmFinalGVProperties**. If you use a different name, you will need to change the references to **frmFinalGVProperties** in the code to that name. Add three text boxes, two command buttons, and a label to the form. Name them as specified in the preceding table under Property Page Form. Label the buttons **OK** and **Cancel**. You may want to add additional labels to identify the text boxes and assign a meaningful caption. Copy the code for **frmFinalGVProperties** in the preceding code example to the code window for the form in the Visual Basic project.
5. Add a form to the project for the runtime display. Change its name to **frmFinalGlobal**. If you use a different name, you will need to change the references to **frmFinalGlobal** in the code to that name. Add a text

box and a timer control to the form. Name them as specified in the table above under Runtime Display Form. Copy the code for **frmFinalGlobal** in the preceding code example to the code window for the form in the Visual Basic project.

6. If you want the task to have an icon other than the default icon, add a resource file to the project and add one or more icons to the resource file.
7. On the **File** menu, click **Make DTSSampleUI.dll** to build the component. To register the task, open DTS Designer, and on the **Task** menu, click **Register Custom Task** and provide the information necessary to register the custom task.

See Also

[CustomTaskUI Object](#)

[Edit Method](#)

[Help Method](#)

[New \(CustomTaskUI\) Method](#)

DTS Example: Running Concurrent Operations in Visual Basic

Custom tasks implemented in Microsoft® Visual Basic® must run on the package main thread because Visual Basic does not support free threading. Therefore, tasks implemented in Visual Basic and run in a Data Transformation Services (DTS) package in DTS Designer run sequentially, even when the package has no precedence constraints.

However, in a DTS package application, one task running on the main thread can run concurrently with others on worker threads. Thus, such a task could be implemented in Visual Basic.

DTS Designer also does not allow a task to display a modeless form or dialog box. However, in a DTS application modeless forms can be displayed. A modeless form is used in this example.

Concurrent Display

The following example code implements a DTS custom task that continuously displays the value of a global variable while other tasks are running. The custom task closes the display when the value of another global variable changes to TRUE.

This Visual Basic project consists of a custom task class and a runtime display form.

Custom Task Class

In the custom task class, called **ShowGlobal**:

- Properties **GVMonitor** and **GVFinish** specify the names of the global variable to be displayed and to the global variable that indicates completion, respectively. There is no property page user interface, as the application sets the properties directly.

- The global variable display is updated continuously. The task raises the **OnProgress** and **OnQueryCancel** events every 3 seconds.
- A log file string and a task record are written when task execution completes.
- The DTS properties provider is explicitly invoked. The **PersistPropertyBag** interface is implemented.

Implementing the ShowGlobal Class

This is the Visual Basic code for the **ShowGlobal** class:

Implements DTS.CustomTask

Implements DTS.PersistPropertyBag

```
Const INVALID_PROP = "Invalid property value."
```

```
Private mstrDescription As String 'Task/ShowAGlobal.Description
```

```
Private mstrTaskName As String 'Task/ShowAGlobal.Name prop
```

```
Private mstrGVMonitorName As String 'ShowAGlobal.GVMonitor
```

```
Private mstrGVFinishName As String 'ShowAGlobal.GVFinish prc
```

```
Private frmShowGV As frmFinalGlobal
```

```
Private objTask As DTS.Task
```

```
Private Sub CustomTask_Execute(ByVal pPackage As Object, _
    ByVal pPackageEvents As Object, ByVal pPackageLog As Obj
    pTaskResult As DTS.DTSTaskExecResult)
```

```
'Display value of global variable until another global indicates display
```

```
Dim objPackage As DTS.Package2
```

```
Dim objMonitor As DTS.GlobalVariable
```

```
Dim objFinished As DTS.GlobalVariable
```

```
Dim blnCancel As Boolean
```

```
Dim datCurrTime As Date
```

Dim datStartTime As Date

'Save reference to package, release parameter reference.

Set objPackage = pPackage

Set pPackage = Nothing

pTaskResult = DTSTaskExecResult_Success

'Initialize times for event generation.

datStartTime = Now

datCurrTime = Now

'Get reference to global variables, exit if already finished.

Set objMonitor = objPackage.GlobalVariables(mstrGVMonitorName)

Set objFinished = objPackage.GlobalVariables(mstrGVFinishName)

If objFinished.Value = True Then Exit Sub

'Display form and use global variable name as title.

Set frmShowGV = New frmFinalGlobal

frmShowGV.MonitorName = mstrGVMonitorName

frmShowGV.Show vbModeless

'Refresh display until finished GV indicates done, user closes form, or

Do Until objFinished.Value Or frmShowGV.Unloaded

 frmShowGV.MonitorValue = objMonitor.Value

 'Every 3 sec, raise OnQueryCancel and OnProgress.

 If DateDiff("s", datCurrTime, Now) >= 3 Then

 datCurrTime = Now

 'Make sure package events object is valid.

 If Not pPackageEvents Is Nothing Then

 'Raise On Progress, OnQueryCancel, exit if response says to

```

        pPackageEvents.OnProgress Me.Description, "3 second noti:
            0, DateDiff("s", datStartTime, Now), 0
        pPackageEvents.OnQueryCancel Me.Description, blnCancel
        If blnCancel Then Exit Do
    End If
End If

    DoEvents
Loop

'Write elapsed time and GV value to log, if log object valid.
If Not pPackageLog Is Nothing Then
    pPackageLog.WriteStringToLog Me.Description & ": " & _
        objMonitor.Name & " = " & objMonitor.Value
    pPackageLog.WriteTaskRecord 0, _
        Me.Description & " elapsed time: " & _
        (DateDiff("s", datStartTime, Now)) & " sec."
End If

'Close and release form.
Unload frmShowGV
Set frmShowGV = Nothing

End Sub

Private Property Get CustomTask_Properties() As DTS.Properties
'Use DTS properties provider to generate collection.
    Dim oPropsProvider As New DTS.PropertiesProvider

    Set CustomTask_Properties = oPropsProvider.GetPropertiesForObjec
    Set oPropsProvider = Nothing

End Property

```

```
Private Property Let CustomTask_Description(ByVal strNewDescr As  
'Implements Task.Description.  
    mstrDescription = strNewDescr  
End Property
```

```
Private Property Get CustomTask_Description() As String  
'Implements Task.Description.  
    CustomTask_Description = mstrDescription  
End Property
```

```
Private Property Let CustomTask_Name(ByVal strNewName As String)  
'Implements Task.Name.  
    mstrTaskName = strNewName  
End Property
```

```
Private Property Get CustomTask_Name() As String  
'Implements Task.Name.  
    CustomTask_Name = mstrTaskName  
End Property
```

```
Private Sub PersistPropertyBag_Save(ByVal propBag As DTS.PropertyBag)  
'Save property values in property bag.
```

```
    'On Error Resume Next  
    propBag.Write "Name", mstrTaskName  
    propBag.Write "Description", mstrDescription  
    propBag.Write "GVMonitor", mstrGVMonitorName  
    propBag.Write "GVFinish", mstrGVFinishName
```

```
End Sub
```

```
Private Sub PersistPropertyBag_Load(ByVal propBag As DTS.PropertyBag)
```

'Load property values from property bag.

'On Error Resume Next.

mstrTaskName = propBag.Read("Name")

mstrDescription = propBag.Read("Description")

mstrGVMonitorName = propBag.Read("GVMonitor")

mstrGVFinishName = propBag.Read("GVFinish")

End Sub

Public Property Get GVMonitor() As String

'Name of global variable to monitor.

GVMonitor = mstrGVMonitorName

End Property

Public Property Let GVMonitor(ByVal strNewName As String)

'Name of global variable to monitor, verify non-empty.

If Len(strNewName) > 0 Then

mstrGVMonitorName = strNewName

Else

Err.Raise 1001 + vbObjectError, Me.Name, INVALID_PROP

End If

End Property

Public Property Get GVFinish() As String

'Name of global variable to indicate finish.

GVFinish = mstrGVFinishName

End Property

Public Property Let GVFinish(ByVal strNewName As String)

'Name of global variable to indicate finish, verify non-empty.

```
If Len(strNewName) > 0 Then
    mstrGVFinishName = strNewName
Else
    Err.Raise 1001 + vbObjectError, Me.Name, INVALID_PROP
End If
```

End Property

```
Public Property Get Name() As String
'Implements FinalGlobal.Name.
    Name = mstrTaskName
End Property
```

```
Public Property Let Name(ByVal strNewName As String)
'Implements FinalGlobal.Name
    mstrTaskName = strNewName
End Property
```

```
Public Property Get Description() As String
'Implements FinalGlobal.Description
    Description = mstrDescription
End Property
```

```
Public Property Let Description(ByVal strNewDescr As String)
'Implements FinalGlobal.Description
    mstrDescription = strNewDescr
End Property
```

To build this DTS custom task

1. In the Visual Basic development environment, create a new ActiveX DLL project.

2. On the **Project** menu, click **References**, and under **Available References**, select the check box for **Microsoft DTSPackage Object Library**. Then, on the **Project** menu, click **Properties** and in the **Project name** box, change the project name from Project1 to something meaningful, like **DTSConcurrentSample**.
3. Copy the code for the **ShowGlobal** class in the preceding code example to the class module in the Visual Basic project. Change the name of the class module from **Class1** to **ShowGlobal**. If you use a different name, you need to change the references to **ShowGlobal** in the code to that name.
4. Add the **frmFinalGlobal** form.

For more information about this form, see [DTS Example: Including a User Interface in Visual Basic](#).
5. On the **File** menu, click **Make DTSConcurrentSample.dll** to build the component.

Do not register the task in DTS Designer.

DTS Package Application

This DTS application uses the **ShowGlobal** custom task to display the number of rows copied, via the **Rows Copied** global variable, while another step copies rows from a table in one database to a table in another. The copy step closes the display by setting the **Copy Complete** global variable to TRUE.

The copy step uses a **DataPumpTask2** object with a **DataPumpTransformScript** transformation to copy certain columns from the **Products** table in the Microsoft SQL Server™ **Northwind** database to a table named **NorthwindProducts** in a database named **DTS_UE**. DTS lookups are used to replace the **CategoryID** field with the **CategoryName** from the **Northwind Categories** table, and to replace the **SupplierID** field with the **CompanyName** from the **Northwind Suppliers** table.

Creating the DTS Package Application

This is the definition of the **NorthwindProducts** table in **DTS_UE**:

```
CREATE TABLE [DTS_UE].[dbo].[NorthwindProducts] (  
    [ProductName] [nvarchar] (40) NULL ,  
    [CategoryName] [nvarchar] (25) NULL ,  
    [CompanyName] [nvarchar] (40) NULL )
```

This is the Visual Basic code for the application:

```
Public Sub Main()  
'Copy Northwind..Products names, categories, suppliers to DTS_UE..N  
    Dim objPackage As DTS.Package2  
    Dim objConnect As DTS.Connection2  
    Dim objStep As DTS.Step2  
    Dim objTask As DTS.Task  
    Dim objPumpTask As DTS.DataPumpTask2  
    Dim objCustTask As DTSCurrentSample.ShowGlobal  
    Dim objTransform As DTS.Transformation2  
    Dim objLookUp As DTS.Lookup  
    Dim objTranScript As DTSPump.DTSTransformScriptProperties2  
    Dim sVBS As String 'VBScript text  
  
    Set objPackage = New DTS.Package  
    objPackage.FailOnError = True  
    objPackage.LogFileName = "C:\Temp\TestConcurrent.Log"  
  
    'Establish connections to data source and destination.  
    Set objConnect = objPackage.Connections.New("SQLOLEDB.1")  
    With objConnect  
        .ID = 1  
        .DataSource = "(local)"  
        .UseTrustedConnection = True  
    End With
```

```
objPackage.Connections.Add objConnect
Set objConnect = objPackage.Connections.New("SQLOLEDB.1")
With objConnect
    .ID = 2
    .DataSource = "(local)"
    .UseTrustedConnection = True
End With
objPackage.Connections.Add objConnect
```

```
'Create copy step and task, link step to task.
Set objStep = objPackage.Steps.New
objStep.Name = "NorthwindProductsStep"
Set objTask = objPackage.Tasks.New("DTSDataPumpTask")
Set objPumpTask = objTask.CustomTask
objPumpTask.Name = "NorthwindProductsTask"
objStep.TaskName = objPumpTask.Name
objStep.ExecuteInMainThread = False
objPackage.Steps.Add objStep
```

```
'Link copy task to connections.
With objPumpTask
    .SourceConnectionID = 1
    .SourceSQLStatement = _
        "SELECT ProductName, CategoryID, SupplierID " & _
        "FROM Northwind..Products"
    .DestinationConnectionID = 2
    .DestinationObjectName = "[DTS_UE].[dbo].[NorthwindProduct"
    .UseFastLoad = False
    .MaximumErrorCount = 99
End With
```

```
'Create lookups for supplier and category.
Set objLookUp = objPumpTask.Lookups.New("CategoryLU")
```

```
With objLookUp
    .ConnectionID = 1
    .Query = "SELECT CategoryName FROM Northwind..Categories
            "WHERE CategoryID = ? "
    .MaxCacheRows = 0
```

```
End With
```

```
objPumpTask.Lookups.Add objLookUp
```

```
Set objLookUp = objPumpTask.Lookups.New("SupplierLU")
```

```
With objLookUp
```

```
    .ConnectionID = 1
```

```
    .Query = "SELECT CompanyName FROM Northwind..Suppliers
            "WHERE SupplierID = ? "
```

```
    .MaxCacheRows = 0
```

```
End With
```

```
objPumpTask.Lookups.Add objLookUp
```

```
'Create and initialize rowcount and completion global variables.
```

```
objPackage.GlobalVariables.AddGlobalVariable "Copy Complete", 0
```

```
objPackage.GlobalVariables.AddGlobalVariable "Rows Copied", 0
```

```
objPackage.ExplicitGlobalVariables = True
```

```
'Create transform to copy row, signal completion.
```

```
Set objTransform = objPumpTask.Transformations. _
```

```
    New("DTSPump.DataPumpTransformScript")
```

```
With objTransform
```

```
    .Name = "CopyNorthwindProducts"
```

```
    .TransformPhases = DTSTransformPhase_Transform + _
        DTSTransformPhase_OnPumpComplete
```

```
    Set objTranScript = .TransformServer
```

```
End With
```

```
With objTranScript
```

```
    .FunctionEntry = "CopyColumns"
```

```
    .PumpCompleteFunctionEntry = "PumpComplete"
```

```
.Language = "VBScript"
sVBS = "Option Explicit" & vbCrLf
sVBS = sVBS & "Function CopyColumns()" & vbCrLf
sVBS = sVBS & "  DTSDestination("""ProductName""") = DTSSc
sVBS = sVBS & "  DTSDestination("""CategoryName""") = DTSI
sVBS = sVBS & "  DTSDestination("""CompanyName""") = DTS
sVBS = sVBS & "  DTSGlobalVariables("""Rows Copied""") = CI
sVBS = sVBS & "  CopyColumns = DTSTransformStat_OK" &
sVBS = sVBS & "End Function" & vbCrLf
```

```
sVBS = sVBS & "Function PumpComplete()" & vbCrLf
sVBS = sVBS & "  DTSGlobalVariables("""Copy Complete""") = '
sVBS = sVBS & "  PumpComplete = DTSTransformStat_OK" &
sVBS = sVBS & "End Function" & vbCrLf
```

```
.Text = sVBS
```

```
End With
```

```
objPumpTask.Transformations.Add objTransform
objPackage.Tasks.Add objTask
```

```
'Create monitor step and task, link step to task.
```

```
Set objStep = objPackage.Steps.New
```

```
objStep.Name = "GVMonitorStep"
```

```
Set objTask = objPackage.Tasks.New("DTSCurrentSample.Show
```

```
objTask.Name = "GVMonitorTask"
```

```
objStep.TaskName = objTask.Name
```

```
Set objCustTask = objTask.CustomTask
```

```
objCustTask.GVMonitor = "Rows Copied"
```

```
objCustTask.GVFinish = "Copy Complete"
```

```
objStep.ExecuteInMainThread = True
```

```
objPackage.Steps.Add objStep
```

```
'Link monitor task to package, run package.
```

```
objPackage.Tasks.Add objTask  
objPackage.Execute
```

End Sub

IMPORTANT This sample application is intentionally implemented to run slowly so the display will be more convenient to view. It uses the same connection for the data source and both lookups, and uses 0 for the **MaxCacheRows** property for both lookups. In an actual application that copies and transforms databases, you should use a separate connection for lookups, or for each lookup. You should also use a nonzero value for **MaxCacheRows**. The default of 100 is recommended for the initial choice.

To build this DTS application

1. Build the **DTSCurrentSample.ShowGlobal** custom task, as described in the preceding example.
2. Create a database named **DTS_UE** containing a table named **NorthwindProducts**, using the definition given above.
3. Create a Standard EXE project in the Visual Basic development environment.
4. On the **Project** menu, click **References**, and under **Available References**, select the check boxes for **DTSCurrentSample**, **Microsoft DTSPump Scripting Object Library** and **Microsoft DTSPackage Object Library**.
5. Add a standard module to the project, and then copy the code for the application in the preceding code example into the module. Remove the form Form1 from the project.
6. On the **Run** menu, click **Start**.

Observe the displayed rowcount.

See Also

[DataPumpTask2 Object](#)

[DataPumpTransformScript Object](#)

[DTS Example: Including a User Interface in Visual Basic](#)

[MaxCacheRows Property](#)

[OnProgress Event](#)

[OnQueryCancel Event](#)

[PersistPropertyBag Object](#)

DTS Programming

Implementing DTS Custom Tasks in Visual C++

This section describes using the Active Template Library (ATL) facility of Microsoft® Visual C++® to implement Data Transformation Services (DTS) custom tasks.

Topic	Description
Building a DTS Custom Task from a Standard ATL Template	Describes how to create a framework for a custom task from a standard ATL template.
Building a DTS Custom Task from the ATL Custom Task Basic Template	Describes how to create a custom task framework from the basic ATL custom task template supplied with Microsoft SQL Server™ 2000.
Adding a User Interface to the Custom Task Framework	Describes how to create a framework that supports a user interface from the basic ATL custom task template.
Building a DTS Custom Task with User Interface from the ATL Custom Task Templates	Describes how to create a custom task that supports a user interface from the ATL custom task templates that have been enabled for a user interface.
Implementing and Testing a DTS Custom Task	Describes how to implement and test a custom task framework and explains how to use the Visual C++ debugger.

Building a DTS Custom Task from a Standard ATL Template

One way to build a custom task is to create a project from a standard Active Template Library (ATL) template, add the interface and other elements required by all Data Transformation Services (DTS) tasks, and then add the features of the specific custom task.

This topic explains how to add the elements required by all DTS tasks. You can also use the basic ATL custom task template supplied as a sample with Microsoft® SQL Server™ 2000 to build the custom task framework. Even if you plan to use the custom task template, you need to understand the features that were added to create the basic custom task template from the standard object template. For more information, see [Building a Custom Task from the ATL Custom Task Basic Template](#).

Building a Standard ATL Component

You can create a standard ATL component that includes a class for the custom task using Microsoft Visual C++® version 6.0.

To build a standard ATL component with a class

1. On the **File** menu, click **New**, and then click the **Projects** tab.
2. Click **ATL COM AppWizard**, and then enter a project name and location.

For this discussion, assume you entered **DTSCusTskBasic** for the project name.
3. Click **Dynamic Link Library (DLL)**, click **Finish**, and in the **New Project Information** dialog box, click **OK**.
4. On the **Insert** menu, click **New ATL Object**, click **Objects**, click **Simple Object** and then click **Next**.

5. On the **Names** tab, enter a short name.

For this discussion, assume you entered **TaskNoUI**. The wizard will fill in the other fields. The **COM/Type** field is the name that will appear in the **Tasks** menu of DTS Designer. You can change it from the default **TaskNoUI Class**.

6. Click the **Attributes** tab, and then do the following:

- Under **Threading Model**, click **Both**.
- Under **Interface**, click **Dual**.
- Under **Aggregation**, click **No**.
- Select the **Support ISupportErrorInfo** check box.

The wizards will create files for the **DTSCusTskBasic** component and the **TaskNoUI** class and save them to the project location folder specified in Step 1.

Adding Custom Task Features

After creating a standard ATL component with **TaskNoUI** class files, you need to add custom task elements. In this section, all files will have the same names you specified in Step 4 of the previous procedure.

File	Features
TaskNoUI.h	Function prototypes, private declarations and COM map entries for the IDTSCustomTask interface elements
TaskNoUI.cpp	Initial function definitions for the IDTSCustomTask interface elements and the CTaskNoUI constructor and destructor
TaskNoUI.rgs	Registry subkeys required for DTS tasks
DTSCusTskBasic.idl	Declaration of the IDTSCustomTask interface

TaskNoUI.h

In this header file, you need to add the following:

- Include statements for the DTS package header file
- Prototypes for the class constructor and destructor
- A COM Map entry for the **IDTSCustomTask** interface
- Prototypes for the **IDTSCustomTask** interface elements

Adding an Include Statement

Immediately preceding the include statement for resource.h:

```
#include "resource.h"    // main symbols
```

add this header file include statement:

```
#include "dtspkg.h"
```

Adding Constructor and Destructor Prototypes

The constructor body will be moved to TaskNoUI.cpp. Replace the following lines:

```
CTaskNoUI()  
{  
}
```

with these prototype declarations:

```
CTaskNoUI();
```

```
~CTaskNoUI();
```

Adding a COM Map Entry

Immediately following the COM map entry for **IDispatch**:

```
COM_INTERFACE_ENTRY(IDispatch)
```

add this COM map entry for **IDTSCustomTask**:

```
COM_INTERFACE_ENTRY2(IDTSCustomTask, ITaskNoUI)
```

Supplying Function Prototypes

You must supply the function prototypes for the **IDTSCustomTask** interface and declarations for storage for the properties.

Immediately after the following lines:

```
// ITaskNoUI  
public:
```

add these lines of code:

```
STDMETHOD(get_Properties)(  
    /* [retval][out] */ IDispatch **pRetVal);
```

```
STDMETHOD(get_Name)(  
    /* [retval][out] */ BSTR *pRetVal);
```

```
STDMETHOD(put_Name)(  
    /* [in] */ BSTR NewValue);
```

```
STDMETHOD(get_Description)(  
    /* [retval][out] */ BSTR *pRetVal);
```

```
STDMETHOD(put_Description)(  
    /* [in] */ BSTR NewValue);
```

```

STDMETHOD(Execute)(
    /* [in] */ IDispatch *pPackage,
    /* [in] */ IDispatch *pPackageEvents,
    /* [in] */ IDispatch *pPackageLog,
    /* [out][in] */ LONG *pTaskResult);

```

private:

```

// Internal storage for Name, Description properties.
BSTR  m_bstrName;
BSTR  m_bstrDescription;

```

TaskNoUI.cpp

This file is where you provide the initial function definitions for the elements of the **IDTSCustomTask** interface and the class constructor and destructor.

Adding Initial Function Definitions

Add these lines of code at the end of the existing file:

```

CTaskNoUI::CTaskNoUI()
{
    m_bstrName = SysAllocString( OLESTR("") );
    m_bstrDescription = SysAllocString( OLESTR("") );
}
CTaskNoUI::~~CTaskNoUI()
{
    if (m_bstrName) SysFreeString(m_bstrName);
    if (m_bstrDescription) SysFreeString(m_bstrDescription);
}

STDMETHODIMP CTaskNoUI::get_Properties(
    /* [retval][out] */ IDispatch **pRetVal)
{

```

```
// You can implement a properties collection if you want or just return  
// DTS will implement a 'PropertiesProvider' utility object which will  
// extract the properties out of your IDispatch typeinfo.
```

```
*pRetVal = NULL;  
return NOERROR;  
}
```

```
STDMETHODIMP CTaskNoUI::get_Name(  
    /* [retval][out] */ BSTR *pRetVal)  
{  
    if (!pRetVal)  
        return E_POINTER;  
    *pRetVal = SysAllocString(m_bstrName);  
    if (!*pRetVal)  
        return E_OUTOFMEMORY;  
    return NOERROR;  
}
```

```
STDMETHODIMP CTaskNoUI::put_Name(  
    /* [in] */ BSTR NewValue)  
{  
    if (m_bstrName)  
        SysFreeString(m_bstrName);  
    m_bstrName = SysAllocString(NewValue);  
    if (!m_bstrName)  
        return E_OUTOFMEMORY;  
    return NOERROR;  
}
```

```
STDMETHODIMP CTaskNoUI::get_Description(  
    /* [retval][out] */ BSTR *pRetVal)  
{
```

```
    if (!pRetVal)
        return E_POINTER;
    *pRetVal = SysAllocString(m_bstrDescription);
    if (!*pRetVal)
        return E_OUTOFMEMORY;
    return NOERROR;
}
```

```
STDMETHODIMP CTaskNoUI::put_Description(
    /* [in] */ BSTR NewValue)
{
    if (m_bstrDescription)
        SysFreeString(m_bstrDescription);
    m_bstrDescription = SysAllocString(NewValue);
    if (!m_bstrDescription)
        return E_OUTOFMEMORY;
    return NOERROR;
}
```

```
STDMETHODIMP CTaskNoUI::Execute(
    /* [in] */ IDispatch *pPackage,
    /* [in] */ IDispatch *pPackageEvents,
    /* [in] */ IDispatch *pPackageLog,
    /* [out][in] */ LONG *pTaskResult)
{
    USES_CONVERSION;    // Needed for functions like A2W, OLE
    HRESULT hr = NOERROR;

    // TODO: Put functionality of custom task here.

    *pTaskResult = DTSTaskExecResult_Success;
    return hr;
}
```

TaskNoUI.rgs

This file contains the registry script for the task class. The lines to be added define the task icon location, localizable task description, and the component category for DTS tasks.

Adding Registry Script

Insert these lines immediately following the definition of the **TypeLib** subkey:

```
'Implemented Categories'
{
    '{10020200-EB1C-11CF-AE6E-00AA004A34D5}'
}
DTSTask
{
    '1033'
    {
        val DTSIconFile = s '%MODULE%'
        val DTSIconIndex = d 0
        val DTSTaskDescription = s 'TaskNoUI Class'
    }
}
val DTSIconFile = s '%MODULE%'
val DTSIconIndex = d 0
val DTSTaskDescription = s 'TaskNoUI Class'
```

To verify that the Implemented Categories globally unique identifier (GUID) is correct, look for it in `dtspkg.h` under the definition for **CATID_DTSCustomTask**. You can verify that it is among the subkeys of `HKEY_CLASSES_ROOT\Component Categories\` in the registry of a computer on which SQL Server 2000 client tools (or the full product) have been installed.

DTSCusTskBasic.idl

This file contains the definitions for the interfaces of the project. You need to

add the definitions of the elements of the **IDTSCustomTask** interface to the **ITaskNoUI** definition.

Adding IDTSCustomTask Element Definitions

Insert the following interface element definitions into the definition of the **ITaskNoUI** interface, immediately following these lines:

```
interface ITaskNoUI : IDispatch
{
```

Add these lines of code:

```
//*****>
// This interface implements IDTSCustomTask. We cannot direct
// because ATL requires us to explicitly inherit from IDispatch.
//*****>

[id(7), propget, helpstring("List of Properties for this object")]
HRESULT Properties([out, retval] IDispatch **pRetVal);
[id(9), propget, helpstring("Name of Task")]
HRESULT Name([out, retval] BSTR *pRetVal);
[id(9), propput]
HRESULT Name([in] BSTR NewValue);
[id(10), propget, helpstring("Description of the task")]
HRESULT Description([out, retval] BSTR *pRetVal);
[id(10), propput]
HRESULT Description([in] BSTR NewValue);

[id(11), helpstring("Execute Task. Reference to Package allows a
HRESULT Execute([in] IDispatch * pPackage, [in] IDispatch * p

//*****>
// Make sure that any elements you add to this interface go at
// the end of the vtable.
```

//*****>

Building a DTS Custom Task from the ATL Custom Task Basic Template

To build a custom task with a user interface, use the ATL custom task basic template. This template is included in Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) sample programs. The basic template does not support a custom user interface. For more information about DTS samples, see [DTS Programming Samples](#).

Installing the ATL Custom Task Basic Template

To install the ATL custom task basic template, do the following:

1. Copy all the files in the DTSTaskBasicTemplate folder except DTSCuTsk.reg to C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template\ATL\. This location will be different if Microsoft® Visual Studio® version 6.0 was not installed to the default location.
2. Double-click DTSCuTsk.reg to run the file.

Building a Custom Task Framework from the Template

You can create an ATL component that includes a custom task class that does not support a user interface by using the Microsoft Visual C++® development environment.

To build a custom task framework from the template

1. On the **File** menu, click **New**, and then click the **Projects** tab.
2. Click **ATL COM AppWizard**, and then enter a project name and location.

For this discussion, assume you entered **DTSCusTskBasic** for the

project name.

3. Click **Dynamic Link Library (DLL)**, click **Finish**, and then in the **New Project Information** dialog box, click **OK**.
4. On the **Insert** menu, click **New ATL Object**.
5. On the **ATL Object Wizard** screen, click **DTS Custom Objects**, click **DTS Task w/o UI**, and then click **Next**.
6. On the **Names** tab, enter a name into the **Short Name** box.

For this discussion, assume you entered **TaskNoUI**. The wizard will fill in the other fields. The **COM/Type** field is the name that will appear in the **Task** menu of DTS Designer. You can change it from the default **TaskNoUI Class**.

7. Click the **Attributes** tab, and then do the following:
 - Under **Threading Model**, click **Both**.
 - Under **Interface**, click **Dual**.
 - Under **Aggregation**, click **No**.
 - Select the **Support ISupportErrorInfo** check box.

The wizards will create files for the **DTSCusTskBasic** component and the **TaskNoUI** class and save them to the project location folder specified in Step 1.

If you build this custom task project from the **Build/Build DTSCusTskBasic.dll** menu before you add any custom code, Visual C++ installs a custom task that appears in the **Task** menu of DTS Designer and can be included in a DTS package. When added to a package, the task uses the DTS default icon and displays the DTS default property grid. However, this task will not perform any function when the package is run. For more information on implementing and

testing a custom task, see [Implementing and Testing a DTS Custom Task](#).

Adding a DTS User Interface to the Custom Task Framework

To build a custom task that supports a custom task user interface, create a project from the Active Template Library (ATL) custom task basic template, add another class and the interface and other elements required by tasks that support a custom user interface, and then add the features of the specific custom task.

This topic explains how to add the elements required by a Data Transformation Services (DTS) task with a custom user interface. You can also use the ATL custom task user interface templates supplied as samples with Microsoft® SQL Server™ 2000. Even if you plan to use the custom task user interface templates, you need to understand the features that were added to create the custom task user interface templates from the basic custom task template. For more information, see [Building a Custom Task with User Interface from the ATL Custom Task Templates](#).

Building a Custom Task Framework with a UI Class

You can create a Custom Task Framework that includes a class for the custom user interface using Microsoft Visual C++® version 6.0.

To build a custom task framework with a user interface class

1. Create a framework for a custom task using the ATL custom task basic template.

Assume you named the component **DTSCusTskWUI** and the custom task class **TaskUISupp**. For more information, see [Building a Custom Task from the ATL Custom Task Basic Template](#).

2. Add another class for the user interface. On the **Insert** menu, click **New ATL Object**. On the **ATL Object Wizard** screen, click **Objects**, and then click **Simple Object**. Click Next.
3. On **Names**, enter a short name.

Assume you entered **UserIF**. The wizard will fill in the other fields.

4. Click the **Attributes** tab, and then do the following:
 - Under **Threading Model**, click **Apartment**.
 - Under **Interface**, click **Dual**.
 - Under **Aggregation**, click **Yes**.

The user interface class will not work unless it can be aggregated.

The wizards will create files for the **DTSCusTskWUI** component and the **TaskUISupp** and **UserIF** classes, and save them to the project location folder specified when you created the framework in the first step.

Adding Features to Support a Custom UI

After creating the custom task framework with **TaskUISupp** and **UserIF** class files, you need to add features to support the user interface. In this section, all files will have the same names you specified for the classes.

File	Features
TaskUISupp.h	Function prototypes, private declarations, and COM map entries for the elements that connect the custom task class to the user interface class.
TaskUISupp.cpp	Definitions for the functions that connect the task class to the user interface class.
UserIF.h	Function prototypes, private declarations, and COM map entries for the IDTSCustomTaskUI interface elements.
UserIF.cpp	Initial function definitions for the IDTSCustomTaskUI interface methods.
DTSCusTskWUI.idl	Declaration of the IDTSCustomTaskUI interface elements in the IUserIF interface.

TaskUISupp.h

In this header file, you need to add the following:

- Prototypes for the functions that connect the custom task class to the user interface class
- COM Map entries for the **IDTSCustomTask** and **IDTSCustomTaskUI** interfaces
- A declaration of an interface pointer variable

Adding Hook Function Prototypes

These prototypes are for functions that are called when a **QueryInterface** request for the **IDTSCustomTaskUI** interface is made to the task class.

Following the constructor and destructor prototypes for the task class:

```
CTaskUISupp();  
~CTaskUISupp();
```

add these lines of code:

```
static HRESULT WINAPI FuncPreQueryInterface(void* pv, REFIID  
LPVOID* ppv, DWORD dw);  
HRESULT PreQueryInterface(REFIID riid, LPVOID *ppv);
```

Adding COM Map Entries

You need to replace the COM map entry for the **ITaskUISupp** interface so that it responds when presented with the interface ID for the **IDTSCustomTask**. You need to add a COM map entry that invokes the hook function when presented with the interface ID for **IDTSCustomTaskUI**.

Replace the following COM map entries:

```
COM_INTERFACE_ENTRY(IDispatch)  
COM_INTERFACE_ENTRY2(IDTSCustomTask, ITaskUISupp)
```

with these lines:

```
COM_INTERFACE_ENTRY2(IDispatch, ITaskUISupp)  
COM_INTERFACE_ENTRY_IID(IID_IDTSCustomTask, ITaskUISupp)  
COM_INTERFACE_ENTRY_FUNC(IID_IDTSCustomTaskUI, 0, 0)
```

Adding an Interface Pointer Variable

This declaration is for a pointer variable for the user interface class, which is set by the hook functions.

Immediately following these lines:

```
BSTR    m_bstrName;  
BSTR    m_bstrDescription;
```

add this declaration:

```
IUnknown * m_pIUnkDTSCustomTaskUI;
```

TaskUISupp.cpp

In this Visual C++ file for the task class, you need to add the following:

- An external reference for the class id of the user interface class
- Code to initialize the user interface class interface pointer variable in the task class constructor
- Code to release the pointer to the user interface class in the task class destructor
- Functions that create an instance of the user interface class and issue a **QueryInterface** for **IDTSCustomTaskUI**

Adding an External Reference for the class id

Following this include statement:

```
#include "TaskUISupp.h"
```

add this external reference:

```
extern const CLSID CLSID_UserIF;
```

Initializing the User Interface Pointer

At the end of the task class constructor (before the right curly bracket):

```
CTaskUISupp::CTaskUISupp()
```

add this line of code:

```
    m_pIUnkDTSCustomTaskUI = NULL;
```

Releasing the Interface Pointer

Release the reference to the **IDTSCustomTaskUI** interface if it exists in the task class destructor.

At the end of the destructor (before the right curly bracket):

```
CTaskUISupp::~~CTaskUISupp()
```

add these lines of code:

```
    if(m_pIUnkDTSCustomTaskUI)
        if(m_pIUnkDTSCustomTaskUI->Release() != 0)
            /* _ASSERT(0) */ NULL;
```

Adding Hook Function Definitions

These functions first obtain a reference to the aggregating object, which is the custom task class. If the **QueryInterface** request is for the **IDTSCustomTaskUI** interface and the user interface has not yet been created, an instance of the user interface is created. Then the **QueryInterface** is requested from the aggregated object, the user interface class.

After the task class destructor:

```
CTaskUISupp::~~CTaskUISupp() { ... }
```

add these lines of code:

```
HRESULT WINAPI CTaskUISupp::FuncPreQueryInterface(void* pv,
{
    HRESULT hr = E_FAIL;
    _ASSERT(pv);
    CTaskUISupp * pDTSCustTask = (CTaskUISupp *)pv;
    return pDTSCustTask->PreQueryInterface(riid, ppv);
}
```

```
HRESULT CTaskUISupp::PreQueryInterface(REFIID riid, LPVOID *
{
    HRESULT hr = S_FALSE;
    IUnknown * pIUnknownOuter;

    if (!ppv)
    {
        hr = E_POINTER;
        goto error;
    }
    if FAILED(hr = QueryInterface(IID_IUnknown, (void **)&pIUnkn
        goto error;

    *ppv = NULL;
    if (IID_IDTSCustomTaskUI == riid)
    {
        if(!m_pIUnkDTSCustomTaskUI)
            if FAILED(hr = CoCreateInstance(CLSID_UserIF,
                pIUnknownOuter, CLSCTX_INPROC_SE
                IID_IUnknown, (LPVOID*)&m_pIUnkDTI
```

```

        goto error;

        hr = m_pIUnkDTSCustomTaskUI->QueryInterface(riid, ppv);
    }
    else
        hr = S_FALSE;

error:
    return hr;
}

```

UserIF.h

In this header file for the user interface class, you need to add the following:

- Include statements for the DTS package header file
- Prototypes for the class constructor and destructor
- A COM Map entry for the **IDTSCustomTaskUI** interface
- Prototypes for the **IDTSCustomTaskUI** interface elements

Adding an Include Statement

Immediately preceding the include statement for resource.h:

```
#include "resource.h"    // main symbols
```

add this header file include statement:

```
#include "dtspkg.h"
```

Adding Constructor and Destructor Prototypes

You need to add prototypes for the class constructor and destructor. The

constructor body will be moved to UserIF.cpp

Replace the following lines:

```
CUserIF()
{
}
```

with these prototype declarations:

```
CUserIF();
~CUserIF();
```

Adding a COM Map Entry

Immediately following the COM map entry for **IDispatch**:

```
COM_INTERFACE_ENTRY(IDispatch)
```

add this COM map entry for **IDTSCustomTaskUI**:

```
COM_INTERFACE_ENTRY_IID(IID_IDTSCustomTaskUI, IUserIF)
```

Adding Function Prototypes

You must supply the function prototypes for the **IDTSCustomTask** interface and declarations for pointers to the generic and custom task interfaces.

Immediately after the following lines:

```
// IUserIF
public:
```

add these lines of code:

```
STDMETHOD(CreateCustomToolTip)(long hwndParent, long x, long y);
STDMETHOD(Help)(long hwndParent);
STDMETHOD(Delete)(long hwndParent);
STDMETHOD(Edit)(long hwndParent);
STDMETHOD(New)(long hwndParent);
```

```
STDMETHOD(GetUIInfo)(BSTR *pbstrToolTip, BSTR *pbstrDesc, BSTR *pbstrImage);
STDMETHOD(Initialize)(IUnknown *pTask);
```

private:

```
    IDTSTask      * m_pIDTSTask;
    IDTSCustomTask * m_pIDTSCustomTask;
```

UserIF.cpp

In this file, you must provide the initial function definitions for the elements of the **IDTSCustomTaskUI** interface and the class constructor and destructor.

Adding Initial IDTSCustomTaskUI Function Definitions

Add these lines of code at the end of the existing file:

```
CUserIF::CUserIF()
{
    m_pIDTSTask = NULL;
    m_pIDTSCustomTask = NULL;
}
CUserIF::~~CUserIF()
{
}

STDMETHODIMP CUserIF::CreateCustomToolTip(long hwndParent,
{
    // TODO: Add your implementation code here.

    return E_NOTIMPL;
}

STDMETHODIMP CUserIF::Help(long hwndParent)
{
    // TODO: Add your implementation code here.
```

```

    return E_NOTIMPL;
}

STDMETHODIMP CUserIF::Delete(long hwndParent)
{
    // TODO: Add your implementation code here.

    return S_OK;
}

// Pop up dialog box to get user's property information.
STDMETHODIMP CUserIF::New(long hwndParent)
{
    //if(!m_pIDTSTask || !m_pIDTSCustomTask)
    // return E_FAIL;
    //return NOERROR;

    // TODO: Add your implementation code here.

    return E_NOTIMPL;
}

// Pop up dialog box to get user's property information.
// Fill the existing information in the controls.
STDMETHODIMP CUserIF::Edit(long hwndParent)
{
    //if(!m_pIDTSTask || !m_pIDTSCustomTask)
    // return E_FAIL;
    //return NOERROR;

    // TODO: Add your implementation code here.

```

```

    return E_NOTIMPL;
}

STDMETHODIMP CUserIF::GetUIInfo(BSTR *pbstrToolTip, BSTR
{
    // TODO: Add your implementation code here.

    return S_OK;
}

// Get custom task interface pointer.
STDMETHODIMP CUserIF::Initialize(IUnknown *pTask)
{
    HRESULT hr = E_FAIL;

    m_pIDTSTask = NULL;

    if FAILED(hr = pTask->QueryInterface(IID_IDTSTask, (void **) &
        return hr;

    // Release it immediately. Do not add ref outer object.
    m_pIDTSTask->Release();

    if FAILED(hr = m_pIDTSTask->GetCustomTask(&m_pIDTSCusto
        return hr;

    // Release it immediately. Do not add ref outer object.
    m_pIDTSCustomTask->Release();

    return NOERROR;
}

```

DTSCusTskWUI.idl

In this file, which contains the definitions for the interfaces of the project, you need to add the definitions of the elements of the **IDTSCustomTaskUI** interface to the **IUserIF** definition.

Adding IDTSCustomTaskUI Element Definitions

Into the definition of the **ITaskNoUI** interface, immediately following these lines:

```
interface IUserIF : IDispatch
{
```

add these lines of code:

```
[id(106), helpstring("Initialize the object at design time with any i
HRESULT Initialize(IUnknown *pTask);
[id(100), helpstring("This method is called to get top level UI info
HRESULT GetUIInfo(BSTR *pbstrToolTip, BSTR *pbstrDescrip
[id(101), helpstring("A New instance of the custom task is to be a
HRESULT New(long hwndParent);
[id(102), helpstring("The 'Edit' command has been invoked on the
HRESULT Edit(long hwndParent);
[id(103), helpstring("The 'Delete' command has been invoked on t
HRESULT Delete(long hwndParent);
[id(104), helpstring("The 'Help' command has been invoked on th
HRESULT Help(long hwndParent);
[id(105), helpstring("If the custom task requested a custom tooltip
HRESULT CreateCustomToolTip(long hwndParent, long x, long :
```

Building a DTS Custom Task with a User Interface from the ATL Custom Task Templates

To build a custom task with a user interface, use the Active Template Library (ATL) custom task templates, which support a user interface. These templates are included in the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) sample programs. The basic template does not support a user interface. For more information about DTS samples, see [DTS Programming Samples](#).

Installing the ATL Custom Task User Interface Templates

To install the ATL custom task user interface templates, do the following:

1. Copy all the files in the DTSTaskUITemplates folder except DTSCuTskUI.reg to C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template\ATL\. This location will be different if Microsoft Visual Studio® version 6.0 was not installed to the default location.
2. Double-click DTSCuTskUI.reg to run the file.

Building a Custom Task Framework from the Templates

You can create an ATL component that includes both a custom task class that supports a custom user interface, as well as a user interface class, by using the Microsoft Visual C++® development environment.

To build a custom task framework from the templates

1. On the **File** menu, click **New**, and then click the **Projects** tab.
2. Click **ATL COM AppWizard**, and then enter a project name and location.

Assume you entered **DTSCusTskWUI** for the project name.

3. Click **Dynamic Link Library (DLL)**, click **Finish**, and in the **New Project Information** dialog box, click **OK**.
4. On the **Insert** menu, click **New ATL Object**.
5. On the **ATL Object Wizard** screen, click **DTS Custom Objects**, click **DTS Task w/ UI Support**, and then click **Next**.
6. On the **Names** tab, enter a short name.

Assume you entered **TaskUISupp**. The wizard will fill in the other fields. The **COM/Type** field is the name that will appear in the **Task** menu of DTS Designer, You can change it from the default **TaskUISupp Class**.

7. Click the **Attributes** tab, and then do the following:
 - Under **Threading Model**, click **Both**.
 - Under **Interface**, click **Dual**.
 - Under **Aggregation**, click **No**.
 - Select the **Support ISupportErrorInfo** check box.
8. Again, on the **Insert** menu, click **New ATL**.
9. On the **ATL Object Wizard** screen, click **DTS Custom Objects**, click **DTS Task w/ UI Support**, and then click **Next**.
10. On the **Names** tab, enter a short name.

Assume you entered **UserIF**. The wizard will fill in the other fields.

11. Click the **Attributes** tab, and then do the following:
 - Under **Threading Model**, click **Apartment**.

 - Under **Interface**, click **Dual**.

 - Under **Aggregation**, click **Yes**.

The wizards will create files for the **DTSCusTskWUI** component and the **TaskUISupp** and **UserIF** classes and save them to the project location folder specified in Step 1.

If you build this custom task project from the **Build/Build DTSCusTskBasic.dll** menu before adding any custom code, you install a custom task that will appear in the **Task** menu of DTS Designer and can be included in a DTS package. When added to a package, the task will use the DTS default icon. Until code is added to the **IDTSCustomTaskUI** methods, it will display the DTS default property grid. However, this task will not perform any function when the package is run. For more information about implementing and testing a custom task, see [Implementing and Testing a DTS Custom Task](#).

Implementing and Testing a DTS Custom Task

To implement and test a Data Transformation Services (DTS) custom task, you need to:

- Install the Microsoft® SQL Server™ 2000 header and library files on your development computer.
- Build a custom task framework.
- Configure Microsoft Visual C++® to build the project.
- Add custom code to the task framework.
- Register and optionally unregister the custom task.
- Debug the custom task.

Installing SQL Server 2000 Header and Library Files

To install the header and library files, you must do a custom installation of either SQL Server 2000 or the SQL Server 2000 client tools on the computer on which you develop the custom task.

To install header and library files during a custom installation

1. In the **Setup Type** dialog box, click **Custom**.
2. In the **Select Component** dialog box, under **Components**, select the **Development Tools** check box.
3. Under **Sub-Components**, check the **Headers and Libraries** and

Debugger Interface check boxes.

Building a Custom Task Framework

To build the task framework, add code to a standard Active Template Library (ATL) template or use the custom task templates included with SQL Server 2000 . For more information, see [Building a Custom Task from the ATL Custom Task Basic Template](#) and [Building a Custom Task with a User Interface from the ATL Custom Task Templates](#).

Configuring Visual C++ to Build the Project

Before you attempt to compile any of the framework files, configure Visual C++ to access SQL Server 2000 header and library files.

To configure Visual C++ to access SQL Server 2000 files

1. On the **Tools** menu, click **Options**.
2. In the **Options** dialog box, click the **Directories** tab.
3. In the **Show directories for** list, enter the paths from the following table at the top of the **Directories** list for each entry.

File type	Path
Executable files	C:\Program Files\Microsoft SQL Server\80\Tools\Binn
Include Files	C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Include
Library files	C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Lib
Source files	C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Include

This only needs to be done once after installing SQL Server 2000. The paths will

be different if SQL Server 2000 components were installed to other than the default locations.

You also need to define the preprocessor symbol `_ATL_NO_UIIDOF`.

To enter preprocessor symbols

1. On the **Project** menu, click **Settings**.
2. Click the **C/C++** tab, and then in the **Preprocessor definitions** box, enter the preprocessor symbols (comma separated) at the end of the list.

Adding Code to the Framework

You need to add code for the logic specific to your custom task.

For all tasks, you need to implement the functionality of your task in the **Execute** method. Typically, the **Name** and **Description** properties do not need to be modified. Logic needs to be added to the **Properties** property only when the functionality of the DTS default properties provider is insufficient. For more information, see [DTS Custom Task Fundamentals](#).

For tasks supporting a user interface, you need to add logic to the **New** and **Edit** methods to display the task user interface. If the task is to display a Help page, you need to add logic to display the page in the **Help** method. The code generated by the templates for these methods returns `E_NOTIMPL`, which causes DTS Designer to display the DTS default property grid or a generic Help page. For more information, see [Including a DTS Custom Task User Interface](#).

To add properties and methods to your custom task

1. In the **Workspace** window, right-click the interface for your custom task class or user interface class, depending on where you want to add the element. Then click **Add Property** or **Add Method**.
2. In the **Add Property to Interface** or **Add Method to Interface** dialog box, enter the name, type and other requested information.

For properties, you will need to specify whether `get_property`,

put_property, or both are supported. Typically, for read/write properties, select the **Get Function** and **Put Function** check boxes and click **PropPut**. For read-only properties, select the **Get Function** check box.

3. Click **Attributes**, and then change the **helpstring** to something meaningful for the property or method.

If you change the **id**, make certain you do not cause the elements you add to come before the elements added by the templates in the interface definition lists in the .idl file.

This procedure adds shells for the **get_property**, **put_property** or *method* functions to your project. You must provide the code to implement them.

The procedure also adds entries to the appropriate interface in the project .idl file. Make sure that entries are added at the end of the list for the interface, because the list order determines the structure of the vtable that Visual C++ uses to navigate to the elements of the interface.

For more information about coding custom task logic, see [DTS Example: Adding Properties and Icons in Visual C++](#) and [DTS Example: Including a User Interface in Visual C++](#).

Registering Custom Tasks

When you build the custom task project in Visual C++, it registers the task as the final step of the build process. If you have enabled DTS component caching, you will need to refresh the cache or DTS Designer will not be able to see the custom task component.

To refresh the cache

1. In SQL Server Enterprise Manager, right-click **Data Transformation Services**, and then click **Properties**.
2. In the **Package Properties** dialog box, click **Refresh Cache**.

When you remove a custom task from your computer, unregister it before deleting the component .dll file.

To unregister a custom task

1. From the command prompt, set the path to the folder that contains the custom task component DLL.
2. Enter:
`regsvr32 /u Component.dll`
3. If DTS caching is enabled, refresh the cache.

Do not rely on utilities like Microsoft RegClean to remove registry entries after you have deleted the corresponding registered files. These utilities often only partially remove registry entries.

Debugging Custom Tasks

It is recommended that you compile both Unicode and non-Unicode versions of your component, even in the absence of a requirement to run on both types of systems. Clean compilation of both Unicode and non-Unicode versions helps ensure that conversion functions such as OLE2T have been used properly. Set the compilation mode from the **Build/Set Active Configuration** menu.

You can debug a custom task by running it from DTS Designer or from a DTS package program (for example, one implemented in Microsoft Visual Basic®). In either case, you must specify the executable name and path, along with any parameters the executable requires, on the **Debug** tab of the **Project Settings** dialog box.

If you use DTS Designer, you typically enter C:\WINNT\system32\mmc.exe in the **Executable for debug session** box and /s "C:\Program Files\Microsoft SQL Server\80\Tools\BINN\SQL Server Enterprise Manager.MSC" in the **Program arguments** box.

To determine the correct debugging settings for your computer, find the shortcut used to launch SQL Server Enterprise Manager from the Start menu. Then extract this information from the Target box on the Shortcut tab of the Enterprise Manager Properties dialog box.

If you get access violations that you cannot trap because they do not occur

within your component, verify that in your .idl file all interface elements are present and that all user-defined properties and methods come at the end of the lists generated by the ATL templates. You can double check your .idl file structure by opening a new project in Visual Basic and referencing your component. View your component in Visual Basic Object Browser and verify all the properties and methods appear as expected.

DTS Programming

DTS Custom Task Examples in Visual C++

This section provides examples of Data Transformation Services (DTS) custom tasks implemented in Microsoft® Visual C++®.

Example	Description
DTS Example: Adding Properties and Icons in Visual C++	Displays a message when executed. The text of the message is specified with a custom property. For more information about a similar task implemented in Microsoft Visual Basic®, see DTS Example: Adding Properties and Icons in Visual Basic
DTS Example: Including a User Interface in Visual C++	Displays the value of a global variable and allows user to update the value. The global variable name and task description are specified through properties using a custom user interface. For more information about a similar task implemented in Visual Basic, see DTS Example: Including a User Interface in Visual Basic .

DTS Example: Adding Properties and Icons in Visual C++

This example, shown in Microsoft® Visual C++®, displays a message when executed. The text of the message is provided by a property you add. To implement this example, do the following:

1. Create a framework for a custom task using the Active Template Library (ATL) custom task basic template.
2. Add a property for the message text.
3. Add an icon that appears when the task is used in Data Transformation Services (DTS) Designer.
4. Add code to implement the message and the property.
5. Build the project and run the custom task.

Creating the Task Framework

Create a custom task framework using the ATL custom task basic template provided with Microsoft SQL Server™ 2000. Name the component **DTSTskPropIcon** and the task class **GenMessage**. Change the **Type** field in **ATL Object Wizard** from **GenMessage Class** to **Generate Message Task**. For more information about using the basic template, see [Building a Custom Task from the ATL Custom Task Basic Template](#).

Adding the Message Property

Add the **Message** property to the custom task.

To add the Message property

1. On the **ClassView** tab of the **Workspace** window, right-click the **IGenMessage** interface, and then click **Add Property**.
2. In the **Add Property to Interface** dialog box, in the **Property Type** list, select **BSTR**, and then in the **Property Name** box, enter **Message**.
3. Click **Attributes**, and then change the **helpstring** from **property Message** to **Message to be displayed**.

Adding an Icon

Select a suitable icon for the task for which you have an .ico file.

To add an icon

1. On the **File** menu, click **Resources**.
2. In the **Insert Resource** dialog box, under **Resource Type**, select **Icon**, and then click **Import**.
3. In the **Import Resource** dialog box, browse to find the .ico file.

When you select a file, the icon editor is displayed. If you make changes to the icon, you must edit both the 16x16 and 32x32 bit images.

This procedure makes a local copy of the icon file in the project directory whether or not you made changes in the icon editor.

Adding Implementation Code

Add the following code segments to the framework files:

- A local variable to hold the value of the **Message** property
- Code to initialize and release the **Message** property value

- Code to retrieve and save the **Message** property value
- Code to display the message when the task is executed

Adding a Local Variable Declaration

The declaration goes in the private section for the **CGenMessage** class, in file `GenMessage.h`.

Immediately after the lines:

```
BSTR  m_bstrName;  
BSTR  m_bstrDescription;
```

insert the line:

```
BSTR  m_bstrMessage;
```

Initializing and Releasing the Message Property Value

The **Message** property must be initialized to a valid value. This is done in the task class constructor in `GenMessage.cpp`.

At the end of the task class constructor (before the right curly bracket):

```
CGenMessage::CGenMessage()
```

add this line:

```
m_bstrMessage = SysAllocString( OLESTR("") );
```

The allocated string must be released before the custom task is removed from memory. This is done in the class destructor, also in `GenMessage.cpp`.

At the end of the destructor (before the right curly bracket):

```
CGenMessage::~~CGenMessage()
```

add this line:

```
if (m_bstrMessage) SysFreeString(m_bstrMessage);
```

Example

The property value must be retrieved in **get_Message** and saved in **put_Message**. These functions are in file GenMessage.cpp.

Replace the // TODO comment in **CGenMessage::get_Message** with the following code:

```
if (!pVal)
    return E_POINTER;
*pVal = SysAllocString(m_bstrMessage);
if (!*pVal)
    return E_OUTOFMEMORY;
```

Replace the // TODO comment in **CGenMessage::put_Message** with the following code:

```
if (m_bstrMessage)
    SysFreeString(m_bstrMessage);
m_bstrMessage = SysAllocString(newVal);
if (!m_bstrMessage)
    return E_OUTOFMEMORY;
```

Displaying the Message

The message is displayed in the **Execute** function in file GenMessage.cpp. The **Description** property is displayed in the message caption and the **Message** property is displayed in the message text.

Replace the // TODO comment in **CGenMessage::Execute** with the following code:

```
MessageBox( NULL, OLE2T( (LPOLESTR)m_bstrMessage ),
            OLE2T( (LPOLESTR)m_bstrDescription ), MB_ICONINFC
```

Running This Example

To build the project, click **Build DTSTskPropIcon.dll** on the **Build** menu. Refresh the DTS cache, if necessary. For more information about preparing the custom task for execution, see [Implementing and Testing a DTS Custom Task](#).

Open DTS Designer and drag the icon for this task onto the design sheet. When the default property grid is displayed, enter or change the values of the **Description** and **Message** properties. The new value for **Description** will be used for the icon title.

Note Do not change the **Name** property. If you do, DTS Designer will generate an error when it is unable find the task using the original name.

When you execute the package, a message box will appear with the **Description** property as its caption and the **Message** property as its text.

DTS Example: Including a User Interface in Visual C++

This example, shown in Microsoft® Visual C++®, shows how to implement a custom user interface and dialog box. The dialog box displays a global variable, the value of which you update. The global variable name and task description are entered through a custom user interface.

Topic	Description
Creating the Custom Task Framework	Describes how to create a framework for the custom task using the Active Template Library (ATL) custom task templates, enabled for a user interface, and explains how to add custom properties for the global variable name and value.
Implementing the Property Page and Display Dialog Box	Describes how to use the ATL Dialog template to implement a custom user interface for the properties and a dialog box for displaying and updating the global variable.
Implementing the Task Class	Describes the code you need to add to the custom task class.
Implementing the User Interface Class	Describes the code you need to add to the user interface class.
Implementing the Property Page Class	Describes the code you need to add to implement the property page.
Implementing the Display Dialog Class	Describes the code you need to add to implement the display and update dialog box.
Building and Running the DTS Custom Task User Interface Example in Visual C++	Describes what you must do to build and use the Data Transformation Services (DTS) user interface custom

task example.

Creating the Custom Task Framework

To include a user interface in a Data Transformation Services (DTS) custom task, you need to create the custom task framework. If you are displaying and updating a global variable through this user interface, add custom properties for the global variable name and value.

Creating the Framework

Create a custom task framework using the Active Template Library (ATL) custom task template, enabled for a user interface, provided with Microsoft® SQL Server™ 2000. Name the component **DTSTskGVUpdate**, the task class **TaskGVUpdate**, and the user interface **GVUserIF**. Change the **Type** field in **ATL Object Wizard** from **TaskGVUpdate Class** to **Global Variable Update Task** when creating the task class. For more information about using the templates with user interface support, see [Building a Custom Task with User Interface from the ATL Custom Task Templates](#).

You can add an icon to the project resource file that will appear when the task is added to the Data Transformation Services (DTS) Designer design sheet. For more information about adding an icon to a custom task, see [DTS Example: Adding Properties and Icons in Visual C++](#).

Adding Custom Properties

Add properties for the global variable name and value according to the following table.

Property name	Type	Parameters	Description
GblVarName	BSTR	None	Name of the global variable to be displayed and updated.
GblVarValue	BSTR	None	Value of the global variable named by GblVarName .

For more information about adding properties to an ATL custom task project, see [Implementing and Testing a DTS Custom Task](#).

Implementing the Property Page and Display Dialog Box

For the Data Transformation Services (DTS) custom task user interface example, you need to implement a property page, which is the user interface for entering custom task properties, and a dialog box for displaying and updating the global variable value.

Implementing the Property Page UI

You can create a property page framework using the Active Template Library (ATL) Dialog template.

To implement the user interface for the properties page

1. On the **Insert** menu in Microsoft® Visual C++®, click **New ATL Object**.
2. On the **ATL Object Wizard** screen, click **Miscellaneous**, click **Dialog**, and then click **Next**.
3. On the **Names** tab, enter **GVPropPage** for the short name.
The dialog box editor is displayed.
4. Add the following controls to the dialog box, which already includes the **OK** and **Cancel** buttons.

Control Type	ID	Description
Static text	IDC_TASK_NAME	Field for display of the task name.
Edit box	IDC_TASK_DESCR	Field for entry and display of the task description.
Edit box	IDC_GV_NAME	Field for entry and display of

the global variable name.

5. Assign a suitable caption to the dialog box, and optionally add static text fields to label the above fields. If you add static text fields, accept the default **ID** of **IDC_STATIC**.

Example

The dialog box editor will add a script for the property page similar to the following to the resource file DTSTskGVUpdate.rc.

```
IDD_GVPROPPAGE DIALOG DISCARDABLE 0, 0, 266, 113
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_S
CAPTION "Global Variable Update Properties"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "OK",1,81,93,50,14
    PUSHBUTTON "Cancel",2,135,93,50,14
    LTEXT "Task Name:",IDC_STATIC,5,5,83,8
    LTEXT "<task name>",IDC_TASK_NAME,5,16,256,8
    LTEXT "Task Description:",IDC_STATIC,5,31,110,8
    EDITTEXT IDC_TASK_DESCR,5,43,255,12,ES_AUTOHSCF
    LTEXT "Global variable name:",IDC_STATIC,5,63,95,8
    EDITTEXT IDC_GV_NAME,5,74,255,12,ES_AUTOHSCROI
END
```

Implementing Display and Update Dialog Box

You can create a dialog box framework using the ATL dialog template for the global variable display and update.

To implement the display and update dialog box

1. Follow the procedure under **Implementing Properties Page UI** and enter **GVDialo** for the short name.

2. Add the following controls to the dialog box, which already includes **OK** and **Cancel** buttons.

Control Type	ID	Description
Static text	IDC_TASK_DESCR	Field for display of the task description.
Static text	IDC_GV_NAME	Field for display of the global variable name.
Edit box	IDC_GV_VALUE	Field for entry and display of the global variable value.

3. Assign a suitable caption and optionally add static text label fields.

4. Click the **More Styles** tab, and then select the **Center** check box.

Example

The dialog box editor will add a script for the display and update dialog box similar to the following to the resource file DTSTskGVUpdate.rc.

```
IDD_GVDIALOG DIALOG DISCARDABLE 0, 0, 256, 65
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION "Global Variable Update"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "OK",1,77,44,50,14
    PUSHBUTTON "Cancel",2,130,44,50,14
    LTEXT "Task:",IDC_STATIC,5,5,19,8
    LTEXT "<task description>",IDC_TASK_DESCR,29,5,219,8
    LTEXT "Global variable name:",IDC_STATIC,5,17,69,8
    LTEXT "<global variable name>",IDC_GV_NAME,81,17,16
    EDITTEXT IDC_GV_VALUE,5,29,246,12,ES_AUTOHSCRO
END
```

Implementing the Task Class

To implement the task class in the Data Transformation Services (DTS) custom task user interface example, add code to the header file `TaskGVUpdate.h` and the Microsoft® Visual C++® file `TaskGVUpdate.cpp`.

Adding Code to `TaskGVUpdate.h`

In this header file for the task class, add declarations for storing the properties added above:

Immediately after the line in the private section:

```
IUnknown * m_pIUnkDTSCustomTaskUI;
```

insert these lines:

```
// Storage for custom properties.  
BSTR m_bstrGblVarName;  
BSTR m_bstrGblVarValue;
```

Adding Code to `TaskGVUpdate.cpp`

Make these additions to the task class code file:

- An **include** statement for the display and update dialog box header file.
- Code to initialize and release the values of the added properties.
- Code to retrieve and save the values of the added properties.
- Code to retrieve the global variable value, display the dialog box, and update the global variable.

Adding an Include Statement

The **include** statement is necessary so the **Execute** method can access the dialog box class.

Immediately after the line:

```
#include "TaskGVUpdate.h"
```

insert the line:

```
#include "GVDialog.h"
```

Initializing and Releasing the Values of the Added Properties

The **GblVarName** and **GblVarValue** properties must be initialized to valid values. This is done in the task class constructor.

At the end of the task class constructor (before the right curly bracket):

```
CTaskGVUpdate::CTaskGVUpdate()
```

add these lines:

```
    m_bstrGblVarName = SysAllocString( OLESTR("") );  
    m_bstrGblVarValue = SysAllocString( OLESTR("") );
```

The allocated strings must be released before the custom task is removed from memory. This is done in the class destructor.

At the end of the destructor (before the right curly bracket):

```
CTaskGVUpdate::~~CTaskGVUpdate()
```

add these lines:

```
    if (m_bstrGblVarName) SysFreeString(m_bstrGblVarName);  
    if (m_bstrGblVarValue) SysFreeString(m_bstrGblVarValue);
```

Retrieving and Saving the Values of the Added Properties

The property values must be retrieved in the **get_property** and saved in the **put_property** functions.

Example

Replace the // TODO comment in **CTaskGVUpdate::get_GblVarName** with the following code:

```
if (!pVal)
    return E_POINTER;
*pVal = SysAllocString(m_bstrGblVarName);
if (!*pVal)
    return E_OUTOFMEMORY;
```

Replace the // TODO comment in **CTaskGVUpdate::put_GblVarName** with the following code:

```
if (m_bstrGblVarName)
    SysFreeString(m_bstrGblVarName);
m_bstrGblVarName = SysAllocString(newVal);
if (!m_bstrGblVarName)
    return E_OUTOFMEMORY;
```

Replace the // TODO comment in **CTaskGVUpdate::get_GblVarValue** with the following code:

```
if (!pVal)
    return E_POINTER;
*pVal = SysAllocString(m_bstrGblVarValue);
if (!*pVal)
    return E_OUTOFMEMORY;
```

Replace the // TODO comment in **CTaskGVUpdate::put_GblVarValue** with the following code:

```
if (m_bstrGblVarValue)
    SysFreeString(m_bstrGblVarValue);
m_bstrGblVarValue = SysAllocString(newVal);
if (!m_bstrGblVarValue)
    return E_OUTOFMEMORY;
```

Global Variable Display and Update Execute Method

The **Execute** method implements the functionality of the custom task. It does a **QueryInterface** on the package reference to validate it, then performs the following steps:

1. Gets a reference to the GlobalVariables collection.
2. Forms a variant from the target global variable name, then gets a reference to the **GlobalVariable** object. The call can fail only if the **ExplicitGlobalVariables** property of the package is TRUE. Otherwise, the global variable is created if it does not exist.
3. Gets the value of the global variable and converts it to BSTR type so it can be displayed. It sets the **GblVarValue** property to this value.
4. Gets a reference to the custom task interface, then displays the dialog box. It passes the interface pointer to the dialog box so that it can retrieve task class properties.
5. When the dialog box is closed by the user and control returns, **Execute** converts the global variable back to its original type and updates the global variable value.

Example

Replace the // TODO comment in **CTaskGVUpdate::Execute** with the following code:

```
IDTSPackage          * pIDTSPackage;
IDTSGlobalVariables  * pGlobVars;
IDTSGlobalVariable   * pGlobVar;
ITaskGVUpdate        * pTaskGVUpdate;
CGVDIALOG            dlgDialog;
VARIANT              vGVName;
```

```

VARIANT          vGVValue;
VARIANT          vGVBSTR;

// Preset for early return.
*pTaskResult = DTSTaskExecResult_Failure;

// Verify the package object.
if FAILED(hr = pPackage->QueryInterface(IID_IDTSPackage, (void*)0,
    return hr;

// Get global variables collection.
if( FAILED( hr = pIDTSPackage->GetGlobalVariables( &pGlobVar
    return hr;
pIDTSPackage->Release());

// Create variant for global variable name.
VariantInit( &vGVName );
V_VT( &vGVName ) = VT_BSTR;
V_BSTR( &vGVName ) = SysAllocString( m_bstrGblVarName );

// Get named global variable.
hr = pGlobVars->Item( vGVName, &pGlobVar );
pGlobVars->Release();
SysFreeString( V_BSTR( &vGVName ) );

// Can just return status on no global variable. DTS puts out a good r
if( FAILED( hr ) ) return hr;

// Get global variable value.
VariantInit( &vGVValue );
if( FAILED( hr = pGlobVar->GetValue( &vGVValue ) ) )
    return hr;

```

```

// Convert GV value to BSTR.
VariantInit( &vGVBSTR );
if( FAILED( hr = VariantChangeType( &vGVBSTR, &vGVValue,
                                     VARIANT_ALPHABOOL, VT_BSTR ) ) )
    return hr;

// Release the value BSTR if necessary.
if( V_VT( &vGVValue ) == VT_BSTR )
    SysFreeString( V_BSTR( &vGVValue ) );

// Release previous vlaue of property, and save current value.
SysFreeString( m_bstrGblVarValue );
m_bstrGblVarValue = V_BSTR( &vGVBSTR );

// Get user interface for this custom task.
if FAILED(hr = QueryInterface(IID_ITaskGVUpdate, (void **) &p
    return hr;

// Put up dialog box to display global variable value.
dlgDialog.DoModal( NULL, (long)pTaskGVUpdate );
pTaskGVUpdate->Release();

// Convert updated property back to variant of original type (or BSTR).
V_BSTR( &vGVBSTR ) = SysAllocString( m_bstrGblVarValue );
if( FAILED( hr = VariantChangeType( &vGVValue, &vGVBSTR, V
                                     ( ( V_VT( &vGVValue ) == VT_EMPTY ) ?
                                       VT_BSTR : V_VT( &vGVValue ) ) ) ) )
    return hr;
SysFreeString( V_BSTR( &vGVBSTR ) );

// Update global variable value.
if( FAILED( hr = pGlobVar->SetValue( vGVValue ) ) )
    return hr;

```

```
pGlobVar->Release();
```

```
// Release the value BSTR if necessary.
```

```
if( V_VT( &vGVValue ) == VT_BSTR )
```

```
    SysFreeString( V_BSTR( &vGVValue ) );
```

Implementing the User Interface Class

To implement the user interface class in the Data Transformation Services (DTS) custom task user interface example, add code to the Microsoft® Visual C++® file GVUserIF.cpp. No changes are necessary to the user interface class header file.

Add the following to the user interface class code file GVUserIF.cpp:

- An **include** statement for the property page header file
- Code to display the property page from the **New** and **Edit** methods

Adding an Include Statement

The **include** statement is necessary so the **New** and **Edit** methods can access the property page class.

Immediately after the line:

```
#include "GVUserIF.h"
```

insert the line:

```
#include "GVPropPage.h"
```

Displaying the Property Page

This code displays the property page and passes it a reference to the custom task so it can retrieve and update properties.

Example

Replace the entire body of both the **CGVUserIF::New** and **CGVUserIF::Edit** methods with the following code:

```
CGVPropPage dlgProp;
```

```
if(!m_pIDTSTask || !m_pIDTSCustomTask)
    return E_FAIL;
```

```
dlgProp.DoModal( (struct HWND__ *)hwndParent, (long)m_pIDTS
```

```
return NOERROR;
```

Implementing the Property Page Class

To implement the property page class in the Data Transformation Services (DTS) custom task user interface example, you need to add code to the header file `GVPropPage.h`. The ATL dialog template puts all the code for the dialog box in the header file. The corresponding code file `TaskGVUpdate.cpp` contains only include statements.

Add the following to the property page header file `GVPropPage.h`:

- An **include** statement for the component header file and some **define** statements
- Code to initialize controls on the property page with values of custom task properties
- Code to validate and save the values of the task properties
- A declaration for the task class interface pointer

Adding an Include and Define Statement

The component header file is generated by Microsoft® Visual C++® from the `.idl` file and contains definitions of all the interfaces of the project. Here, the definition of the task class interface is needed. The **define** statements are for a buffer length and **MessageBox** caption.

Immediately after the line:

```
#include <atlhost.h>
```

insert these lines:

```
#include "DTSTskGVUpdate.h"
```

```
#define MAX_PROP_LEN 2048
#define GVM_CAPTION_T("Global Variable Monitor Task")
```

Initializing Controls

The **Name**, **Description** and **GblVarName** properties must be retrieved from the task class and set into controls.

Example

At the head of the **OnInitDialog** function (after the left curly bracket), insert the following code:

```
USES_CONVERSION;
BSTR      bstrProperty;

m_pCustTask = (ITaskGVUpdate *)lParam;

// Fetch values for Description, Name and GblVarName properties
m_pCustTask->get_Description( &bstrProperty );
SetDlgItemText(IDC_TASK_DESCR, OLE2T( (LPOLESTR)bstrProperty ));
SysFreeString(bstrProperty);

m_pCustTask->get_Name( &bstrProperty );
SetDlgItemText(IDC_TASK_NAME, OLE2T( (LPOLESTR)bstrProperty ));
SysFreeString(bstrProperty);

m_pCustTask->get_GblVarName( &bstrProperty );
SetDlgItemText(IDC_GV_NAME, OLE2T( (LPOLESTR)bstrProperty ));
SysFreeString(bstrProperty);
```

Validating and Updating Properties

The **Description** and **GblVarName** properties must be validated (to verify a value was entered) and saved back to the task class.

Example

At the head of the **OnOK** function, insert the following code:

```
USES_CONVERSION;
TCHAR      atcProperty[ MAX_PROP_LEN ];

// Get task description and generate error if empty.
if( GetDlgItemText( IDC_TASK_DESCR, atcProperty, MAX_PR
    m_pCustTask->put_Description( T2BSTR( atcProperty ) );
else
{
    MessageBox( _T("Description must not be blank."),
        GVM_CAPTION, MB_ICONEXCLAMATION );
    return 0;
}

// Get global variable name and generate error if empty.
if( GetDlgItemText( IDC_GV_NAME, atcProperty, MAX_PROP
    m_pCustTask->put_GblVarName( T2BSTR( atcProperty ) );
else
{
    MessageBox( _T("Global variable name must be entered."),
        GVM_CAPTION, MB_ICONEXCLAMATION );
    return 0;
}
```

Adding an Interface Pointer Statement

The declaration for the task class interface pointer must be added.

Immediately ahead of the lines:

```
};
```

```
#endif //__GVPROPPAGE_H_
```

insert these lines:

```
private:
```

```
    ITaskGVUpdate    * m_pCustTask;
```

Implementing the Display Dialog Class

To implement the display and update dialog box in the Data Transformation Services (DTS) custom task user interface example, add code to the header file `GVDIALOG.h`. The corresponding code file `TaskGVUpdate.cpp` contains only **include** statements.

Add the following to the display and update dialog header file `GVDIALOG.h`:

- An **include** statement for the component header file and a **define** statement.
- Code to initialize controls on the dialog box with values of custom task properties.
- Code to validate and save the value of the **GblVarValue** property.
- A declaration for the task class interface pointer.

Adding an Include and Define Statement

The definition of the task class interface is needed from the component header file. The **define** is for a buffer length.

Immediately after the line:

```
#include <atlhost.h>
```

insert these lines:

```
#include "DTSTskGVUpdate.h"
```

```
#define MAX_PROP_LEN 2048
```

Initializing Controls

The **Description**, **GblVarName** and **GblVarValue** properties must be retrieved from the task class and set into controls.

Example

At the head of the **OnInitDialog** function, insert the following code:

```
USES_CONVERSION;
BSTR      bstrProperty;

m_pCustTask = (ITaskGVUpdate *)lParam;

m_pCustTask->get_GblVarName( &bstrProperty );
SetDlgItemText(IDC_GV_NAME, OLE2T( (LPOLESTR)bstrPr
SysFreeString(bstrProperty);

m_pCustTask->get_GblVarValue( &bstrProperty );
SetDlgItemText(IDC_GV_VALUE, OLE2T( (LPOLESTR)bstrPr
SysFreeString(bstrProperty);

m_pCustTask->get_Description( &bstrProperty );
SetDlgItemText(IDC_TASK_DESCR, OLE2T( (LPOLESTR)bstr
SysFreeString(bstrProperty);
```

Validating and Updating Properties

The **GblVarValue** property must be saved back to the task class.

Example

At the head of the **OnOK** function, insert the following code:

```
USES_CONVERSION;
TCHAR      atcProperty[ MAX_PROP_LEN ];

// Return updated (?) value of global variable.
```

```
GetDlgItemText( IDC_GV_VALUE, atcProperty, MAX_PROP_I  
m_pCustTask->put_GblVarValue( T2BSTR( atcProperty ) );
```

Adding an Interface Pointer Statement

The declaration for the task class interface pointer must be added.

Immediately ahead of the lines:

```
};
```

```
#endif // __GVDIALOG_H_
```

insert these lines:

```
private:
```

```
    ITaskGVUpdate    * m_pCustTask;
```

Building and Running the DTS Custom Task User Interface Example in Visual C++

To build the Data Transformation Services (DTS) custom task user interface example, click **Build DTSTskGVUpdate.dll** on the **Build** menu. Refresh the Data Transformation Services (DTS) cache, if necessary. For more information about preparing the custom task for execution, see [Implementing and Testing a DTS Custom Task](#).

Open DTS Designer and drag the icon for this task onto the design sheet. When the property page you implemented is displayed, change the values of the **Description** property and enter a value for the global variable name. The **Name** property was made read-only on the property page because to change it in DTS designer causes an error.

When you execute the DTS package, the dialog box you implemented will appear with the value of the global variable. If the global variable did not exist, you will see a blank edit box (DTS will have created the global variable), or you will get an error, depending on whether the **ExplicitGlobalVariables** property is set.

Enter or change the global variable value. If you enter an invalid value (for example, substituting alpha characters for a numeric global variable), the task will fail when you close the dialog box.

Test the update feature by placing two copies of this task in a package and connecting them with an **OnSuccess** precedence constraint.

DTS Programming

Building a DTS Custom Transformation

The Data Transformation Services (DTS) data pump, which is the engine for the Transform Data, Data Driven Query and Parallel Data Pump tasks, reads rows from a source connection, transforms the row data as necessary, and writes rows to a destination connection. The data pump uses separate components called transformations to transform the data. The transformation performs specific conversions for which it was designed and that are made necessary by the source and destination column data types. One or more transformations are always required, even when the row data is simply copied.

Several transformations are supplied with Microsoft® SQL Server™ 2000. For more information, see [DTS Transformations](#) or [Transformation Objects](#). Custom transformations can also be implemented by users and third-party vendors. To implement a custom transformation, you must:

- Use Microsoft Visual C++®. The DTS data pump does not support the interfaces necessary to use components implemented in Microsoft Visual Basic®. Most of the constants, structures and interfaces you will need are defined only in Visual C++ header (.h) files that are supplied with SQL Server 2000 and Visual C++.
- Implement the **IDTSDataPumpTransform** interface. If the transformation is to be used with SQL Server 2000, you must also implement the **IDTSDataPumpTransform2** interface. Other optional custom transformation interfaces can be implemented as well.
- Implement the API functions that COM dynamic-link libraries (DLLs) require. You must modify the registration functions, or a registration script, to add and remove the component category globally unique identifier (GUID) for DTS transformations from the system registry.

This section explains how to implement custom transformations and provides examples.

Topic	Description
-------	-------------

<u>DTS Custom Transformation Fundamentals</u>	Describes DTS transformation infrastructure, interfaces, data structures and registration.
<u>Implementing DTS Custom Transformations</u>	Describes the Active Template Library (ATL) custom transformation template, how to add code and how to test the transformation.
<u>DTS Custom Transformation Examples</u>	Provides examples of DTS custom transformations.

DTS Programming

DTS Custom Transformation Fundamentals

The following topics describe functions, interfaces, data structures and registration issues you must consider when you implement Data Transformation Services (DTS) custom transformations.

Topic	Description
COM DLL Infrastructure	Describes the functions involved with creating, loading and registering the component that all COM DLLs must implement.
IDTSDataPumpTransform Interface	Describes the interface that all custom transformations must implement. IDTSDataPumpTransform supports initialization, schema validation, data transformation and termination.
IDTSDataPumpTransform2 Interface	Describes the interface that custom transformations used with Microsoft® SQL Server™ 2000 must implement. IDTSDataPumpTransform2 supports pre-validation and multiple phases of data transformation.
Column Information Structures in DTS Transformations	Describes the data structures that contain information about the source and destination columns. These data structures are arguments to the methods of IDTSDataPumpTransform and IDTSDataPumpTransform2 .
Registration Requirements for Transformations	Describes the information that must be stored in the system registry that allows DTS to find and run the transformation.

COM DLL Infrastructure

A Data Transformation Services (DTS) custom transformation is implemented as a COM DLL. All COM DLLs must implement several functions that are involved with creating, loading, unloading and registering the component.

Function	Description
DLLMain	Initializes the DLL. DLLMain is called by the operating system when it first loads the DLL.
DLLGetClassObject	Creates a class factory for the transformation and returns a pointer to its interface. COM calls through the interface to create the transformation
DLLCanUnloadNow	Returns a code indicating whether the DLL can be unloaded. It will be unloaded if no other application is using any transformation the DLL contains.
DLLRegisterServer	Inserts information about each transformation into the system registry under the key HKEY_CLASSES_ROOT\CLSID. This registry information specifies the location of the DLL executable file, the transformation ProgID, and the DTS transformation component category.
DLLUnregisterServer	Removes the registry information inserted by DLLRegisterServer .

These functions are generated automatically if you use the Active Template Library (ATL) facility of Microsoft® Visual C++® and the ATL wizards to create the framework for your custom transformation.

For **DLLRegisterServer** and **DLLUnregisterServer**, you will need to either add code to the functions or add entries to the registry script in order to include information about DTS transformations in the registration for the transformation. This is not necessary if you use the custom transformation template supplied with Microsoft SQL Server™ 2000. For more information, see [Registration Requirements for Transformations](#).

IDTSDataPumpTransform Interface

All Data Transformation Services (DTS) custom transformations must implement the **IDTSDataPumpTransform** interface. Custom transformations that implement **IDTSDataPumpTransform** but do not implement **IDTSDataPumpTransform2** can only be used with Microsoft® SQL Server™ version 7.0. Such transformations only support a single phase of execution.

The **IDTSDataPumpTransform** interface has the following elements.

Element	Description
Initialize method	Initializes the transformation for the current transform operation.
AddVariable method	Adds a variable to the transformations execution context.
ValidateSchema method	Validates the schemas that are to be transformed.
Execute method	Executes the transformation for a single row.
OnRowComplete method	Performs any processing that is necessary after each row has been transformed.
OnTransformComplete method	Performs any processing that is necessary after all rows have been transformed.

As required by COM, all these methods must be present, but they can be placeholders returning **NOERROR**.

Initialize Method

Initialize can be used to allocate local storage for the transform operation. The output parameter of **Initialize** is passed to all the other methods so that it can store the state of the particular transform operation. This allows a single instance of the custom transformation to process multiple operations. This is only done when the task that creates the custom transformation passes a reference to a single instance in multiple **IDTSDataPump::AddTransform** calls.

AddVariable Method

AddVariable is always called at least once to add the **DTSErrorRecords** collection so that subsequent methods can report errors. It is also called to add the collections used in Microsoft ActiveX® scripts, such as **DTSGlobalVariables** and **DTSLookups**.

ValidateSchema Method

ValidateSchema is called after the data pump has opened the source and destination rowsets but before any source rows are fetched. It verifies that the meta data of the columns to be transformed is consistent with the needs of the transformation. Data structures defining the source and destination columns are passed to the method. For more information, see [Column Information Structures in DTS Transformations](#).

ValidateSchema verifies that the number and types of the source and destination columns are appropriate. It also determines whether the transformations being performed are consistent with the specified transform flags. For example, if a column for which NULLs are allowed is being copied to a column defined as NOT NULL, **ValidateSchema** indicates an error unless the **DTSTransformFlag_AllowNullChange** is specified. Likewise, it indicates an error if, for example, an **int** column is being copied to a **smallint** column, unless **DTSTransformFlag_AllowDemotion** is specified.

Execute Method

Execute is called to perform the transformation once for each source row. The method returns a code other than **NOERROR** from the function only in the event of a fatal error, which terminates the data pump. **Execute** indicates row data errors or specifies the data driven query by returning an appropriate value from **DTSTransformStatus**. After a successful return from **Execute**, the data pump attempts to insert the row data into, or perform the indicated data driven query on, the destination rowset.

OnRowComplete Method

OnRowComplete is called after the row data is applied to the destination rowset in an insert operation or a data driven query. It is called even when the insert

fails or is not attempted because **Execute** specified the data not be inserted, or when **Execute** itself failed. The primary function of **OnRowComplete** is to release allocations made by **Execute** that need to be retained until after the row data is applied to the destination.

OnTransformComplete Method

OnTransformComplete is called after all the source rows have been processed. It gives the transformation an opportunity to release allocations and perform any other post-processing that might be necessary. It is not called if **IDTSDDataPumpTransform2** is implemented and the transformation supports **DTSTransformPhase_OnPumpComplete** phase, which is called instead. For more information, see [IDTSDDataPumpTransform2 Interface](#).

See Also

[DTSTransformStatus](#)

[IDTSDDataPump::AddTransform](#)

[IDTSDDataPumpTransform::AddVariable](#)

[IDTSDDataPumpTransform::Execute](#)

[IDTSDDataPumpTransform::Initialize](#)

[IDTSDDataPumpTransform::OnRowComplete](#)

[IDTSDDataPumpTransform::OnTransformComplete](#)

[IDTSDDataPumpTransform::ValidateSchema](#)

IDTSDataPumpTransform2 Interface

All Data Transformation Services (DTS) custom transformations that are to be used with Microsoft® SQL Server™ 2000 must implement the **IDTSDataPumpTransform2** interface. Although **IDTSDataPumpTransform2** inherits and implements all the elements of **IDTSDataPumpTransform**, the transformation must still respond to **QueryInterface** for **IDTSDataPumpTransform** as well as for **IDTSDataPumpTransform2**.

The **IDTSDataPumpTransform2** interface has the following elements, in addition to those implemented by **IDTSDataPumpTransform**.

Element	Description
GetTransformServerInfo method	Returns supported phases and other information about the transformation.
PreValidateSchema method	Validates the schemas that are to be transformed at the time a custom transformation is created.
SetExtendedInfo method	Reserved for future use.
ProcessPhase method	Executes a phase of the custom transformation for a single source row.
SetExecuteThreadComplete method	Performs post-processing on a thread prior to executing on another thread.

As required by COM, all these methods must be present, but they can be placeholders returning **NOERROR**.

GetTransformServerInfo Method

GetTransformServerInfo returns a bitmask that defines the phases supported by the custom transformation. It returns a help string that can be displayed in a user interface to explain the function of the custom transformation.

Note The **ProcessPhase** method is not called for a phase specified by **GetTransformServerInfo** unless the value specified for the **TransformPhases** property of the **Transformation2** object also specifies the phase. For DTS

packages built in DTS Designer, specify phases on the **Phases** tab of the **Transformation Options** dialog box.

PreValidateSchema Method

PreValidateSchema is used to provide validation at the time a package is built. It is called from DTS Designer when the custom transformation is created or edited. It can perform part or all of the validation on the source and destination column meta data that **IDTSDataPumpTransform::ValidateSchema** performs. However, there are limitations that may justify deferring some complex validations to **ValidateSchema**. For more information, see [DTS Custom Transformation Example: Format Names](#).

SetExtendedInfo Method

SetExtendedInfo is reserved for future use. Implement it as a placeholder that returns **NOERROR**.

ProcessPhase Method

ProcessPhase is called to perform each phase of the custom transformation. It is responsible for transforming the source column data to the destination columns. It writes any header or trailer rows that are necessary. It handles transformation, insert operations, data driven queries, and batch errors. For more information, see [IDTSDataPumpTransform2::ProcessPhase](#).

SetExecuteThreadComplete Method

SetExecuteThreadComplete is called when the data pump is to switch execution threads. The custom transformation closes any thread-affinitive processes and prepares to reopen them on the new thread.

See Also

[IDTSDataPumpTransform::ValidateSchema](#)

[IDTSDataPumpTransform2::PreValidateSchema](#)

[IDTSDataPumpTransform2::GetTransformServerInfo](#)

[IDTSDataPumpTransform2::SetExecuteThreadComplete](#)

[IDTSDataPumpTransform2::SetExtendedInfo](#)

[TransformPhases Property](#)

Column Information Structures in DTS Transformations

When you build a custom transformation, you must consider the Data Transformation Services (DTS) transformation methods that are used to validate and process the source and destination columns being transformed. The **ValidateSchema**, **Execute**, **OnRowComplete** and **OnTransformComplete** methods of the **IDTSDataPumpTransform** interface and the **ProcessPhase** method of the **IDTSDataPumpTransform2** interface need to access structures that define these columns. This access is provided by pointers to a **DTSTransformColumnInfo** structure for the source columns and another **DTSTransformColumnInfo** for the destination columns, which are passed as parameters to each of these methods.

DTSTransformColumnInfo Important Fields

The following are the important fields in **DTSTransformColumnInfo**.

Field	Description
cColumns	Count of source or destination columns.
rgColumnData	Pointer to an array of DTSColumnData structures. There is one array element for each column.

The remaining fields are associated with binary large object (BLOB) processing, and only need to be referenced if the transformation processes BLOB types.

For the complete definition of this structure, search the include file `dtspump.h` for **DTSTransformColumnInfo**. This file is installed by default in `C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\` during a custom installation of Microsoft® SQL Server™ 2000 client tools.

DTSColumnData Important Fields

The array referenced by **DTSTransformColumnInfo.rgColumnData** contains a **DTSColumnData** structure for each source or destination column. The

following are the important fields in **DTSColumnData**.

Field	Description
pDBCColumnInfo	Pointer to an OLE DB DBCOLUMNINFO structure for the column.
pDBBinding	Pointer to an OLE DB DBBINDING structure for the column.
pvData	Pointer to the data space for the column. Includes fields for the data (or a pointer to the data), data length, and status.

When **IDTSDatapumpTransform::ValidateSchema** is called, both **pDBBinding** and **pvData** are NULL. Thus, the **DBBINDING** structure and the data space are not available in **ValidateSchema**.

For the complete definition of this structure, search the include file **dtspump.h** for **DTSColumnData**.

DBCOLUMNINFO Important Fields

Each **DTSColumnData** structure references an OLE DB **DBCOLUMNINFO** structure, which contains the meta data for the column. The following are the important fields in **DBCOLUMNINFO**.

Field	Description
pwszName	The name of the column.
iOrdinal	The numeric position of the column within the source or destination row.
dwFlags	The sum of flags for meta data attributes (for example, ISNULLABLE, ISROWID, KEYCOLUMN).
ulColumnSize	The width of the column, in characters for wide character types and in bytes for other types.
wType	The data type of the column.

The information in the **DBCOLUMNINFO** structure is not generally modified

by the transformation. For the complete definition of this structure, search the include file OLEDB.h for **DBCOLUMNINFO**.

To find the valid values for **dwFlags**, search OLEDB.h for **DBCOLUMNFLAGS_**. The symbols containing **DBCOLUMNFLAGS_** are defined in enumerations named **DBCOLUMNFLAGSENUMxx**, where **xx** is an optional OLE DB version number.

To find the valid values for **wType**, search OLEDB.h for **DBTYPE_**. The symbols containing **DBTYPE_** are defined in enumerations named **DBTYPExx**, where **xx** is an optional OLE DB version number.

DBBINDING Important Fields

Each **DTSColumnData** structure also references an OLE DB **DBBINDING** structure. A binding associates a single column to the buffer referenced by the **pvData** field of the **DTSColumnData** structure, and it contains information about that buffer. The following are the important fields in **DBBINDING**.

Field	Description
iOrdinal	The numeric position of the column within the source or destination row.
obValue	The offset within the buffer referenced by DTSColumnData.pvData where the data value, or a pointer to the data value, is stored.
obLength	The offset within the buffer referenced by DTSColumnData.pvData where the actual data length, in bytes, is stored.
obStatus	The offset within the buffer referenced by DTSColumnData.pvData where the data status is stored.
dwPart	Flags that specify which parts of the buffer are to be bound to the column. The flags will indicate a combination of data length, status and value.
cbMaxLen	The size of the data area of the buffer, which is the maximum length of the data. For character types, this is usually the width of the column in bytes, plus one character.
wType	The data type of the column.

For the complete definition of this structure, search the include file OLEDB.h for **DBBINDING**.

To find the valid values for the field whose offset is specified by **obStatus**, search OLEDB.h for **DBSTATUS_**. The symbols containing **DBSTATUS_** are defined in enumerations named **DBSTATUSENUMxx**, where **xx** is an optional OLE DB version number.

To find the valid values for **dwPart**, search OLEDB.h for **DBPART_**. These symbols containing **DBPART_** are defined in an enumeration named **DBPARTENUM**.

The valid values for **wType** are the same as for the **DBCOLUMNINFO.wType** field. If **wType** includes the flags **DBTYPE_ARRAY**, **DBTYPE_BYREF** or **DBTYPE_VECTOR**, then the field in **DTSColumnData.pvData** at offset **obValue** contains a pointer to the data, not the data itself.

IDTSDataPumpTransform2::PreValidateSchema Method

In **PreValidateSchema** the column information parameters are **DTSTransformColumnMetadata** structures. The following are the important fields in **DTSTransformColumnMetadata**.

Field	Description
cColumns	Count of source or destination columns.
rgDBCColumnInfo	Pointer to an array of OLE DB DBCOLUMNINFO structures. There is one array element for each column. DBCOLUMNINFO was described above.

The information in the **DTSTransformColumnMetadata** structures is the same information that is available to **ValidateSchema**, packaged differently.

Registration Requirements for DTS Transformations

Data Transformation Services (DTS) custom transformations require entries in their class registration to identify them as DTS transformations. You need to add code to the **DLLRegisterServer** and **DLLUnregisterServer** functions or add script to the registry script (.rgs) file of the custom transformation project so that it can create these entries.

DTS Transformation Registry Entries

The **DLLRegisterServer** function needs to create a set of registry keys under `\HKEY_CLASSES_ROOT\CLSID\` with the following structure:

- *{Class ID for Component.CXFormClass}*
(Default) *transformation description*
 - DTSTransform
 - 1033
DTSTransformDescription *transformation description*
 - Implemented Categories
 - *{GUID for DTS Transformations component category}*
 - InprocHandler32
(Default) ole32.dll
 - InprocServer32
(Default) *path\Component.dll*
Threading Model Both
 - ProgID
(Default) *Component.CXFormClass.version*

- VersionIndependentProgID
(Default) *Component.CTaskClass*

The *transformation description* is the name that appears in the **Create New Transformation** dialog box of DTS Designer. The subkeys under the DTSTransform key provide locale-specific versions of the *transformation description*. You can add a subkey for each locale in which you expect your transformation to be used.

DTS defines a component category for DTS transformations, which is a globally unique identifier (GUID) that is added to the system registry when Microsoft® SQL Server™ 2000 client tools are installed on your system. To make a custom transformation registration visible to DTS Designer, you must provide the **Implemented Categories** key and a subkey that contains this component category.

In addition, **DLLRegisterServer** needs to map the VersionIndependentProgID and ProgID to the class ID by creating these keys directly under \HKEY_CLASSES_ROOT\:

- *Component.CXFormClass*
(Default) *transformation description*
 - CLSID
(Default) {Class ID for Component.CXFormClass}
- *Component.CXFormClass.version*
(Default) *transformation description*
 - CLSID
(Default) {Class ID for Component.CXFormClass}

DLLUnregisterServer removes the registry entries added by **DLLRegisterServer**.

Registry Script File

If you use the Active Template Library (ATL) to create the framework for your custom transformation, it provides a registry script that creates these registry keys under \HKEY_CLASSES_ROOT\CLSID\:

- {*Class ID for Component.CXFormClass*}
 (Default) *CXFormClass Class*
 - InprocServer32
 (Default) *path\Component.dll*
 Threading Model *Both*
 - ProgID
 (Default) *Component.CXFormClass.version*
 - Programmable
 - TypeLib
 (Default) {*Component type library GUID*}
 - VersionIndependentProgID
 (Default) *Component.CTaskClass*

You can add the required DTSTransform, Implemented Categories and InprocHandler32 subkeys by editing the registry script file. There is no need to remove the Programmable and TypeLib keys. For more information, see [Building a Custom Transformation from a Standard ATL Template](#).

DTS Programming

Implementing DTS Custom Transformations

This section describes use of the Active Template Library (ATL) facility of Microsoft® Visual C++® to implement Data Transformation Services (DTS) custom transformations.

Topic	Description
Building a Custom Transformation from a Standard ATL Template	Describes how to create a framework for a custom transformation from a standard ATL template.
Building a Custom Transformation from the ATL Custom Transformation Template	Describes how to create a custom transformation framework from the ATL custom transformation template supplied with Microsoft SQL Server™ 2000.
Implementing and Testing a DTS Custom Transformation	Describes how to add code to a custom transformation framework and explains how to use the Visual C++ debugger.

Building a Custom Transformation from a Standard ATL Template

To build a custom transformation, create a project from a standard Active Template Library (ATL) template, add the interfaces and other elements required by all Data Transformation Services (DTS) transformations, and then add the features of the specific transformation.

This topic explains how to add the elements required by all DTS transformations. You can also use the ATL custom transformation template supplied as a sample with Microsoft® SQL Server™ 2000 to build a custom transformation framework. Even if you plan to use the custom transformation template, it is recommended that you understand the features that were added to create the custom template from the standard object template. For more information, see [Building a Custom Transformation from the ATL Custom Transformation Template](#).

Building a Standard ATL Component

To create a standard ATL component that includes a class for the custom transformation using Microsoft Visual C++® version 6.0, do the following:

To build a standard ATL component

1. On the **File** menu, click **New**, and then click the **Projects** tab.
2. Click **ATL COM AppWizard**, and then enter a project name and location.

For this discussion, assume you entered **DTSTrans** for the project name.
3. Click **Dynamic Link Library (DLL)**, click **Finish**, and in the **New Project Information** dialog box, click **OK**.
4. On the **Insert** menu, click **New ATL Object**, click **Objects**, click

Simple Object, and then click **Next**.

5. On the **Names** tab, enter a short name.

For this discussion, assume you entered **CustomXFm**. The wizard will fill in the other fields. The **COM/Type** field is the name that will appear in the **Create New Transformation** dialog box of DTS Designer. You can change it from the default **CustomXFm Class**.

6. Click the **Attributes** tab, and then do the following:

- Under **Threading Model**, click **Both**.
- Under **Interface**, click **Dual**.
- Under **Aggregation**, click **No**.
- Select the **Support ISupportErrorInfo** check box.

The wizards will create files for the **DTSTrans** component and the **CustomXFm** class and save them to the project location folder specified in Step 1.

Adding Custom Transformation Features

After creating a standard ATL component with **CustomXFm** class files, you need to add custom transformation elements. In this section, all files will have the same names you specified in Step 4 of the previous procedure.

File	Features
CustomXFm.h	Function prototypes and COM map entries for the IDTSDataPumpTransform and IDTSDataPumpTransform2 interfaces
CustomXFm.cpp	Initial function definitions for the IDTSDataPumpTransform and IDTSDataPumpTransform2 interfaces
CustomXFm.rgs	Registry subkeys required for DTS transformations

CustomXFm.h

In this header file, you need to add the following:

- Include statements for other header files
- An entry to the list of interfaces from which the class inherits
- COM Map entries
- Function prototypes

Adding Include Statements

Add these header file include statements:

```
#include <oledb.h>
#include <msdadc.h>
#include <comdef.h>
#include "dtspump.h"
```

immediately preceding the include statement for resource.h:

```
#include "resource.h"    // main symbols
```

Adding Interface List Entry

To the list of interfaces from which class **CCustomXFm** inherits, add this reference to **IDTSDDataPumpTransform2**:

```
public IDTSDDataPumpTransform2,
```

immediately preceding:

```
public ISupportErrorInfo
```

Adding COM Map Entries

Add these COM map entries for **IDTSDDataPumpTransform** and **IDTSDDataPumpTransform2**:

```
COM_INTERFACE_ENTRY(IDTSDDataPumpTransform) // Mus
COM_INTERFACE_ENTRY(IDTSDDataPumpTransform2) // eve
```

Immediately preceding the COM map entry for **IDispatch**:

```
COM_INTERFACE_ENTRY(IDispatch)
```

Adding Function Prototypes

You must provide the function prototypes for the **IDTSDDataPumpTransform** and **IDTSDDataPumpTransform2** interfaces. Immediately after the following lines:

```
// ICustomXFm
public:
```

add these lines of code:

```
// IDTSDDataPumpTransform members
STDMETHOD(Initialize)(THIS_
    DP_IN LPCOLESTR pwzName, // Transform
    DP_IN VARIANT ServerParameters, // Parameters
    DP_OUT LPBYTE *ppvTransformServerData // Trans
);
STDMETHOD(ValidateSchema)(THIS_
    DP_IN LPBYTE pvTransformServerData, // Transfo
    DP_INOUT LPCDTSTransformColumnInfo pSrcColumnInfo,
    DP_INOUT LPCDTSTransformColumnInfo pDestColumnInfo
    DP_IN IDTSDDataConvert *pIDTSDDataConvert, // Point
    DP_IN DTSTransformFlags eTransformFlags // Input F
);
STDMETHOD(AddVariable)(THIS_
```

```

    DP_IN LPBYTE pvTransformServerData,          // Transform
    DP_IN LPCOLESTR pwzName,                    // Variable name
    DP_IN BOOL bGlobal,                         // For ActiveX script
                                                // methods must be qualified by
    DP_IN VARIANT Variable                      // Variable value;
);
STDMETHOD(Execute)(THIS_
    DP_IN LPBYTE pvTransformServerData,        // Transform
    DP_IN LPCDTSTransformColumnInfo pSrcColumnInfo, // Source
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,
    DP_IN IDTSDataConvert *pIDTSDataConvert,   // Pointer
    DP_OUT LPDTSTransformStatus pTransformStatus // Result
) {
    return ProcessPhase(pvTransformServerData
        , pSrcColumnInfo
        , pDestColumnInfo
        , pIDTSDataConvert
        , NULL
        , pTransformStatus
    );
}
STDMETHOD(OnRowComplete)(THIS_
    DP_IN LPBYTE pvTransformServerData,        // Transform
    DP_INOUT LPDTSTransformColumnInfo pSrcColumnInfo, // Source
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,
    DP_IN IDTSDataConvert *pIDTSDataConvert,   // Pointer
    DP_IN DTSTransformStatus eTransformStatus, // Result
    DP_IN HRESULT hrInsert                      // Result of IRow
);
STDMETHOD(OnTransformComplete)(THIS_
    DP_IN LPBYTE pvTransformServerData,        // Transform
    DP_INOUT LPDTSTransformColumnInfo pSrcColumnInfo, // Source
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,

```

```

        DP_IN IDTSDDataConvert *pIDTSDDataConvert        // Point
    );
// IDTSDDataPumpTransform2 members
STDMETHOD(GetTransformServerInfo)(THIS_
    DP_OUT BSTR *pbstrHelpString,                        // Description
    DP_OUT LPDTSTransformPhaseEnum peSupportedPhases    /
);
STDMETHOD(PreValidateSchema)(THIS_
    DP_IN LPCDTSTransformColumnMetadata pSrcMetadata, //
    DP_IN LPCDTSTransformColumnMetadata pDestMetadata, /
    DP_IN DTSTransformFlags eTransformFlags,           // Input F
    DP_IN DTSTransformPhaseEnum ePhases                // Phase(s
);
STDMETHOD(SetExtendedInfo)(THIS_
    DP_IN IUnknown *pUnkExtendedInfo                   // Pointer to
) {
    return NOERROR;
}
STDMETHOD(ProcessPhase)(THIS_
    DP_IN LPBYTE pvTransformServerData,                // Transfo
    DP_IN LPCDTSTransformColumnInfo pSrcColumnInfo,    // :
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,
    DP_IN IDTSDDataConvert *pIDTSDDataConvert,        // Point
    DP_IN LPCDTSTransformPhaseInfo pPhaseInfo,         // Point
    DP_OUT LPDTSTransformStatus peTransformStatus      // Re
);
STDMETHOD(SetExecuteThreadComplete)(THIS)
{
    return NOERROR;
}

```

CustomXFm.cpp

You must provide the initial function definitions for the

IDTSDDataPumpTransform and **IDTSDDataPumpTransform2** interfaces.

Adding Initial Function Definitions

Add these lines of code at the end of the existing file:

```
// IDTSDDataPumpTransform members
STDMETHODIMP CCustomXFm::Initialize(THIS_
    DP_IN LPCOLESTR pwzName,                // Transform
    DP_IN VARIANT ServerParameters,        // Parameters
    DP_OUT LPBYTE *ppvTransformServerData  // Trans
)
{
    return NOERROR;
}

STDMETHODIMP CCustomXFm::ValidateSchema(THIS_
    DP_IN LPBYTE pvTransformServerData,    // Transfo
    DP_INOUT LPCDTSTransformColumnInfo pSrcColumnInfo,
    DP_INOUT LPCDTSTransformColumnInfo pDestColumnInfo
    DP_IN IDTSDDataConvert *pIDTSDDataConvert, // Point
    DP_IN DTSTransformFlags eTransformFlags // Input F
)
{
    return NOERROR;
}

STDMETHODIMP CCustomXFm::AddVariable(THIS_
    DP_IN LPBYTE pvTransformServerData,    // Transfo
    DP_IN LPCOLESTR pwzName,                // Variable n
    DP_IN BOOL bGlobal,                    // For ActiveX scri
                                           // methods must be qualified by
    DP_IN VARIANT Variable                 // Variable value;
)
{
    return NOERROR;
}
```

```

}
STDMETHODIMP CCustomXFm::OnRowComplete(THIS_
    DP_IN LPBYTE pvTransformServerData,          // Transform
    DP_INOUT LPDTSTransformColumnInfo pSrcColumnInfo,
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,
    DP_IN IDTSDataConvert *pIDTSDataConvert,    // Pointer
    DP_IN DTSTransformStatus eTransformStatus,   // Result
    DP_IN HRESULT hrInsert                       // Result of IRow
    )
{
    return NOERROR;
}
STDMETHODIMP CCustomXFm::OnTransformComplete(THIS_
    DP_IN LPBYTE pvTransformServerData,          // Transform
    DP_INOUT LPDTSTransformColumnInfo pSrcColumnInfo,
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,
    DP_IN IDTSDataConvert *pIDTSDataConvert    // Pointer
    )
{
    return NOERROR;
}
// IDTSDataPumpTransform2 members
STDMETHODIMP CCustomXFm::GetTransformServerInfo(THIS_
    DP_OUT BSTR *pbstrHelpString,                // Description
    DP_OUT LPDTSTransformPhaseEnum peSupportedPhases
    )
{
    BSTR bstrHelp = _bstr_t("Helpstring for Custom Transformation Fr

// If help string pointer valid, define help string.
if (pbstrHelpString)
    *pbstrHelpString = bstrHelp;

```

```

// If supported phases pointer valid, define supported phases
if (peSupportedPhases)
    *peSupportedPhases = DTSTransformPhase_Transform;

return NOERROR;
}
STDMETHODIMP CCustomXFm::PreValidateSchema(THIS_
    DP_IN LPCDTSTransformColumnMetadata pSrcMetadata, //
    DP_IN LPCDTSTransformColumnMetadata pDestMetadata, /
    DP_IN DTSTransformFlags eTransformFlags, // Input F
    DP_IN DTSTransformPhaseEnum ePhases // Phase(s
    )
{
    return NOERROR;
}
STDMETHODIMP CCustomXFm::ProcessPhase(THIS_
    DP_IN LPBYTE pvTransformServerData, // Transfor
    DP_IN LPCDTSTransformColumnInfo pSrcColumnInfo, //
    DP_INOUT LPDTSTransformColumnInfo pDestColumnInfo,
    DP_IN IDTSDDataConvert *pIDTSDDataConvert, // Point
    DP_IN LPCDTSTransformPhaseInfo pPhaseInfo, // Point
    DP_OUT LPDTSTransformStatus pTransformStatus // Res
    )
{
    return NOERROR;
}

```

CustomXFm.rgs

This file contains the registry script for the transformation class. The lines to be added define the localizable transformation description and the component category for DTS transformations.

Adding Registry Script

Insert these lines immediately following the definition of the **VersionIndependentProgID** subkey:

```
DTSTransform
{
    '1033'
    {
        val DTSTransformDescription = s 'Custom Transformation'
    }
}
'Implemented Categories'
{
    {10010100-740B-11D0-AE7B-00AA004A34D5}
}
InprocHandler32 = s 'ole32.dll'
```

To verify that the Implemented Categories globally unique identifier (GUID) is correct, look for it in dtspump.h under the definition for **CATID_DTSCustomXform**. You can verify that it is among the subkeys of HKEY_CLASSES_ROOT\Component Categories\ in the registry of a computer on which either SQL Server or SQL Server 2000 client tools have been installed.

Building a Custom Transformation from the ATL Custom Transformation Template

To build a Data Transformation Services (DTS) custom transformation, use the Active Template Library (ATL) custom transformation template. This template, which enables you to build custom transformations more quickly than if you used the ATL standard template, is included in the Microsoft® SQL Server™ 2000 DTS sample programs. For more information, see [DTS Programming Samples](#).

Installing the ATL Custom Transformation Template

To install the ATL custom transformation template, do the following:

1. Copy all the files in the DTSXFormTemplate folder except DTSCuXFm.reg to C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template\ATL\. This location will be different if Microsoft Visual Studio® version 6.0 was not installed to the default location.
2. Double-click DTSCuXFm.reg to run the file.

Building a Custom Transformation Framework from the Template

You can create an ATL component that includes a custom transformation class by using the Microsoft Visual C++® development environment.

To create the ATL component

1. On the **File** menu, click **New**, and then click the **Projects** tab.
2. Click **ATL COM AppWizard**, and then enter a project name and location.

For this discussion, assume **DTSTrans** was entered for the project name.

3. Click **Dynamic Link Library (DLL)**, click **Finish**, and in the **New Project Information** dialog box, click **OK**.
4. On the **Insert** menu, click **New ATL Object**, click **DTS Custom Objects**, click **DTS Transformation**, and then click **Next**.
5. On the **Names** tab, enter a short name.

For this discussion, assume **CustomXFm** was entered. The wizard will fill in the other fields. The **COM/Type** field is the name that will appear in the **Create New Transformation** dialog box of DTS Designer, You can change it from the default **CustomXFm Class**.

6. Click the **Attributes** tab, and then do the following:
 - Under **Threading Model**, click **Both**.
 - Under **Interface**, click **Dual**.
 - Under **Aggregation**, click **No**.
 - Select the **Support ISupportErrorInfo** check box.

The wizards will create files for the **DTSTrans** component and the **CustomXFm** class and save them to the project location folder specified in Step 1.

Building this transformation project from the **Build/Build DTSTrans.dll** menu, before adding any custom code, installs a custom transformation that will appear in the **Create New Transformation** dialog box and can be included in a DTS package. However, this custom transformation will not copy or transform the source columns when the package is run. For more information about building and debugging a custom transformation, see [Implementing and Testing a DTS Custom Transformation](#).

Implementing and Testing a DTS Custom Transformation

To implement and test a Data Transformation Services (DTS) custom transformation, you need to:

- Install the Microsoft® SQL Server® 2000 header and library files on your development computer.
- Build the custom transformation framework.
- Add properties, if necessary, to the custom transformation framework.
- Configure Microsoft Visual C++® to build the project.
- Add custom code to the custom transformation framework.
- Register and optionally unregister the custom transformation.
- Debug the custom transformation.

Installing SQL Server 2000 Header and Library Files

To install the header and library files, you must do a custom installation of SQL Server 2000 or the SQL Server 2000 client tools on the computer on which you develop the custom transformation.

To install header and library files during a custom installation

1. In the **Setup Type** dialog box, click **Custom**.
2. In the **Select Component** dialog box, under **Components**, select the

Development Tools check box.

3. Under **Sub-Components**, select **Headers and Libraries** and **Debugger Interface**.

Building the Custom Transformation Framework

You can add code to a standard Active Template Library (ATL) template or use the custom transformation template included with SQL Server 2000 to build the transformation framework. For more information, see [Building a Custom Transformation from a Standard ATL Template](#) and [Using the ATL Custom Transformation Template](#).

Adding Properties to a Custom Transformation

Your custom transformation may require properties that are not supplied by the custom transformation framework.

To add properties to a custom transformation

1. In the **Workspace** window, right-click the interface for your transformation class, and then click **Add Property**.
2. In the **Add Property to Interface** dialog box, enter the name and type, as well as other requested information.

This procedure adds shells for the **get_property** and **put_property** functions to your project. You must provide the code to implement them.

Configuring Visual C++ to Build the Project

Before you attempt to compile any of the framework files, you need to configure Visual C++ to look for SQL Server 2000 header and library files.

To configure Visual C++ to build the project

1. On the **Tools** menu, click **Options**.

2. In the **Options** dialog box, click the **Directories** tab.
3. In the **Show directories for** list, enter the paths from the following table at the top of the **Directories** list for each entry.

File type	Path
Executable files	C:\Program Files\Microsoft SQL Server\80\Tools\Binn
Include Files	C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Include
Library files	C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Lib
Source files	C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Include

This only needs to be done once after installing SQL Server 2000. The paths will be different if SQL Server 2000 components were installed to other than the default locations.

You also need to define the preprocessor symbol `_ATL_NO_UUIDOF`.

To enter preprocessor symbols

1. On the **Project** menu, click **Settings**.
2. Click the **C/C++** tab, and then in the **Preprocessor definitions** box, enter the preprocessor symbols (comma separated) at the end of the list.

Adding Code to the Framework

You need to add code for the logic specific to your transformation.

All transformations need `IDTSDDataPumpTransform::ValidateSchema`, plus either `IDTSDDataPumpTransform::Execute` or `IDTSDDataPumpTransform2::ProcessPhase`, in order to be functional beyond a

placeholder that returns **NOERROR**. If the transformation is to run in DTS Designer, you also need to provide **IDTSDataPumpTransform2::PreValidateSchema**. For more information, see [IDTSDataPumpTransform Interface](#) and [IDTSDataPumpTransform2 Interface](#).

For more information about coding transformation logic, see [DTS Custom Transformation Example: Copy One Column](#) and [DTS Custom Transformation Example: Format Names](#).

Registering Custom Transformations

When you build the custom transformation project in Visual C++, it registers the transformation as the final step of the build process. If you have enabled DTS component caching, you will need to refresh the cache or DTS Designer will not be able to see the transformation component.

To refresh the cache

1. In SQL Server Enterprise Manager, right-click **Data Transformation Services**, and then click **Properties**.
2. In the **Package Properties** dialog box, click **Refresh Cache**.

If you want to remove a custom transformation from your computer, you must unregister it before deleting the component .dll file.

To unregister a custom transformation

1. From the command prompt, set the path to the folder that contains the transformation component DLL.
2. Enter:
`regsvr32 /u Component.dll`
3. If DTS caching is enabled, refresh the cache.

Debugging Custom Transformations

You can debug a custom transformation by running it from DTS Designer or from a DTS package program (for example, one implemented in Microsoft Visual Basic®). Using DTS Designer may give you more testing options, as **IDTSDataPumpTransform2::PreValidateSchema** is called and a custom user interface can be displayed. In either case, you must specify the executable name and path, along with any parameters the executable requires, on the **Debug** tab of the **Project Settings** dialog box.

If you use DTS Designer, you typically enter C:\WINNT\system32\mmc.exe in the **Executable for debug session** box and /s "c:\Program Files\Microsoft SQL Server\80\Tools\BINN\SQL Server Enterprise Manager.MSC" in the **Program arguments** box.

To determine the correct debugging settings for your computer

1. Find the shortcut used to launch SQL Server Enterprise Manager from the **Start** menu.
2. Extract this information from the **Target** box on the **Shortcut** tab of the **Enterprise Manager Properties** dialog box.

When a transformation commits an access violation or other fatal error, the data pump terminates it and reports that the task using the transformation failed. For example, when an access violation occurs, the message "Access is denied" is displayed. It is recommended you place a breakpoint at the entry to **IDTSDataPumpTransform::OnTransformComplete** or the **OnPumpComplete** code in **IDTSDataPumpTransform2::ProcessPhase**. If this breakpoint is reached unexpectedly before all rows are processed, a likely cause is a transformation fatal error.

DTS Programming

DTS Custom Transformation Examples

This section provides examples of Data Transformation Services (DTS) custom transformations.

Example	Description
DTS Custom Transformation Example: Copy One Column	Takes one source and one destination column, verifies that source and destination are the same simple type, and then copies source to destination.
DTS Custom Transformation Example: Format Names	Takes two source columns and one destination column, verifies that the columns are string types, formats the source columns (LastName and FirstName), and then copies the combined field to destination. Converts ANSI <=> wide character when necessary. Performs certain transform flag validation.

DTS Custom Transformation Example: Copy One Column

The following code example in Microsoft® Visual C++® implements a custom transformation that copies a single source column to a destination column. The source and destination columns must be the same simple type. The transformation verifies that:

- There is exactly one source and one destination column.
- The columns are the same type.
- The columns are not complex types like binary large objects (BLOBs).

To implement this example, use the Active Template Library (ATL) custom transformation template to create the transformation framework. Name the component **DTSCopy** and the transformation class **Copy1Column**. For more information, see [Using the ATL Custom Transformation Template](#).

Add the following code segments to the framework files:

- **CCopy1Column::PreValidateSchema** method
- **CCopy1Column::ValidateSchema** method
- **CCopy1Column::ProcessPhase** method
- Error code definitions

PreValidateSchema

The code for **PreValidateSchema** checks the number of source and destination columns. It also checks that the types match and that they are simple types.

Adding PreValidateSchema Code

Insert the following code immediately ahead of the

```
return NOERROR;
```

statement in **CCopy1Column::PreValidateSchema** in file Copy1Column.cpp:

```
// Validate the count of source and destination columns.
if (pDestMetadata->cColumns != 1 || pSrcMetadata->cColumns != 1
    return DTSCopy_E_WrongNumCols;

// Validate that the destination column type is simple. Remove BYR
const DBCOLUMNINFO* pDestDBCColumnInfo = &(pDestMetad
WORD          wDestType          = (pDestDBCColumnInfo->wTyp

if( wDestType & ( DBTYPE_ARRAY | DBTYPE_VECTOR | DBT
    return DTSCopy_E_NotSimpleType;

// Validate that the source column type is simple.
const DBCOLUMNINFO* pSrcDBCColumnInfo = &(pSrcMetadat
WORD          wSourceType          = (pSrcDBCColumnInfo->wTyp

if( wSourceType & ( DBTYPE_ARRAY | DBTYPE_VECTOR | DE
    return DTSCopy_E_NotSimpleType;

// Source and destination columns must be the same type.
if( wDestType != wSourceType )
    return DTSCopy_E_NotSameType;
```

ValidateSchema

The code for **ValidateSchema** performs the same logic as **PreValidateSchema** in this custom transformation. This will typically be the case in simple transformations that do not have properties needing to be set before validation can occur.

ValidateSchema does not support promotion or demotion between similar column data types (for example, **int** and **smallint**). It also does not reference the transformation flags.

Adding ValidateSchema Code

Insert the following code immediately ahead of the

```
return NOERROR;
```

statement in **CCopy1Column::ValidateSchema** in file Copy1Column.cpp.

```
// Validate the count of source and destination columns.
if (pDestColumnInfo->cColumns != 1 || pSrcColumnInfo->cColumns
    return DTSCopy_E_WrongNumCols;

// Validate that the destination column type is simple. Remove BYR
const DBCOLUMNINFO* pDestDBColumnInfo = pDestColumnInfo
WORD                wDestType        = (pDestDBColumnInfo->wType

if( wDestType & ( DBTYPE_ARRAY | DBTYPE_VECTOR | DBT
    return DTSCopy_E_NotSimpleType;

// Validate that the source column type is simple.
const DBCOLUMNINFO* pSrcDBColumnInfo = pSrcColumnInfo
WORD                wSourceType       = (pSrcDBColumnInfo->wType

if( wSourceType & ( DBTYPE_ARRAY | DBTYPE_VECTOR | DE
    return DTSCopy_E_NotSimpleType;

// Source and destination columns must be the same type.
if( wDestType != wSourceType )
    return DTSCopy_E_NotSameType;
```

ProcessPhase

The code for **ProcessPhase** immediately returns if called for a phase other than **DTSTransformPhase_Transform**, although it should only be called for those phases specified by **PreValidateSchema**.

For the destination column, **ProcessPhase** code gets the buffer size from the binding structure and creates pointers to the buffer fields where the data, length and status are to be stored. For the source column, this code gets the length and status and creates a data pointer. It must check the source type for **DBTYPE_BYREF** to see whether an additional level of indirection is required.

If the source status is **DBSTATUS_S_ISNULL**, **ProcessPhase** sets the destination status to this value. Length and data do not need to be set when the status is set to **DBSTATUS_S_ISNULL**. Otherwise, it calculates the length of data to be moved, which is the shorter of the source actual data length or the size of the destination buffer (reduced by the width of a NULL character, for string types). It copies the data, sets status, and writes a trailing NULL character, if there is room (there will be, for string types).

Adding ProcessPhase Code

Insert the following code immediately ahead of the

```
return NOERROR;
```

statement in **CCopy1Column::ProcessPhase** in file **Copy1Column.cpp**:

```
// Only do something for the Transform phase.
if( pPhaseInfo && !( pPhaseInfo->eCurrentPhase & DTSTransform
    return NOERROR;

// Get destination binding and information structures.
DTSColumnData*   pDTSDestColumnData =  &( pDestColumnI
const DBBINDING*  pDBDestBinding =     pDTSDestColumnD

// Set the destination length to maximum length. Initialize to empty s
ULONG  ulDestMaxLen  = pDBDestBinding->cbMaxLen;
LPBYTE pDestData     = (pDTSDestColumnData->pvData + pDE
```

```

// Pointers to destination length and status buffers
ULONG* pulLength    = (ULONG*)(pDTSDestColumnData->p
ULONG* pulStatus    = (ULONG*)(pDTSDestColumnData->p

// Get source binding and information structures.
DTSColumnData*    pDTSSourceColumnData = &(amp; pSrcColumnI
const DBBINDING*  pDBSourceBinding =    pDTSSourceColun

// Get source type, length and status.
ULONG  wSourceType  = (pDBSourceBinding->wType);
ULONG  ulSourceStatus = *(ULONG*)(pDTSSourceColumnData
ULONG  ulSourceLength = *(ULONG*)(pDTSSourceColumnData
LPBYTE pSourceData;

// Get pointer to source data.
if( wSourceType & DBTYPE_BYREF )
    pSourceData = *(LPBYTE*)(pDTSSourceColumnData->pvData
else
    pSourceData = (pDTSSourceColumnData->pvData + pDBSource

// Copy source to destination if source is not Null.
if( ulSourceStatus != DBSTATUS_S_ISNULL )
{
    // Calculate maximum actual data space (allow room for \0).
    wSourceType &= ~DBTYPE_BYREF;
    ULONG ulMaxDataLen = ulDestMaxLen - ( ( wSourceType == I
        ( ( wSourceType == DBTYPE_STR ) ? 1

// Calculate length of data, then move it and set status.
*pulLength = min( ulSourceLength, ulMaxDataLen );
memcpy( pDestData, pSourceData, *pulLength );
*pulStatus = DBSTATUS_S_OK;

```

```

// Add one or two NULLs if there is room ( for ANSI/Unicode str
if( *pulLength < ulDestMaxLen )
    *( pDestData + *pulLength ) = '\0';
if( *pulLength + 1 < ulDestMaxLen )
    *( pDestData + *pulLength + 1 ) = '\0';
}
else
    *pulStatus = DBSTATUS_S_ISNULL;

```

Error Code Definitions

These error codes are returned by functions in the transformation.

Adding Transformation Error Codes

Insert the following code immediately ahead of the

```
import "ocidl.idl";
```

statement in file Copy1Column.idl.

```

//Error codes for this custom transformation
typedef [helpstring("Error codes generated by the DTSCopy transfo
    DTSCopy_E_WrongNumCols      = 0x80041001,
    DTSCopy_E_NotSimpleType     = 0x80041002,
    DTSCopy_E_NotSameType      = 0x80041003,
} DTSCopyError, *LPDTSCopyError;

```

Building and Testing Copy1Column

For more information about building and testing this project, see [Implementing and Testing a DTS Custom Transformation](#).

DTS Custom Transformation Example: Format Names

The following code example in Microsoft® Visual C++® implements a custom transformation that merges two source columns that are presumed to be a first and last name. It formats them *LastName*, *FirstName* and copies the combined name to a destination column.

The source and destination columns must be string types, but they can be ANSI or wide character strings. If some columns are ANSI and others are wide character, the source columns are converted to the character width of the destination column as they are copied. The destination column is set to NULL if both source columns are NULL.

The transformation verifies there are exactly two source and one destination column, and that they are string types. It validates that if both source columns can contain NULLs, either the destination can contain NULL or the **DTSTransformFlag_AllowNullChange** transform flag has been set. It also verifies either that the destination column is large enough to hold any name that will fit in the source columns or that the **DTSTransformFlag_AllowStringTruncation** transform flag has been set.

Implementing the Format Names Example

To implement this example, use the Active Template Library (ATL) custom transformation template to create the transformation framework. Name the component **DTSStrings** and the transformation class **FormatName**. For more information, see [Building a Custom Transformation from the ATL Custom Transformation Template](#).

Add Custom Properties

After creating the transformation framework, you need to add two properties to the transformation.

Property name	Property type	Description
---------------	---------------	-------------

FirstNameColumn	BSTR	The name of the source column that contains the first name.
RemoveTrailingSpaces	VARIANT_BOOL	A boolean that indicates whether trailing spaces are to be trimmed from the first and last names.

To add properties in Visual C++

1. On the **ClassView** tab of the **Workspace** window, right-click the **IFormatName** interface, and then click **Add Property**.
2. In the **Property Name** box, enter a name, and then in the **Property Type** list, select or enter the type of property you want to add. No parameters are needed.
3. Select the **Get Function** and **Put Function** check boxes, and then click **PropPut**.

Add Custom Code

Add the following code segments to the framework:

- Initializations in the **CFormatName** class constructor
- Declarations of module level variables in the **CFormatName** class
- Overloaded function **RemoveTrailingSpace**
- **CFormatName::PreValidateSchema** method
- **CFormatName::ValidateSchema** method
- **CFormatName::AddVariable** method

- **CFormatName::GetTransformServerInfo** method
- **CFormatName::ProcessPhase** method
- Property **get_** and **put_** functions
- Error code definitions

Initializations in CFormatName Constructor

This code provides initial values for the transformation properties.

Immediately after the following lines in FormatName.h:

```
public:  
    CFormatName()  
    {
```

Add these code lines:

```
        // Initialize the properties  
        m_bstrFirstNameColumn.m_str = NULL;  
        m_vbRemoveTrailingSpaces = FALSE;
```

Declaration of CFormatName Variables

These are the declarations of the internal storage for the properties and other module level variables.

Adding CFormatName Declarations

Immediately after these lines in FormatName.h:

```
    STDMETHODCALLTYPE(SetExecuteThreadComplete)(THIS)  
    {
```

```

    return NOERROR;
}

```

add these lines of code for the private section :

private:

```

// Local variables.
LONG          m_lFirstNameOrd;    // Ordinal of first-name column
BOOL          m_bNullIntoNonNull; // True if can get Null into
ULONG        m_ulSrcLength;      // Combined maximum source length
BOOL          m_bFirstWide;      // First name is wide chars.
BOOL          m_bLastWide;       // Last name is wide chars.
BOOL          m_bDestWide;       // Destination column is wide
LPBYTE        m_pstrBuffer;      // Intermediate buffer.
IDTSErrorRecords* m_pErrorRecords;

// Properties
CComBSTR      m_bstrFirstNameColumn;
VARIANT_BOOL  m_vbRemoveTrailingSpaces;

```

Function RemoveTrailingSpace

This function removes trailing spaces from ANSI and wide character strings.

Adding Code for RemoveTrailingSpace

Immediately ahead of the following line near the end of FormatName.h:

```
#endif // __FORMATNAME_H_
```

add these code lines:

```

// Overloaded function to remove trailing spaces.
inline void RemoveTrailingSpace( LPSTR lpstr, int iLength )
{
    for( ULONG ii = iLength - 1; ii >= 0; ii-- )

```

```

        if( !isspace( lpstr[ ii ] ) && lpstr[ ii ] != '\0')
            break;
        lpstr[ ii + 1 ] = '\0';
    }
inline void RemoveTrailingSpace( LPWSTR lpwstr, int iLength )
{
    for( ULONG ii = iLength - 1; ii >= 0; ii-- )
        if( !isspace( lpwstr[ ii ] ) && lpwstr[ ii ] != L'\0')
            break;
        lpwstr[ ii + 1 ] = L'\0';
}

```

PreValidateSchema

The code for **PreValidateSchema** checks that there are two source columns and one destination column, and checks that each column is an ANSI or wide character string type. Validation of the **FirstNameColumn** property and the transform flags is deferred to the **ValidateSchema** method.

Adding PreValidateSchema Code

Immediately ahead of the

```
return NOERROR;
```

statement in **CFormatName::PreValidateSchema** in file FormatName.cpp, add these code lines:

```
// Validate the count of source and destination columns.
```

```
if (pDestMetadata->cColumns != 1)
    return DTSSStrings_E_NumDestCols;
```

```
if (pSrcMetadata->cColumns != 2)
    return DTSSStrings_E_NumSourceCols;
```

```
// Validate that the destination column is ANSI or Unicode. Remove
```

```

const DBCOLUMNINFO* pDestDBColumnInfo = &(pDestMetadata->DBColumnInfo)
WORD wDestType = (pDestDBColumnInfo->wType);

if( wDestType != DBTYPE_STR && wDestType != DBTYPE_WSTR )
    return DTSSStrings_E_OnlyStringCols;

// Validate that the source columns are ANSI or Unicode. They can't
for (UINT i = 0; i < pSrcMetadata->cColumns; i++)
{
    const DBCOLUMNINFO* pSrcDBColumnInfo = &(pSrcMetadata->DBColumnInfo)
    WORD wSourceType = (pSrcDBColumnInfo->wType);

    if( wSourceType != DBTYPE_STR && wSourceType != DBTYPE_WSTR )
        return DTSSStrings_E_OnlyStringCols;
}

```

ValidateSchema

The code for **ValidateSchema** duplicates the logic of **PreValidateSchema**. In addition, it does the following:

- Checks to see whether a NULL value might be moved to the destination column where NULLs are not allowed and DTSTransformFlag_AllowNullChange is not set.
- Checks to see whether the combined source columns plus the separator exceed the destination column width, and DTSTransformFlag_AllowStringTruncation is not set.
- Verifies that the **FirstNameColumn** property contains the name of one of the source columns.

Adding ValidateSchema Code

Immediately ahead of the

```
return NOERROR;
```

statement in **CFormatName::ValidateSchema** in file FormatName.cpp, add these code lines:

```
// Mark that first name column unknown.
```

```
m_lFirstNameOrd = -1;
```

```
// Validate the count of source and destination columns.
```

```
if (pDestColumnInfo->cColumns != 1)
```

```
    return DTSSStrings_E_NumDestCols;
```

```
if (pSrcColumnInfo->cColumns != 2)
```

```
    return DTSSStrings_E_NumSourceCols;
```

```
// The pointers to the binding structures and the data area are Null in  
const DBCOLUMNINFO* pDestDBCColumnInfo = pDestColumnInfo;
```

```
WORD          wDestType      = (pDestDBCColumnInfo->wType
```

```
ULONG         ulDestLength   = pDestDBCColumnInfo->ulCol
```

```
BOOL          bDestNullable  = ( ( pDestDBCColumnInfo->dwF  
                                ( DBCOLUMNFLAGS_ISNULLABLE
```

```
BOOL          bSrcNullable    = TRUE;
```

```
// This is length of ", ".
```

```
m_ulSrcLength = 2;
```

```
// Validate the destination column is ANSI or Unicode.
```

```
if( wDestType != DBTYPE_STR && wDestType != DBTYPE_WS'
```

```
    return DTSSStrings_E_OnlyStringCols;
```

```
// Make sure column not a BLOB type.
```

```
if( ulDestLength > DTS_DEFAULT_INMEMORY_BLOB_SIZE )
```

```
    return DTSSStrings_E_NoBLOBCols;
```

```

// Validate that the source columns are ANSI or Unicode. They can't
for (UINT i = 0; i < pSrcColumnInfo->cColumns; i++)
{
    const DBCOLUMNINFO* pSrcDBColumnInfo = pSrcColumnIn
WORD          wSrcType = (pSrcDBColumnInfo->wType);
LPCOLESTR     pwzColName = (pSrcDBColumnInfo->psz

// Accumulate total of source column widths.
m_ulSrcLength += pSrcDBColumnInfo->ulColumnSize;

// Accumulate nullability of combined source columns. All colum
bSrcNullable &= ( ( pSrcDBColumnInfo->dwFlags &
                    ( DBCOLUMNFLAGS_ISNULLABLE | DBCOLUM

// Save index if this is specified first name column.
#if defined(_WIN32) && !defined(OLE2ANSI)
    if( !wcscmp( pwzColName, (m_bstrFirstNameColumn.m_str) ) )
#else
    if( !strcmp( pwzColName, (m_bstrFirstNameColumn.m_str) ) )
#endif
    m_lFirstNameOrd = i;

    if( wSrcType != DBTYPE_STR && wSrcType != DBTYPE_WS
        return DTSSStrings_E_OnlyStringCols;

// Make sure column not a BLOB type.
if( m_ulSrcLength > DTS_DEFAULT_INMEMORY_BLOB_SIZ
    return DTSSStrings_E_NoBLOBCols;
}

// Error if first name column is not found.
if( m_lFirstNameOrd < 0 )
    return DTSSStrings_E_NoFirstNameCol;

```

```

// Error if combined source cols might overflow destination, unless s
if( m_ulSrcLength > ulDestLength )
    if( !( eTransformFlags & DTSTransformFlag_AllowStringTrunca
        return DTSSStrings_E_StrTruncPossible;

// Error if combined source cols are nullable while destination is not
if( m_bNullIntoNonNull = ( bSrcNullable && !bDestNullable ) )
    if( !( eTransformFlags & DTSTransformFlag_AllowNullChange )
        return DTSSStrings_E_NullNotNullPossible;

```

AddVariable

The code for **AddVariable** looks for the **DTSErrorRecords** variable and uses it to obtain a pointer, via QueryInterface, to the **IDTSErrorRecords** interface. Other variables are ignored.

Adding AddVariable Code

Immediately ahead of the

```
return NOERROR;
```

statement in **CFormatName::AddVariable** in file FormatName.cpp,

add these code lines:

```

// Dig out the error variable if this is it. Others, ignore.
HRESULT hr = NOERROR;
#ifdef _WIN32 && !defined(OLE2ANSI)
    if (!wcscmp(wzDTSErrorRecords, pwzName)) {
#else
    if (!strcmp(wzDTSErrorRecords, pwzName)) {
#endif
    //m_pErrorRecords->Clear();
    if (!V_DISPATCH(&Variable))

```

```

        return E_POINTER;
    hr = V_DISPATCH(&Variable)->QueryInterface(IID_IDTSError)
    if( hr == NOERROR )
        m_pErrorRecords->Clear();
}

```

GetTransformServerInfo

The code for **GetTransformServerInfo** returns a help string and specifies the phases the custom transformation supports. In addition to **DTSTransformPhase_Transform**, this transformation uses **DTSTransformPhase_PreSourceData** and **DTSTransformPhase_PostSourceData**.

Adding GetTransformServerInfo Code

Replace the body of **CFormatName::GetTransformServerInfo** in file **FormatName.cpp** with the following code:

```

    BSTR bstrHelp = _bstr_t("Format source column names as Last, Fir

// If help string pointer is valid, define help string.
if (pbstrHelpString)
    *pbstrHelpString = bstrHelp;

// If supported phases pointer is valid, define supported phases.
if (peSupportedPhases)
    *peSupportedPhases = DTSTransformPhase_Transform +
        DTSTransformPhase_PreSourceData +
        DTSTransformPhase_PostSourceData;
return NOERROR;

```

ProcessPhase

The code for **ProcessPhase** supports the phases **DTSTransformPhase_PreSourceData**, **DTSTransformPhase_PostSourceData** and

DTSTransformPhase_Transform.

In the DTSTransformPhase_PreSourceData phase, **ProcessPhase** allocates an intermediate buffer and determines which columns are wide character strings. It uses DTSTransformPhase_PostSourceData to release the buffer.

In the DTSTransformPhase_Transform phase, **ProcessPhase** first copies the last name to the intermediate buffer, converting character width and truncating spaces, if necessary. It appends a comma and space, although only if neither last name nor first name is NULL. It appends the first name to the intermediate buffer, converting character width and truncating spaces, if necessary. It then moves the intermediate buffer to the destination, only copying the part of the string that will fit in the destination buffer. It sets the destination to NULL only if both first and last name are NULL.

Adding ProcessPhase Code

Immediately ahead of the

```
return NOERROR;
```

statement in **CFormatName::ProcessPhase** in file FormatName.cpp,

add these code lines:

```
DTSColumnData*    pDTSDestColumnData;  
const DBBINDING*  pDBDestBinding;
```

```
DTSColumnData*    pDTSSourceColumnData;  
const DBBINDING*  pDBSourceBinding;
```

```
// Process the current transform phase.
```

```
switch( pPhaseInfo ? pPhaseInfo->eCurrentPhase : DTSTransformP  
{
```

```
// Delete intermediate buffer.
```

```
case DTSTransformPhase_PostSourceData:
```

```
delete( m_pstrBuffer );
```

```
break;
```

```
// Allocate intermediate buffer and gather column widths.
```

```
case DTSTransformPhase_PreSourceData:
```

```
// Get destination char width.
```

```
pDTSDestColumnData = &( pDestColumnInfo->rgColumnDa
```

```
pDBDestBinding = pDTSDestColumnData->pDBBinding;
```

```
m_bDestWide = ( pDBDestBinding->wType == DBTYPE
```

```
// Get last name char width.
```

```
pDTSSourceColumnData = &( pSrcColumnInfo->rgColumnDa
```

```
pDBSourceBinding = pDTSSourceColumnData->pDBBindir
```

```
m_bLastWide = ( ( pDBSourceBinding->wType & ~DBT
```

```
// Get first name char width.
```

```
pDTSSourceColumnData = &( pSrcColumnInfo->rgColumnDa
```

```
pDBSourceBinding = pDTSSourceColumnData->pDBBindir
```

```
m_bFirstWide = ( ( pDBSourceBinding->wType & ~DBT
```

```
// Allocate intermediate buffer w/ space for null term.
```

```
m_pstrBuffer = (LPBYTE)new char[ ( m_ulSrcLength + 1
```

```
break;
```

```
case DTSTransformPhase_Transform:
```

```
USES_CONVERSION;
```

```
// Get last name info structures.
```

```
pDTSSourceColumnData = &( pSrcColumnInfo->rgColumnDa
```

```
pDBSourceBinding = pDTSSourceColumnData->pDBBindir
```

```

// Get last name status, data length, and data ptr.
ULONG  ulLastNStatus  = *(ULONG *) ( pDTSSourceColumnI
ULONG  ulSourceLength = *(ULONG *) ( pDTSSourceColumn
LPBYTE pSourceString = ( pDBSourceBinding->wType & DB
                        *(LPBYTE *) (pDTSSourceColumnData->pvDa
                        (LPBYTE) (pDTSSourceColumnData->pvData -

// If last name not null, move to interm buffer.
if( ulLastNStatus != DBSTATUS_S_ISNULL )
    if( m_bDestWide )
    {
        // Move to buffer, and convert to wide if necessary.
        if( m_bLastWide )
            wcscpy( (LPWSTR)m_pstrBuffer, (LPCWSTR)pSourceSt
        else
            wcscpy( (LPWSTR)m_pstrBuffer, (LPCWSTR)A2W( LI

        // Remove trailing spaces, if specified.
        if( m_vbRemoveTrailingSpaces )
            RemoveTrailingSpace( (LPWSTR)m_pstrBuffer,
                ( m_bLastWide ? ulSourceLength / 2 : ulSou
    }
else
    {
        // Move to buffer, and convert to ANSI if necessary.
        if( m_bLastWide )
            strcpy( (LPSTR)m_pstrBuffer, (LPCSTR)W2A( (LPCWS
        else
            strcpy( (LPSTR)m_pstrBuffer, (LPCSTR)pSourceString );

        // Remove trailing spaces, if specified.
        if( m_vbRemoveTrailingSpaces )

```

```

        RemoveTrailingSpace( (LPSTR)m_pstrBuffer,
                            ( m_bLastWide ? ulSourceLength / 2 : ulSou
    }

// Otherwise put a null terminator. Works for both ANSI and wide
else
    *(LPWSTR)m_pstrBuffer = L'\0';

// Get first name length, status and data ptr.
pDTSSourceColumnData = &( pSrcColumnInfo->rgColumnDa
pDBSourceBinding      = pDTSSourceColumnData->pDBBindir
ulSourceLength        = *(ULONG*)( pDTSSourceColumnData-
pSourceString         = ( pDBSourceBinding->wType & DBTYPE
                        *(LPBYTE*)(pDTSSourceColumnData->pvDa
                        (LPBYTE)(pDTSSourceColumnData->pvData -

ULONG ulFirstNStatus = *(ULONG*)( pDTSSourceColumnD

// If first name not null, append it to interm buffer.
if( ulFirstNStatus != DBSTATUS_S_ISNULL )
{
    // If neither name null, append ", ".
    if( ulLastNStatus != DBSTATUS_S_ISNULL )
        if( m_bDestWide )
            wcscat( (LPWSTR)m_pstrBuffer, L", " );
        else
            strcat( (LPSTR)m_pstrBuffer, ", " );

    // Now append the first name.
    if( m_bDestWide )
    {
        // Move to buffer, and convert to wide if necessary.
        if( m_bFirstWide )

```

```

        wcscat( (LPWSTR)m_pstrBuffer, (LPCWSTR)pSourceSt
else
        wcscat( (LPWSTR)m_pstrBuffer, (LPCWSTR)A2W( (LP

// Remove trailing spaces, if specified.
if( m_vbRemoveTrailingSpaces )
    RemoveTrailingSpace( (LPWSTR)m_pstrBuffer, wcslen(
}
else
{
    // Move to buffer, and convert to ANSI if necessary.
    if( m_bFirstWide )
        strcat( (LPSTR)m_pstrBuffer, (LPCSTR)W2A( (LPCWSTR)
    else
        strcat( (LPSTR)m_pstrBuffer, (LPCSTR)pSourceString );

// Remove trailing spaces, if specified.
if( m_vbRemoveTrailingSpaces )
    RemoveTrailingSpace( (LPSTR)m_pstrBuffer, strlen( (LP
}
}

// Destination information structures.
pDTSDestColumnData    = &( pDestColumnInfo->rgColumnDa
pDBDestBinding        = pDTSDestColumnData->pDBBinding;

// Destination buffer size, ptrs to data, length, and status.
ULONG  ulDestLength   = pDBDestBinding->cbMaxLen;
LPBYTE pDestString    = (LPBYTE)(pDTSDestColumnData->]
ULONG* ulLength       = (ULONG*)( pDTSDestColumnData->
ULONG* ulStatus       = (ULONG*)( pDTSDestColumnData->

// If both first and last name Null, set destination to Null.

```

```

if( ulLastNStatus == DBSTATUS_S_ISNULL && ulFirstNStatus
    *ulStatus = DBSTATUS_S_ISNULL;

// Otherwise move data to destination buffer, write length and stat
else
{
    // Calculate length of data (in bytes) to be moved.
    *ulLength = ( m_bDestWide ?
        min( ulDestLength - 2, wcslen( (LPCWSTR)m_pstrE
        min( ulDestLength - 1, strlen( (LPCSTR)m_pstrBuff

// Set good status and move data.
*ulStatus = DBSTATUS_S_OK;
memcpy( pDestString, m_pstrBuffer, *ulLength );

// Null terminate the string.
if( m_bDestWide )
    *(LPWSTR)( pDestString + *ulLength ) = L'\0';
else
    *(LPSTR)( pDestString + *ulLength ) = '\0';
}
break;
}
}

```

Property `get_` and `put_` Functions

The property `get_` functions return the property value through their pointer parameter after verifying the pointer is non-null. The property `put_` functions save the property value, or a pointer to the value, in local storage.

You need to add the code that does this to the `get_` and `put_` functions of the **FirstNameColumn** and **RemoveTrailingSpaces** properties. In each case, replace the comment

```
// TODO: Add your implementation code here
```

with the appropriate code. The **get_** and **put_** functions were added by Visual C++ when the properties were added to the project. They are located near the end of FormatName.cpp.

CFormatName::get_FirstNameColumn

Replace the comment in **CFormatName::get_FirstNameColumn** with the following code:

```
if( !pVal )  
    return E_POINTER;  
*pVal = m_bstrFirstNameColumn.Copy( );
```

CFormatName::put_FirstNameColumn

Replace the comment in **CFormatName::put_FirstNameColumn** with the following code:

```
SysFreeString( m_bstrFirstNameColumn.m_str );  
m_bstrFirstNameColumn.m_str = SysAllocString( newVal );
```

CFormatName::get_RemoveTrailingSpaces

Replace the comment in **CFormatName::get_RemoveTrailingSpaces** with the following code:

```
if( !pVal )  
    return E_POINTER;  
*pVal = m_vbRemoveTrailingSpaces;
```

CFormatName::put_RemoveTrailingSpaces

Replace the comment in **CFormatName::put_RemoveTrailingSpaces** with the following code:

```
m_vbRemoveTrailingSpaces = newVal;
```

Error Code Definitions

These error codes are returned by methods in the transformation.

Adding Transformation Error Codes

Immediately following the

```
import "ocidl.idl";
```

statement in file FormatName.idl,

add these code lines:

```
//Error codes for this custom transformation
typedef [helpstring("Error codes generated by the DTSSStrings transf
    DTSSStrings_E_NumDestCols          = 0x80041001,
    DTSSStrings_E_NumSourceCols        = 0x80041002,
    DTSSStrings_E_OnlyStringCols       = 0x80041003,
    DTSSStrings_E_NoFirstNameCol        = 0x80041004,
    DTSSStrings_E_StrTruncPossible      = 0x80041005,
    DTSSStrings_E_NullNotNullPossible  = 0x80041006,
    DTSSStrings_E_NoBLOBCols           = 0x80041007,
} DTSSStringsError, *LPDTSSStringsError;
```

Building and Testing FormatName

For more information about building and testing this project, see [Implementing and Testing a DTS Custom Transformation](#).

DTS Programming

DTS Scripting Reference

This section documents the objects and collections, and their properties and methods, that are provided by the Data Transformation Services (DTS) data pump for the scripts of Microsoft® ActiveX® Script transformations. They can also be referenced in custom transformations implemented in Microsoft Visual C++® and the C language. The data pump is the engine for the Transform Data task, the Data Driven Query task, and the Parallel Data Pump task.

These objects and collections can generally not be used in the scripts associated with the ActiveX Script task or with DTS package steps. Moreover, the name you use within an ActiveX Script transformation is generally different from the object name. This table specifies the names to be used within scripts, the types of scripts in which they are valid, and a reference to the underlying object.

Scripting Name	Validity	Reference
DTSErrorRecords	ActiveX Script transformations	DTSErrorRecords Collection
DTSGlobalVariables	All DTS ActiveX scripts	GlobalVariables Collection
DTSLookups	ActiveX Script transformations	DTSDataPumpLookups Collection
DTSPackageLog	ActiveX Script tasks	PackageLog Object
DTSSource DTSDestination	ActiveX Script transformations	DTSDataPumpColumns Collection
DTSTransformPhaseInfo	ActiveX Script transformations	DTSTransformPhaseInfo Object

To reference any other object in the DTS object model hierarchy from an ActiveX script, use **DTSGlobalVariables.Parent** to return a reference to the **Package2** object, from which you can reference any other object in the hierarchy. For example, to cause a step named DTSStep_DTSPackageLogTask_1 to execute again after it has already completed execution, set:

```
DTSGlobalVariables.Parent.Steps("DTSStep_DTSCBulkInsertTask_1").  
    DTSStepExecStat_Waiting
```

See Also

[Package2 Object](#)

DTS Programming

Scripting Objects

This section documents the Data Transformation Services (DTS) objects exposed by the DTS data pump for the scripts of Microsoft® ActiveX® Script transformations. They can also be referenced in custom transformations implemented in Microsoft Visual C++® and the C language.

Topic	Description
DTSDataPumpColumn Object	Provides access to a column for a transformation or ActiveX script.
DTSDataPumpColumn2 Object	Extends the functionality of the DTSDataPumpColumn object.
DTSDataPumpLookup Object	Specifies a named, parameterized query string for a transformation or ActiveX script.
DTSTransformPhaseInfo Object	Makes status information available to a transformation or ActiveX script.

DTS Programming

DTSDataPumpColumn Object

The **DTSDataPumpColumn** object provides access to a column value and its meta data to a Microsoft® ActiveX® script.

Properties

ActualSize Property	OriginalValue Property
Attributes Property	Precision Property
DefinedSize Property	Type Property
Name Property	UnderlyingValue Property
NumericScale Property	Value Property

Methods

AppendChunk Method	GetChunk Method
------------------------------------	---------------------------------

Remarks

The **DTSDataPumpColumn** object belongs to the **DTSSource** and **DTSDestination** collections. The **DTSDataPumpColumn** object is identical to the **ADO.Field** interface.

The **DTSDataPumpColumn** object is compatible with Microsoft® SQL Server™ version 7.0. For more information about an extended version of this object, see [DTSDataPumpColumn2 Object](#).

Reference the **DTSDataPumpColumn** object from within ActiveX script transformations by referencing an element of the **DTSSource** or **DTSDestination** collections. If you must remain compatible with SQL Server 7.0, use only the properties specified above.

See Also

[Column Object](#)

[DTSDatapumpColumns Collection](#)

DTS Programming

DTSDataPumpColumn2 Object

The **DTSDataPumpColumn2** object provides access to a column value and its meta data to a Microsoft® ActiveX® script.

Extended Properties

Status Property

Remarks

The **DTSDataPumpColumn2** object extends the functionality of the **DTSDataPumpColumn** object and inherits the properties and methods of that object. In addition, the **Status** property indicates whether the data value is to be used as the value of the column and whether the data pump was able to get or set the value.

For more information about when to use the **DTSDataPumpColumn** object instead of the **DTSDataPumpColumn2** object, see [Extended DTS Objects](#).

Reference the **DTSDataPumpColumn2** object from within ActiveX script transformations by referencing an element of the **DTSSource** or **DTSDestination** collections.

See Also

[Column Object](#)

[DTSDataPumpColumn Object](#)

[DTSDataPumpColumns Collection](#)

DTS Programming

DTSDataPumpLookup Object

The **DTSDataPumpLookup** object provides information about columns in a Data Transformation Services (DTS) lookup query to a Microsoft® ActiveX® script. The **DTSDataPumpLookup** object belongs to the **DTSDataPumpLookups** collection.

Properties

LastRowCount Property	Name Property
---------------------------------------	-------------------------------

Methods

AddToCache Method	RemoveFromCache Method
Execute Method	

Remarks

DTSDataPumpLookup returns a variant or an array of variants (if multivalued) corresponding to the sequence of columns in the single output row resulting from the execution of a query. If multiple rows are returned, only the value of the first row is returned in the output variant. An application can call **LastRowCount** to assert that only one row was returned.

See Also

[DTSDataPumpLookups Collection](#)

[Lookup Object](#)

DTS Programming

DTSTransformPhaseInfo Object

The **DTSTransformPhaseInfo** object makes status information available to Data Transformation Services (DTS) transformations and Microsoft® ActiveX® Script transformations.

Properties

CurrentPhase Property	ErrorCode Property
CurrentSourceRow Property	ErrorRows Property
DestinationRowsComplete Property	TransformStatus Property

Remarks

The following information is available from the **DTSTransformPhaseInfo** object:

- Current source row being processed; first row is row 1.
- Destination rows inserted or Data driven queries executed.
- Total number of error rows encountered.
- Error code of operation preceding current phase.
- Transform status of most recently completed transformation.
- Current transformation phase.

The **DTSTransformPhaseInfo** object is available within ActiveX Script transformations using the same name, **DTSTransformPhaseInfo**.

See Also

[Adding DTS Transformations](#)

DTS Programming

Scripting Collections

Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) scripting collections contain groups of related data pump scripting objects. They can be used from custom transformations or Microsoft ActiveX® script transformations.

Topic	Description
DTSDataPumpColumns Collection	Contains descriptions of source and destination columns for ActiveX script transformations.
DTSDataPumpLookups Collection	Specifies named, parameterized query strings for a transformation or ActiveX script.
DTSErrorRecords Collection	Details errors that a custom transformation or ActiveX script transformation has added to data pump error stack.

DTS Programming

DTSDataPumpColumns Collection

The **DTSDataPumpColumns** collection contains groups of columns that provide source and destination column information to a Microsoft® ActiveX® Script transformation or a custom transformation.

Properties

[Count Property](#)

Methods

[Item Method](#)

Remarks

Reference the **DTSDataPumpColumns** collection from within transformation ActiveX scripts by referencing the **DTSSource** or **DTSDestination** collections.

See Also

[Columns Collection](#)

[DTSDataPumpColumn Object](#)

DTS Programming

DTSDataPumpLookups Collection

The **DTSDataPumpLookups** collection contains **DTSDataPumpLookup** objects that provide Data Transformation Services (DTS) lookup column information to a Microsoft® ActiveX® script.

Properties

[Count Property](#)

Methods

[Item Method](#)

Remarks

Reference the **DTSDataPumpLookups** collection from within ActiveX script transformations by referencing the **DTSLookups** collection.

See Also

[Lookups Collection](#)

[DTSDataPumpLookup Object](#)

DTS Programming

DTSErrorRecords Collection

The **DTSErrorRecords** collection allows an application to append error records to the OLE DB **IErrorInfo** interface of the current thread. This information can be provided by a Microsoft® ActiveX® script.

Methods

Add Method	Clear Method
----------------------------	------------------------------

See Also

[Handling DTS Events and Errors](#)

[IDTSDataPumpErrorSink](#)

DTS Programming

Scripting Properties

This section defines the properties of the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) scripting objects and collections. With these properties, you can retrieve and set the attributes of objects within Microsoft ActiveX® scripts.

DTS Programming

ActualSize Property

The **ActualSize** property returns the actual size of a column value for the current row.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.ActualSize

Part	Description
<i>Object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_ActualSize(long *pl);
```

Remarks

The **ActualSize** property may be less than the **DefinedSize** of the column.

See Also

[DefinedSize Property](#)

Type Property

DTS Programming

Attributes Property

The **Attributes** property returns one or more characteristics of a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.Attributes

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_Attributes(long *pl);
```

Remarks

For more information about the valid values for the **Attributes** property, search for **DBCOLUMNFLAGS_** in include file OLEDB.h. This default location for this file is c:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\if, during a Custom installation of Microsoft SQL Server, Development Tools/Headers and Libraries was selected from the **Select Components** dialog box.

See Also

[Flags Property](#)

[Type Property](#)

DTS Programming

Count Property

The **Count** property specifies the number of items in a scripting collection.

Applies To

DTSDataPumpColumns Collection	DTSDataPumpLookups Collection
---	---

Syntax

object.Count

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCount(long *pRetVal);
```

See Also

[Item Method](#)

DTS Programming

CurrentPhase Property

The **CurrentPhase** property specifies the current transformation phase.

Applies To

[DTSTransformPhaseInfo Object](#)

Syntax

object.**CurrentPhase**

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformPhaseInfo object

Data Type

[DTSTransformPhaseEnum](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_CurrentPhase(LPDTSTransformPhaseEnum *pRetVal);
```

Remarks

The **DTSTransformPhaseInfo** object is referenced within a transformation script or custom transformation.

See Also

[Adding DTS Transformations](#)

[DTSTransformScriptProperties2 Object](#)

[Transformation2 Object](#)

DTS Programming

CurrentSourceRow Property

The **CurrentSourceRow** property specifies the current source row being processed by a Transform Data task, Data Driven Query task, or Parallel Data Pump task.

Applies To

[DTSTransformPhaseInfo Object](#)

Syntax

object.**CurrentSourceRow**

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformPhaseInfo object

Data Type

Variant/vt_decimal

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_CurrentSourceRow(VARIANT *pRetVal);
```

Remarks

The **DTSTransformPhaseInfo** object is referenced within a transformation script or custom transformation.

Some scripting languages, for example Microsoft® Visual Basic® Scripting Edition (VBScript), do not support the **vt_decimal** data type of the

CurrentSourceRow property. In VBScript, convert **CurrentSourceRow** to Long before using it. For example, use the following code to assign **CurrentSourceRow** to a global variable:

```
DTSGlobalVariables("GV1") = CLng(DTSTransformPhaseInfo.Curren
```

See Also

[Data Driven Query Task](#)

[DestinationRowsComplete Property](#)

[DTSTransformScriptProperties2 Object](#)

[ErrorRows Property](#)

[ParallelDataPumpTask Object](#)

[Transform Data Task](#)

DTS Programming

DefinedSize Property

The **DefinedSize** property specifies the maximum size of a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.**DefinedSize**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_DefinedSize(long *pl);
```

Remarks

The **ActualSize** property specifies the size of the data in the current row of the column.

See Also

[ActualSize Property](#)

Type Property

DTS Programming

DestinationRowsComplete Property

The **DestinationRowsComplete** property specifies the number of destination rows inserted or data-driven queries executed so far for the current rowset by a Transform Data task, Data Driven Query task, or Parallel Data Pump task.

Applies To

[DTSTransformPhaseInfo Object](#)

Syntax

object.**DestinationRowsComplete**

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformPhaseInfo object

Data Type

Variant/vt_decimal

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_DestinationRowsComplete(VARIANT *pRetVal);
```

Remarks

The **DTSTransformPhaseInfo** object is referenced within a transformation script or custom transformation.

Some scripting languages, for example Microsoft® Visual Basic® Scripting Edition (VBScript), do not support the **vt_decimal** data type of the

DestinationRowsComplete property. In VBScript, you must convert **DestinationRowsComplete** to Long before using it. For example, use the following code to compare **DestinationRowsComplete** to a global variable in VBScript:

```
If DTSGlobalVariables("GV1") <> CLng(DTSTransformPhaseInfo.De
```

DestinationRowsComplete is incremented after the Transform and OnTransformFailure phases, but prior to the OnInsertSuccess phase. It is not incremented for this row if the OnInsertFailure phase occurs.

See Also

[CurrentSourceRow Property](#)

[Data Driven Query Task](#)

[DTSTransformScriptProperties2 Object](#)

[ErrorRows Property](#)

[ParallelDataPumpTask Object](#)

[Transform Data Task](#)

DTS Programming

ErrorRows Property

The **ErrorRows** property specifies the number of error rows encountered for the current rowset by a transformation in a Transform Data task, Data Driven Query task, or Parallel Data Pump task.

Applies To

[DTSTransformPhaseInfo Object](#)

Syntax

object.**ErrorRows**

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformPhaseInfo object

Data Type

Variant/vt_decimal

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_ErrorRows(VARIANT *pRetVal);
```

Remarks

The **DTSTransformPhaseInfo** object is referenced within a Microsoft® ActiveX® Script transformation or custom transformation.

Some scripting languages, for example Microsoft Visual Basic® Scripting Edition (VBScript), do not support the **vt_decimal** data type of the **ErrorRows**

property. In VBScript, you should convert **ErrorRows** to Long before using it. For example, use the following code to compare **ErrorRows** to a global variable in VBScript:

```
If DTSGlobalVariables("GV1") > CLng(DTSTransformPhaseInfo.Errc
```

ErrorRows includes the current row if in the OnTransformFailure or OnInsertFailure phases.

See Also

[CurrentSourceRow Property](#)

[Data Driven Query Task](#)

[DestinationRowsComplete Property](#)

[DTSTransformScriptProperties2 Object](#)

[ParallelDataPumpTask Object](#)

[Transform Data Task](#)

DTS Programming

LastRowCount Property

The **LastRowCount** property returns the number of rows returned during the last operation of this lookup.

Applies To

[DTSDataPumpLookup Object](#)

Syntax

object.LastRowCount

Part	Description
<i>object</i>	Expression that evaluates to a DTSDataPumpLookup object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_LastRowCount(LONG *pRetVal);
```

Remarks

If multiple rows are retrieved by the lookup, only the fields of the first row are returned by the **Execute** method. Use **LastRowCount** to determine whether one or more rows were retrieved.

See Also

[Execute Method](#)

DTS Programming

Name Property

The **Name** property specifies the name of a scripting object.

Applies To

DTSDataPumpColumn Object	DTSDataPumpLookup Object
DTSDataPumpColumn2 Object	

Syntax

object.Name

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_Name(*pbstr);
```

DTS Programming

NumericScale Property

The **NumericScale** property specifies the scale for numeric values in a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.**NumericScale** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Byte

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_NumericScale(*pbNumericScale);
```

See Also

[Precision Property](#)

[Type Property](#)

DTS Programming

OriginalValue Property

The **OriginalValue** property specifies the value of a column before it was modified.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.**OriginalValue**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Variant

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_OriginalValue(*pvar);
```

Remarks

For source columns, this property is the same as the current value because the source values cannot be modified. For destination columns, this is always empty because the original value of a destination column (prior to transformation) is not set.

See Also

[Underlying Value Property](#)

[Value Property](#)

DTS Programming

Precision Property

The **Precision** property specifies the precision of numeric values in a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.Precision

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Byte

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_Precision( BYTE *pbPrecision);
```

See Also

[NumericScale Property](#)

[Type Property](#)

DTS Programming

Status Property

The **Status** property indicates whether the data value or some other value, such as NULL, is to be used as the value of the column. It may also indicate whether the data pump was able to get or set the value.

Applies To

[DTSDataPumpColumn2 Object](#)

Syntax

dpcolumn.**Precision**

Part	Description
<i>dpcolumn</i>	Expression that evaluates to a DTSDataPumpColumn2 object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_Status( long *pl );
```

Remarks

The values for the **Status** property are from the OLE DB DBSTATUS enumeration. See the definition for DBSTATUSENUM in include file OLEDB.h. This file is installed by default in C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Include\ during a custom installation of Microsoft®

SQL Server™ if Development Tools/Headers and Libraries was selected from the **Select Components** dialog box.

See Also

[Attributes Property](#)

[Type Property](#)

DTS Programming

TransformStatus Property

The **TransformStatus** property specifies the status of the most recently completed transformation for the current row in a Transform Data task, Data Driven Query task, or Parallel Data Pump task.

Applies To

[DTSTransformPhaseInfo Object](#)

Syntax

object.**TransformStatus**

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformPhaseInfo object

Data Type

[DTSTransformStatus](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_TransformStatus(DTSTransformStatus *pRetVal);
```

Remarks

The **DTSTransformPhaseInfo** object is referenced within a Microsoft® ActiveX® Script transformation or custom transformation.

When there are multiple transformations scheduled to run during a particular transform phase, they are executed sequentially in the order in which the

Transformation2 objects were added to the **Transformations** collection. The **TransformStatus** property allows a transformation to pass on the status code from previous transformation or to generate one of its own. In Microsoft Visual Basic® Scripting Edition (VBScript), this would look like:

```
Function Transform_Next( )  
    Transform_Next = DTSTransformPhaseInfo.TransformStatus  
    {logic which may or may not assign a function return value}  
End Function
```

TransformStatus is not updated after the execution of the insert or data-driven queries on the destination rowset.

See Also

[Data Driven Query Task](#)

[ParallelDataPumpTask Object](#)

[Transform Data Task](#)

[Transformation2 Object](#)

[Transformations Collection](#)

DTS Programming

Type Property

The **Type** property specifies the data type of a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.Type

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_Type(Long *pDataTypes);
```

Remarks

For more information about the valid values for the **Type** property, search the include file oledb.h for DBTYPEENUM. Oledb.h is installed to c:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\ by default during a custom installation of Microsoft® SQL Server™ if Development Tools/Headers and Libraries was selected from the **Select Components** dialog box.

See Also

[ActualSize Property](#)

[Attributes Property](#)

[DefinedSize Property](#)

[Status Property](#)

DTS Programming

UnderlyingValue Property

The **UnderlyingValue** property specifies the committed value of a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.**UnderlyingValue**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Variant

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT get_UnderlyingValue(*pvar);
```

Remarks

In Microsoft® SQL Server™ 2000, the **UnderlyingValue** property is the same as the **OriginalValue** property.

See Also

[OriginalValue Property](#)

Value Property

DTS Programming

Value Property

The **Value** property specifies the current value of a column.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.**Value** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Current value of a column

Data Type

Variant

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT get_Value(*pRetVal);
```

```
HRESULT put_Value(NewValue);
```

See Also

[OriginalValue Property](#)

[UnderlyingValue Property](#)

DTS Programming

Scripting Methods

This section defines the data pump scripting methods of Microsoft® SQL Server™ 2000 Data Transformation Services (DTS). The methods control the operation of Microsoft ActiveX® scripts in DTS objects.

DTS Programming

Add Method

The **Add** method adds an error record to the errors collection for a transformation.

Applies To

[DTSErrorRecords Collection](#)

Syntax

object.**Add**(
Number,
NativeError,
Description,
Source,
Helpfile,
Helpid)

Part	Description
<i>object</i>	Expression that evaluates to a DTSErrorRecords collection
<i>Number</i>	Error number
<i>NativeError</i>	Native error code
<i>Description</i>	Description of the error
<i>Source</i>	Source of the error
<i>Helpfile</i>	Name of the help file
<i>Helpid</i>	Help context ID within the help file

Prototype (C/C++)

HRESULT Add(
 long Number,
 long NativeError,
 BSTR Description,

BSTR Source,
BSTR Helpfile,
long Helpid);

See Also

[Clear Method](#)

DTS Programming

AddToCache Method

The **AddToCache** method adds a key and value mapping to the lookup object cache.

Applies To

[DTSDDataPumpLookup Object](#)

Syntax

```
object.AddToCache(  
    DataValues,  
    ParamArray KeyValues())
```

Part	Description
<i>object</i>	Expression that evaluates to a DTSDDataPumpLookup object
<i>DataValues</i>	Data values
<i>KeyValues</i>	Key values

Prototype (C/C++)

```
HRESULT AddToCache(  
    VARIANT DataValues,  
    SAFEARRAY * KeyValues);
```

Remarks

Either the key or value parameters may be a variant array.

See Also

[Execute Method](#)

[RemoveFromCache Method](#)

DTS Programming

AppendChunk Method

The **AppendChunk** method adds a segment to a binary large object (BLOB) column value.

Applies To

DTSDataPumpColumn Object	DTSDataPumpColumn2 Object
--	---

Syntax

object.**AppendChunk**(*Data*)

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>Data</i>	The segment to be added to the BLOB column

Prototype (C/C++)

```
HRESULT AppendChunk(  
    VARIANT Data);
```

Remarks

If the column value is not a BLOB type, **AppendChunk** sets the value of the column.

See Also

[GetChunk Method](#)

DTS Programming

Clear Method

The **Clear** method clears the error records collection for the current transformation.

Applies To

[DTSErrorRecords Collection](#)

Syntax

object.Clear()

Part	Description
<i>object</i>	Expression that evaluates to a DTSErrorRecords collection

Prototype (C/C++)

HRESULT Clear();

See Also

[Add Method](#)

DTS Programming

Execute Method

The **Execute** method returns a value or row of values from the lookup based on the lookup keys provided.

Applies To

[DTSDataPumpLookup Object](#)

Syntax

```
DTSDataPumpLookup.Execute(  
ParamArray KeyValues())
```

Part	Description
<i>object</i>	Expression that evaluates to a DTSDataPumpLookup object
<i>KeyValues</i>	An array of key values associated with a lookup operation

Prototype (C/C++)

```
HRESULT Execute(  
SAFEARRAY * KeyValues,  
VARIANT *pRetVal);
```

See Also

[AddToCache Method](#)

[RemoveFromCache Method](#)

DTS Programming

GetChunk Method

The **GetChunk** method retrieves the next segment of a binary large object (BLOB) column value.

Applies To

DTSDumpColumn Object	DTSDumpColumn2 Object
--------------------------------------	---------------------------------------

Syntax

object.**GetChunk**(*Length*)

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>Length</i>	Length of the BLOB segment

Prototype (C/C++)

```
HRESULT GetChunk(  
    long Length,  
    VARIANT *pvar);
```

Remarks

If the column value is not a BLOB type, **GetChunk** gets the value of the column.

See Also

[AppendChunk Method](#)

DTS Programming

Item Method

The **Item** method retrieves an object from a scripting collection.

Applies To

DTSDataPumpColumns Collection	DTSDataPumpLookups Collection
---	---

Syntax

object.**Item**(*Index*)

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>Index</i>	Item name or number

Prototype (C/C++)

```
HRESULT Item(  
    VARIANT Index,  
    interface **pRetVal);
```

interface is IDTSDataPumpColumn or IDTSDataPumpLookup.

Remarks

Index is the object name or ordinal.

See Also

[Count Property](#)

[DTSDataPumpColumn2 Object](#)

[DTSDataPumpLookup Object](#)

DTS Programming

RemoveFromCache Method

The **RemoveFromCache** method removes a key and value mapping to the lookup object cache.

Applies To

[DTSDDataPumpLookup Object](#)

Syntax

```
DTSDDataPumpLookup.RemoveFromCache(  
ParamArray KeyValues())
```

Part	Description
<i>object</i>	Expression that evaluates to a DTSDDataPumpLookup object
<i>KeyValues</i>	Key to value mapping to remove from the cache

Prototype (C/C++)

```
HRESULT RemoveFromCache(SAFEARRAY * KeyValues);
```

Remarks

The *KeyValues* array should have the same number of elements as are used for **Execute**.

See Also

[AddToCache Method](#)

[Execute Method](#)

DTS Programming

Scripting Constants

These are the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) constants that are frequently used in Microsoft ActiveX® scripts.

Constants	Description
DTStepExecResult	Specifies the result from the execution of a step
DTStepExecStatus	Specifies the current status of a step
DTStepPrecedenceBasis	Specifies whether execution status or result is used to specify precedence basis
DTStepScriptResult	Specifies the disposition of the task associated with a step script
DTTaskExecResult	Specifies the result from the execution of a task
DTSTransformFlags	Specifies the exception handling options for transformations
DTSTransformPhaseEnum	Specifies the available phases for transformations
DTSTransformStatus	Specifies the Insert operation or data driven query to be taken by data pump after a transformation completes

DTS Programming

DTS Programming Reference

This section documents the objects and collections, as well as their associated properties, methods, events, and constants, of the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) object model. For more information about the object model and a graphical representation, see [DTS Object Model Diagram](#).

For more information about the system requirements and configuration instructions for developing DTS applications in a particular programming environment, see [Creating DTS Packages in Visual Basic](#).

DTS Programming

Task Objects

The following table describes the Data Transformation Services (DTS) task classes supplied with Microsoft® SQL Server™ 2000.

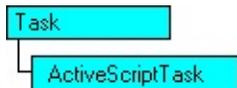
DTS Task	Description
ActiveScriptTask Object	Runs a Microsoft ActiveX® script in the context of a DTS task.
BulkInsertTask Object	Performs the function of Transact-SQL BULK INSERT statements.
CreateProcessTask Object	Runs a Microsoft Win32® executable or batch file.
CreateProcessTask2 Object	Extends the CreateProcessTask object.
DataDrivenQueryTask Object	Transforms source data and writes it to the destination through user-specified queries.
DataDrivenQueryTask2 Object	Extends the DataDrivenQueryTask object.
DataPumpTask Object	Transforms source data and copies it to the destination.
DataPumpTask2 Object	Extends the DataPumpTask object.
DynamicPropertiesTask Object	Changes the values of the properties of DTS objects at runtime.
ExecutePackageTask Object	Runs another DTS package.
ExecuteSQLTask Object	Runs a sequence of SQL statements.
ExecuteSQLTask2 Object	Runs a sequence of SQL statements. Extended ExecuteSQLTask object.
DTSFTPTask Object	Transfers files using File Transfer Protocol (FTP).
DTSMessagesQueueTask Object	Sends and receives Message Queuing messages.
ParallelDataPumpTask Object	Transforms hierarchical source rowsets and writes to destination.
SendMailTask Object	Sends e-mail in the context of a DTS task.
TransferObjectsTask Object	Transfers SQL Server objects between source

	and destination.
TransferObjectsTask2 Object	Transfers SQL Server objects between source and destination. Extended TransferObjectsTask object.

DTS Programming

ActiveScriptTask Object

The **ActiveScriptTask** object defines a task that is a Microsoft® ActiveX® script. ActiveX Script tasks do not use the data pump and therefore do not have access to the **Connections** collection or Data Transformation Services (DTS) source and destination collections. However, **ActiveScriptTask** objects have full access to the **GlobalVariables** collection, which provides a way to share information across tasks.



Collections

Properties Collection

Properties

ActiveXScript Property	FunctionName Property
AddGlobalVariables Property	Name Property
Description Property	

Methods

CheckSyntax Method	Execute Method
------------------------------------	--------------------------------

Remarks

Script languages available on a particular system can be determined by enumerating the **ScriptingLanguageInfos** collection of the **Application** object. For more information about the scripting language appropriate for use with DTS, see [ScriptingLanguageInfo Object](#).

Example

The following Microsoft Visual Basic® code creates **ActiveScriptTask** and **Step** objects. The ActiveX script returns **DTSTaskExecResult_Success** or **DTSTaskExecResult_Failure**, depending on the value of a global variable. The success or failure return can be used to direct the workflow of subsequent steps in the package.

```
Dim objPackage    As DTS.Package2
Dim objStep       As DTS.Step
Dim objTask       As DTS.Task
Dim objScripTask  As DTS.ActiveScriptTask
...
'create step and task, specify script, func name and language
Set objStep = objPackage.Steps.New
Set objTask = objPackage.Tasks.New("DTSActiveScriptTask")
Set objScripTask = objTask.CustomTask
With objScripTask
    .Name = "AXScr_Task"
    .ActiveXScript = _
        "Function Main()" & vbCrLf & _
        "If DTSGlobalVariables( ""GlobalOne"" ) > 0 Then" & vbCrLf & _
        vbTab & "Main = DTSTaskExecResult_Success" & vbCrLf & _
        "Else" & vbCrLf & _
        vbTab & "Main = DTSTaskExecResult_Failure" & vbCrLf & _
        "End If" & vbCrLf & _
        "End Function"
    .FunctionName = "Main"
    .ScriptLanguage = "VBScript"
End With

'link step to task to package
objStep.TaskName = objScripTask.Name
objStep.Name = "AXScr_Step"
With objPackage
```

```
.Steps.Add objStep  
.Tasks.Add objTask  
.FailOnError = False  
End With
```

Note If an ActiveX script returns **DTSTaskExecResult_Failure** and the **FailOnError** property of the **Package2** object is TRUE, the entire package will fail.

See Also

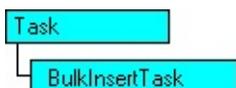
[Application Object](#)

[ScriptingLanguageInfos Collection](#)

DTS Programming

BulkInsertTask Object

The **BulkInsertTask** object, based on the Transact-SQL BULK INSERT statement, provides the fastest method for copying large amounts of data from a text file to Microsoft® SQL Server™. Use **BulkInsertTask** for copying operations, and in situations where performance is the most important consideration. It is not used in conjunction with transformations during data import operations.



Collections

[Properties Collection](#)

Properties

BatchSize Property	FormatFile Property
CheckConstraints Property	KeepIdentity Property
Codepage Property	KeepNulls Property
ConnectionID Property	LastRow Property
DataFile Property	MaximumErrors Property
DataFileType Property	Name Property
Description Property	RowTerminator Property
DestinationTableName Property	SortedData Property
FieldTerminator Property	TableLock Property
FirstRow Property	

Methods

[Execute Method](#)

Remarks

A **Connection2** object must be used to access the database into which data is inserted. You can specify the format of the input data file directly using the **FieldTerminator** and **RowTerminator** properties, or indirectly through a **bcp** format file.

Example

The following Microsoft Visual Basic® code uses the **BulkInsertTask** object to insert data from file D:\DTS_UE\BCPData\Payroll.txt into table Payroll of database DTS_UE.

```
Public Sub Main()  
'initialize Payroll table in DTS_UE db with bulk data  
Dim objPackage As DTS.Package2  
Dim objConnect As DTS.Connection2  
Dim objStep As DTS.Step  
Dim objTask As DTS.Task  
Dim objBulkCopy As DTS.BulkInsertTask  
  
Set objPackage = New DTS.Package  
  
'create database connection  
Set objConnect = objPackage.Connections.New("SQLOLEDB.1")  
With objConnect  
    .ID = 1  
    .DataSource = "(local)"  
    .UseTrustedConnection = True  
End With  
objPackage.Connections.Add objConnect  
'create step and task, specify data file and format  
Set objStep = objPackage.Steps.New  
Set objTask = objPackage.Tasks.New("DTSBulkInsertTask")  
Set objBulkCopy = objTask.CustomTask
```

```
With objBulkCopy
    .Name = "BulkInsTask"
    .DataFile = "D:\DTS_UE\BCPData\Payroll.txt"
    .ConnectionID = 1
    .DestinationTableName = "DTS_UE..Payroll"
    .FieldTerminator = "|"
    .RowTerminator = "\r\n"
End With
```

```
'link step to task to package, run package
objStep.TaskName = objBulkCopy.Name
objStep.Name = "BulkInsStep"
```

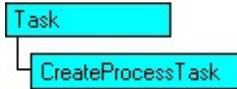
```
With objPackage
    .Steps.Add objStep
    .Tasks.Add objTask
    .FailOnError = True
```

```
    .Execute
End With
End Sub
```

DTS Programming

CreateProcessTask Object

The **CreateProcessTask** object runs a Microsoft® Win32® executable or batch file in the context of the Data Transformation Services (DTS) package.



Use the **ProcessCommandLine** property to specify the file to be executed and command line parameters. You can set a **Timeout** for the executed process. You can specify **TerminateProcessAfterTimeout**, or also **FailPackageOnTimeout**.

Collections

[Properties Collection](#)

Properties

Description Property	SuccessReturnCode Property
FailPackageOnTimeout Property	TerminateProcessAfterTimeout Property
Name Property	Timeout Property
ProcessCommandLine Property	

Methods

[Execute Method](#)

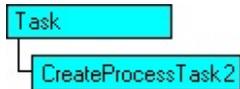
Remarks

The **CreateProcessTask** object is compatible with Microsoft SQL Server™ version 7.0. For information about an extended version of this object, see [CreateProcessTask2 Object](#).

DTS Programming

CreateProcessTask2 Object

The **CreateProcessTask2** object runs a Microsoft® Win32® executable or batch file in the context of the Data Transformation Services (DTS) package. It is called the **Execute Process Task** in DTS Designer.



Extended Methods

[GetExpandedProcessCommandLine Method](#)

Remarks

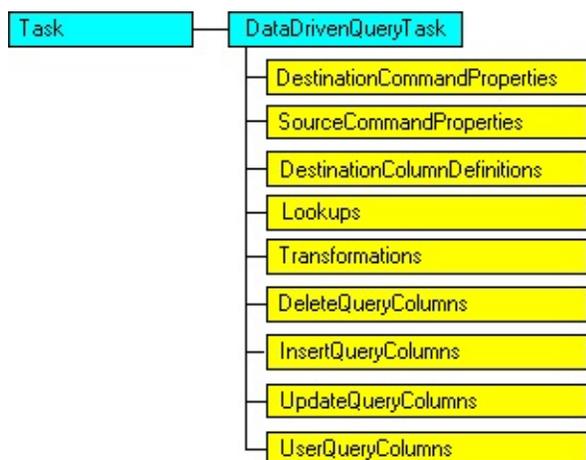
The **CreateProcessTask2** object extends the functionality of the [CreateProcessTask Object](#) and inherits the properties and methods of that object. In addition, the **GetExpandedProcessCommandLine** method returns the process command line parameter string with environment variables expanded.

For more information about when to use the **CreateProcessTask** object instead of the **CreateProcessTask2** object, see [Extended DTS Objects](#).

DTS Programming

DataDrivenQueryTask Object

The **DataDrivenQueryTask** object reads data through a source **Connection** object and transforms it using one or more **Transformation** objects. One of the transformations, a **DataPumpTransformScript** object, returns an indicator that determines which of four parameterized queries is executed on the destination **Connection** object. For more information, see [Adding DTS Transformations](#) and [Adding DTS Column Objects](#).



The queries are called the Insert query, Update query, Delete query and User query, although they do not actually need to be used for these purposes. Any sequence of SQL action statements and stored procedure calls can be used for any of the queries. The query parameters are columns from the destination connection. For more information, see [Adding DTS Query Strings](#).

Collections

Lookups Collection	Transformations Collection
Properties Collection	

Properties

DeleteQuery Property	InsertQueryColumns Property
DeleteQueryColumns Property	LastRow Property
Description Property	MaximumErrorCount Property

DestinationColumnDefinitions Property	Name Property
DestinationCommandProperties Property	ProgressRowCount Property
DestinationConnectionID Property	SourceCommandProperties Property
DestinationObjectName Property	SourceConnectionID Property
DestinationSQLStatement Property	SourceObjectName Property
ExceptionFileColumnDelimiter Property	SourceSQLStatement Property
ExceptionFileName Property	UpdateQuery Property
ExceptionFileRowDelimiter Property	UpdateQueryColumns Property
FetchBufferSize Property	UserQuery Property
FirstRow Property	UserQueryColumns Property
InsertQuery Property	

Methods

Execute Method

Remarks

The query selection indicator must be from the DTSTransformStatus constants. Do not use a sum or logical OR of these values. If more than one **DataPumpTransformScript** object returns an indicator, all but the last will be overwritten and lost.

The **DataDrivenQueryTask** object is compatible with Microsoft® SQL Server™ version 7.0. For information about an extended version of this object, see [DataDrivenQueryTask2 Object](#).

For more information about how to create a **DataDrivenQueryTask** object and assign query strings and parameters, see [DTS Query Strings in Visual Basic](#).

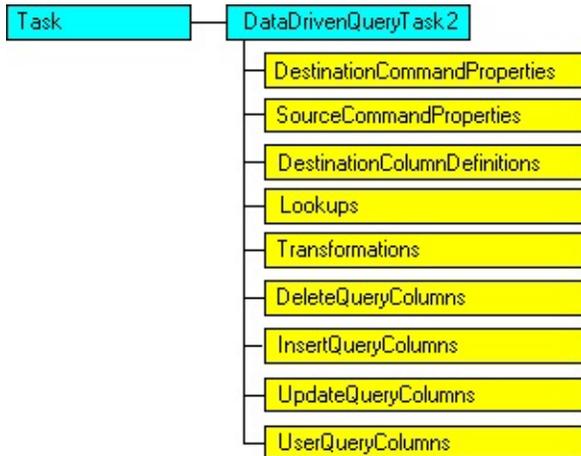
See Also

[DTSTransformStatus](#)

DTS Programming

DataDrivenQueryTask2 Object

The **DataDrivenQueryTask2** object transforms data from a source connection and invokes user-defined queries to write data to a destination connection.



Extended Properties

ExceptionFileOptions Property	RowsComplete Property
ExceptionFileTextQualifier Property	RowsInError Property
InputGlobalVariableNames Property	

Remarks

The **DataDrivenQueryTask2** object extends the functionality of the **DataDrivenQueryTask** object and inherits the properties and methods of that object. In addition, Microsoft® SQL Server™ 2000 provides the following properties:

- The **ExceptionFileOptions** property specifies how errors and exception rows are to be written to the appropriate files. The **ExceptionFileTextQualifier** property specifies the text qualifier for the data in the exception file.
- The **SourceSQLStatement** property specifies parameters that can now be coded in the query. The **InputGlobalVariableNames** property

specifies a list of package global variable names whose values are to be substituted for the query parameters.

- The **RowsComplete** property returns the count of transformed rows and the **RowsInError** property returns the number of transformation error rows.

For more information about when to use the **DataDrivenQueryTask** object instead of the **DataDrivenQueryTask2** object, see [Extended DTS Objects](#).

See Also

[DataDrivenQueryTask Object](#)

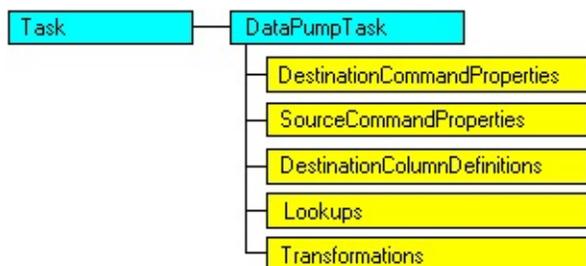
[SourceSQLStatement Property](#)

DTS Programming

DataPumpTask Object

The Data Transformation Services (DTS) data pump is an OLE DB service provider that provides the means to import, export, and transform data between heterogeneous data sources.

The **DataPumpTask** object makes the features of the data pump available as a DTS task.



Collections

Lookups Collection	Transformations Collection
Properties Collection	

Properties

AllowIdentityInserts Property	FirstRow Property
Description Property	InsertCommitSize Property
DestinationColumnDefinitions Property	LastRow Property
DestinationCommandProperties Property	MaximumErrorCount Property
DestinationConnectionID Property	Name Property
DestinationObjectName Property	ProgressRowCount Property
DestinationSQLStatement Property	SourceCommandProperties Property
ExceptionFileColumnDelimiter Property	SourceConnectionID Property
ExceptionFileName Property	SourceObjectName Property

ExceptionFileRowDelimiter Property	SourceSQLStatement Property
FastLoadOptions Property	UseFastLoad Property
FetchBufferSize Property	

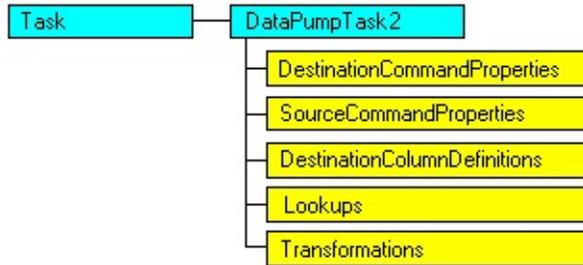
Methods

Execute Method

DTS Programming

DataPumpTask2 Object

The **DataPumpTask2** object imports, exports, and transforms data between heterogeneous data sources.



Extended Properties

DataPumpOptions Property	InputGlobalVariableNames Property
ExceptionFileOptions Property	RowsComplete Property
ExceptionFileTextQualifier Property	

Remarks

The **DataPumpTask2** object extends the functionality of the **DataPumpTask** object and inherits the properties and methods of that object. In addition, Microsoft® SQL Server™ 2000 provides the following properties:

- The **ExceptionFileOptions** property specifies how errors and exception rows are to be written to the appropriate files. The **ExceptionFileTextQualifier** property specifies the text qualifier for the data in the exception file.
- The **SourceSQLStatement** property specifies parameters that can now be coded in the query. The **InputGlobalVariableNames** property specifies a list of package global variable names whose values are to be substituted for the query parameters.
- The **RowsComplete** property returns the count of transformed rows and

the **RowsInError** property returns the number of transformation error rows.

For more information about when to use the **DataPumpTask** object instead of the **DataPumpTask2** object, see [Extended DTS Objects](#).

See Also

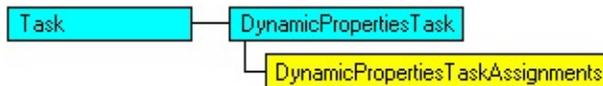
[DataPumpTask Object](#)

[SourceSQLStatement Property](#)

DTS Programming

DynamicPropertiesTask Object

The **DynamicPropertiesTask** object changes the values of package objects properties at runtime. Any property of any object in the package can be modified. This is useful for packages created with Data Transformation Services (DTS) Designer and the DTS Import/Export Wizard because many package object properties are fixed after the design process is complete.



The **Dynamic Properties Task** object provides several different sources for the new value of a property:

- A constant
- The contents of a data file
- An environment variable
- A DTS global variable
- A field in an .ini file
- An SQL query

In an application that creates and manipulates DTS objects, it is often easier to modify the values of properties directly in code rather than use a **Dynamic Properties Task** object. However, if part of a DTS package is contained within a module that cannot be modified easily, the **Dynamic Properties Task** object may be useful.

Collections

--	--

DynamicPropertiesTaskAssignments Collection	Properties Collection
---	---------------------------------------

Properties

Assignments Property	Name Property
Description Property	

Methods

Execute Method

Remarks

To use the **Dynamic Properties Task** object, for each package object property that is to be modified, a **DynamicPropertiesTaskAssignment** object is created. The properties of this object are set to specify the package object property to be changed and the source of the new value. The **DynamicPropertiesTaskAssignment** object is added to the **DynamicPropertiesTaskAssignments** collection.

The **New** method of the **Tasks** collection of the **Package** object returns a reference to a **Task** object. The **CustomTask** property of the **Task** object returns a reference to the appropriate custom task object. For more information, see [Creating DTS Package Workflow and Tasks](#).

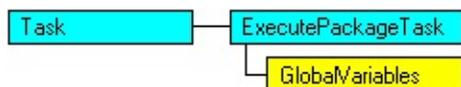
See Also

[DynamicPropertiesTaskAssignment Object](#)

DTS Programming

ExecutePackageTask Object

The **ExecutePackageTask** object runs another Data Transformation Services (DTS) package. The package can be located in Microsoft® SQL Server™ 2000 Meta Data Services, in SQL Server, or in a file. The package can be specified by name or by package or version globally unique identifier (GUID).



DTS global variables can be passed to the target package. For each such global variable, a **GlobalVariable** object, which defines the name of the variable and value, is added to the **GlobalVariables** collection of the **ExecutePackageTask** object. These global variables are distinct from the members of the **GlobalVariables** collection of the **Package2** object that contains the **ExecutePackageTask** object. Use the **InputGlobalVariableNames** property to specify members of the parent package **GlobalVariables** collection that are to be created or set in the child package.

Steps in child packages can join the transactions of parent packages, if Microsoft Distributed Transaction Coordinator (MS DTC) is running.

Collections

GlobalVariables Collection	Properties Collection
--	---------------------------------------

Properties

Description Property	RepositoryDatabaseName Property
FileName Property	ServerName Property
InputGlobalVariableNames Property	ServerPassword Property
Name Property	ServerUserName Property
PackageID Property	UseRepository Property
PackageName Property	UseTrustedConnection Property
PackagePassword Property	

Methods

Execute Method

Remarks

The **New** method of the **Tasks** collection of the **Package** object returns a reference to a **Task** object. The **CustomTask** property of the **Task** object returns a reference to the appropriate custom task object.

Example

The following Microsoft Visual Basic® code uses the **Execute Package Task** object to run a package located in the file VarPubsFields.dts, which is encrypted and requires the password "user" to access.

```
Public Sub Main()
```

```
'Run package stored in file C:\DTS_UE\TestPkg\VarPubsFields.dts.
```

```
    Dim oPackage As DTS.Package
```

```
    Dim oStep As DTS.Step
```

```
    Dim oTask As DTS.Task
```

```
    Dim oCustTask As DTS.ExecutePackageTask
```

```
    Set oPackage = New DTS.Package
```

```
'Create step and task, link step to task.
```

```
    Set oStep = oPackage.Steps.New
```

```
    oStep.Name = "ExecPkgStep"
```

```
    Set oTask = oPackage.Tasks.New("DTSExecutePackageTask")
```

```
    Set oCustTask = oTask.CustomTask
```

```
    oCustTask.Name = "ExecPkgTask"
```

```
    oStep.TaskName = oCustTask.Name
```

```
    oPackage.Steps.Add oStep
```

```
    Set oStep = Nothing
```

'Specify package to be run.

oCustTask.PackagePassword = "user"

oCustTask.FileName = "C:\DTS_UE\TestPkg\VarPubsFields.dts"

'Link task to package, run package.

oPackage.Tasks.Add oTask

Set oCustTask = Nothing

Set oTask = Nothing

oPackage.Execute

Set oPackage = Nothing

End Sub

DTS Programming

ExecuteSQLTask Object

The **ExecuteSQLTask** object allows you to execute a sequence of one or more SQL statements on a connection. Use the **ConnectionID** property to specify the connection and the **SQLStatement** property to specify the sequence of SQL statements.



Collections

[Properties Collection](#)

Properties

CommandProperties Property	Description Property
CommandTimeout Property	Name Property
ConnectionID Property	

Methods

[Execute Method](#)

DTS Programming

ExecuteSQLTask2 Object

The **ExecuteSQLTask2** object allows you to execute a sequence of one or more SQL statements on a connection.



Extended Properties

InputGlobalVariableNames Property	OutputGlobalVariableNames Property
OutputAsRecordset Property	

Remarks

The **ExecuteSQLTask2** object extends the functionality of the **ExecuteSQLTask** object and inherits the properties and methods of that object. In addition, the **ExecuteSQLTask2** adds the **InputGlobalVariableNames**, **OutputAsRecordset** and **OutputGlobalVariableNames** properties.

The **InputGlobalVariableNames** property specifies a list of Data Transformation Services (DTS) global variable names that are used as parameters for the sequence of SQL statements specified by the **SQLStatement** property. The **OutputGlobalVariableNames** property specifies a list of global variable names that receive fields from the first row of the first rowset produced by the **SQLStatement** query. If the **OutputAsRecordset** property is set, the entire rowset is written to the first global variable specified by **OutputGlobalVariableNames** as a disconnected Microsoft® ActiveX® Data Objects (ADO) recordset.

For more information about when to use the **ExecuteSQLTask** object instead of the **ExecuteSQLTask2** object, see [Extended DTS Objects](#).

See Also

[ExecuteSQLTask Object](#)

[SQLStatement Property](#)

DTS Programming

DTSFTPTask Object

The **DTSFTPTask** object transfers one or more files from a specified Internet FTP site or network directory to a destination directory.



Collections

[Properties Collection](#)

Properties

Description Property	SourceFilename Property
DestSite Property	SourceLocation Property
Name Property	SourcePassword (DTSFTPTask) Property
NonOverwritable Property	SourceSite Property
NumRetriesOnSource Property	

Methods

[Execute Method](#)

Remarks

The **New** method of the **Tasks** collection of the **Package** object returns a reference to a **Task** object. The **CustomTask** property of the **Task** object returns a reference to the appropriate custom task object.

Example

The following Microsoft® Visual Basic® code uses the **DTSFTPTask** object to

copy the files File3.dat and NWProdWiz.xls from the directory I:\DTS\TestData to D:\DTS_UE\Dest.

Public Sub Main()

'Copy files from I:\DTS\TestData to D:\DTS_UE\Dest.

Dim oPackage As DTS.Package

Dim oStep As DTS.Step

Dim oTask As DTS.Task

Dim oCustTask As DTSCustTasks.DTSFTPTask

Set oPackage = New DTS.Package

oPackage.FailOnError = True

'Create step and task, link step to task.

Set oStep = oPackage.Steps.New

oStep.Name = "FTPSrcDirStep"

Set oTask = oPackage.Tasks.New("DTSFTPTask")

Set oCustTask = oTask.CustomTask

oCustTask.Name = "FTPSrcDirTask"

oStep.TaskName = oCustTask.Name

oPackage.Steps.Add oStep

Set oStep = Nothing

'Specify files, source and destination directories.

oCustTask.SourceLocation = DTSFTPSourceLocation_Directory

oCustTask.SourceSite = "I:\DTS\TestData"

oCustTask.SourceFilename = _

 "File3.dat";';123';NWProdWiz.XLS';';458240';"

oCustTask.DestSite = "D:\DTS_UE\Dest"

'Link task to package, run package.

oPackage.Tasks.Add oTask

Set oCustTask = Nothing

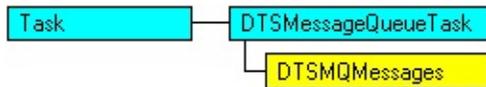
Set oTask = Nothing

```
oPackage.Execute  
Set oPackage = Nothing  
End Sub
```

DTS Programming

DTSMMessageQueueTask Object

The **DTSMMessageQueueTask** object sends and receives messages to or from a queue of Message Queuing. It allows participation in distributed transactions when Microsoft® Distributed Transaction Coordinator (MS DTC) is running.



A single instance of the **DTSMMessageQueueTask** either sends one or more messages to a specified queue or receives a single message from a specified queue, waiting, if necessary, for the message to arrive.

Three types of messages can be sent:

- A string message, which is supplied as the value of a property.
- A data file message, which is generated from the contents of a specified data file.
- A global variables message, which is generated from the names and values of one or more Data Transformation Services (DTS) global variables.

Two types of messages can be received:

- A string message. Various comparisons can be specified to determine whether the task returns success or failure.
- A global variables message. The values of one or more global variables (specified by the message) in the local package can be updated. If any of the global variables do not exist in the local package, they are created.

Collections

[DTSMQMessages Collection](#)

[Properties Collection](#)

Properties

DataFileNonOverwritable Property	ReceiveMessageTimeout Property
Description Property	ReceiveMessageType Property
DTSMMessageLineageID Property	RemoveFromQueue Property
DTSMMessagePackageID Property	SaveDataFileName Property
DTSMMessageVersionID Property	StringCompareType Property
ErrorIfReceiveMessageTimeout Property	StringCompareValue Property
Name Property (DTS)	TaskType Property
QueuePath Property	

Methods

Execute Method	Messages Method
--------------------------------	---------------------------------

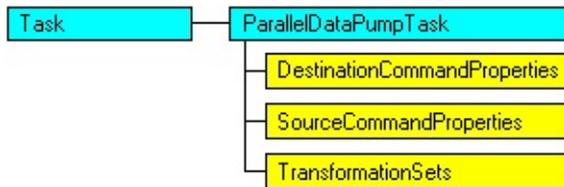
Remarks

The **New** method of the **Tasks** collection of the **Package** object returns a reference to a **Task** object. The **CustomTask** property of the **Task** object returns a reference to the appropriate custom task object.

DTS Programming

ParallelDataPumpTask Object

The **ParallelDataPumpTask** object copies and transforms data from source to destination rowsets. It performs the same functions as the **DataPumpTask2** and **DataDrivenQueryTask2**, except that it will also copy and transform hierarchical rowsets. The **FastLoad** option of the **DataPumpTask2** is not supported, however.



In the **ParallelDataPumpTask** object, a **TransformationSets** collection is populated with one or more **TransformationSet** objects, each of which includes a **Transformations** collection and contains all the information necessary to transform a component rowset of the source hierarchical rowset to the corresponding component rowset in the destination.

The component rowsets are scanned and matched source to destination through recursive descent. The columns of the parent rowset are in column-ordinal order. When a child rowset column is encountered, it is scanned before the remaining columns of the parent. Child rowsets are similarly processed; their own children are scanned when encountered, before their remaining rows, with greater column ordinal.

Using Transformation Modes

The **ParallelDataPumpTask** operates in one of the following modes:

- In flattened mode, the component rowsets are copied without regard to the chapter values. All the rows of each child rowset are copied, including those not referenced by any chapters.
- In hierarchical mode, rowsets are copied a row at a time. The rows of a child rowset referenced by the chapter in the parent rowset row are copied. Thus, child rowset rows can be copied multiple times, or not at

all.

- In data driven query mode, rowsets are processed in the same way as in flattened mode, except that one of four queries, typically an INSERT, UPDATE or DELETE SQL statement, or stored procedure, can be executed based on the return code of a script transform.

The mode is specified with the **TransformationSetOptions** property of the **TransformationSet** object.

Collections

Properties Collection	TransformationSets Collection
---------------------------------------	---

Properties

Description Property	Name Property
DestinationCommandProperties Property	SourceCommandProperties Property
DestinationConnectionID Property	SourceConnectionID Property
DestinationObjectName Property	SourceObjectName Property
DestinationSQLStatement Property	SourceSQLStatement Property
InputGlobalVariableNames Property	

Methods

Execute Method

Remarks

A failure of any **TransformationSet** (including failure due to the maximum

number of error rows being exceeded for that **TransformationSet**, regardless of the error counts in other TransformationSets) results in the failure of the **ParallelDataPumpTask**.

In Microsoft® SQL Server™ 2000, the **ParallelDataPumpTask** is available only through the Data Transformation Services (DTS) object model. It cannot be accessed through DTS Designer or the DTS Import/Export Wizard.

Examples

Parallel Data Driven Query Example	Parallel Data Pump Example
--	--

See Also

[Hierarchical Rowsets](#)

[TransformationSet Object](#)

Hierarchical Rowsets

A rowset is an OLE DB object that consists of data organized as a grid of rows and named, typed columns. Typically, a rowset contains a result set from a database query, but the data can come from any source.

In a hierarchical rowset, one or more columns are themselves rowsets. The individual column values are references to subsets, called chapters, of the column rowset. A chapter can include none, some, or all of its rows. The column rowsets can themselves have one or more columns that are rowsets, nested to an arbitrary level.

Using the Data Shaping Service for OLE DB

Hierarchical rowsets are often generated with the Microsoft® Data Shaping Service for OLE DB. This provider supports the Shape language, which allows rowset hierarchies to be constructed from rowsets obtained from an OLE DB data provider. The **Shape Append** command appends one or more child rowsets as columns to a parent rowset, and assigns a reference to a chapter to each row value in each appended column. For example:

```
SHAPE {SELECT au_id, au_lname, au_fname FROM authors}  
APPEND ({SELECT au_id, title FROM titleauthor TA, titles TS  
        WHERE TA.title_id = TS.title_id}  
        AS title_chap RELATE au_id TO au_id)
```

This command creates a parent rowset from table **authors** and appends a child rowset in a column named **title_chap**. Each row value in **title_chap** is a reference to the subset of the child rowset that has the same value in its **au_id** column as in the **au_id** column of the parent rowset for that row. The tables referenced by this command are in the **pubs** database that is supplied with Microsoft SQL Server™ 2000.

Writing sGetTitleAuthors Code

This is the Microsoft Visual Basic® source code for the **sGetTitleAuthors** function:

```

Private Function sGetTitleAuthors() As String
Dim rstParent As ADODB.Recordset
Dim rstChild As ADODB.Recordset
Dim sBuf As String

Const CONNECT_PUBS = "PROVIDER=MSDataShape;DATA PRO'
"SERVER=;DATABASE=pubs;UID=sa;PWD=;"
Const SHAPE_TITLEAUTHORS = _
"SHAPE {SELECT au_id, au_lname, au_fname FROM authors} " &
"APPEND ({SELECT au_id, title FROM titleauthor TA, titles TS "
"WHERE TA.title_id = TS.title_id} " & _
"AS title_chap RELATE au_id TO au_id)"

'----- create rowsets
Set rstParent = New ADODB.Recordset
rstParent.Open SHAPE_TITLEAUTHORS, CONNECT_PUBS

'----- process parent rowset
Do While Not rstParent.EOF
sBuf = sBuf & rstParent("au_id") & vbTab & _
rstParent("au_lname") & ", " & rstParent("au_fname") & vbCrLf

'----- process chapter of child rowset
Set rstChild = rstParent("title_chap").Value
Do While Not rstChild.EOF
sBuf = sBuf & vbTab & vbTab & rstChild("title") & vbCrLf
rstChild.MoveNext
Loop
rstParent.MoveNext
Loop
sGetTitleAuthors = sBuf
End Function

```

Running sGetTitleAuthors

This Microsoft Visual Basic function creates and processes the hierarchical rowset described earlier. It returns a string consisting of each author name, followed by the titles associated with that author in the **pubs** database.

This example can be run on a computer on which Visual Basic 6.0 and SQL Server 2000 have been installed.

The steps for running this example are as follows:

1. Create a new Standard EXE project in the Visual Basic development environment.
2. In the **Project/References** dialog box, select **Microsoft ActiveX Data Objects 2.5 Library**.
3. Place a **command button** and a **textbox** on the form **Form1**. Set the **ScrollBars** property of the textbox to **3 – Both** and the **MultiLine** property to **TRUE**.
4. Copy the following code for function **sGetTitleAuthors** to the code window for **Form1**.
5. In the **_Click** sub for the command button, call **sGetTitleAuthors**, and assign the string it returns to the **Text** property of the text box.
6. Run the project and click the command button.

Parallel Data Pump Example

This sample Microsoft® Visual Basic® function **sCopyCustOrderProd** creates and runs a package that transforms parts of the **Customers**, **Orders**, **Order Details**, and **Products** tables from the **Northwind** database that ships with Microsoft SQL Server™ 2000. The function generates a hierarchical rowset consisting of the customers located in the U.K., their orders, the order details, and the products.

Creating the sCopyCustOrderProd Rowset

This function copies the data to tables in a database called **DTSTest** that have the following structure:

```
CREATE TABLE dbo.customers (  
    customer_key NCHAR (5) NOT NULL ,  
    company_name NVARCHAR (40) NOT NULL )
```

```
CREATE TABLE dbo.orders (  
    customer_key NCHAR (5) NULL ,  
    order_key INT NOT NULL )
```

```
CREATE TABLE dbo.products (  
    product_key INT NOT NULL ,  
    product_name NVARCHAR (40) NOT NULL )
```

```
CREATE TABLE dbo.order_details (  
    order_key INT NOT NULL ,  
    product_key INT NOT NULL ,  
    discount REAL NOT NULL )
```

The number of rows copied depends on whether **Flattened** or **Hierarchical** mode is used. In **Flattened** mode, the entire **Orders**, **Order Details**, and

Products tables are copied. In **Hierarchical** mode, only the rows referenced by the U.K. customers are copied, although there are many duplicates of these rows in the **products** table in the destination database.

The Visual Basic Function for sCopyCustOrderProd

This is the Visual Basic source code for the **sCopyCustOrderProd** code:

```
Private Function sCopyCustOrderProd( _
    ByVal TranSetOpt As DTS.DTSTransformationSetOptions) As Strir
Dim oPackage          As New DTS.Package
Dim oConnection      As DTS.Connection
Dim oTask            As DTS.Task
Dim oStep            As DTS.Step
Dim oTransform       As DTS.Transformation
Dim oTransformationSet As DTS.TransformationSet
Dim oParallelPumpTask As DTS.ParallelDataPumpTask

Const SHAPE_NW_CUST_ORDER_PROD = _
    "SHAPE {SELECT CustomerID, CompanyName " & _
        "FROM Customers WHERE Country = 'UK'} " & _
    "APPEND ((SHAPE {SELECT OrderID, CustomerID FROM Order
        "APPEND ((SHAPE {SELECT OrderID, ProductID, Discount
            "FROM [Order Details]} " & _
                "APPEND ({SELECT ProductID, ProductName " & _
                    "FROM Products} " & _
                        "AS ProductChap " & _
                            "RELATE ProductID TO ProductID)) " & _
                                "AS DetailChap RELATE OrderID TO OrderID)) " & _
                                    "AS OrderChap RELATE CustomerID TO CustomerID)"

Const SHAPE_UE_CUST_ORDER_PROD = _
    "SHAPE {SELECT * FROM customers} " & _
    "APPEND ((SHAPE {SELECT * FROM orders} " & _
```

```
"APPEND ((SHAPE {SELECT * FROM order_details} " & _
"APPEND ({SELECT * FROM products} " & _
"AS product_chap " & _
"RELATE product_key TO product_key)) " & _
"AS detail_chap RELATE order_key TO order_key)) " &
"AS order_chap RELATE customer_key TO customer_key)"
```

```
'----- define source connection - Northwind
Set oConnection = oPackage.Connections.New("MSDataShape")
With oConnection
    .ConnectionProperties("Data Provider") = "SQLOLEDB"
    .ID = 1
    .Catalog = "Northwind"
    .UserID = "sa"
End With
oPackage.Connections.Add oConnection
```

```
'----- define destination connection - (local) DTSTest
Set oConnection = oPackage.Connections.New("MSDataShape")
With oConnection
    .ConnectionProperties("Data Provider") = "SQLOLEDB"
    .ID = 2
    .DataSource = "(local)"
    .Catalog = "DTSTest"
    .UseTrustedConnection = True
End With
oPackage.Connections.Add oConnection
```

```
'----- set hierarchical/flattened, set connections and commands
Set oTask = oPackage.Tasks.New("DTSParallelDataPumpTask")
Set oParallelPumpTask = oTask.CustomTask
With oParallelPumpTask
    .TransformationSetOptions = TranSetOpt
```

```
.SourceConnectionID = 1
.SourceSQLStatement = SHAPE_NW_CUST_ORDER_PROD
.DestinationConnectionID = 2
.DestinationSQLStatement = SHAPE_UE_CUST_ORDER_PROD
End With
```

```
'----- create TransformationSet for customers
Set oTransformationSet = oParallelPumpTask. _
    TransformationSets.New("TransformSet_Customers")
oParallelPumpTask.TransformationSets.Add oTransformationSet
Set oTransform = oTransformationSet. _
    Transformations.New("DTS.DataPumpTransformCopy")
With oTransform
    .SourceColumns.AddColumn "CustomerID", 1
    .SourceColumns.AddColumn "CompanyName", 2
    .DestinationColumns.AddColumn "customer_key", 1
    .DestinationColumns.AddColumn "company_name", 2
    .Name = "Transform"
End With
oTransformationSet.Transformations.Add oTransform
```

```
'----- create TransformationSet for orders
Set oTransformationSet = oParallelPumpTask. _
    TransformationSets.New("TransformSet_Orders")
oParallelPumpTask.TransformationSets.Add oTransformationSet
Set oTransform = oTransformationSet. _
    Transformations.New("DTS.DataPumpTransformCopy")
With oTransform
    .SourceColumns.AddColumn "OrderID", 1
    .SourceColumns.AddColumn "CustomerID", 2
    .DestinationColumns.AddColumn "order_key", 1
    .DestinationColumns.AddColumn "customer_key", 2
    .Name = "Transform"
```

End With

```
oTransformationSet.Transformations.Add oTransform
```

```
'----- create TransformationSet for order details
```

```
Set oTransformationSet = oParallelPumpTask. _
```

```
TransformationSets.New("TransformSet_Details")
```

```
oParallelPumpTask.TransformationSets.Add oTransformationSet
```

```
Set oTransform = oTransformationSet. _
```

```
Transformations.New("DTS.DataPumpTransformCopy")
```

```
With oTransform
```

```
.SourceColumns.AddColumn "OrderID", 1
```

```
.SourceColumns.AddColumn "ProductID", 2
```

```
.SourceColumns.AddColumn "Discount", 3
```

```
.DestinationColumns.AddColumn "order_key", 1
```

```
.DestinationColumns.AddColumn "product_key", 2
```

```
.DestinationColumns.AddColumn "discount", 3
```

```
.Name = "Transform"
```

End With

```
oTransformationSet.Transformations.Add oTransform
```

```
'----- create TransformationSet for products
```

```
Set oTransformationSet = oParallelPumpTask. _
```

```
TransformationSets.New("TransformSet_Products")
```

```
oParallelPumpTask.TransformationSets.Add oTransformationSet
```

```
Set oTransform = oTransformationSet. _
```

```
Transformations.New("DTS.DataPumpTransformCopy")
```

```
With oTransform
```

```
.SourceColumns.AddColumn "ProductID", 1
```

```
.SourceColumns.AddColumn "ProductName", 2
```

```
.DestinationColumns.AddColumn "product_key", 1
```

```
.DestinationColumns.AddColumn "product_name", 2
```

```
.Name = "Transform"
```

End With

```
oTransformationSet.Transformations.Add oTransform
```

```
'----- add task, step to package
```

```
oTask.Name = "ParallelDPTask"
```

```
With oPackage
```

```
Set oStep = oPackage.Steps.New
```

```
oStep.Name = "ParallelDPStep"
```

```
oStep.TaskName = oTask.Name
```

```
.Tasks.Add oTask
```

```
.Steps.Add oStep
```

```
.Name = "ParallelDataPumpTask Package"
```

```
.FailOnError = True
```

```
.Execute      'run the package
```

```
End With
```

```
End Function
```

Running sCopyCustOrderProd

This example can be run on a computer on which Microsoft Visual Basic 6.0 and SQL Server 2000 have been installed.

The basic steps for running **sCopyCustOrderProd** are as follows:

1. Create a database named **DTSTest** using SQL Server Enterprise Manager, and then create the tables defined above in **DTSTest**. If you use another database, change the line in the example that sets the database name for the destination connection.
2. Create a new Standard EXE project in the Visual Basic development environment. In the **Project/References** dialog box, select **Microsoft DTSPackage Object Library**.
3. Copy the following code for function **sCopyCustOrderProd** to the

code window for **Form1**.

4. Place a **command button** and another control, such as a check box, on the form **Form1**. In the **_Click** sub for the command button, call **sCopyCustOrderProd**, and use the other control to provide values for the parameter **TranSetOpt**.
5. You can add completion notification, such as a message box, and an error handler. For more information about returning meaningful error information, see [Handling DTS Errors in Visual Basic](#).
6. If you are using a database other than **DTSTest**, change the setting of the **Catalog** property of connection 2.
7. Run the project, providing the value **DTSTranSetOpt_Flattened** for the **TranSetOpt** parameter. View, truncate the destination tables, and then run the sample again with **TranSetOpt** set to **DTSTranSetOpt_Hierarchical**.

See Also

[Hierarchical Rowsets](#)

Parallel Data Driven Query Example

This sample Microsoft® Visual Basic® function, **sDDQTitleAuthors**, creates and runs a package that transforms parts of the **authors**, **titleauthor**, and **titles** tables from the **pubs** database that ships with Microsoft SQL Server™ 2000. The function generates a hierarchical rowset consisting of the authors from the **pubs** database and the titles with which they are associated.

Creating sDDQTitleAuthors Rowset

This function copies the data to tables in a database called **DTSTest** that have the following structure:

```
CREATE TABLE dbo.AuthNames (  
    AuthID VARCHAR (11) NOT NULL ,  
    LastName VARCHAR (40) NOT NULL ,  
    FirstName VARCHAR (20) NOT NULL )
```

```
CREATE TABLE dbo.TitleNames (  
    AuthID VARCHAR (11) NOT NULL ,  
    TitleName VARCHAR (80) NOT NULL )
```

As in flattened mode, the component rowsets are copied without regard to the chapters.

Running sDDQTitleAuthors

This example can be run on a computer on which Visual Basic 6.0 and SQL Server 2000 have been installed.

The steps for running **sDDQTitleAuthors** are as follows:

1. Create a database named **DTSTest** using SQL Server Enterprise Manager, and then create the tables defined earlier in **DTSTest**. If you use another database, change the line in the example that sets the database name for the destination connection.

2. Create a new Standard EXE project in the Visual Basic development environment. In the **Project/References** dialog box, check **Microsoft DTSPackage Object Library** and **Microsoft DTSPump Scripting Object Library**.
3. Copy the following code for function **sDDQTitleAuthors** to the code window for **Form1**.
4. Place a **command button** on the form **Form1**. In the **_Click** sub for the command button, call **sDDQTitleAuthors**.
5. You can add completion notification, such as a message box, and an error handler. For more information about returning meaningful error information, see [Handling DTS Errors in Visual Basic](#).
6. If you are using a database other than **DTSTest**, change the setting of the **Catalog** property of connection 2.
7. Run the project, click the command button, and then view the destination tables.

Writing sDDQTitleAuthors Code

This is the Visual Basic source code for the **sDDQTitleAuthors** function:

```
Private Function sDDQTitleAuthors() As String
    Dim oPackage      As New DTS.Package
    Dim oConnection   As DTS.Connection
    Dim oTask         As DTS.Task
    Dim oStep         As DTS.Step
    Dim oTransform    As DTS.Transformation
    Dim oScriptTransform As DTSPump.DataPumpTransformScript
```

```
Dim oTransformationSet As DTS.TransformationSet
Dim oParallelPumpTask As DTS.ParallelDataPumpTask
Dim sScript(1 To 3) As String
Dim sScriptLanguage As String
Dim sScriptFunction As String
```

```
Const SHAPE_PUBS_TITLEAUTHORS = _
    "SHAPE {SELECT au_id, au_lname, au_fname FROM authors} " & _
    "APPEND ({SELECT au_id, title FROM titleauthor TA, titles TS " & _
    "WHERE TA.title_id = TS.title_id} " & _
    "AS title_chap RELATE au_id TO au_id)"
```

```
Const SHAPE_DTSUE_TITLEAUTHORS = _
    "SHAPE {SELECT * FROM AuthNames} " & _
    "APPEND ({SELECT * FROM TitleNames} " & _
    "AS TitleChap RELATE AuthID TO AuthID)"
```

```
'----- generate scripts, one needs 2 col, other needs 3
```

```
sScriptLanguage = "VBScript"
```

```
sScriptFunction = "Transform"
```

```
sScript(1) = "Function Transform()" & vbCrLf & _
    "DTSDestination(1) = DTSSource(1)" & vbCrLf & _
    "DTSDestination(2) = DTSSource(2)" & vbCrLf
```

```
sScript(2) = "DTSDestination(3) = DTSSource(3)" & vbCrLf
```

```
sScript(3) = "Transform = DTSTransformStat_InsertQuery" & _
    vbCrLf & "End Function"
```

```
'----- define source connection - pubs
```

```
Set oConnection = oPackage.Connections.New("MSDataShape")
```

```
With oConnection
```

```
    .ConnectionProperties("Data Provider") = "SQLOLEDB"
```

```
    .ID = 1
```

```
    .Catalog = "pubs"
```

```

        .UserID = "sa"
    End With
    oPackage.Connections.Add oConnection

'----- define destination connection - (local) DTSTest
Set oConnection = oPackage.Connections.New("MSDataShape")
With oConnection
    .ConnectionProperties("Data Provider") = "SQLOLEDB"
    .ID = 2
    .DataSource = "(local)"
    .Catalog = "DTSTest"
    .UseTrustedConnection = True
End With
oPackage.Connections.Add oConnection

'----- Create ParallelDPTask set DDQ, connections and commands
Set oTask = oPackage.Tasks.New("DTSParallelDataPumpTask")
Set oParallelPumpTask = oTask.CustomTask
With oParallelPumpTask
    .TransformationSetOptions = DTSTranSetOpt_DataDrivenQuerie
    .SourceConnectionID = 1
    .SourceSQLStatement = SHAPE_PUBS_TITLEAUTHORS
    .DestinationConnectionID = 2
    .DestinationSQLStatement = SHAPE_DTSUE_TITLEAUTHOR
End With

'----- create TransformationSet for parent rowset
Set oTransformationSet = oParallelPumpTask. _
    TransformationSets.New("TransformSet_author")
oParallelPumpTask.TransformationSets.Add oTransformationSet
Set oTransform = oTransformationSet. _
    Transformations.New("DTS.DataPumpTransformScript")
Set oScriptTransform = oTransform.TransformServer

```

```
With oScriptTransform
    .Language = sScriptLanguage
    .FunctionEntry = sScriptFunction
    .Text = sScript(1) & sScript(2) & sScript(3)
End With
```

```
'----- define source/dest columns for parent
```

```
With oTransform
    .SourceColumns.AddColumn "au_id", 1
    .SourceColumns.AddColumn "au_lname", 2
    .SourceColumns.AddColumn "au_fname", 3
    .DestinationColumns.AddColumn "AuthID", 1
    .DestinationColumns.AddColumn "LastName", 2
    .DestinationColumns.AddColumn "FirstName", 3
    .Name = "Transform"
End With
```

```
'----- define INSERT query, params for parent
```

```
With oTransformationSet
    .InsertQuery = "INSERT AuthNames VALUES (?, ?, ?)"
    .InsertQueryColumns.AddColumn "AuthID", 1
    .InsertQueryColumns.AddColumn "LastName", 2
    .InsertQueryColumns.AddColumn "FirstName", 3
    .Transformations.Add oTransform
End With
```

```
'----- create TransformationSet for child rowset
```

```
Set oTransformationSet = oParallelPumpTask. _
    TransformationSets.New("TransformSet_title")
oParallelPumpTask.TransformationSets.Add oTransformationSet
Set oTransform = oTransformationSet. _
    Transformations.New("DTS.DataPumpTransformScript")
Set oScriptTransform = oTransform.TransformServer
```

```
With oScriptTransform
    .Language = sScriptLanguage
    .FunctionEntry = sScriptFunction
    .Text = sScript(1) & sScript(3)
End With
```

```
'----- define source/dest columns for child
With oTransform
    .SourceColumns.AddColumn "au_id", 1
    .SourceColumns.AddColumn "title", 2
    .DestinationColumns.AddColumn "AuthID", 1
    .DestinationColumns.AddColumn "TitleName", 2
    .Name = "Transform"
End With
```

```
'----- define INSERT query, params for child
With oTransformationSet
    .InsertQuery = "INSERT TitleNames VALUES (?, ?)"
    .InsertQueryColumns.AddColumn "AuthID", 1
    .InsertQueryColumns.AddColumn "TitleName", 2
    .Transformations.Add oTransform
End With
```

```
'----- add task, step to package
oTask.Name = "ParallelDDQTask"
With oPackage
    Set oStep = .Steps.New
    oStep.Name = "ParallelDPStep"
    oStep.TaskName = oTask.Name
    .Tasks.Add oTask
    .Steps.Add oStep
    .Name = "ParallelDDQTask Package"
    .FailOnError = True
```

.Execute 'run the package

End With
End Function

See Also

[Hierarchical Rowsets](#)

DTS Programming

SendMailTask Object

The **SendMailTask** object lets you send an e-mail as a task. For example, if you want to notify a database administrator about the success or failure of a particular task (such as a backup), you can link a **SendMailTask** object with a precedence constraint to the previous task. To use a **SendMailTask**, the computer must have the Microsoft® messaging API installed with a valid user profile.



A **SendMailTask** 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 and exception files, which contain information that changes constantly, and for which the file may not exist when the package is created (at design time).

Note If you enter an attachment file name and path that does not exist when the package is run, with some versions of the messaging API you receive the message: "Error sending mail: Internal MAPI error: the address book has no directories that contain names." This message indicates the file does not exist at the specified location, or that access permissions are not granted for the file. To fix the error, make sure that the file is available at the specified location when the package is run, or that access is granted.

Collections

[Properties Collection](#)

Properties

CCLine Property	Password Property
Description Property	Profile Property
FileAttachments Property	SaveMailInSentItemsFolder Property
IsNTService Property	Subject Property

MessageText Property	ToLine Property
Name Property	

Methods

Execute Method	Logon Method
GetDefaultProfileName Method	ResolveName Method
InitializeMAPI Method	ShowAddressBook Method
Logoff Method	

Example

The Microsoft Visual Basic® Sub **SendMailMsg** creates a Data Transformation Services (DTS) step and a **SendMailTask** object. It configures the task to send an e-mail message with attachment to a recipient named "IT Managers" and CC to "Data Center Operations":

```
Private Sub SendMailMsg(ByVal objPackage As DTS.Package2)
Dim objStep      As DTS.Step
Dim objTask      As DTS.Task
Dim objSendMail  As DTS.SendMailTask

'create step and task
Set objStep = objPackage.Steps.New
Set objTask = objPackage.Tasks.New("DTSSendMailTask")
Set objSendMail = objTask.CustomTask

'configure send mail task
With objSendMail
    .Name = "ErrorMailTask"
    .Profile = "Microsoft Outlook"
    .ToLine = "IT Managers"
```

```
.CCLine = "Data Center Operations"  
.Subject = "Error in DTS Nightly Job"  
.MessageText = "An error occurred loading data " & _  
                "warehouse. See attachment for details."  
.FileAttachments = "D:\DTS_UE\Messages\DTSError.txt"  
.IsNTService = True  
.SaveMailInSentItemsFolder = True  
End With
```

```
'link step to task  
objStep.TaskName = objSendMail.Name  
objStep.Name = "ErrorMailStep"  
objPackage.Steps.Add objStep  
objPackage.Tasks.Add objTask  
End Sub
```

DTS Programming

TransferObjectsTask Object

The **TransferObjectsTask** object allows you transfer one or more Microsoft® SQL Server™ objects between source and destination databases. An object can represent:

- A table, or table data.
- A view.
- A referential integrity constraint.
- A stored procedure.
- An index.
- A default or a rule.
- A user-defined data type.
- In addition, you can transfer all users or all logins (roles) for the source database. You can also transfer all objects dependent on the requested objects.

Note The source and destination must both be Microsoft SQL Server version 7.0 or later databases.



Collections

[Properties Collection](#)

Properties

CopyAllObjects Property	IncludeLogins Property
CopyData Property	IncludeUsers Property
CopySchema Property	Name Property
Description Property	ScriptFileDirectory Property
DestinationDatabase Property	ScriptOption Property
DestinationLogin Property	ScriptOptionEx Property
DestinationPassword Property	SourceDatabase Property
DestinationServer Property	SourceLogin Property
DestinationUseTrustedConnection Property	SourcePassword Property
DropDestinationObjectsFirst Property	SourceServer Property
IncludeDependencies Property	

Methods

AddObjectForTransfer Method	GetObjectForTransfer Method
CancelExecution Method	OnError Event
Execute Method	

Remarks

Certain errors can occur that are documented in an error message written to a log file named *server.database.LOG*, in the directory specified by the **ScriptFileDirectory** property. In some cases, these errors may not raise the **OnError** event, and may not be recorded in the Data Transformation Services (DTS) error file or the SQL Server log.

The **TransferObjectsTask** object is compatible with SQL Server 7.0. For

information about an updated version of this object, see [TransferObjectsTask2 Object](#).

Example

The Microsoft Visual Basic® Sub **RunTransfer** creates a DTS step and a **TransferObjectsTask** object. It configures the task to copy the tables **authors** and **employee**, the view **titleview**, and the stored procedure **byroyalty**, and all objects dependent on these, from the **pubs** database supplied with SQL Server 2000 to a database named **SomeOfPubs**.

```
Private Sub RunTransfer(ByVal objPackage As DTS.Package2)
Dim objStep      As DTS.Step
Dim objTask      As DTS.Task
Dim objXferObj   As DTS.TransferObjectsTask

'create step and task
Set objStep = objPackage.Steps.New
Set objTask = objPackage.Tasks.New("DTSTransferObjectsTask")
Set objXferObj = objTask.CustomTask

'configure transfer objects task
With objXferObj
    .Name = "XferObjTask"
    .SourceServer = "(local)"
    .SourceUseTrustedConnection = True
    .SourceDatabase = "pubs"
    .DestinationServer = "(local)"
    .DestinationUseTrustedConnection = True
    .DestinationDatabase = "SomeOfPubs"
    .ScriptFileDirectory = "D:\DTS_UE\Scripts"
    .CopyAllObjects = False
    .IncludeDependencies = True
    .IncludeLogins = False
    .IncludeUsers = False
End With
```

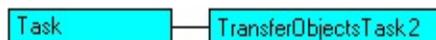
```
.DropDestinationObjectsFirst = True
.CopySchema = True
.CopyData = DTSTransfer_AppendData
.AddObjectForTransfer "authors", "dbo", DTSSQLObj_UserTable
.AddObjectForTransfer "employee", "dbo", DTSSQLObj_UserTable
.AddObjectForTransfer "titleview", "dbo", DTSSQLObj_View
.AddObjectForTransfer "byroyalty", "dbo", DTSSQLObj_StoredPro
End With
```

```
'link step to task
objStep.TaskName = objXferObj.Name
objStep.Name = "XferObjStep"
objPackage.Steps.Add objStep
objPackage.Tasks.Add objTask
End Sub
```

DTS Programming

TransferObjectsTask2 Object

The **TransferObjectsTask2** object transfers objects between instances of Microsoft® SQL Server™.



Extended Properties

DestTranslateChar Property	SourceTranslateChar Property
DestUseTransaction Property	

Remarks

The **TransferObjectsTask2** object extends the functionality of the existing **TransferObjectsTask** object and inherits the properties and methods of that object.

The **SourceTranslateChar** and **DestTranslateChar** properties turn character translation on or off at the source and destination, respectively. However, the **SourceTranslateChar** and **DestTranslateChar** properties are now largely unused as they only support translation of non-Unicode characters.

The transfer of SQL Server objects is done within a transaction on the destination server if the **DestUseTransaction** property is set.

If the **UseCollation** property is set, column-level collation settings on the source table are used when transferring data between computers running instances of SQL Server 2000.

For more information about when to use the **TransferObjectsTask** object instead of the **TransferObjectsTask2** object, see [Extended DTS Objects](#).

See Also

[TransferObjectsTask Object](#)

DTS Programming

Transformation Objects

This section describes the Data Transformation Services (DTS) transformation classes supplied with Microsoft® SQL Server™ 2000.

Topic	Description
DataPumpTransformCopy Object	Copies multiple source columns to destination columns.
DataPumpTransformDateTimeString Object	Converts datetime string formats.
DataPumpTransformLowerString Object	Converts multiple source columns to lowercase characters and copies them to destination columns.
DataPumpTransformMidString Object	Extracts substrings from a source column; optionally trims white space and changes case.
DataPumpTransformReadFile Object	Copies data read from files into destination columns.
DataPumpTransformScript Object	Transforms data with user-supplied Microsoft ActiveX® scripts.
DataPumpTransformTrimString Object	Removes white-space characters from data and optionally changes case.
DataPumpTransformUpperString Object	Converts multiple source columns to uppercase characters and copies them to destination columns.
DataPumpTransformWriteFile Object	Writes data from a source column into files.
DTSTransformScriptProperties2 Object	Supports multiphase transformations with an extended DataPumpTransformScript object.

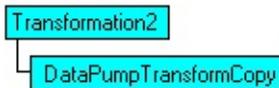
DTS Programming

DataPumpTransformCopy Object

The **DataPumpTransformCopy** object converts a source column to the destination column data type and moves the data to the destination column.

DataPumpTransformCopy supports multiple source and destination columns. Destination truncation is possible by setting

DTSTransformFlag_AllowStringTruncation in the **TransformFlags** property of the **Transformation2** object. There are no transformation properties.



For more information, see [DataPumpTransformLowerString Object](#). The **DataPumpTransformCopy** object is used the same way as the **DataPumpTransformLowerString** object in the example in that section, except that it is created as follows:

```
Set oTransform = oCustTask.Transformations. _  
    New("DTSPump.DataPumpTransformCopy")
```

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[Transformation2 Object](#)

[TransformFlags Property](#)

DTS Programming

DataPumpTransformDateTimeString Object

The **DataPumpTransformDateTimeString** object converts a datetime string in one format to another datetime format. It requires one source and one destination column, both of data types compatible with the OLE DB data type **DBTIMESTAMP**. The transformation properties **InputFormat** and **OutputFormat** specify the formats of the source and destination columns, respectively.



Properties

AMSymbol Property	Month??ShortName Property
Day?LongName Property	OutputFormat Property
Day?ShortName Property	PMSymbol Property
InputFormat Property	ShortYear2000Cutoff Property
Month??LongName Property	

Methods

GetDayLongName Method	SetDayLongName Method
GetDayShortName Method	SetDayShortName Method
GetMonthLongName Method	SetMonthLongName Method
GetMonthShortName Method	

Remarks

The **New** method of the **Transformations** collection of the **DataPumpTask2**, **DataDrivenQueryTask2**, and **TransformationSet** objects returns a reference to a **Transformation2** object. The **TransformServer** property of the **Transformation2** object returns a reference to the **DataPumpTransformDateTimeString** object.

Example

This example Microsoft® Visual Basic® program transforms a date column in the **employee** table of the **pubs** database, where dates are in a short date format, to column **HireDate** in table **Employee** in a Microsoft Access database **D:\DTS_UE\Data\jetPubs.mdb**. For example, **5/1/94** is converted to **May 01, 1994 (Sunday)**:

```
Public Sub Main()
```

```
'Copy/reformat pubs..employee.hire_date to Access DB.
```

```
Dim oPackage As DTS.Package
```

```
Dim oConnect As DTS.Connection
```

```
Dim oStep As DTS.Step
```

```
Dim oTask As DTS.Task
```

```
Dim oCustTask As DTS.DataPumpTask
```

```
Dim oTransform As DTS.Transformation
```

```
Dim oColumn As DTS.Column
```

```
Dim oDateTime As DTSPump.DataPumpTransformDateTimeStrin
```

```
Set oPackage = New DTS.Package
```

```
oPackage.FailOnError = True
```

```
'Establish connection to SQL Server DB.
```

```
Set oConnect = oPackage.Connections.New("SQLOLEDB.1")
```

```
With oConnect
```

```
    .ID = 1
```

```
    .DataSource = "(local)"
```

```
    .UseTrustedConnection = True
```

```
End With
```

```
oPackage.Connections.Add oConnect
```

```
'Establish connection to Access database.
```

```
Set oConnect = oPackage.Connections.New( _  
    "Microsoft.Jet.OLEDB.4.0")
```

```
oConnect.ID = 2
```

```
oConnect.DataSource = "D:\DTS_UE\Data\JetPubs.mdb"  
oPackage.Connections.Add oConnect
```

```
'Create step and task, link step to task.  
Set oStep = oPackage.Steps.New  
oStep.Name = "DateTimeStep"  
Set oTask = oPackage.Tasks.New("DTSDataPumpTask")  
Set oCustTask = oTask.CustomTask  
oCustTask.Name = "DateTimeTask"  
oStep.TaskName = oCustTask.Name  
oPackage.Steps.Add oStep
```

```
'Link task to connections.  
With oCustTask  
    .SourceConnectionID = 1  
    .SourceObjectName = "[pubs].[dbo].[employee]"  
    .DestinationConnectionID = 2  
    .DestinationObjectName = "Employee"  
End With
```

```
'Create custom transform, link to source and dest columns.  
Set oTransform = oCustTask.Transformations. _  
    New("DTSPump.DataPumpTransformDateTimeString")  
oTransform.Name = "DateTimeTransform"  
Set oColumn = oTransform.SourceColumns.New("hire_date", 1)  
oTransform.SourceColumns.Add oColumn  
Set oColumn = oTransform.DestinationColumns. _  
    New("HireDate", 1)  
oTransform.DestinationColumns.Add oColumn
```

```
'Define source and destination date formats.  
Set oDateTime = oTransform.TransformServer  
oDateTime.InputFormat = "M/d/yy"
```

```
oDateTime.OutputFormat = "MMMM dd, yyyy (dddd)"
```

```
'Link transform to task, task to package, and then run package.
```

```
oCustTask.Transformations.Add oTransform
```

```
oPackage.Tasks.Add oTask
```

```
oPackage.Execute
```

```
End Sub
```

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[New \(ID\) Method](#)

[Transformation2 Object](#)

[Transformations Collection](#)

[TransformationSet Object](#)

[TransformServer Property](#)

DTS Programming

DataPumpTransformLowerString Object

The **DataPumpTransformLowerString** object converts a source column to lowercase characters and, if necessary, to the destination column data type. It requires source and destination columns to be of string data types (**char**, **varchar**, **text**, **nchar**, **nvarchar**, **ntext**, and flat file strings). Like the **DataPumpTransformCopy** object, this transformation object supports multiple source and destination columns. Destination truncation is possible by setting **DTSTransformFlag_AllowStringTruncation** in the **TransformFlags** property of the **Transformation2** object. There are no custom transformation properties.



Remarks

Conversion to lowercase characters is also a feature of the **DataPumpTransformTrimString** and **DataPumpTransformMidString** objects.

Example

This example Microsoft® Visual Basic® program converts two columns from the **authors** table in the **pubs** database to lowercase characters while copying them to a table named **AuthNames** in a database named **DTS_UE**.

```
Public Sub Main()
```

```
'copy pubs..authors names to DTS_UE..AuthNames, making lower cas
```

```
    Dim oPackage As DTS.Package
```

```
    Dim oConnect As DTS.Connection
```

```
    Dim oStep As DTS.Step
```

```
    Dim oTask As DTS.Task
```

```
    Dim oCustTask As DTS.DataPumpTask
```

```
    Dim oTransform As DTS.Transformation
```

```
    Dim oColumn As DTS.Column
```

```
Set oPackage = New DTS.Package
oPackage.FailOnError = True
```

```
'establish connection to source server
Set oConnect = oPackage.Connections.New("SQLOLEDB.1")
With oConnect
    .ID = 1
    .DataSource = "(local)"
    .UseTrustedConnection = True
End With
oPackage.Connections.Add oConnect
```

```
'establish connection to destination server
Set oConnect = oPackage.Connections.New("SQLOLEDB.1")
With oConnect
    .ID = 2
    .DataSource = "(local)"
    .UseTrustedConnection = True
End With
oPackage.Connections.Add oConnect
```

```
'create step and task, link step to task
Set oStep = oPackage.Steps.New
oStep.Name = "LowerCaseStep"
Set oTask = oPackage.Tasks.New("DTSDDataPumpTask")
Set oCustTask = oTask.CustomTask
oCustTask.Name = "LowerCaseTask"
oStep.TaskName = oCustTask.Name
oPackage.Steps.Add oStep
```

```
'link task to connections
With oCustTask
    .SourceConnectionID = 1
```

```
.SourceObjectName = "pubs..authors"  
.DestinationConnectionID = 2  
.DestinationObjectName = "[DTS_UE].[dbo].[AuthNames]"  
End With
```

```
'create custom transform, link to source and dest columns  
Set oTransform = oCustTask.Transformations. _  
    New("DTSPump.DataPumpTransformLowerString")  
With oTransform  
    .Name = "LowerCaseTransform"  
    .SourceColumns.AddColumn "au_fname", 1  
    .SourceColumns.AddColumn "au_lname", 2  
    .DestinationColumns.AddColumn "FirstName", 1  
    .DestinationColumns.AddColumn "LastName", 2  
End With
```

```
'link transform to task, task to package, run package  
oCustTask.Transformations.Add oTransform  
oPackage.Tasks.Add oTask  
oPackage.Execute  
End Sub
```

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataPumpTransformCopy Object](#)

[DataPumpTransformMidString Object](#)

[DataPumpTransformTrimString Object](#)

[DataPumpTransformUpperString Object](#)

[Transformation2 Object](#)

TransformFlags Property

DTS Programming

DataPumpTransformMidString Object

The **DataPumpTransformMidString** object extracts a substring from the source column and converts it, if necessary, to the destination column data type. This object requires one source column and one destination column, both of a string data type (**char**, **varchar**, **text**, **nchar**, **nvarchar**, **ntext**, and flat file strings). The properties **CharacterStart** and **CharacterCount** specify the position of the substring.

Optionally, the transformation converts the extracted substring to uppercase or lowercase characters, as specified by the **UpperCaseString** and **LowerCaseString** properties. It also optionally trims white-space characters, as specified by the **TrimLeadingWhiteSpace**, **TrimTrailingWhiteSpace**, and **TrimEmbeddedWhiteSpace** properties. Substring extraction occurs before the trimming of white space characters.

Destination truncation is possible by setting **DTSTransformFlag_AllowStringTruncation** in the **TransformFlags** property of the **Transformation2** object.



Properties

CharacterCount Property	TrimLeadingWhiteSpace Property
CharacterStart Property	TrimTrailingWhiteSpace Property
LowerCaseString Property	UpperCaseString Property
TrimEmbeddedWhiteSpace Property	

Remarks

If only case conversion is required, the **DataPumpTransformLowerString** or **DataPumpTransformUpperString** objects can be used. These transformations accept multiple source and destination columns. If only case conversion and trimming of white space is required, the **DataPumpTransformTrimString** object can be used.

The **New** method of the **Transformations** collection of the **DataPumpTask**, **DataDrivenQueryTask**, and **TransformationSet** objects returns a reference to a **Transformation** object. The **TransformServer** property of the **Transformation** object returns a reference to the appropriate custom transformation object.

Example

This example Microsoft® Visual Basic® program splits the phone column from the **authors** table in the **pubs** database into two columns named **AreaCode** and **LocalPhone** while copying it to a Microsoft Excel worksheet named **Phones** in **PubsAuthors.xls**. The worksheet had been previously created by the Microsoft OLE DB Provider for Jet from:

```
CREATE TABLE `Phones` (  
  `AreaCode` VarChar (255) ,  
  `LocalPhone` VarChar (255) )
```

The basic steps for manually creating a worksheet are as follows:

1. Open a new workbook in Excel and rename one of the blank sheets **Phones**.
2. Enter **AreaCode** in cell A1 and **LocalNumber** in cell B1.
3. Save the workbook as **PubsAuthors.xls**, and then close Excel.

```
Public Sub Main()
```

```
'Copy pubs..authors.phone to Excel. Split out area code and local numl
```

```
  Dim oPackage As DTS.Package
```

```
  Dim oConnect As DTS.Connection
```

```
  Dim oStep As DTS.Step
```

```
  Dim oTask As DTS.Task
```

```
  Dim oCustTask As DTS.DataPumpTask
```

```
  Dim oTransform As DTS.Transformation
```

```
  Dim oColumn As DTS.Column
```

```
  Dim oMidString As DTSPump.DataPumpTransformMidString
```

```
Set oPackage = New DTS.Package
oPackage.FailOnError = True
```

'Establish a connection to the source server.

```
Set oConnect = oPackage.Connections.New("SQLOLEDB.1")
```

```
With oConnect
```

```
    .ID = 1
```

```
    .DataSource = "(local)"
```

```
    .UseTrustedConnection = True
```

```
End With
```

```
oPackage.Connections.Add oConnect
```

```
Set oConnect = Nothing
```

'Establish a connection to an Excel worksheet.

```
Set oConnect = oPackage.Connections.New("Microsoft.Jet.OLEDB
```

```
With oConnect
```

```
    .ID = 2
```

```
    .DataSource = "D:\DTS_UE\Data\PubsAuthors.xls"
```

```
    .ConnectionProperties("Extended Properties").Value = _
        "Excel 8.0;HDR=YES;"
```

```
End With
```

```
oPackage.Connections.Add oConnect
```

```
Set oConnect = Nothing
```

'Create a step and task, and then link the step to the task.

```
Set oStep = oPackage.Steps.New
```

```
oStep.Name = "MidStringStep"
```

```
Set oTask = oPackage.Tasks.New("DTSDataPumpTask")
```

```
Set oCustTask = oTask.CustomTask
```

```
oCustTask.Name = "MidStringTask"
```

```
oStep.TaskName = oCustTask.Name
```

```
oPackage.Steps.Add oStep
```

'Link the task to the connections.

With oCustTask

```
.SourceConnectionID = 1  
.SourceObjectName = "[pubs].[dbo].[authors]"  
.DestinationConnectionID = 2  
.DestinationObjectName = "Phones$"
```

End With

'Create an area code transform, and link it to source and destination

Set oTransform = oCustTask.Transformations. _

```
    New("DTSPump.DataPumpTransformMidString")
```

With oTransform

```
.Name = "AreaCodeTransform"  
.SourceColumns.AddColumn "phone", 1  
.DestinationColumns.AddColumn "AreaCode", 1
```

End With

'Define start and width for area code, and then link transform to task

Set oMidString = oTransform.TransformServer

```
oMidString.CharacterStart = 1
```

```
oMidString.CharacterCount = 3
```

```
oCustTask.Transformations.Add oTransform
```

'Create local numeric transform, and then link to source and destination

Set oTransform = oCustTask.Transformations. _

```
    New("DTSPump.DataPumpTransformMidString")
```

With oTransform

```
.Name = "LocalNumTransform"  
.SourceColumns.AddColumn "phone", 1  
.DestinationColumns.AddColumn "LocalNumber", 1
```

End With

```
'Define start and width for local number.  
Set oMidString = oTransform.TransformServer  
oMidString.CharacterStart = 5  
oMidString.CharacterCount = 8
```

```
'Link transform to task and task to package. Then run the package.  
oCustTask.Transformations.Add oTransform  
oPackage.Tasks.Add oTask  
oPackage.Execute
```

End Sub

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DataPumpTransformLowerString Object](#)

[DataPumpTransformTrimString Object](#)

[DataPumpTransformUpperString Object](#)

[New \(ID\) Method](#)

[Transformation2 Object](#)

[Transformations Collection](#)

[TransformationSet Object](#)

[TransformFlags Property](#)

[TransformServer Property](#)

DTS Programming

DataPumpTransformReadFile Object

The **DataPumpTransformReadFile** object copies the contents of a file, the name of which is specified by a source column, to a destination column.

Data conversion is controlled by the **OEMFile** and **UnicodeFile** properties. If the file named by the source column contains the Unicode prefix bytes (hex FFFE), the file is assumed to be Unicode regardless of the value of **UnicodeFile**, and the prefix bytes are skipped.



Properties

ErrorIfFileNotFound Property	OEMFile Property
FilePath Property	

Remarks

If the file name column contains a path, it can use either a drive letter or a universal naming convention (UNC) file specification. If no path is present, the **FilePath** property can be used to supply the path. However, **FilePath** is always used when it is nonempty, even when the file name column contains a path.

The **New** method of the **Transformations** collection of the **DataPumpTask**, **DataDrivenQueryTask**, and **TransformationSet** objects returns a reference to a **Transformation2** object. The **TransformServer** property of the **Transformation2** object returns a reference to the appropriate custom transformation object.

Example

This example Microsoft® Visual Basic® program transforms a column in a Microsoft Excel sheet to a database column using the **Read File** custom transformation. Column **file_name** (row 1 of the column contains the label "file_name") of worksheet **FileSpecTwo** in **D:\DTS_UE\Source\FileSpecs.xls**

contains the file names. The transformation writes the file data to column **file_data** in table **FileDataOut** in database **DTS_UE** on the local server:

Public Sub Main()

'Read file names from Excel worksheet. Write file data to database col

```
Dim oPackage As DTS.Package
Dim oConnect As DTS.Connection
Dim oStep As DTS.Step
Dim oTask As DTS.Task
Dim oCustTask As DTS.DataPumpTask
Dim oTransform As DTS.Transformation
Dim oColumn As DTS.Column
Dim oReadFile As DTSPump.DataPumpTransformReadFile
```

```
Set oPackage = New DTS.Package
oPackage.FailOnError = True
```

'Establish connection to source Excel worksheet.

```
Set oConnect = oPackage.Connections.New("Microsoft.Jet.OLEDB
```

```
With oConnect
```

```
  .ID = 1
```

```
  .DataSource = "D:\DTS_UE\Source\FileSpecs.xls"
```

```
  .ConnectionProperties("Extended Properties") = _
    "Excel 8.0;HDR=YES;"
```

```
End With
```

```
oPackage.Connections.Add oConnect
```

'Establish connection to the destination server.

```
Set oConnect = oPackage.Connections.New("SQLOLEDB.1")
```

```
With oConnect
```

```
  .ID = 2
```

```
  .DataSource = "(local)"
```

```
  .UseTrustedConnection = True
```

```
End With
```

oPackage.Connections.Add oConnect

'Create the step and task, and link the step to the task

Set oStep = oPackage.Steps.New

oStep.Name = "ReadFileStep"

Set oTask = oPackage.Tasks.New("DTSDDataPumpTask")

Set oCustTask = oTask.CustomTask

oCustTask.Name = "ReadFileTask"

oStep.TaskName = oCustTask.Name

oPackage.Steps.Add oStep

'Link the task to the connections, and specify worksheet, database ar

With oCustTask

.SourceConnectionID = 1

.SourceObjectName = "FileSpecTwo\$"

.DestinationConnectionID = 2

.DestinationObjectName = "DTS_UE..FileDataOut"

End With

'Create transform, and link it to the source and destination columns.

Set oTransform = oCustTask.Transformations. _

New("DTSPump.DataPumpTransformReadFile")

oTransform.Name = "ReadFileTransform"

Set oColumn = oTransform.SourceColumns.New("file_name", 1)

oTransform.SourceColumns.Add oColumn

Set oColumn = oTransform.DestinationColumns. _

New("file_data", 1)

oTransform.DestinationColumns.Add oColumn

'Define error action and path prefix.

Set oReadFile = oTransform.TransformServer

oReadFile.ErrorIfFileNotFound = False

oReadFile.FilePath = "D:\DTS_UE"

```
'Link transform to task and task to package. Then run the package.  
oCustTask.Transformations.Add oTransform  
oPackage.Tasks.Add oTask  
oPackage.Execute  
End Sub
```

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[New \(ID\) Method](#)

[Transformation2 Object](#)

[Transformations Collection](#)

[TransformationSet Object](#)

[TransformServer Property](#)

DTS Programming

DataPumpTransformScript Object

The **DataPumpTransformScript** object transforms source columns and moves data to the destination columns using a Microsoft® ActiveX® script. Columns can be transformed in any way supported by the scripting language. The driver for the specific ActiveX scripting language must be installed.

The **DataPumpTransformScript** object supports properties that are used to specify the script text, scripting language, and entry point name.



Properties

FunctionEntry Property	Text Property
Language Property	

Remarks

Valid script languages available on a particular system can be determined by enumerating the **ScriptingLanguageInfos** collection of the **Application** object. For more information about the scripting language appropriate for use with Data Transformation Services (DTS), see [ScriptingLanguageInfo Object](#).

The **DataPumpTransformScript** object is compatible with Microsoft SQL Server™ version 7.0. For more information about an extended version of this object, see [DTSTransformScriptProperties2 Object](#).

See Also

[Adding DTS ActiveX Scripts](#)

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[Application Object](#)

[DTS Scripting Reference](#)

[ScriptingLanguageInfos Collection](#)

DTS Programming

DataPumpTransformTrimString Object

The **DataPumpTransformTrimString** object converts the source column to uppercase or lowercase characters, as specified by the **UpperCaseString** and **LowerCaseString** properties. It trims white-space characters, as specified by the **TrimLeadingWhiteSpace**, **TrimTrailingWhiteSpace** and **TrimEmbeddedWhiteSpace** properties. It converts, if necessary, to the destination column data type. It requires one source column and one destination column, both of a string data type (**char**, **varchar**, **text**, **nchar**, **nvarchar**, **ntext**, and flat file strings).

Destination truncation is possible by setting **DTSTransformFlag_AllowStringTruncation** in the **TransformFlags** property of the **Transformation** object.



Properties

LowerCaseString Property	TrimTrailingWhiteSpace Property
TrimEmbeddedWhiteSpace Property	UpperCaseString Property
TrimLeadingWhiteSpace Property	

Remarks

The **DataPumpTransformMidString** object also performs case changing and white space removal functions. If only case conversion is required, the **DataPumpTransformLowerString** or **DataPumpTransformUpperString** objects can be used. These transformations accept multiple source and destination columns.

The **New** method of the **Transformations** collection of the **DataPumpTask**, **DataDrivenQueryTask**, and **TransformationSet** objects returns a reference to a **Transformation2** object. The **TransformServer** property of the **Transformation2** object returns a reference to the appropriate custom transformation object.

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DataPumpTransformLowerString Object](#)

[DataPumpTransformMidString Object](#)

[DataPumpTransformUpperString Object](#)

[New \(ID\) Method](#)

[Transformation2 Object](#)

[Transformations Collection](#)

[TransformationSet Object](#)

[TransformFlags Property](#)

[TransformServer Property](#)

DTS Programming

DataPumpTransformUpperString Object

The **DataPumpTransformUpperString** object converts a source column to uppercase characters and, if necessary, to the destination column data type. It requires source and destination columns to be of string data types (**char**, **varchar**, **text**, **nchar**, **nvarchar**, **ntext**, and flat file strings). Like the **DataPumpTransformCopy** object, this transformation object supports multiple source and destination columns. Destination truncation is possible by setting **DTSTransformFlag_AllowStringTruncation** in the **TransformFlags** property of the **Transformation2** object. There are no custom transformation properties.



Remarks

Conversion to uppercase characters is also a feature of the **DataPumpTransformTrimString** and **DataPumpTransformMidString** objects.

For more information, see [DataPumpTransformLowerString Object](#). The **DataPumpTransformUpperString** object is used the same way as the **DataPumpTransformLowerString** object in the example in that section, except that it is created as follows:

```
Set oTransform = oCustTask.Transformations. _  
    New("DTSPump.DataPumpTransformUpperString")
```

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataPumpTransformCopy Object](#)

[DataPumpTransformMidString Object](#)

[DataPumpTransformTrimString Object](#)

[Transformation2 Object](#)

[TransformFlags Property](#)

DTS Programming

DataPumpTransformWriteFile Object

The **DataPumpTransformWriteFile** object converts a field from one source column into a file, the path of which is specified by another source column. Columns in the destination connection of the task are not written, although the connection must exist.

Data conversion is controlled by the **OEMFile** and **UnicodeFile** properties. If **UnicodeFile** is set to TRUE, the Unicode file header (hex FFFE) is prepended to the file, if it is not already there. The default behavior is to overwrite the destination file if it exists already.



Properties

AppendIfFileExists Property	FilePath Property
ErrorIfFileExists Property	OEMFile Property
FileColumnName Property	

Remarks

The data column must be a string or binary data type. If it is NULL, no file is written. If **AppendIfFileExists** is set to FALSE and the file exists, it is deleted. If the file is empty, a zero-length file is created. The file name column cannot be NULL or empty. If the file name column contains a path, it can use either a drive letter or a universal naming convention (UNC) file specification.

If no path is present, the **FilePath** property can be used to supply the path. However, **FilePath** is always used when it is nonempty, even when the file name column contains a path.

This transformation object does not write destination columns, but a destination connection must still be provided. If no other transformations write columns, no rows are written.

You must explicitly add the source columns to the **SourceColumns** collection,

even if the source connection has only two columns. If you do not add the columns, the transformation assumes you are including all the source and destination columns. This causes an error because the transformation cannot reference destination columns.

The **New** method of the **Transformations** collection of the **DataPumpTask**, **DataDrivenQueryTask** and **TransformationSet** objects returns a reference to a **Transformation2** object. The **TransformServer** property of the **Transformation2** object returns a reference to the appropriate custom transformation object.

Example

This example Microsoft® Visual Basic® program transforms a database column to written flat files with a **Write File** custom transformation. In table **WriteFileData** in database **DTS_UE**, column **file_spec** provides the file names, and **file_data** provides the data.

```
Public Sub Main()
```

```
'Write the data in DTS_UE. WriteFileData to specified files.
```

```
    Dim oPackage As DTS.Package
```

```
    Dim oConnect As DTS.Connection
```

```
    Dim oStep As DTS.Step
```

```
    Dim oTask As DTS.Task
```

```
    Dim oCustTask As DTS.DataPumpTask
```

```
    Dim oTransform As DTS.Transformation
```

```
    Dim oColumn As DTS.Column
```

```
    Dim oWriteFile As DTSPump.DataPumpTransformWriteFile
```

```
    Set oPackage = New DTS.Package
```

```
    oPackage.FailOnError = True
```

```
'Establish a connection to the source server.
```

```
    Set oConnect = oPackage.Connections.New("SQLOLEDB.1")
```

```
    With oConnect
```

```
        .ID = 1
```

```
.DataSource = "(local)"  
.UseTrustedConnection = True  
End With  
oPackage.Connections.Add oConnect
```

```
'Establish connection to the (dummy) destination server.  
Set oConnect = oPackage.Connections.New("SQLOLEDB.1")  
With oConnect  
.ID = 2  
.DataSource = "(local)"  
.UseTrustedConnection = True  
End With  
oPackage.Connections.Add oConnect
```

```
'Create a step and task, and then link the step to the task.  
Set oStep = oPackage.Steps.New  
oStep.Name = "WriteFileStep"  
Set oTask = oPackage.Tasks.New("DTSDataPumpTask")  
Set oCustTask = oTask.CustomTask  
oCustTask.Name = "WriteFileTask"  
oStep.TaskName = oCustTask.Name  
oPackage.Steps.Add oStep
```

```
'Link the task to the connections, and specify tables.  
With oCustTask  
.SourceConnectionID = 1  
.SourceObjectName = "DTS_UE.dbo.WriteFileData"  
.DestinationConnectionID = 2  
.DestinationObjectName = "DTS_UE.dbo.WriteFileData"  
End With
```

```
'Create transform, and link it to source columns.  
Set oTransform = oCustTask.Transformations. _
```

```

        New("DTSPump.DataPumpTransformWriteFile")
With oTransform
    .Name = "WriteFileTransform"
    .SourceColumns.AddColumn "file_spec", 1
    .SourceColumns.AddColumn "file_data", 2
End With

'Define error action and path prefix.
Set oWriteFile = oTransform.TransformServer
With oWriteFile
    .ErrorIfFileExists = False
    .FileColumnName = "file_spec"
    .AppendIfFileExists = True
    .FilePath = "D:\DTS_UE"
End With

'Link transform to the task, the task to the package, and run the pack
oCustTask.Transformations.Add oTransform
oPackage.Tasks.Add oTask
oPackage.Execute
End Sub

```

See Also

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[New \(ID\) Method](#)

[Transformation2 Object](#)

[Transformations Collection](#)

[TransformationSet Object](#)

[TransformServer Property](#)

DTS Programming

DTSTransformScriptProperties2 Object

The **DTSTransformScriptProperties2** object transforms source columns and moves data to the destination columns using a Microsoft® ActiveX® script. Columns can be transformed in any way supported by the scripting language being used.



Extended Properties

BatchCompleteFunctionEntry Property	PreSourceDataFunctionEntry Property
InsertFailureFunctionEntry Property	PumpCompleteFunctionEntry Property
InsertSuccessFunctionEntry Property	TransformFailureFunctionEntry Property
PostSourceDataFunctionEntry Property	

Remarks

The **DTSTransformScriptProperties2** object extends the functionality of the **DataPumpTransformScript** and inherits the properties and methods of that object. In addition, the extended object supports multiple transformation phases and adds properties to specify the script function entry point for each supported phase.

The following table specifies the transformation phases and the property that specifies the entry point for the phase. For a phase to be supported by a **DTSTransformScriptProperties2** object, the function entry point must be specified, and the phase must be specified using the **TransformPhases** property of the **Transformation2** object.

Phase	Description	Entry Point Property

PreSourceData	Occurs before first row is fetched from source connection.	PreSourceDataFunc
Transform	Occurs after each source row is fetched and before the destination row is written.	FunctionEntry
OnTransformFailure	Occurs after a failure in the Transform phase, indicated by the return of DTSTransformStat_Error or DTSTransformStat_ExceptionRow . Typically this failure is caused by conversion errors.	TransformFailureFu
OnInsertSuccess	Occurs after each data row is written successfully to the destination connection.	InsertSuccessFunci
OnInsertFailure	Occurs after each attempt to write a data row to the destination connection fails (for example, by attempting to write a duplicate value to a primary key field, or a NULL to a NOT NULL field).	InsertFailureFunci
OnBatchComplete	Occurs in DataPumpTask2 when using the FastLoad option after each batch is written, successfully or unsuccessfully.	BatchCompleteFunc
PostSourceData	Occurs after the last row is written to the destination connection.	PostSourceDataFun
OnPumpComplete	Occurs at the end of the execution of the task.	PumpCompleteFunc

To create the **DTSTransformScriptProperties2** object, declare an object variable or pointer of type **DTSTransformScriptProperties2**, but use the **ProgID** of the **DataPumpTransformScript** object as the parameter for the **New** method of the **Transformations** collection. The following Microsoft Visual Basic® code illustrates this:

```

Dim objCustTask As DTS.DataPumpTask2
Dim objTransform As DTS.Transformation2
Dim objTranScript As DTSPump.DTSTransformScriptProperties2
...
Set objTransform = _
    objCustTask.Transformations.New("DTSPump.DataPumpTransfc
Set objTranScript = objTransform.TransformServer

```

For more information about when to use the **DataPumpTransformScript** object instead of the **DTSTransformScriptProperties2** object, see [Extended DTS Objects](#).

Example

The following Visual Basic program uses multiphased transformations to insert data into a table and correct certain errors.

The data source is a table named **Transactions** in a database named **DailyRuns**. The following is the definition of **Transactions**:

```

CREATE TABLE dbo.Transactions (
    CustID INT PRIMARY KEY ,
    CustName VARCHAR (50) NULL ,
    CustAddr VARCHAR (100) NULL ,
    TransAmount VARCHAR (50) NOT NULL )

```

The data destination is two tables named **Transactions** and **ErrorAmounts** in a database named **DataPerm**. The following are the definitions of these tables:

```

CREATE TABLE dbo.Transactions (
    CustID INT PRIMARY KEY ,
    CustName VARCHAR (50) NOT NULL ,
    CustAddr VARCHAR (50) NOT NULL ,
    TransAmount MONEY NOT NULL ,
    ErrorCount INT NOT NULL )

```

```
CREATE TABLE dbo.ErrorAmounts (  
    CustID INT NOT NULL ,  
    TransAmount VARCHAR (50) NOT NULL )
```

The example program inserts the data from **DailyRuns..Transactions**

- If the **CustName** or **CustAddr** columns in the source contain Null values, they cannot be inserted into the corresponding columns in the destination, which are specified NOT NULL.
- An error occurs if the **TransAmount** column in the source, which is **varchar**, cannot be converted to **money**.
- An error occurs if the destination already contains a row that has the same value for the primary key **CustID** as the source row.

One or more of these errors can occur in a single source row.

The example uses a **DataPumpTransformCopy** transformation that supports the Transform phase. The transformation inserts rows from **DailyRuns..Transactions** into **DataPerm..Transactions**. A **SourceSQLStatement** provides an initial value for the **ErrorCount** column and correctly maps the data source to the destination table.

The example also uses a **DTSTransformScriptProperties2** transformation that supports the OnTransformFailure, OnInsertFailure and PreSourceData phases. The transformation processes the errors described above. It contains these ActiveX script functions to support the transformation phases.

Script Function	Transformation Phase
TransformFailed	OnTransformFailure
InsertFailed	OnInsertFailure
InitializeGV	PreSourceData

The **TransformFailed** function determines if a conversion error to the **money** data type occurred. If it did, the function opens a Microsoft ActiveX Data

Objects (ADO) recordset on the **ErrorAmounts** table and writes a record containing the **CustID** and **TransAmount** columns from the data source. It sets the destination columns **TransAmount** to 0.0 and **ErrorCount** to 1. It saves the current source row number in a global variable so the **InsertFailed** function can determine if **TransformFailed** found an error converting to **money** for the current row. If either of the source columns **CustName** or **CustAddr** are NULL, it sets the corresponding destination column to "<unknown>".

The **InsertFailed** function executes when a duplicate primary key error occurs. It opens an ADO recordset and queries for the existing record in the **DataPerm..Transactions** table. It updates the **CustName** and **CustAddr** columns only if the original values in the source row were not NULL. If the **TransformFailed** function found that an error converting to **money** occurred, **InsertFailed** increments the **ErrorCount** column. Otherwise it adds the **TransAmount** column from the source row to the **TransAmount** column of the existing record.

The **InitializeGV** function initializes the global variable that is used to determine if both a transform failure and an insert failure occurred on the same row.

For more information about these script functions and a view of them separated from their text string, see [Phased Transformation Samples](#).

Note Database operations using ADO from ActiveX scripts cannot be made to join the package transaction that DTS manages.

Building the Multiphased Example

This is the Visual Basic code for the multi-phased transformation example:

```
Public Sub Main()  
'Copy DailyRuns..Transactions to DataPerm..Transactions.  
    Dim objPackage    As DTS.Package2  
    Dim objConnect    As DTS.Connection2  
    Dim objStep       As DTS.Step2  
    Dim objTask       As DTS.Task  
    Dim objCustTask   As DTS.DataPumpTask2  
    Dim objTransform  As DTS.Transformation2
```

```
Dim objTranScript As DTSPump.DTSTransformScriptProperties2
Dim strVBS As String 'Assemble VBScript text.
```

```
Set objPackage = New DTS.Package
objPackage.FailOnError = True
```

```
'Establish connection to source server.
Set objConnect = objPackage.Connections.New("SQLOLEDB")
With objConnect
    .ID = 1
    .DataSource = "(local)"
    .UseTrustedConnection = True
End With
objPackage.Connections.Add objConnect
```

```
'Establish connection to destination server.
Set objConnect = objPackage.Connections.New("SQLOLEDB")
With objConnect
    .ID = 2
    .DataSource = "(local)"
    .UseTrustedConnection = True
End With
objPackage.Connections.Add objConnect
```

```
'Create step and task, and link step to task.
Set objStep = objPackage.Steps.New
objStep.Name = "CopyNValidateStep"
Set objTask = objPackage.Tasks.New("DTSDDataPumpTask")
Set objCustTask = objTask.CustomTask
objCustTask.Name = "CopyNValidateTask"
objStep.TaskName = objCustTask.Name
objPackage.Steps.Add objStep
```

'Link task to connections.

With objCustTask

.SourceConnectionID = 1

.SourceSQLStatement = _

"SELECT CustID, CustName, CustAddr, TransAmount, 0 AS I
"FROM DailyRuns..Transactions"

.DestinationConnectionID = 2

.DestinationObjectName = "[DataPerm].[dbo].[Transactions]"

.UseFastLoad = False

.MaximumErrorCount = 4

End With

'Create transform to copy source to destination data.

Set objTransform = objCustTask.Transformations. _

New("DTSPump.DataPumpTransformCopy")

With objTransform

.Name = "CopyData"

.TransformPhases = DTSTransformPhase_Transform

.TransformFlags = DTSTransformFlag_Default

End With

objCustTask.Transformations.Add objTransform

'Create transform to handle error conditions.

Set objTransform = objCustTask.Transformations. _

New("DTSPump.DataPumpTransformScript")

With objTransform

.Name = "InsertFailure"

.TransformPhases = DTSTransformPhase_OnInsertFailure + _

DTSTransformPhase_OnTransformFailure + DTSTransformPh

Set objTranScript = .TransformServer

End With

'Define the ActiveX script functions.

With objTranScript

```
.InsertFailureFunctionEntry = "InsertFailed"
```

```
.TransformFailureFunctionEntry = "TransformFailed"
```

```
.PreSourceDataFunctionEntry = "InitializeGV"
```

```
.Language = "VBScript"
```

```
strVBS = "Option Explicit" & vbCrLf
```

```
strVBS = strVBS & "Function InsertFailed()" & vbCrLf
```

```
strVBS = strVBS & "  Dim rstCustomers " & vbCrLf & "  Dim s
```

```
strVBS = strVBS & "  strConnect = ""Provider=SQLOLEDB;Da
```

```
    "Initial Catalog=DataPerm;User Id=sa;Password=; ""
```

```
strVBS = strVBS & "  strQuery = ""SELECT CustID, CustName
```

```
    "FROM Transactions WHERE CustID = """" & vbCrLf
```

```
strVBS = strVBS & "  'open recordset" & vbCrLf
```

```
strVBS = strVBS & "  Set rstCustomers = CreateObject(""ADOD
```

```
strVBS = strVBS & "  rstCustomers.LockType = 3
```

```
strVBS = strVBS & "  strQuery = strQuery & DTSSource(""Cust
```

```
strVBS = strVBS & "  rstCustomers.Open strQuery, strConnect, ,
```

```
strVBS = strVBS & "  If rstCustomers.EOF Then" & vbCrLf
```

```
strVBS = strVBS & "    rstCustomers.AddNew" & vbCrLf
```

```
strVBS = strVBS & "    rstCustomers(""CustID"") = DTSSource
```

```
strVBS = strVBS & "  End If" & vbCrLf
```

```
strVBS = strVBS & "  If DTSSource(""CustName"") <> ""<unkr
```

```
strVBS = strVBS & "    rstCustomers(""CustName"") = DTSSou
```

```
strVBS = strVBS & "  End If" & vbCrLf
```

```
strVBS = strVBS & "  If DTSSource(""CustAddr"") <> ""<unkn
```

```
strVBS = strVBS & "    rstCustomers(""CustAddr"") = DTSSou
```

```
strVBS = strVBS & "  End If" & vbCrLf
```

```
strVBS = strVBS & "  If CLng(DTSTransformPhaseInfo.Curren
```

```
strVBS = strVBS & "    rstCustomers(""TransAmount"") = rstCu
```

```
strVBS = strVBS & "  Else" & vbCrLf
```

```
strVBS = strVBS & "    rstCustomers(""ErrorCount"") = rstCust
```

```
strVBS = strVBS & "  End If" & vbCrLf
```

```
strVBS = strVBS & " rstCustomers.Update" & vbCrLf
strVBS = strVBS & " rstCustomers.Close" & vbCrLf
strVBS = strVBS & " InsertFailed = DTSTransformStat_OK" &
strVBS = strVBS & "End Function" & vbCrLf
```

```
strVBS = strVBS & "Function TransformFailed()" & vbCrLf
strVBS = strVBS & " Dim rstErrors " & vbCrLf & " Dim strCo
strVBS = strVBS & " DTSDestination("CustID") = DTSSourc
strVBS = strVBS & " DTSDestination("ErrorCount") = 0 " &
strVBS = strVBS & " On Error Resume Next" & vbCrLf
strVBS = strVBS & " DTSDestination("TransAmount") = CCu
strVBS = strVBS & " If Err.Number <> 0 Then " & vbCrLf
strVBS = strVBS & " On Error GoTo 0 " & vbCrLf
strVBS = strVBS & " strConnect = "Provider=SQLOLEDB;I
"Initial Catalog=DataPerm;User Id=sa;Password=;
strVBS = strVBS & " Set rstErrors = CreateObject("ADODB
strVBS = strVBS & " rstErrors.LockType = 3 'a
strVBS = strVBS & " rstErrors.Open "ErrorAmounts", strCo
strVBS = strVBS & " rstErrors.AddNew" & vbCrLf
strVBS = strVBS & " rstErrors("CustID") = DTSSource("C
strVBS = strVBS & " rstErrors("TransAmount") = DTSSour
strVBS = strVBS & " rstErrors.Update" & vbCrLf
strVBS = strVBS & " rstErrors.Close" & vbCrLf
strVBS = strVBS & " DTSDestination("TransAmount") = 0.
strVBS = strVBS & " DTSDestination("ErrorCount") = 1" &
strVBS = strVBS & " DTSGlobalVariables("LastErrorRow")
strVBS = strVBS & " End If" & vbCrLf
strVBS = strVBS & " On Error GoTo 0 " & vbCrLf
strVBS = strVBS & " If IsNull(DTSSource("CustName").Valu
strVBS = strVBS & " DTSDestination("CustName") = ""<u
strVBS = strVBS & " Else" & vbCrLf
strVBS = strVBS & " DTSDestination("CustName") = DTS:
strVBS = strVBS & " End If" & vbCrLf
```

```
strVBS = strVBS & " If IsNull(DTSSource("""CustAddr""")) Then  
strVBS = strVBS & "     DTSDestination("""CustAddr""") = ""<un  
strVBS = strVBS & " Else" & vbCrLf  
strVBS = strVBS & "     DTSDestination("""CustAddr""") = DTSS  
strVBS = strVBS & " End If" & vbCrLf  
strVBS = strVBS & " TransformFailed = DTSTransformStat_OI  
strVBS = strVBS & "End Function" & vbCrLf
```

```
strVBS = strVBS & "Function InitializeGV()" & vbCrLf  
strVBS = strVBS & "     DTSGlobalVariables("""LastErrorRow""") =  
strVBS = strVBS & "     InitializeGV = DTSTransformStat_OK" &  
strVBS = strVBS & "End Function" & vbCrLf  
.Text = strVBS
```

End With

objCustTask.Transformations.Add objTransform

'Link task to package, run package.

objPackage.Tasks.Add objTask

objPackage.Execute

Exit Sub

End Sub

See Also

[Adding DTS ActiveX Scripts](#)

[Adding DTS Column Objects](#)

[Adding DTS Transformations](#)

[DTS Scripting Reference](#)

[New \(ID\) Method](#)

[SourceSQLStatement Property](#)

[Transformation2 Object](#)

[Transformations Collection](#)

[TransformPhases Property](#)

Phased Transformation Samples

These Microsoft® Visual Basic® Scripting Edition (VBScript) functions support a Data Transformation Services (DTS) package program that uses multiphase transformations. For more information, see [DTSTransformScriptProperties2 Object](#).

TransformFailed Function

If an error occurred converting to **money**, **TransformFailed** opens a Microsoft ActiveX® Data Objects (ADO) recordset on an error records table. Then it writes a record containing the primary key from the data source and the invalid **money** field. It sets destination columns to indicate the error occurred. It saves the current source row number in a global variable to indicate the conversion error occurred for the current row. If source columns are Null, it sets the corresponding destination column to "<unknown>".

Example

The following is the VBScript for the **TransformFailed** function:

```
Function TransformFailed()
```

```
'Called on transform failure, usually conversion error or Null -> NOT I
```

```
    Dim rstErrors
```

```
    Dim strConnect
```

```
    DTSDestination("CustID") = DTSSource("CustID")
```

```
    DTSDestination("ErrorCount") = 0
```

```
'See if transaction amount conversion error occurred.
```

```
On Error Resume Next
```

```
DTSDestination("TransAmount") = CCur(DTSSource("TransAmour
```

```
'Conversion error occurred. Write bad transaction amount to error tab
```

```

If Err.Number <> 0 Then
    On Error GoTo 0
    strConnect = "Provider=SQLOLEDB;Data Source=(local);Initial C
    Set rstErrors = CreateObject("ADODB.Recordset")
    rstErrors.LockType = 3                'adLockOptimistic
    rstErrors.Open "ErrorAmounts", strConnect, , , 2 'adCmdTable
    rstErrors.AddNew
    rstErrors("CustID") = DTSSource("CustID")
    rstErrors("TransAmount") = DTSSource("TransAmount")
    rstErrors.Update
    rstErrors.Close

    'Indicate error in destination table, and flag that that transform erro
    DTSDestination("TransAmount") = 0.0
    DTSDestination("ErrorCount") = 1
    DTSGlobalVariables("LastErrorRow") = _
        CLng(DTSTransformPhaseInfo.CurrentSourceRow)
End If
On Error GoTo 0

'If NULL is in Name or Address, write <unknown>. Otherwise updat
If IsNull(DTSSource("CustName").Value) Then
    DTSDestination("CustName") = "<unknown>"
Else
    DTSDestination("CustName") = DTSSource("CustName")
End If
If IsNull(DTSSource("CustAddr")) Then
    DTSDestination("CustAddr") = "<unknown>"
Else
    DTSDestination("CustAddr") = DTSSource("CustAddr")
End If

TransformFailed = DTSTransformStat_OK

```


'If name or address not NULL in source, update destination.

```
If DTSSource("CustName") <> "<unknown>" Then  
    rstCustomers("CustName") = DTSSource("CustName")
```

```
End If
```

```
If DTSSource("CustAddr") <> "<unknown>" Then  
    rstCustomers("CustAddr") = DTSSource("CustAddr")
```

```
End If
```

'If no transform failure occurred, add to transaction amount.

```
If CLng(DTSTransformPhaseInfo.CurrentSourceRow) <> (DTSGlo  
    rstCustomers("TransAmount") = rstCustomers("TransAmount") + 1
```

'If transform failed in this row, increment error count.

```
Else
```

```
    rstCustomers("ErrorCount") = rstCustomers("ErrorCount") + 1
```

```
End If
```

'Update and close recordset.

```
rstCustomers.Update
```

```
rstCustomers.Close
```

```
InsertFailed = DTSTransformStat_OK
```

```
End Function
```

InitializeGV Function

InitializeGV initializes a global variable.

Example

The following is the VBScript for the InitializeGV function:

```
Function InitializeGV()
```

```
    'Write row 0 into transform error row indicator.
```

```
    DTSGlobalVariables("LastErrorRow") = _
```

```
        CLng(DTSTransformPhaseInfo.CurrentSourceRow)
```

InitializeGV = DTSTransformStat_OK
End Function

DTS Programming

Other Objects

This section describes the objects of the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) object model.

For more information about the task objects and transformation objects supplied with SQL Server 2000, see [Task Objects](#) and [Transformation Objects](#).

Topic	Description
Application Object	Provides access to system properties and information about system components.
Column Object	Contains information about a source or destination column, or a data-driven query parameter.
Connection Object	Contains information about connections to OLE DB data sources.
Connection2 Object	Extends the functionality of the Connection object.
CustomTask Object	Allows developers to implement DTS custom tasks.
CustomTaskUI Object	Allows developers to implement a user interface for a custom task.
DTSMQMessage Object	Holds the definition of a DTSMQMessageTask message to be sent.
DynamicPropertiesTaskAssignment Object	Holds the definition of a DTS object property for a DynamicPropertiesTask object.
GlobalVariable Object	Defines a DTS global variable.
GlobalVariable2 Object	Extends the functionality of the GlobalVariable object.
IDTSStdObject	Serves as the base object from which all other DTS objects are derived.
Lookup Object	Specifies named, parameterized

	query string.
OLEDBProperty Object	Specifies property to be set in an OLE DB service provider at run time.
OLEDBProperty2 Object	Extends the functionality of the OLEDBProperty object.
OLEDBProviderInfo Object	Supplies information about an OLE DB provider.
Package Object	Heads the hierarchy of objects.
Package2 Object	Extends the functionality of the Package object.
PackageInfo Object	Provides information about a DTS package in persistent storage.
PackageLineage Object	Provides the contents of a SQL Server 2000 Meta Data Services package lineage record.
PackageLog Object	Allows a custom task or task script to write task log records.
PackageLogRecord Object	Provides the contents of a package log record.
PackageRepository Object	Provides access to the DTS components on Meta Data Services.
PackageSQLServer Object	Provides access to the components on an instance of SQL Server.
PersistPropertyBag Object	Defines a persistent property storage interface for a custom task
PrecedenceConstraint Object	Limits when a DTS step can begin execution.
PropertiesProvider Object	Defines an object supplying a DTS Properties collection.
Property Object	Exposes the attributes of an object property.
PropertyBag Object	Defines a name-indexed container for property values.

SavedPackageInfo Object	Contains information about a package saved in a COM-structured storage file.
ScriptingLanguageInfo Object	Provides information about a Microsoft ActiveX® scripting language registered on the system.
Step Object	Controls the execution of a task in the package.
Step2 Object	Extends the functionality of the Step object.
StepLineage Object	Provides the contents of a step lineage record from Meta Data Services.
StepLogRecord Object	Provides the contents of a step log record from an instance of SQL Server.
Task Object	Defines a unit of work to be performed as part of a package.
TaskInfo Object	Provides information about a task class registered on the computer system.
TaskLogRecord Object	Provides the contents of a task log record from an instance of SQL Server.
Transformation Object	Contains information about the transformation class and the source and destination columns.
Transformation2 Object	Extends the functionality of the Transformation object.
TransformationInfo Object	Provides information about a registered DTS transformation class.
TransformationSet Object	Defines the transformations to be performed on a component of a hierarchical rowset.

See Also

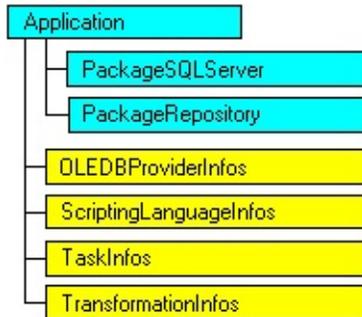
[Creating a DTS Package](#)

[Programming DTS Applications](#)

DTS Programming

Application Object

The **Application** object provides access to system properties and information about system components. It returns connections to Microsoft® SQL Server™ storage or to SQL Server 2000 Meta Data Services instances that contain Data Transformation Services (DTS) packages.



Collections

OLEDBProviderInfos Collection	TaskInfos Collection
Properties Collection	TransformationInfos Collection
ScriptingLanguageInfos Collection	

Properties

DesignerSettings Property	Parent Property
JITDebug Property	

Methods

GetPackageRepository Method	GetPackageSQLServer Method
---	--

Remarks

The **Application** object is not derived from another DTS component. Instead, it is created. For example, do this with the **New** operator in Microsoft Visual Basic®:

```
Dim objDTSApp1 As DTS.Application
...
Set objDTSApp1 = New DTS.Application
```

The system properties accessible through the **Application** object are:

- Whether Phased Transformation features are visible in DTS Designer.
- Whether script run-time errors cause the scripting debugger to be entered.

The system components about which information is accessible through the **Application** object are:

- The set of OLE DB providers available on the system.
- The scripting languages that can be used in ActiveX® Script tasks, ActiveX Script transformations, and step scripts.
- The DTS tasks available on the system, including the tasks provided with SQL Server 2000 and custom tasks implemented by users and other vendors.
- The DTS transformations available on the system, including the transformations provided with SQL Server 2000 and custom transformations implemented by users and other vendors.

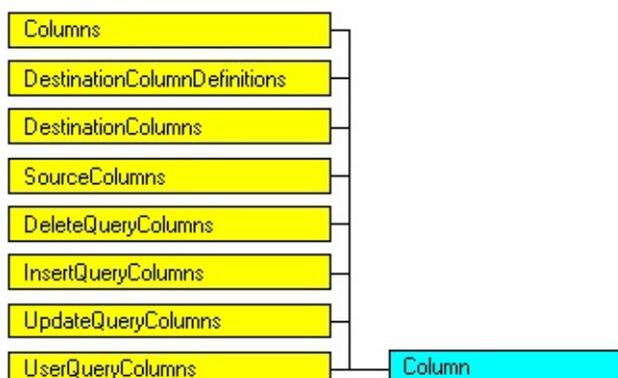
Examples

For more information about the **Application** object and examples of its use, see [Retrieving DTS System, Package and Log Data](#).

DTS Programming

Column Object

The **Column** object contains information about a source or destination column, or a data driven query parameter. If no source or destination columns are specified for a transformation, then all columns are implied by default.



Collections

[Properties Collection](#)

Properties

ColumnID Property	NumericScale Property
DataType Property	Ordinal Property
Flags Property	Parent Property
Name Property	Precision Property
Nullable Property	

See Also

[Adding DTS Column Objects](#)

[Columns Collection](#)

[DeleteQueryColumns Property](#)

[DestinationColumnDefinitions Property](#)

[DestinationColumns Property](#)

[InsertQueryColumns Property](#)

[SourceColumns Property](#)

[UpdateQueryColumns Property](#)

[UserQueryColumns Property](#)

DTS Programming

Connection Object

The **Connection** object contains information about connections to data sources through OLE DB service providers. **Connection** objects allow connection pooling and reuse for connections within a package so that only one connection must be established for multiple steps or tasks.



Properties

Catalog Property	InUse Property
Connected Property	LastOwnerTaskName Property
ConnectImmediate Property	Name Property
ConnectionProperties Property	Parent Property
ConnectionTimeout Property	Password Property
DataSource Property	ProviderID Property
Description Property	Reusable Property
ID Property	UserID Property
InTransaction Property	

Methods

AcquireConnection Method	ReleaseConnection Method
--	--

Remarks

The **Connection** object is compatible with Microsoft® SQL Server™ version 7.0. For more information about an extended version of this object, see [Connection2 Object](#)

See Also

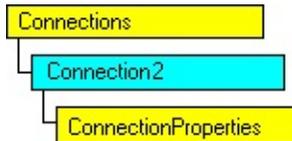
[Connections Collection](#)

[Creating DTS Package Objects and Connections](#)

DTS Programming

Connection2 Object

The **Connection2** object contains information about connections to data sources through OLE DB service providers.



Extended Properties

[UDLPath Property](#)

Remarks

The **Connection2** object extends the functionality of the **Connection** Object and inherits the properties and methods of that object. In addition, the **UDLPath** property is read/write, while it is read-only in the **Connection** object.

For more information about when to use the **Connection** object instead of the **Connection2** object, see [Extended DTS Objects](#).

See Also

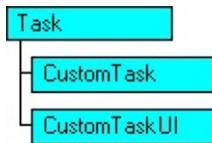
[Connection Object](#)

[Creating DTS Package Objects and Connections](#)

DTS Programming

CustomTask Object

The **CustomTask** object is an interface that all Data Transformation Services (DTS) tasks must implement. This allows programmers to create their own custom tasks, which can be controlled by the DTS package.



Collections

[Properties Collection](#)

Properties

[Description Property](#)

[Name Property](#)

Methods

[Execute Method](#)

Remarks

After inheriting from the **CustomTask** object, a DTS custom task must implement the following:

- **Description** property
- **Execute** method
- **Name** property

- **Properties** collection

To inherit from the **CustomTask** interface in Microsoft® Visual Basic®, the program references the interface with the Implements statement:

```
Implements DTS.CustomTask
```

Then prototypes for the elements the custom task must implement can be selected from the **Procedures/Events** box in the code window in the Visual Basic integrated development environment (IDE), after selecting **CustomTask** from the Object Box.

DTS implements a default properties provider for the **Properties** collection if the user-supplied **Property Get CustomTask_Properties(.)** function returns **Nothing**.

To implement a user interface in a custom task, the program must also inherit from the **CustomTaskUI** interface.

See Also

[CustomTaskUI Object](#)

DTS Programming

CustomTaskUI Object

The **CustomTaskUI** object is an interface that allows you to optionally specify a custom dialog box for a Data Transformation Services (DTS) custom task that can be used in DTS Designer. The **CustomTaskUI** interface is only used in conjunction with custom tasks. If the **CustomTaskUI** is not implemented, DTS Designer displays a default user interface for task properties in a simple grid format.



Methods

CreateCustomToolTip Method	Help Method
Delete Method	Initialize Method
Edit Method	New Method
GetUIInfo Method	

Remarks

After inheriting from the **CustomTaskUI** interface, a custom task must implement the following:

- **CreateCustomToolTip** method
- **Delete** method
- **Edit** method
- **GetUIInfo** method
- **Help** method

- **Initialize** method
- **New** method

Some of these methods can be placeholders that do nothing.

To inherit from the **CustomTaskUI** interface in Microsoft® Visual Basic®, the program references the interface with the Implements statement:

```
Implements DTS.CustomTaskUI
```

Then prototypes for the elements the custom task must implement can be selected from the Procedures/Events Box in the code window in the Visual Basic integrated development environment (IDE), after selecting **CustomTaskUI** from the Object Box.

All DTS tasks must implement the **CustomTask** interface.

See Also

[CustomTask Object](#)

DTS Programming

DTSMQMessage Object

The **DTSMQMessage** object holds the definition of a single message to be sent by a **DTSMessageQueueTask** object.



Collections

[Properties Collection](#)

Properties

MessageDataFile Property	MessageType Property
MessageGlobalVariables Property	UseTransaction Property
MessageString Property	

Methods

[Reset Method](#)

Remarks

DTSMQMessage objects are used only to define the messages to be sent by a **DTSMessageQueueTask** object. The single message to be received by a receiving task is defined by the task properties.

See Also

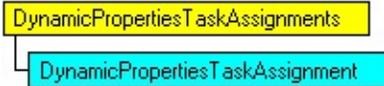
[DTSMQMessages Collection](#)

[DTSMessageQueueTask Object](#)

DTS Programming

DynamicPropertiesTaskAssignment Object

The **DynamicPropertiesTaskAssignment** object holds the definition of a single package object property to be modified by a **Dynamic Properties Task** object, and the source of the new property value.



Collections

[Properties Collection](#)

Properties

DestinationPropertyID Property	SourceIniFileKey Property
SourceConstantValue Property	SourceIniFileSection Property
SourceDataFileFileName Property	SourceQueryConnectionID Property
SourceEnvironmentVariable Property	SourceQuerySQL Property
SourceGlobalVariable Property	SourceType Property
SourceIniFileFileName Property	

Methods

[Reset Method](#)

Remarks

One of the following sources for the new property value can be specified:

- A constant

- The contents of a data file
- An environment variable
- A Data Transformation Services (DTS) global variable
- A field in an .ini file
- An SQL query

See Also

[DynamicPropertiesTask](#)

[DynamicPropertiesTaskAssignments Collection](#)

DTS Programming

GlobalVariable Object

The **GlobalVariable** object defines a variable that allows data to be shared across steps and Microsoft® ActiveX® scripts.



Collections

[Properties Collection](#)

Properties

Name Property	Value Property
Parent Property	

Remarks

Global variables can be shared between Data Transformation Services (DTS) packages by using the **ExecutePackageTask** or **DTSMessageQueueTask** objects.

The **GlobalVariable** object is compatible with Microsoft® SQL Server™ version 7.0. For more information about an extended version of this object, see [GlobalVariable2 Object](#).

See Also

[Adding DTS Lookups and Global Variables](#)

[DTSMessageQueueTask Object](#)

[ExecutePackageTask Object](#)

[GlobalVariables Collection](#)

DTS Programming

GlobalVariable2 Object

The **GlobalVariable2** object defines a variable that allows data to be shared across steps and Microsoft® ActiveX® scripts.

Extended Methods

Lock Method	Unlock Method
-----------------------------	-------------------------------

Remarks

The **GlobalVariable2** object extends the functionality of the **GlobalVariable** Object and inherits the properties and methods of that object. In addition, the **Lock** and **Unlock** methods allow a task to acquire a **GlobalVariable2** object for exclusive use and to later release it.

For more information about when to use the **GlobalVariable** object instead of the **GlobalVariable2** object, see [Extended DTS Objects](#).

See Also

[Adding DTS Lookups and Global Variables](#)

[GlobalVariable Object](#)

[GlobalVariables Collection](#)

DTS Programming

IDTSStdObject

The **IDTSStdObject** is the base object from which all other Data Transformation Services (DTS) package objects are derived. It has no properties, methods, or events.

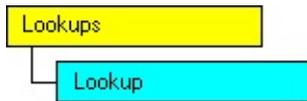
See Also

[Parent Property](#)

DTS Programming

Lookup Object

The **Lookup** object allows a data pump consumer, for example a **DataDrivenQueryTask2**, **DataPumpTask2** or **ParallelDataPumpTask** object, to specify one or more named, parameterized query strings that allow a transformation to retrieve data from locations other than the row being transformed. For example, a **Lookup** object might reference data in a Microsoft® Excel worksheet.



Collections

[Properties Collection](#)

Properties

ConnectionID Property	Parent Property
MaxCacheRows Property	Query Property
Name Property	

See Also

[Adding DTS Lookups and Global Variables](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

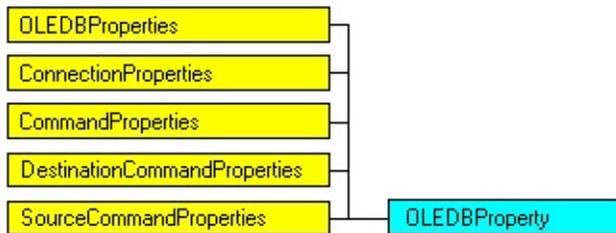
[Lookups Collection](#)

[ParallelDataPumpTask Object](#)

DTS Programming

OLEDBProperty Object

OLEDBProperty objects are used by the **Connection** object to specify properties of sessions and rowsets to be set in the OLE DB service provider at run time. These properties are set automatically by the Data Transformation Services (DTS) package at run time. Connection properties of each OLE DB service provider can also be set this way.



Collections

[Properties Collection](#)

Properties

Name Property	PropertySet Property
Parent Property	Value Property
PropertyID Property	

Remarks

The **OLEDBProperty** object is compatible with Microsoft® SQL Server™ version 7.0. For more information about an extended version of this object, see [OLEDBProperty2 Object](#).

See Also

[CommandProperties Property](#)

[ConnectionProperties Property](#)

[Creating DTS Package Objects and Connections](#)

[DestinationCommandProperties Property](#)

[OLEDBProperties Collection](#)

[SourceCommandProperties Property](#)

DTS Programming

OLEDBProperty2 Object

OLEDBProperty2 objects are used by the **Connection** object to specify properties of sessions and rowsets to be set in the OLE DB service provider at run-time.

Extended Properties

[IsDefaultValue Property](#)

Remarks

The **OLEDBProperty2** object extends the functionality of the **OLEDBProperty** object and inherits the properties and methods of that object. In addition, the **IsDefaultValue** property indicates whether the value of the **OLEDBProperty2** object has been explicitly set to a value other than the code for "restore default value".

For more information about when to use the **OLEDBProperty** object instead of the **OLEDBProperty2** object, see [Extended DTS Objects](#).

See Also

[CommandProperties Property](#)

[ConnectionProperties Property](#)

[Creating DTS Package Objects and Connections](#)

[DestinationCommandProperties Property](#)

[OLEDBProperty Object](#)

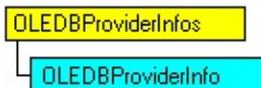
[OLEDBProperties Collection](#)

[SourceCommandProperties Property](#)

DTS Programming

OLEDBProviderInfo Object

The **OLEDBProviderInfo** object provides information about an OLE DB provider that is registered on the computer system.



Properties

ClassID Property	Name Property
Description Property	Parent Property
ImplementationFileName Property	ParseName Property
ImplementationFileVersionString Property	

Remarks

Access the **OLEDBProviderInfo** objects by creating the Data Transformation Services (DTS) **Application** object and iterating through the **OLEDBProviderInfos** collection.

See Also

[Application Object](#)

[OLEDBProviderInfos Collection](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

Package Object

The **Package** object is the main transformation-defining object from which all other objects stem.



Collections

Connections Collection	Steps Collection
GlobalVariables Collection	Tasks Collection
Properties Collection	

Properties

AutoCommitTransaction Property	PackageID Property
CreationDate Property	PackagePriorityClass Property
CreatorComputerName Property	Parent Property
CreatorName Property	RepositoryMetadataOptions Property
Description Property	TransactionIsolationLevel Property
FailOnError Property	UseOLEDBServiceComponents Property
LineageOptions Property	UseTransaction Property
LogFileName Property	VersionID Property
MaxConcurrentSteps Property	WriteCompletionStatusToNTEventLog Property
Name Property	

Methods

Execute Method	RemoveFromRepository Method
GetDTSVersionInfo Method	RemoveFromSQLServer Method
GetLastExecutionLineage Method	SaveAs Method
GetSavedPackageInfos Method	SaveToRepository Method
LoadFromRepository Method	SaveToSQLServer Method
LoadFromSQLServer Method	SaveToStorageFile Method
LoadFromStorageFile Method	

Events

OnError Event	OnQueryCancel Event
OnFinish Event	OnStart Event
OnProgress Event	

Remarks

The **Package** object is compatible with Microsoft® SQL Server™ version 7.0. For more information about an extended version of this object, see [Package2 Object](#).

See Also

[Creating DTS Package Objects and Connections](#)

DTS Programming

Package2 Object

The **Package2** object is the parent object of a Data Transformation Services (DTS) package. Most of the new properties support logging to the **msdb** database of a specified instance of Microsoft® SQL Server™.



Extended Properties

ExplicitGlobalVariables Property	LogServerUserName Property
FailPackageOnLogFailure Property	LogToSQLServer Property
LogServerFlags Property	NestedExecutionLevel Property
LogServerName Property	PackageType Property
LogServerPassword Property	

Extended Methods

SaveToRepositoryAs Method	SaveToStorageFileAs Method
SaveToSQLServerAs Method	

Remarks

The **Package2** object extends the functionality of the existing **Package** object and inherits the properties and methods of that object. In addition, several extended methods and properties have been added.

The **ExplicitGlobalVariables** property inhibits automatic creation of global variables on first reference. The **NestedExecutionLevel** property helps detect

uncontrolled recursive package execution through the **ExecutePackageTask** object. The **PackageType** property provides information about the package creator.

The **LogToSQLServer**, **LogServerFlags**, **LogServerName**, **LogServerPassword**, **LogServerUserName**, and **FailPackageOnLogFailure** properties enable logging to an instance of SQL Server, identify the server, and provide authentication information.

The **SaveToRepositoryAs**, **SaveToSQLServerAs**, and **SaveToStorageFileAs** methods assign a new name and package ID to a **Package2** object, and then save it to the specified persistent storage.

Note In SQL Server 2000, if a **Package2** object variable is declared, **WithEvents**, event handlers should be provided for all of the **Package** events, and the **ExecuteInMainThread** property should be set to TRUE for all steps.

From within ActiveX® scripts, use **DTSGlobalVariables.Parent** to reference the **Package2** object. From the **Package2** object, you can reference any other object in the hierarchy.

For more information about when to use the **Package** object instead of the **Package2** object, see [Extended DTS Objects](#).

See Also

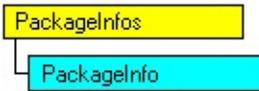
[Creating DTS Package Objects and Connections](#)

[Package Object](#)

DTS Programming

PackageInfo Object

The **PackageInfo** object provides information about a package stored in Microsoft® SQL Server™ 2000 Meta Data Services or SQL Server storage.



Properties

CreationDate Property	PackageID Property
Description Property	PackageType Property
IsOwner Property	Parent Property
Name Property	Properties Collection
Owner Property	VersionID Property
PackageDataSize Property	

Remarks

Access the **PackageInfo** objects by using the **EnumPackageInfos** method of the **PackageRepository** or **PackageSQLServer** objects and iterating through the **PackageInfos** collection the method returns.

See Also

[EnumPackageInfos Method](#)

[PackageInfos Collection](#)

[PackageRepository Object](#)

[PackageSQLServer Object](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PackageLineage Object

The **PackageLineage** object provides the contents of a package lineage record from Microsoft® SQL Server™ 2000 Meta Data Services. The record contains information about a Data Transformation Services (DTS) package execution hosted by Meta Data Services.



Properties

Computer Property	Operator Property
ExecutionDate Property	PackageID Property
LineageFullID Property	Parent Property
LineageShortID Property	Properties Collection
Name Property	

Remarks

A package lineage record is written each time a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this be done.

Access the **PackageLineage** objects by using the **EnumPackageLineages** method of the **PackageRepository** object and iterating through the **PackageLineages** collection the method returns.

See Also

[EnumPackageLineages Method](#)

[LineageOptions Property](#)

[PackageLineages Collection](#)

[PackageRepository Object](#)

Retrieving DTS System, Package and Log Data

DTS Programming

PackageLog Object

The **PackageLog** object allows a Data Transformation Services (DTS) custom task or **ActiveScriptTask** object to write task log records in the database or write log messages to the log file.



Methods

WriteStringToLog Method	WriteTaskRecord Method
---	--

Remarks

The reference to the **PackageLog** object is a parameter of the task **Execute** method, which you implement in a custom task, and thus have access. It is also available in task Microsoft® ActiveX® scripts, through the **DTSPackageLog** object.

See Also

[EnumTaskLogRecords Method](#)

[Execute Method](#)

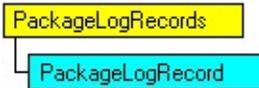
[PackageLogRecords Collection](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PackageLogRecord Object

The **PackageLogRecord** object provides the contents of a package log record from an instance of Microsoft® SQL Server™. The log record contains information about a package execution.



Properties

Computer Property	LogDate Property
Description Property	Name Property
ErrorCode Property	Operator Property
ErrorDescription Property	PackageID Property
ExecutionTime Property	Parent Property
FinishTime Property	Properties Collection
LineageFullID Property	StartTime Property
LineageShortID Property	

Remarks

A package log record is written to the **msdb** database on the server specified by the package **LogServerName** property each time a Data Transformation Services (DTS) package is executed, if the package **LogToSQLServer** property has been set.

Access the **PackageLogRecord** objects by using the **EnumPackageLogRecords** method of the **PackageSQLServer** object and iterating through the **PackageLogRecords** collection the method returns.

See Also

[EnumPackageLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageLogRecords Collection](#)

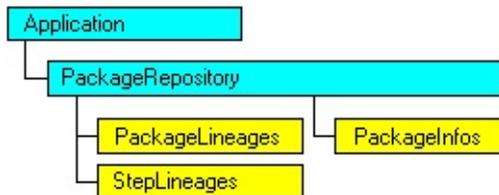
[PackageSQLServer Object](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PackageRepository Object

The **PackageRepository** object provides access to the Data Transformation Services (DTS) components hosted by an instance of Microsoft® SQL Server™ 2000 Meta Data Services. Through methods of this object, you can obtain information about the DTS packages stored in Meta Data Services and access the contents of the package and step lineage data for these packages.



Properties

Name Property	Properties Collection
Parent Property	

Methods

EnumPackageInfos Method	EnumStepLineages Method
EnumPackageLineages Method	

Remarks

A package lineage record is written each time a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this to be done.

Access the **PackageRepository** object by using the **GetPackageRepository** method of the **Application** object. Provide the necessary login information to access the instance of SQL Server that hosts the desired Meta Data Services instance.

See Also

[Application Object](#)

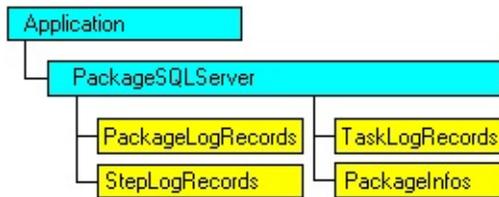
[GetPackageRepository Method](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PackageSQLServer Object

The **PackageSQLServer** object provides access to the Data Transformation Services (DTS) components hosted by an instance of Microsoft® SQL Server™. Through methods of this object, you can obtain information about the DTS packages stored in SQL Server storage and access the contents of the package, step, and task log records stored on that server.



Properties

Name Property	Properties Collection
Parent Property	

Methods

EnumPackageInfos Method	RemoveAllLogRecords Method
EnumPackageLogRecords Method	RemovePackageLogRecords Method
EnumStepLogRecords Method	RemoveStepLogRecords Method
EnumTaskLogRecords Method	

Remarks

Access the **PackageSQLServer** object by using the **GetPackageSQLServer** method of the **Application** object. Provide the necessary login information to access the instance of SQL Server you want.

See Also

[Application Object](#)

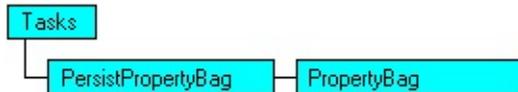
[GetPackageSQLServer Method](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PersistPropertyBag Object

The **PersistPropertyBag** object defines a persistent property storage interface for an object implementing a Data Transformation Services (DTS) custom task. The **PropertyBag** object is a name-indexed container object for object properties. When implemented, the **PropertyBag** and **PersistPropertyBag** objects allow simple object property handling.



Methods

Load Method	Save Method
-----------------------------	-----------------------------

Remarks

DTS provides these options for custom task object property storage. You can:

- Do nothing. DTS will save and load properties when **Package** object store and retrieve methods are called. The object can, optionally, implement a **PropertiesProvider** object to expose its properties as a DTS **Properties** collection.
- Implement a **PersistPropertyBag** object.
- If using Microsoft® Visual C++®, implement the **IPersistStorage** interface on the custom task object.

If implemented, DTS will call the **PersistPropertyBag** object **Load** method when a **Package** retrieving method is called. The **Save** method is called when a **Package** method storing a package is called.

See Also

PropertyBag Object

DTS Programming

PrecedenceConstraint Object

The **PrecedenceConstraint** object contains information about a condition that must occur before a Data Transformation Services (DTS) step can be released for execution. The **PrecedenceConstraint** objects of all the package steps control the order in which steps are executed.



Collections

[Properties Collection](#)

Properties

Parent Property	StepName Property
PrecedenceBasis Property	

See Also

[Creating DTS Package Workflow and Tasks](#)

[PrecedenceConstraints Collection](#)

DTS Programming

PropertiesProvider Object

The **PropertiesProvider** object defines an object supplying a Data Transformation Services (DTS) **Properties** collection. When exposed, DTS will retrieve the **Properties** collection as required.

Methods

[GetPropertiesForObject Method](#)

See Also

[CustomTask Object](#)

[Properties Collection](#)

DTS Programming

Property Object

The **Property** object exposes the attributes of a Data Transformation Services (DTS) object property.

Most DTS objects have **Properties** collections, which contain a **Property** object for each property the object has. By referencing the **Properties** collection, a user of the object can determine whether the object supports a particular property without causing a program error if it does not.



Collections

[Properties Collection](#)

Properties

Get Property	Set Property
Name Property	Type Property
Parent Property	

Remarks

Development environments, such as Microsoft® Visual Basic®, typically provide syntax completion and other development aids. Because it exposes the attributes of object properties, the **Property** object supports such automated developer assistance.

Note The **Property** object is implemented for automation controllers. Microsoft Visual C++® DTS applications have no direct access to the **Property** object.

See Also

[TransformServerProperties Property](#)

DTS Programming

PropertyBag Object

The **PropertyBag** object defines a name-indexed container for property values for an object implementing a Data Transformation Services (DTS) custom task. Use the **PropertyBag** object as part of custom task object implementation when the custom task maintains storage for task properties.

DTS can read and write values of simple data types, such as String, in a **PropertyBag** object. DTS cannot support objects and other more complex data types as values in a **PropertyBag** container.



Methods

Read Method	Write Method
-----------------------------	------------------------------

See Also

[PersistPropertyBag Object](#)

DTS Programming

SavedPackageInfo Object

The **SavedPackageInfo** object contains information about packages that are saved in COM-structured storage files. This information is returned by the **GetSavedPackageInfos** method.



Properties

Description Property	PackageName Property
IsVersionEncrypted Property	VersionID Property
PackageCreationDate Property	VersionSaveDate Property
PackageID Property	

See Also

[GetSavedPackageInfos Method](#)

[Retrieving DTS System, Package and Log Data](#)

[SavedPackageInfos Collection](#)

DTS Programming

ScriptingLanguageInfo Object

The **ScriptingLanguageInfo** object provides information about a Microsoft® ActiveX® scripting language that is registered on the computer system.



Properties

ClassID Property	Name Property
Description Property	Parent Property
ImplementationFileName Property	Properties Collection
ImplementationFileVersionString Property	

Remarks

Access the **ScriptingLanguageInfo** objects by creating the Data Transformation Services (DTS) **Application** object and iterating through the **ScriptingLanguageInfos** collection.

Typically, the available scripting languages will include the following.

Scripting Language	Name	Description
Microsoft JScript®	JScript	JScript Language
PerlScript	PerlScript	PerlScript Language
Microsoft Visual Basic® Scripting Edition (VBScript)	VBScript	VBScript Language
Extensible Markup Language	XML	XML Script Engine

The values in the **Name** and **Description** columns are the values that are returned by the **Name** and **Description** properties of the **ScriptingLanguageInfo** object. Use a value from the **Name** property to set the

Language or **ScriptLanguage** property of scripting objects.

Note Encoding versions of scripting languages, which may be installed on a computer system to support scripts embedded in Web pages, are not appropriate for use with DTS.

See Also

[Application Object](#)

[Retrieving DTS System, Package and Log Data](#)

[ScriptingLanguageInfos Collection](#)

[ScriptLanguage Property](#)

[Language Property](#)

DTS Programming

Step Object

The **Step** object controls the flow and execution of tasks within the Data Transformation Services (DTS) package. Each step is associated with a single task, although association with no task is possible. Step execution sequence is determined by the precedence constraints. A step cannot start execution until all its precedence constraints are satisfied.



Collections

PrecedenceConstraints Collection	Properties Collection
--	---------------------------------------

Properties

ActiveXScript Property	FunctionName Property
AddGlobalVariables Property	IsPackageDSORowset Property
CloseConnection Property	JoinTransactionIfPresent Property
CommitSuccess Property	Name Property
Description Property	Parent Property
DisableStep Property	RelativePriority Property
ExecuteInMainThread Property	RollbackFailure Property
ExecutionResult Property	ScriptLanguage Property
ExecutionStatus Property	StartTime Property
ExecutionTime Property	TaskName Property
FinishTime Property	

Methods

Execute (Package) Method	GetExecutionErrorInfo Method
--	--

Remarks

The **Step** object is compatible with Microsoft® SQL Server™ version 7.0. For more information about an extended version of this object, see [Step2 Object](#).

See Also

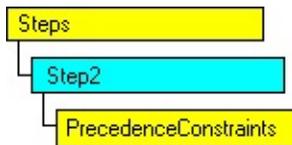
[Creating DTS Package Workflow and Tasks](#)

[Steps Collection](#)

DTS Programming

Step2 Object

The **Step2** object controls the flow and execution of tasks within the Data Transformation Services (DTS) package. Each step is associated with a single task, although association with no task is possible. Step execution sequence is determined by the precedence constraints. A step cannot start execution until all its precedence constraints are satisfied.



Extended Properties

[FailPackageOnError Property](#)

Remarks

The **Step2** object extends the functionality of the **Step** Object and inherits the properties and methods of that object. In addition, the **FailPackageOnError** property causes the package to fail if the step fails.

For more information about when to use the **Step** object instead of the **Step2** object, see [Extended DTS Objects](#).

See Also

[Creating DTS Package Workflow and Tasks](#)

[FailOnError Property](#)

[Step Object](#)

DTS Programming

StepLineage Object

The **StepLineage** object provides the contents of a step lineage record from Microsoft® SQL Server™ 2000 Meta Data Services. The record contains information about the execution of a step in a Data Transformation Services (DTS) package hosted by Meta Data Services.



Properties

ErrorCode Property	Name Property
ErrorDescription Property	Parent Property
ErrorHelpContext Property	Properties Collection
ErrorHelpFile Property	StartTime Property
ErrorSource Property	StepExecutionResult Property
ExecutionTime Property	StepExecutionStatus Property
FinishTime Property	

Remarks

A step lineage record is written each time a step of a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this to be done.

Access the **StepLineage** objects by using the **EnumStepLineages** method of the **PackageRepository** object and iterating through the **StepLineages** collection the method returns.

See Also

[EnumStepLineages Method](#)

[LineageOptions Property](#)

[PackageRepository Object](#)

[Retrieving DTS System, Package and Log Data](#)
[StepLineages Collection](#)

DTS Programming

StepLogRecord Object

The **StepLogRecord** object provides the contents of a step log record from an instance of Microsoft® SQL Server™. The log record contains information about the execution of a step in a Data Transformation Services (DTS) package.



Properties

ErrorCode Property	Parent Property
ErrorDescription Property	ProgressCount Property
ExecutionTime Property	Properties Collection
FinishTime Property	StartTime Property
LineageFullID Property	StepExecutionID Property
Name Property	StepExecutionResult Property

Remarks

A step log record is written to the **msdb** database on the server specified by the package **LogServerName** property each time a step in a package is executed, if the package **LogToSQLServer** property has been set.

Access the **StepLogRecord** objects by using the **EnumStepLogRecords** method of the **PackageSQLServer** object and iterating through the **StepLogRecords** collection the method returns.

See Also

[EnumStepLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageSQLServer Object](#)

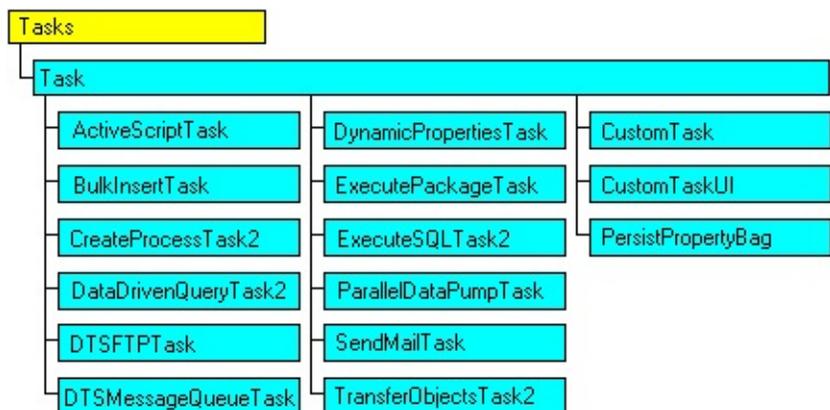
[Retrieving DTS System, Package and Log Data](#)

StepLogRecords Collection

DTS Programming

Task Object

The **Task** object defines a unit of work to be performed as part of a Data Transformation Services (DTS) package. A **Task** object is created in a package when the **Tasks.New** method is invoked with the program ID or class ID of the desired task class, followed by **Tasks.Add**.



Collections

[Properties Collection](#)

Properties

CustomTask Property	Name Property
CustomTaskID Property	Parent Property
Description Property	

Methods

[Execute Method](#)

See Also

[Creating DTS Package Workflow and Tasks](#)

[Tasks Collection](#)

DTS Programming

TaskInfo Object

The **TaskInfo** object provides information about a Data Transformation Services (DTS) task class that is registered on the computer system.



Properties

ClassID Property	ImplementationFileVersionString Property
Description Property	Name Property
IconFile Property	Parent Property
IconIndex Property	Properties Collection
ImplementationFileName Property	

Remarks

The registered task classes include those provided with Microsoft® SQL Server™ 2000 and custom tasks provided by other vendors and implemented by users.

Access the **TaskInfo** objects by creating the DTS **Application** object and iterating through the **TaskInfos** collection.

See Also

[Application Object](#)

[Retrieving DTS System, Package and Log Data](#)

[TaskInfos Collection](#)

DTS Programming

TaskLogRecord Object

The **TaskLogRecord** object provides the contents of a task log record from an instance of Microsoft® SQL Server™. The log record contains information about the execution of a task in a Data Transformation Services (DTS) package that has been implemented to write task log records.



Properties

ErrorCode Property	Properties Collection
ErrorDescription Property	SequenceID Property
Parent Property	

Remarks

Task log records are not automatically written by the task classes supplied with SQL Server 2000, but the **PackageLog** interface is available so that a custom task or the script of an **ActiveScriptTask** object can write them. They are written to the **msdb** database on the server specified by the package **LogServerName** property each time a task in a package that has been implemented to write them is executed, if the package **LogToSQLServer** property has been set.

Access the **TaskLogRecord** objects by using the **EnumTaskLogRecords** method of the **PackageSQLServer** object and iterating through the **TaskLogRecords** collection the method returns.

See Also

[ActiveScriptTask Object](#)

[EnumTaskLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageSQLServer Object](#)

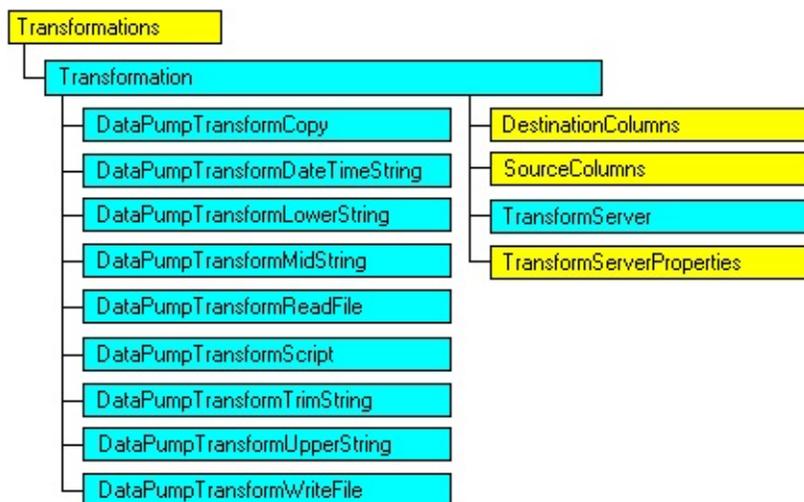
[Retrieving DTS System, Package and Log Data](#)

[TaskLogRecords Collection](#)

DTS Programming

Transformation Object

The generic **Transformation** object contains information about the class-specific transformation object and the source and destination columns it manipulates.



Collections

[Properties Collection](#)

Properties

DestinationColumns Property	SourceColumns Property
ForceBlobsInMemory Property	TransformFlags Property
ForceSourceBlobsBuffered Property	TransformServer Property
InMemoryBlobSize Property	TransformServerID Property
Name Property	TransformServerParameter Property
Parent Property	

Remarks

The **Transformation** object is compatible with Microsoft® SQL Server™

version 7.0. For more information about an extended version of this object, see [Transformation2 Object](#).

See Also

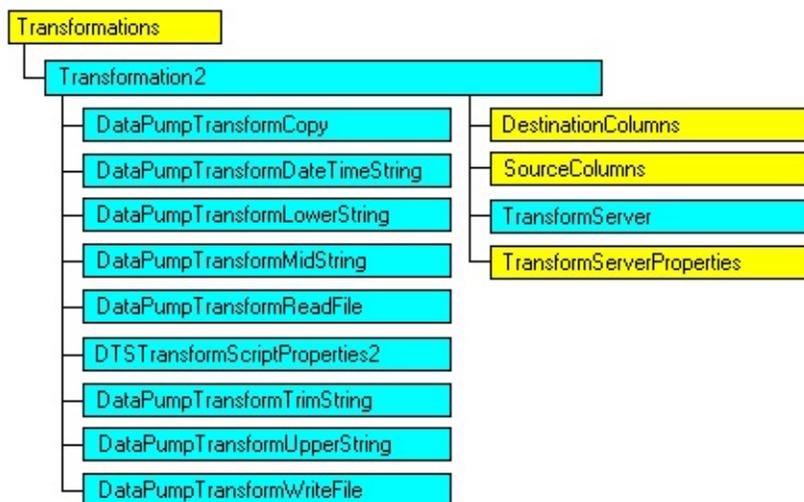
[Adding DTS Transformations](#)

[Transformations Collection](#)

DTS Programming

Transformation2 Object

The generic **Transformation2** object contains information about the class-specific transformation object and the source and destination columns it manipulates.



Extended Properties

[TransformPhases Property](#)

Remarks

The **Transformation2** object extends the functionality of the **Transformation** object and inherits the properties and methods of that object. In addition, the **TransformPhases** property specifies which transform phases this transformation supports.

The **Transformation2** object supports multiple transform phases, as shown in this table.

Phase	Description
PreSourceData	Occurs before first row is fetched from source connection.
Transform	Occurs after each source row is fetched, before the destination row is written.

OnTransformFailure	Occurs after a failure in the Transform phase, indicated by return of DTSTransformStat_Error or DTSTransformStat_ExceptionRow . Typically caused by conversion errors.
OnInsertSuccess	Occurs after each data row is written successfully to the destination connection.
OnInsertFailure	Occurs after each attempt to write a data row to the destination connection failed (for example, by attempting to write a duplicate value to a primary key field, or a NULL to a NOT NULL field).
OnBatchComplete	Occurs in DataPumpTask2 when using the FastLoad option after each batch is written, successfully or unsuccessfully.
PostSourceData	Occurs after the last row is written to the destination connection.
OnPumpComplete	Occurs at the end of the task's execution.

The **Transformation** object only supports the Transform phase.

Only the **DTSTransformScriptProperties2** object and custom transformations can support phases other than the Transform phase.

For more information about when to use the **Transformation** object instead of the **Transformation2** object, see [Extended DTS Objects](#).

See Also

[Adding DTS Transformations](#)

[DTSTransformScriptProperties2 Object](#)

[Transformation Object](#)

[Transformations Collection](#)

DTS Programming

TransformationInfo Object

The **TransformationInfo** object provides information about a Data Transformation Services (DTS) transformation class that is registered on the computer system.



Properties

ClassID Property	Name Property
Description Property	Parent Property
ImplementationFileName Property	Properties Collection
ImplementationFileVersionString Property	

Remarks

The registered transformation classes include those provided with Microsoft® SQL Server™ 2000 and custom transformations provided by other vendors and implemented by users.

Access the **TransformationInfo** objects by creating the DTS **Application** object and iterating through the **TransformationInfos** collection.

See Also

[Application Object](#)

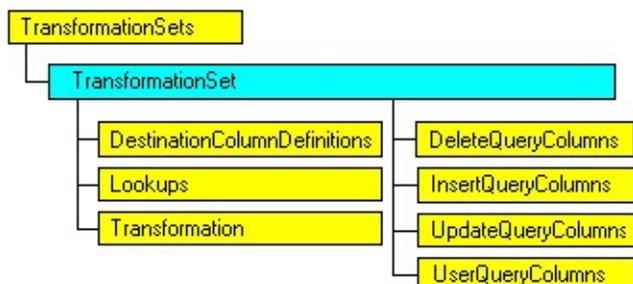
[Retrieving DTS System, Package and Log Data](#)

[TransformationInfos Collection](#)

DTS Programming

TransformationSet Object

The **TransformationSet** object defines the transformations to be performed on the columns of a component rowset in a hierarchical rowset by the **Parallel Data Pump Task** object.



Collections

Lookups Collection	Transformations Collection
Properties Collection	

Properties

DeleteQuery Property	InsertQueryColumns Property
DeleteQueryColumns Property	LastRow Property
Description Property	MaximumErrorCount Property
DestinationColumnDefinitions Property	Name Property
ExceptionFileColumnDelimiter Property	Parent Property
ExceptionFileName Property	ProgressRowCount Property
ExceptionFileOptions Property	RowsComplete Property
ExceptionFileRowDelimiter Property	RowsInError Property
ExceptionFileTextQualifier Property	UpdateQuery Property
FetchBufferSize Property	UpdateQueryColumns Property
FirstRow Property	UserQuery Property
InsertQuery Property	

Remarks

The **TransformationSet** object includes a **Transformations** collection and contains all the information necessary to copy and transform a component rowset, such as data driven queries and their parameter collections, row and error counts, and exception file information.

The ordinal position of the **TransformationSet** object in the **TransformationSets** collection determines the component rowset to which the object is mapped. For more information about the mapping process, see [TransformationSets Collection](#).

See Also

[ParallelDataPumpTask Object](#)

DTS Programming

Collections

This section describes the collections of the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) object model. Collections contain groups of related DTS objects.

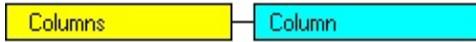
Topic	Description
Columns Collection	Contains descriptions of meta data for columns in a data source.
Connections Collection	Contains information about connections to OLE DB service providers.
DTSMQMessages Collection	Defines the messages to be sent by a DTSMessagesQueueTask object.
DynamicPropertiesTaskAssignments Collection	Defines the properties to be changed by a DynamicPropertiesTask object.
GlobalVariables Collection	Contains information about data to be shared across DTS steps.
Lookups Collection	Defines query strings that allow data retrieval from other than the row being transformed.
OLEDBProperties Collection	Contains properties for an OLE DB service provider.
OLEDBProviderInfos Collection	Provides information about available OLE DB service providers.
PackageInfos Collection	Provides information about packages stored in SQL Server 2000 Meta Data Services or SQL Server storage.
PackageLineages Collection	Provides the contents of package lineage records from Meta Data Services.
PackageLogRecords Collection	Provides the contents of package log records from an instance of SQL

	Server.
PrecedenceConstraints Collection	Contains conditions that must occur before a step can execute.
Properties Collection	Contains a collection of properties for an object.
SavedPackageInfos Collection	Contains information about packages saved in files.
ScriptingLanguageInfos Collection	Provides information about Microsoft ActiveX® scripting languages available on the system.
StepLineages Collection	Provides the contents of step lineage records from Meta Data Services.
StepLogRecords Collection	Provides the contents of step log records from an instance of SQL Server.
Steps Collection	Contains information about the flow and execution of DTS tasks.
TaskInfos Collection	Provides information about the tasks available on the system.
TaskLogRecords Collection	Provides the contents of task log records from an instance of SQL Server.
Tasks Collection	Contains information about the tasks in a DTS package.
TransformationInfos Collection	Provides information about the DTS transformations available on the system.
Transformations Collection	Defines the transformations used by a task.
TransformationSets Collection	Defines the sets of transformations used to process components of a hierarchical rowset.

DTS Programming

Columns Collection

A **Columns** collection is a group of **Column** objects containing a description of all available meta data about a column in a data source. This includes name, description, data type, precision, scale, nullability, and numeric base.



Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	Item Method
AddColumn Method	New (Columns) Method
Insert Method	

Remarks

Columns collections are referenced by the properties of the objects indicated in the table.

Property	Object
DestinationColumns Property SourceColumns Property	Transformation2
DeleteQueryColumns Property InsertQueryColumns Property UpdateQueryColumns Property UserQueryColumns Property	DataDrivenQueryTask2 TransformationSet
DestinationColumnDefinitions Property	DataDrivenQueryTask2 DataPumpTask2 TransformationSet

See Also

[Adding DTS Column Objects](#)

[Column Object](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[Transformation2 Object](#)

[TransformationSet Object](#)

DTS Programming

Connections Collection

The **Connections** collection is a group of **Connection** objects containing information about connections to OLE DB service providers. This collection allows connection pooling and reuse across steps and tasks in a DTS package.



Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	Item Method
BeginAcquireMultipleConnections Method	New (ID) Method
EndAcquireMultipleConnections Method	NewDataLink Method
Insert Method	

Remarks

When implementing a custom task that must acquire more than one connection, do the following to avoid deadlocks:

1. Call **BeginAcquireMultipleConnections**.

2. For each *connection* to be acquired:
 - Verify that *connection.InUse* is FALSE.
 - Call *connection.AcquireConnection*.
3. Call **EndAcquireMultipleConnections**.

Prototype (C/C++)

HRESULT GetConnections(IDTSConnections **pRetVal);

See Also

[Connection Object](#)

[Creating DTS Package Objects and Connections](#)

DTS Programming

DTSMQMessages Collection

The **DTSMQMessages** collection contains the **DTSMQMessage** objects that define the messages to be sent by a **DTSMessageQueueTask** object.



Applies To

[DTSMessageQueueTask Object](#)

Properties

[Count Property](#)

Methods

Add Method	New Method
Item Method	

Remarks

The **Messages** method of a **DTSMessageQueueTask** object is used to get a reference to the **DTSMQMessages** collection.

See Also

[DTSMQMessage Object](#)

[Messages Method](#)

DTS Programming

DynamicPropertiesTaskAssignments Collection

The **DynamicPropertiesTaskAssignments** collection contains the **DynamicPropertiesTaskAssignment** objects that define the source of the new value and the properties to be changed by a **DynamicPropertiesTask** object.



Applies To

[DynamicPropertiesTask Object](#)

Properties

[Count Property](#)

Methods

[Add Method](#)

[New Method](#)

[Item Method](#)

Remarks

The **Assignments** property of a **DynamicPropertiesTask** object is used to get a reference to the **DynamicPropertiesTaskAssignments** collection.

See Also

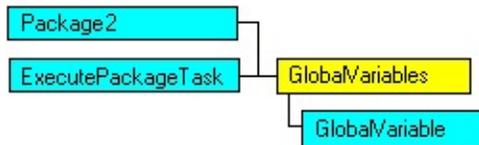
[Assignments Property](#)

[DynamicPropertiesTaskAssignment Object](#)

DTS Programming

GlobalVariables Collection

The **GlobalVariables** collection is a group of **GlobalVariable** objects containing information about variants that allow data to be shared across steps and Microsoft® ActiveX® scripts.



Applies To

ExecutePackageTask Object	Package2 Object
Package Object	

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	Item Method
AddGlobalVariable Method	New (Name) Method
Insert Method	

Remarks

The **GlobalVariables** collection is dynamic; values may be added to the collection as the package executes. If the package **ExplicitGlobalVariables** property is not set, a global variable is created on first reference by an ActiveX script, if it does not already exist.

The **GlobalVariables** collection of the **ExecutePackageTask** object is exported

to the target package, but it is not part of the parent package's **GlobalVariables** collection. To export global variables from the parent package to the target package, use the **InputGlobalVariableNames** property of the **ExecutePackageTask** object.

Reference the **GlobalVariables** collection from within ActiveX scripts with the name **DTSGlobalVariables**.

Prototype (C/C++)

```
HRESULT GetGlobalVariables(IDTSGlobalVariables **pRetVal);
```

See Also

[Adding DTS Lookups and Global Variables](#)

[ExplicitGlobalVariables Property](#)

[GlobalVariable Object](#)

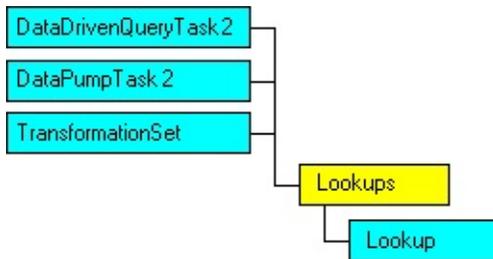
[InputGlobalVariableNames Property](#)

[Package2 Object](#)

DTS Programming

Lookups Collection

The **Lookups** collection is a group of **Lookup** object definitions. A **Lookup** object defines a named, parameterized query string that allows a transformation to retrieve data from a location other than the row being transformed.



Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	Item Method
AddLookup Method	New (Name) Method
Insert Method	

Remarks

In a Microsoft® ActiveX® script, the lookup should be referenced with the **Execute** method of an element of the **DTSLookups** collection. If the lookup

rowset has more than one column, the **Execute** method returns a Variant array. The script may need to iterate through the array to use multiple values.

Prototype (C/C++)

```
HRESULT GetLookups(IDTSLookups **pRetVal);
```

See Also

[Adding DTS Lookups and Global Variables](#)

[DTSDataPumpLookups Collection](#)

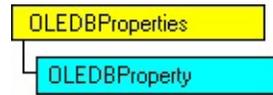
[Execute Method \(Script\)](#)

[Lookup Object](#)

DTS Programming

OLEDBProperties Collection

The **OLEDBProperties** collection is a group of **OLEDBProperty** objects containing information about an OLE DB service provider.



Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

[Item Method](#)

See Also

[CommandProperties Property](#)

[ConnectionProperties Property](#)

[Creating DTS Package Objects and Connections](#)

[DestinationCommandProperties Property](#)

[OLEDBProperty Object](#)

[SourceCommandProperties Property](#)

DTS Programming

OLEDBProviderInfos Collection

The **OLEDBProviderInfos** collection contains **OLEDBProviderInfo** objects that provide information about each OLE DB provider available on the computer system.



Applies To

[Application Object](#)

Properties

Count Property	UseCache Property
Parent Property	

Methods

Item Method	Refresh Method
-----------------------------	--------------------------------

Remarks

Obtain a reference to the **OLEDBProviderInfos** collection from the **Application** object.

You can iterate through the objects of the **OLEDBProviderInfos** collection using the **Item** method and **Count** property. However, it is faster to use **For Each ... Next** in Microsoft® Visual Basic®.

DTS maintains a cache, in the operating system registry, which holds the OLE DB provider information. If the **UseCache** property is TRUE, the cache is scanned, rather than all registered classes, when iterating through the

OLEDBProviderInfos collection. Use the **Refresh** method to update the cache from the registered classes section of the system registry.

See Also

[Application Object](#)

[Item Method](#)

[OLEDBProviderInfo Object](#)

[Refresh Method](#)

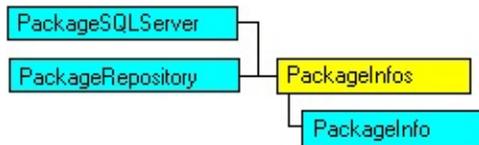
[Retrieving DTS System, Package and Log Data](#)

[UseCache Property](#)

DTS Programming

PackageInfos Collection

The **PackageInfos** collection contains **PackageInfo** objects that provide information about a DTS package stored in Microsoft® SQL Server™ 2000 Meta Data Services or SQL Server storage.



Properties

[EOF Property](#)

Methods

[Next Method](#)

Remarks

Use the **EnumPackageInfos** method of the **PackageRepository** or **PackageSQLServer** objects to return the **PackageInfos** collection.

Iterate through the objects of the **PackageInfos** collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all the elements will have been fetched. You can also use **For Each ... Next** in Microsoft Visual Basic®.

See Also

[EnumPackageInfos Method](#)

[PackageInfo Object](#)

[PackageRepository Object](#)

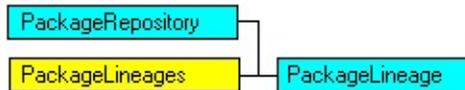
[PackageSQLServer Object](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PackageLineages Collection

The **PackageLineages** collection contains **PackageLineage** objects that provide the contents of package lineage records from Microsoft® SQL Server™ 2000 Meta Data Services. The records contain information about a DTS package execution hosted by Meta Data Services.



Properties

[EOF Property](#)

Methods

[Next Method](#)

Remarks

A package lineage record is written each time a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this to be done.

Use the **EnumPackageLineages** method of the **PackageRepository** object to return the **PackageLineages** collection.

Iterate through the objects of the **PackageLineages** collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all the elements will have been fetched. You can also use **For Each ... Next** in Microsoft Visual Basic®.

See Also

[EnumPackageLineages Method](#)

[LineageOptions Property](#)

[PackageLineage Object](#)

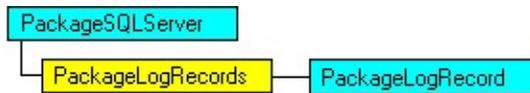
[PackageRepository Object](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PackageLogRecords Collection

The **PackageLogRecords** collection contains **PackageLogRecord** objects that provide the contents of package log records from an instance of Microsoft® SQL Server™. The log records contain information about a DTS package execution.



Properties

[EOF Property](#)

Methods

[Next Method](#)

Remarks

A package log record is written to the **msdb** database on the server specified by the package **LogServerName** property each time a package is executed, if the package **LogToSQLServer** property has been set.

Use the **EnumPackageLogRecords** method of the **PackageSQLServer** object to return the **PackageLogRecords** collection.

Iterate through the objects of the **PackageLogRecords** collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all the elements will have been fetched. You can also use **For Each ... Next** in Microsoft Visual Basic®.

See Also

[EnumPackageLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageLogRecord Object](#)

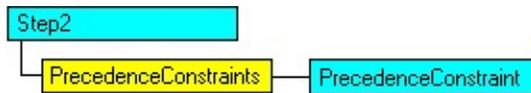
[PackageSQLServer Object](#)

[Retrieving DTS System, Package and Log Data](#)

DTS Programming

PrecedenceConstraints Collection

The **PrecedenceConstraints** collection is a group of **PrecedenceConstraint** objects containing information about conditions that must occur before a DTS step can be released for execution.



Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	Item Method
AddConstraint Method	New (Name) Method
Insert Method	

See Also

[Creating DTS Package Workflow and Tasks](#)

[PrecedenceConstraint Object](#)

DTS Programming

Properties Collection

The **Properties** collection contains **Property** objects exposing the attributes of a DTS object property.



Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

[Item Method](#)

Remarks

Development environments, such as Microsoft® Visual Basic®, typically provide syntax completion and other development aids. Because the **Property** object exposes the attributes of object properties, it supports such automated developer assistance.

When using the **Item** method, the **Properties** collection supports member identification using either name or ordinal reference syntax. For example:

```
Set oProperty = oCustomTask.Properties("Name")
```

-or-

```
Set oProperty = oCustomTask.Properties(1)
```

Note The **Properties** collection is implemented for automation controllers. DTS applications written in Microsoft Visual C++® and C have no direct access to the **Property** object.

See Also

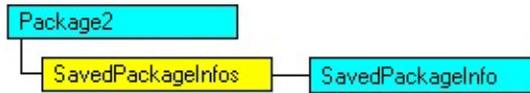
[Property Object](#)

[TransformServerProperties Property](#)

DTS Programming

SavedPackageInfos Collection

The **SavedPackageInfos** collection is a group of **SavedPackageInfo** objects containing information about DTS packages saved in files.



Properties

[Count Property](#)

Method

[Item Method](#)

See Also

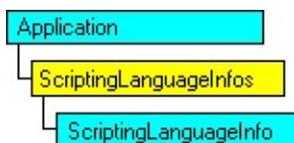
[Retrieving DTS System, Package and Log Data](#)

[SavedPackageInfo Object](#)

DTS Programming

ScriptingLanguageInfos Collection

The **ScriptingLanguageInfos** collection contains **ScriptingLanguageInfo** objects that provide information about each Microsoft® ActiveX® scripting language that is available on the system. You can use these scripting languages in ActiveX Script tasks, ActiveX Script transformations, and step scripts.



Properties

Count Property	UseCache Property
Parent Property	

Methods

Item Method	Refresh Method
-----------------------------	--------------------------------

Remarks

Obtain a reference to the **ScriptingLanguageInfos** collection from the **Application** object.

You can iterate through the objects of the **ScriptingLanguageInfos** collection using the **Item** method and **Count** property. However, it is faster to use **For Each ... Next** in Microsoft Visual Basic®.

DTS maintains a cache, in the operating system registry, which holds the scripting language information. If the **UseCache** property is TRUE the cache is scanned, rather than all registered classes, when iterating through the **ScriptingLanguageInfos** collection. Use the **Refresh** method to update the cache from the registered classes section of the system registry.

See Also

[Application Object](#)

[Item Method](#)

[Refresh Method](#)

[Retrieving DTS System, Package and Log Data](#)

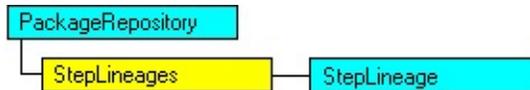
[ScriptingLanguageInfo Object](#)

[UseCache Property](#)

DTS Programming

StepLineages Collection

The **StepLineages** collection contains **StepLineage** objects that provide the contents of step lineage records from Microsoft® SQL Server™ 2000 Meta Data Services. These records contain information about the execution of a step in a DTS package hosted by Meta Data Services.



Properties

[EOF Property](#)

Methods

[Next Method](#)

Remarks

A step lineage record is written each time a step of a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this to be done.

Use the **EnumStepLineages** method of the **PackageRepository** object to return the **StepLineages** collection.

Iterate through the objects of the **StepLineages** collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all the elements will have been fetched. You can also use **For Each ... Next** in Microsoft Visual Basic®.

See Also

[EnumStepLineages Method](#)

[LineageOptions Property](#)

[PackageRepository Object](#)

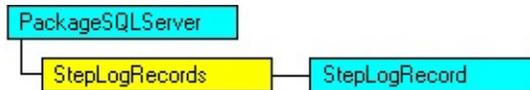
[Retrieving DTS System, Package and Log Data](#)

[StepLineage Object](#)

DTS Programming

StepLogRecords Collection

The **StepLogRecords** collection contains **StepLogRecord** objects that provide the contents of step log records from an instance of Microsoft® SQL Server™. The log records contain information about the execution of a step in a DTS package.



Properties

[EOF Property](#)

Methods

[Next Method](#)

Remarks

A step log record is written to the **msdb** database on the server specified by the package **LogServerName** property each time a step in a package is executed, if the package **LogToSQLServer** property has been set.

Use the **EnumStepLogRecords** method of the **PackageSQLServer** object to return the **StepLogRecords** collection.

Iterate through the objects of the **StepLogRecords** collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all the elements will have been fetched. You can also use **For Each ... Next** in Microsoft Visual Basic®.

See Also

[EnumStepLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageSQLServer Object](#)

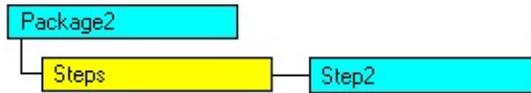
[Retrieving DTS System, Package and Log Data](#)

[StepLogRecord Object](#)

DTS Programming

Steps Collection

The **Steps** collection is a group of **Step2** objects that contain information about the flow and execution of tasks within a DTS package.



Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	New Method
Insert Method	Remove Method
Item Method	

Remarks

The **Step** objects control the flow and execution of tasks within the package. Step execution sequence is determined by their precedence constraints.

Prototype (C/C++)

```
HRESULT GetSteps(IDTSSSteps **pRetVal);
```

See Also

[Creating DTS Package Workflow and Tasks](#)

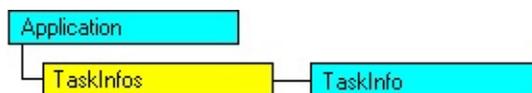
[PrecedenceConstraint Object](#)

[Step2 Object](#)

DTS Programming

TaskInfos Collection

The **TaskInfos** collection contains **TaskInfo** objects that provide information about each DTS task available on the system. These include the tasks supplied by Microsoft® SQL Server™ 2000 and custom tasks implemented by users or other vendors.



Properties

Count Property	UseCache Property
Parent Property	

Methods

Item Method	Refresh Method
-----------------------------	--------------------------------

Remarks

Obtain a reference to the **TaskInfos** collection from the **Application** object. For more information about the tasks supplied with SQL Server 2000, see [Task Objects](#).

You can iterate through the objects of the **TaskInfos** collection using the **Item** method and **Count** property. However, it is faster to use **For Each ... Next** in Microsoft Visual Basic®.

DTS maintains a cache, in the operating system registry, that holds the task information. If the **UseCache** property is TRUE, the cache, rather than all registered classes, is scanned when iterating through the **XXXXs** collection. Use the **Refresh** method to update the cache from the registered classes section of the system registry.

See Also

[Application Object](#)

[Item Method](#)

[Refresh Method](#)

[Retrieving DTS System, Package and Log Data](#)

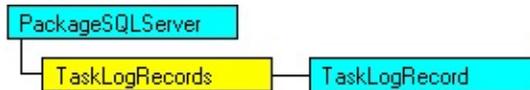
[TaskInfo Object](#)

[UseCache Property](#)

DTS Programming

TaskLogRecords Collection

The **TaskLogRecords** collection contains **TaskLogRecord** objects that provide the contents of task log records from an instance of Microsoft® SQL Server™. The log record contains information about the execution of a task that has been implemented in a Data Transformation Services (DTS) package to write task log records.



Properties

[EOF Property](#)

Methods

[Next Method](#)

Remarks

Task log records are not automatically written by the task classes supplied with SQL Server 2000, but the **PackageLog** interface is available so that a custom task or the script of an **ActiveScriptTask** object can write them. They are written to the **msdb** database on the server specified by the package **LogServerName** property each time a task in a package that has been implemented to write them is executed, if the package **LogToSQLServer** property has been set.

Use the **EnumTaskLogRecords** method of the **PackageSQLServer** object to return the **TaskLogRecords** collection.

Iterate through the objects of the **TaskLogRecords** collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all the elements will have been fetched. You can also use **For Each ... Next** in Microsoft Visual Basic®.

See Also

[ActiveScriptTask Object](#)

[EnumTaskLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageSQLServer Object](#)

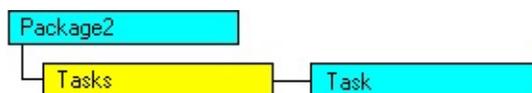
[Retrieving DTS System, Package and Log Data](#)

[TaskLogRecord Object](#)

DTS Programming

Tasks Collection

The **Tasks** collection is a group of **Task** objects that contain information about units of work to be performed as part of the transformation process. The **Tasks** collection contains all of the defined tasks in a DTS package.



Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	New (ID) Method
Insert Method	Remove Method
Item Method	

Remarks

Tasks specifies a **Tasks** collection.

Prototype (C/C++)

```
HRESULT GetTasks(IDTSTasks **pRetVal);
```

See Also

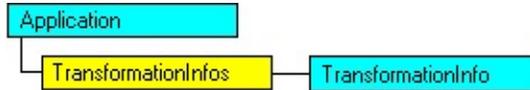
[Creating DTS Package Workflow and Tasks](#)

[Task Object](#)

DTS Programming

TransformationInfos Collection

The **TransformationInfos** collection contains **TransformationInfo** objects that provide information about each DTS transformation available on the system. These include the transformations supplied by Microsoft® SQL Server™ 2000 and custom transformations implemented by users or other vendors.



Properties

Count Property	UseCache Property
Parent Property	

Methods

Item Method	Refresh Method
-----------------------------	--------------------------------

Remarks

Obtain a reference to the **TransformationInfos** collection from the **Application** object. For more information about the transformations supplied with SQL Server 2000, see [Transformation Objects](#).

You can iterate through the objects of the **TransformationInfos** collection using the **Item** method and **Count** property. However, it is faster to use **For Each ... Next** in Microsoft Visual Basic®.

DTS maintains a cache, in the operating system registry, that holds the transformation information. If the **UseCache** property is TRUE the cache is scanned, rather than all registered classes, when iterating through the **TransformationInfos** collection. Use the **Refresh** method to update the cache from the registered classes section of the system registry.

See Also

[Application Object](#)

[Item Method](#)

[Refresh Method](#)

[Retrieving DTS System, Package and Log Data](#)

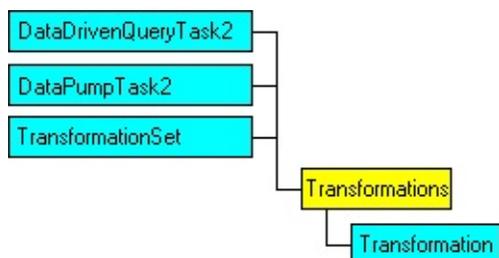
[TransformationInfo Object](#)

[UseCache Property](#)

DTS Programming

Transformations Collection

The **Transformations** collection is a group of **Transformation2** objects that contain information about the transformation, and about source and destination columns.



Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	New (ID) Method
Insert Method	Remove Method
Item Method	

Remarks

Transformations specifies the collection of transformations that transfer data from the data source to the data destination.

Prototype (C/C++)

HRESULT GetTransformations(IDTSTransformations **pRetVal);

See Also

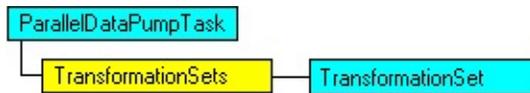
[Adding DTS Transformations](#)

[Transformation2 Object](#)

DTS Programming

TransformationSets Collection

The **TransformationSets** collection contains the **TransformationSet** objects that define the transformations to be performed on the columns of a component rowset in a hierarchical rowset by the **Parallel Data Pump Task** object.



Applies To

ParallelDataPumpTask Object

Properties

Count Property	Parent Property
--------------------------------	---------------------------------

Methods

Add Method	New (Name) Method
Insert Method	Remove Method
Item Method	

Remarks

The **New** method creates a new **TransformationSet** object. The **Add** method adds a **TransformationSet** object to the collection at the last ordinal position. The **Insert** method adds a **TransformationSet** object to the collection at a specified ordinal position, or just ahead of a referenced object in the collection.

The ordinal position of the **TransformationSet** objects in the collection determines the order in which they are mapped to the component rowsets of the source and destination hierarchical rowsets.

Hierarchical rowsets are scanned in column-ordinal order. When a child rowset is encountered, it is mapped to the next **TransformationSet** object, and its columns are scanned before the remaining columns of the parent rowset are scanned. This process is continued recursively until the entire hierarchical rowset is scanned.

See Also

[TransformationSet Object](#)

DTS Programming

Properties

This section defines the properties of the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) objects and collections. Use these properties to retrieve and set the attributes of the DTS components.

DTS Programming

ActiveXScript Property

The **ActiveXScript** property specifies a Microsoft® ActiveX® script text string for an object to execute.

Applies To

ActiveScriptTask Object	Step2 Object
Step Object	

Syntax

object.**ActiveXScript** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	ActiveX script for an object to execute

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetActiveXScript(BSTR *pRetVal);
```

```
HRESULT SetActiveXScript(BSTR NewValue);
```

Remarks

ActiveX scripts also are used by the **DTSTransformScriptProperties2** and

DataPumpTransformScript objects.

See Also

[FunctionName Property](#)

[ScriptLanguage Property](#)

DTS Programming

AddGlobalVariables Property

The **AddGlobalVariables** property specifies whether global variables can be referenced from the current Microsoft® ActiveX® script.

Applies To

ActiveScriptTask Object	Step2 Object
Step Object	

Syntax

object.AddGlobalVariables [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether global variables can be referenced from the current ActiveX script

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetAddGlobalVariables(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetAddGlobalVariables(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE. You access the **GlobalVariables** collection from within ActiveX scripts using the name **DTSGlobalVariables**.

See Also

[GlobalVariables Collection](#)

DTS Programming

AllowIdentityInserts Property

The **AllowIdentityInserts** property specifies whether the SET IDENTITY_INSERT Transact-SQL statement is set to ON before and OFF after the data pump execution.

Applies To

DataPumpTask Object	DataPumpTask2 Object
-------------------------------------	--------------------------------------

Syntax

object.**AllowIdentityInserts**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether SET IDENTITY_INSERT is set to ON during data pump execution

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetAllowIdentityInserts(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetAllowIdentityInserts(VARIANT_BOOL NewValue);
```

Remarks

AllowIdentityInserts only applies to Microsoft® SQL Server™.

See Also

[KeepIdentity Property](#)

DTS Programming

AMSymbol Property

The **AMSymbol** property specifies or returns the string indicating the time format before 12:00 noon (for example, A.M.) when a 12-hour time format is specified.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**AMSymbol** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>string</i>	Suffix string that indicates a time format before 12:00 noon for a 12-hour time format

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT AMSymbol(BSTR* pRetVal);
```

```
HRESULT AMSymbol(BSTR pRetVal);
```

Remarks

The default value is the English "AM".

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("AMSymbol") [= string]
```

See Also

[InputFormat Property](#)

[OutputFormat Property](#)

[PMSymbol Property](#)

DTS Programming

AppendIfFileExists Property

The **AppendIfFileExists** property specifies or returns a value indicating whether data written to a destination file is appended to or is written over data that was present when the file was opened.

Applies To

[DataPumpTransformWriteFile Object](#)

Syntax

object.**AppendIfFileExists** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformWriteFile object.
<i>boolean</i>	If TRUE, data is appended to data that already exists in the file. If FALSE, existing data is overwritten.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT AppendIfFileExists(VARIANT_BOOL* pRetVal);
```

```
HRESULT AppendIfFileExists(VARIANT_BOOL pRetVal);
```

Remarks

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops"AppendIfFileExists") [= boolean]
```

See Also

[ErrorIfFileExists Property](#)

[ErrorIfFileNotFound Property](#)

DTS Programming

Assignments Property

The **Assignments** property returns a reference to the **DynamicPropertiesTaskAssignments** collection.

Applies To

[DynamicPropertiesTask](#)

Syntax

[Set *collection* =] *object*.**Assignments**

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTask object
<i>collection</i>	An object variable compatible with type DTSCustTasks.DynamicPropertiesTaskAssignments

Data Type

[DynamicPropertiesTaskAssignments Collection](#)

Modifiable

Read-only

Prototype (C/C++)

HRESULT Assignments(DynamicPropertiesTaskAssignments** pVal);

See Also

[DynamicPropertiesTaskAssignment Object](#)

DTS Programming

AutoCommitTransaction Property

The **AutoCommitTransaction** property specifies whether an active transaction is committed or rolled back on completion of **Package.Execute**.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**AutoCommitTransaction**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Whether an active transaction is committed or rolled back

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetAutoCommitTransaction(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetAutoCommitTransaction(VARIANT_BOOL NewValue);
```

See Also

[TransactionIsolationLevel Property](#)

[UseTransaction Property](#)

DTS Programming

BatchCompleteFunctionEntry Property

The **BatchCompleteFunctionEntry** property specifies or returns the name of the script function that is to be called for the OnBatchComplete transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**BatchCompleteFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports the OnBatchComplete phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT BatchCompleteFunctionEntry(BSTR* pRetVal);
```

```
HRESULT BatchCompleteFunctionEntry(BSTR pRetVal);
```

Remarks

The `OnBatchComplete` phase is available only in the **DataPumpTask2** object, not in the **DataDrivenQueryTask2** or **ParallelDataPumpTask** objects. The transform is called on success or failure of the batch.

The `OnBatchComplete` script function has no access to the columns of the **DTSSource** and **DTSDestination** collections. The only valid return values are **DTSTransformStat_OK** and **DTSTransformStat_AbortPump**.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("BatchCompleteFunctionEntry") [= string]
```

See Also

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertFailureFunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PreSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

BatchSize Property

The **BatchSize** property specifies the number of rows to load in a batch.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**BatchSize** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Number of rows to load

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetBatchSize(LONG *pRetVal);
```

```
HRESULT SetBatchSize(LONG NewValue);
```

Remarks

The default is 0, which specifies that all rows are to be loaded as a single transaction.

DTS Programming

Catalog Property

The **Catalog** property specifies the name of the catalog in which the connection is initially established.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**Catalog** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>value</i>	Name of a catalog (for example, a database).

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCatalog(BSTR *pRetVal);
```

```
HRESULT SetCatalog(BSTR NewValue);
```

Remarks

When the connection uses the Microsoft® OLE DB Provider for SQL Server, the **Catalog** property is a database name.

See Also

[DataSource Property](#)

[UDLPath Property](#)

DTS Programming

CCLine Property

The **CCLine** property specifies e-mail addresses to include on the CC: line.

Applies To

[SendMailTask Object](#)

Syntax

object.**CCLine**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	E-mail addresses to include on the CC: line

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCCLine(BSTR *pRetVal);
```

```
HRESULT SetCCLine(BSTR NewValue);
```

Remarks

E-mail addresses must be separated by semicolons.

See Also

[FileAttachments Property](#)

[MessageText Property](#)

[Subject Property](#)

[ToLine Property](#)

DTS Programming

CharacterCount Property

The **CharacterCount** property specifies or returns the number of characters in the substring of the source column to be copied by custom transformations.

Applies To

[DataPumpTransformMidString Object](#)

Syntax

object.**CharacterCount** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformMidString object
<i>value</i>	Number of characters in the substring to be copied

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT CharacterCount(long* pRetVal);
```

```
HRESULT CharacterCount(long pRetVal);
```

Remarks

If a value less than 1 is provided for **CharacterCount**, a zero-length string is

copied. If a value greater than the number of characters available to be copied is provided, the entire source string from the specified **CharacterStart** to the end of the string is copied.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("CharacterCount") [= value]
```

See Also

[CharacterStart Property](#)

DTS Programming

CharacterStart Property

The **CharacterStart** property specifies or returns the starting position of the substring of the source column to be copied by custom transformations.

Applies To

[DataPumpTransformMidString Object](#)

Syntax

object.**CharacterStart** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformMidString object
<i>value</i>	Position of the starting character of the substring to be copied

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT CharacterStart(long* pRetVal);
```

```
HRESULT CharacterStart(long pRetVal);
```

Remarks

If the substring starts at the first character of the source column, the property has

the value of 1. If a value less than 1 is provided, 1 is used. If a value greater than the number of characters in the source string is provided, a zero-length string is copied.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("CharacterStart") [= value]
```

See Also

[CharacterCount Property](#)

DTS Programming

CheckConstraints Property

The **CheckConstraints** property specifies whether any constraints must be checked while data is loaded.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**CheckConstraints** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Boolean that specifies whether any constraints are checked

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCheckConstraints(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetCheckConstraints(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

DTS Programming

ClassID Property

The **ClassID** property returns the class ID and a globally unique identifier (GUID), under which a class of a component used by Data Transformation Services (DTS) is registered in the operating system registry.

Applies To

OLEDBProviderInfo Object	TaskInfo Object
ScriptingLanguageInfo Object	TransformationInfo Object

Syntax

object.ClassID

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetClassID(BSTR* pRetVal);
```

Remarks

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits.

See Also

[ImplementationFileName Property](#)

[ImplementationFileVersionString Property](#)

[UseCache Property](#)

DTS Programming

CloseConnection Property

The **CloseConnection** property specifies whether to close a connection on completion of a step.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**CloseConnection** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Whether to close the connection on completion

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCloseConnection(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetCloseConnection(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[Connection2 Object](#)

DTS Programming

Codepage Property

The **Codepage** property specifies the code page to use while loading data.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**Codepage** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Code page to use

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCodepage(BSTR *pRetVal);
```

```
HRESULT SetCodepage(BSTR NewValue);
```

Remarks

Use one of these values:

- ACP

- OEM (default)
- RAW
- A code page number (for example, 850)

DTS Programming

ColumnID Property

The **ColumnID** property specifies the column ID of a source or destination column.

Applies To

[Column Object](#)

Syntax

object.**ColumnID** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Column identifier

Data Type

Variant

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetColumnID(VARIANT *pRetVal);
```

```
HRESULT SetColumnID(VARIANT NewValue);
```

Remarks

The **ColumnID** property is typically the same as the column **Name**. The column **Ordinal** is a number that determines the position of the column in the column

order.

See Also

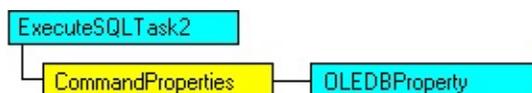
[Name Property](#)

[Ordinal Property](#)

DTS Programming

CommandProperties Property

The **CommandProperties** property returns a reference to an **OLEDBProperties** collection, which contains an **OLEDBProperty** object for each OLE DB command property for the connection.



Applies To

ExecuteSQLTask Object	ExecuteSQLTask2 Object
---------------------------------------	--

Syntax

object.**CommandProperties**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[OLEDBProperties Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCommandProperties(IDTSOleDbProperties **pRetVal);
```

See Also

[Connection2 Object](#)

[OLEDBProperty Object](#)

DTS Programming

CommandTimeout Property

The **CommandTimeout** property specifies the amount of time, in seconds, before the command is presumed to have failed.

Applies To

ExecuteSQLTask Object	ExecuteSQLTask2 Object
---------------------------------------	--

Syntax

object.**CommandTimeout** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Time, in seconds, before the command is presumed to have failed

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCommandTimeout(LONG *pRetVal);
```

```
HRESULT SetCommandTimeout(LONG NewValue);
```

Remarks

A value of 0 (default) indicates no time-out period.

DTS Programming

CommitSuccess Property

The **CommitSuccess** property specifies whether to commit a step if it completes successfully.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**CommitSuccess** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether to commit the step

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCommitSuccess(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetCommitSuccess(VARIANT_BOOL NewValue);
```

See Also

[JoinTransactionIfPresent Property](#)

[RollbackFailure Property](#)

DTS Programming

Computer Property

The **Computer** property specifies the network node name of the computer on which a Data Transformation Services (DTS) package was executed. This property applies only to packages for which a lineage or log record was written

Applies To

PackageLineage Object	PackageLogRecord Object
---------------------------------------	---

Syntax

object.**Computer**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetComputer(BSTR *pRetVal);
```

See Also

[Operator Property](#)

DTS Programming

Connected Property

The **Connected** property indicates whether a connection is currently active.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**Connected**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetConnected(VARIANT_BOOL *pRetVal);
```

See Also

[AcquireConnection Method](#)

[ConnectImmediate Property](#)

[ReleaseConnection Method](#)

DTS Programming

ConnectImmediate Property

The **ConnectImmediate** property specifies whether to make an immediate connection, either when a Data Transformation Services (DTS) package starts running or at the time a step that references a task using this connection executes.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**ConnectImmediate** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Value indicating whether to make an immediate connection

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetConnectImmediate(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetConnectImmediate(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[Connected Property](#)

DTS Programming

ConnectionID Property

The **ConnectionID** property specifies the ID of a **Connection** object you use when connecting to a database or another data source.

Applies To

BulkInsertTask Object	ExecuteSQLTask2 Object
ExecuteSQLTask Object	Lookup Object

Syntax

object.**ConnectionID** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	ID of the Connection object you want to use

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetConnectionID(LONG *pRetVal);
```

```
HRESULT SetConnectionID(LONG NewValue);
```

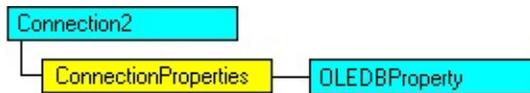
See Also

[Connection2 Object](#)

DTS Programming

ConnectionProperties Property

The **ConnectionProperties** property returns a reference to an **OLEDBProperties** collection used to establish the characteristics of a connection.



Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**ConnectionProperties**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[OLEDBProperties Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetConnectionProperties(IDTSOLEDBProperties **pRetVal);
```

Remarks

These properties are specific to the OLE DB provider used by the connection.

See Also

[OLEDBProperty2 Object](#)

DTS Programming

ConnectionTimeout Property

The **ConnectionTimeout** property returns or sets the number of seconds to wait while establishing a connection. After that, an error is generated.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**ConnectionTimeout** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Number of seconds to wait while establishing a connection

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetConnectionTimeout(LONG *pRetVal);
```

```
HRESULT SetConnectionTimeout(LONG NewValue);
```

Remarks

The default is 60 seconds.

See Also

[AcquireConnection Method](#)

[ConnectImmediate Property](#)

DTS Programming

CopyAllObjects Property

The **CopyAllObjects** property specifies whether to transfer all objects from an instance of Microsoft® SQL Server™.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**CopyAllObjects**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether to transfer all objects

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCopyAllObjects(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetCopyAllObjects(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE.

See Also

[CopyData Property](#)

[CopySchema Property](#)

[DropDestinationObjectsFirst Property](#)

[IncludeDependencies Property](#)

[IncludeLogins Property](#)

[IncludeUsers Property](#)

DTS Programming

CopyData Property

The **CopyData** property specifies whether data is copied and whether existing data is replaced or appended to.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**CopyData**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Code that specifies whether data is copied and whether existing data is replaced or appended to

Data Type

[DTSTransfer_CopyDataOption](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCopyData(DTSTransfer_CopyDataOption *pRetVal);
```

```
HRESULT SetCopyData(DTSTransfer_CopyDataOption NewValue);
```

Remarks

CopyData must be set to one of the DTSTransfer_CopyDataOption values. The

default is DTSTransfer_ReplaceData.

See Also

[CopyAllObjects Property](#)

[CopySchema Property](#)

[DropDestinationObjectsFirst Property](#)

[IncludeDependencies Property](#)

[IncludeLogins Property](#)

[IncludeUsers Property](#)

DTS Programming

CopySchema Property

The **CopySchema** property specifies whether Microsoft® SQL Server™ database objects are copied.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**CopySchema**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Value indicating whether data is copied

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetCopySchema(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetCopySchema(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE. If **CopySchema** is set to FALSE, the objects are not copied. If **CopySchema** is FALSE, data is copied only if the **CopyData** property

is TRUE.

See Also

[CopyAllObjects Property](#)

[CopyData Property](#)

[DropDestinationObjectsFirst Property](#)

[IncludeDependencies Property](#)

[IncludeLogins Property](#)

[IncludeUsers Property](#)

DTS Programming

Count Property

The **Count** property specifies the number of items in a Data Transformation Services (DTS) collection.

Applies To

Columns Collection	Properties Collection
Connections Collection	SavedPackageInfos Collection
DTSMQMessages Collection	ScriptingLanguageInfos Collection
DynamicPropertiesTaskAssignments Collection	Steps Collection
GlobalVariables Collection	TaskInfos Collection
Lookups Collection	Tasks Collection
OLEDBProperties Collection	TransformationInfos Collection
OLEDBProviderInfos Collection	Transformations Collection
PrecedenceConstraints Collection	TransformationSets Collection

Syntax

object.**Count**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCount(long *pRetVal);
```

See Also

[Item Method](#)

DTS Programming

CreationDate Property

The **CreationDate** property specifies the date and time the Data Transformation Services (DTS) package was created.

Applies To

Package Object	PackageInfo Object
Package2 Object	

Syntax

object.CreationDate

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCreationDate(DATE *pRetVal);
```

See Also

[CreatorComputerName Property](#)

[CreatorName Property](#)

[PackageCreationDate Property](#)

DTS Programming

CreatorComputerName Property

The **CreatorComputerName** property specifies the network name of the computer on which the Data Transformation Services (DTS) package was created.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**CreatorComputerName**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCreatorComputerName(BSTR *pRetVal);
```

See Also

[CreationDate Property](#)

[CreatorName Property](#)

DTS Programming

CreatorName Property

The **CreatorName** property specifies the name of the user who created the Data Transformation Services (DTS) package.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.CreatorName

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCreatorName(BSTR *pRetVal);
```

Remarks

An application must not rely on the **CreatorName** property being set. For example, if a package is created on a computer running Microsoft® Windows® 98, in some cases, no value is entered for the user name.

See Also

[CreationDate Property](#)

[CreatorComputerName Property](#)

DTS Programming

CustomTask Property

The **CustomTask** property returns a reference to the class-specific task object.

Applies To

[Task Object](#)

Syntax

object.**CustomTask** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Task object
<i>value</i>	Returns a reference to the class-specific task object

Data Type

Depends on the Data Transformation Services (DTS) task class being used.

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCustomTask(IDTSCustomTask **pRetVal);
```

Remarks

The **New** method of the **Tasks** collection creates a (generic) **Task** object and a class-specific task object.

Example

The relationship between the (generic) **Task** object, the class-specific task object and the **Step2** object is illustrated by the following Microsoft® Visual Basic® code:

```
Dim objPackage As DTS.Package2
Dim objTask As DTS.Task 'This is the generic Task object
Dim objStep As DTS.Step2
Dim objDataPump As DTS.DataPumpTask2 'This is the class-specific task object
...
Set objStep = objPackage.Steps.New
objStep.Name = "LowerCaseStep"
Set objTask = objPackage.Tasks.New("DTSDataPumpTask")
Set objDataPump = objTask.CustomTask
objDataPump.Name = "LowerCaseTask"
objStep.TaskName = objDataPump.Name
objPackage.Steps.Add objStep
objPackage.Tasks.Add objTask
```

See Also

[New \(ID\) Method](#)

[Step2 Object](#)

[Tasks Collection](#)

DTS Programming

CustomTaskID Property

The **CustomTaskID** property returns the programmatic identifier (ProgID) or class identifier (CLSID) of the class-specific object for this task.

Applies To

[Task Object](#)

Syntax

object.CustomTaskID

Part	Description
<i>object</i>	Expression that evaluates to a Task object

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetCustomTaskID(BSTR *pRetVal);
```

Remarks

This property returns the ProgID or CLSID used in the call to the **New** method or **CoCreateInstance** function that created the **Task** object.

See Also

New (ID) Method

DTS Programming

DataFile Property

The **DataFile** property specifies the universal naming convention (UNC) path of the file from which to load the data.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**DataFile**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	UNC path of the file

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetDataFile(BSTR *pRetVal);

HRESULT SetDataFile(BSTR NewValue);

Remarks

The UNC path is relative to the server on which the **bulk insert** command will run.

See Also

[DataFileType Property](#)

[FormatFile Property](#)

DTS Programming

DataFileNonOverwritable Property

The **DataFileNonOverwritable** property returns or sets a value indicating whether a data file message can overwrite an existing data file.

Applies To

[DTSMessagesQueueTask Object](#)

Syntax

object.**DataFileNonOverwritable** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessagesQueueTask object
<i>value</i>	Boolean that specifies whether an existing data file can be overwritten

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDataFileNonOverwritable(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetDataFileNonOverwritable(VARIANT_BOOL NewValue);
```

Remarks

The step fails if the data file already exists and **DataFileNonOverwritable** is

TRUE. **DataFileNonOverwritable** is ignored if **ReceiveMessageType** is other than **DTSMQMessageType_DataFile**.

See Also

[DTSMQMessageType](#)

[ReceiveMessageType Property](#)

[SaveDataFileName Property](#)

DTS Programming

DataFileType Property

The **DataFileType** property specifies the type of the data file to insert.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**DataFileType** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Type of the data file

Data Type

[DTSBulkInsert_DataFileType](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDataFileType(DTSBulkInsert_DataFileType *pRetVal);
```

```
HRESULT SetDataFileType(DTSBulkInsert_DataFileType NewValue);
```

Remarks

DataFileType must be set to one of the [DTSBulkInsert_DataFileType](#) values. The default is [DTSBulkInsert_DataFileType_Char](#).

See Also

[DataFile Property](#)

DTS Programming

DataPumpOptions Property

The **DataPumpOptions** property returns or sets extended Data Transformation Services (DTS) data pump options.

Applies To

[DataPumpTask2 Object](#)

Syntax

object.**DataPumpOptions** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTask2 object
<i>value</i>	Value that specifies the extended options

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetDataPumpOptions(long *pRetVal);

HRESULT SetDataPumpOptions(long NewValue);

Remarks

The following table lists the valid values for the **DataPumpOptions** property.

Value	Description
-------	-------------

0	Default
1	Commits all successful batches including the final batch, even if the data pump terminates. Use this option to support restartability.

See Also

[FastLoadOptions Property](#)

[InsertCommitSize Property](#)

[UseFastLoad Property](#)

DTS Programming

DataSource Property

The **DataSource** property specifies a data source name appropriate to the OLE DB provider being used.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.DataSource [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Data source name

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDataSource(BSTR *pRetVal);
```

```
HRESULT SetDataSource(BSTR NewValue);
```

Remarks

The data source can be a Microsoft® SQL Server™ name, file name, or some other specification meaningful to the provider.

See Also

[Catalog Property](#)

DTS Programming

Data Type Property

The **Data Type** property specifies the data type of a **Column** object.

Applies To

[Column Object](#)

Syntax

object.**Data Type** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Column data type

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDataType(LONG *pRetVal);
```

```
HRESULT SetDataType(LONG NewValue);
```

Remarks

Data Transformation Services (DTS) column data types match those used in Microsoft® ActiveX® Data Objects (ADO), both in name and value, and OLE DB type identifiers (DBTYPES).

You can find the valid values for the **Data Type** property by searching for **DBTYPEENUM** in the include file OLEDB.h. OLEDB.h is installed in c:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\ by default.

See Also

[Flags Property](#)

[NumericScale Property](#)

[Precision Property](#)

[Size Property](#)

DTS Programming

Day?LongName Property

The **Day?LongName** property specifies or returns the string to be used for the full name of the indicated day of the week. ? is a number from 1 through 7 that indicates the day of the week.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**Day?LongName** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
?	Day of week number from 1 through 7
<i>name</i>	Full name of the specified day of week

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDay?LongName(BSTR* pRetVal);
```

```
HRESULT SetDay?LongName(BSTR pRetVal);
```

Remarks

The default value is the English day of week name.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("Day?LongName") [= name]
```

See Also

[Day?ShortName Property](#)

[GetDayLongName Method](#)

[GetDayShortName Method](#)

[InputFormat Property](#)

[OutputFormat Property](#)

[SetDayLongName Method](#)

[SetDayShortName Method](#)

DTS Programming

Day?ShortName Property

The **Day?ShortName** property specifies or returns the string to be used for the short (3-character abbreviation) name of the indicated day of the week. ? is a number from 1 through 7 that indicates the number of the week.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**Day?ShortName** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
?	Day of week number from 1 through 7
<i>name</i>	Short (3-character abbreviation) name of the specified day of week

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDay?ShortName(BSTR* pRetVal);
```

```
HRESULT SetDay?ShortName(BSTR pRetVal);
```

Remarks

The default value is the English abbreviation for the day of the week.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("Day?ShortName") [= name]
```

See Also

[Day?LongName Property](#)

[GetDayLongName Method](#)

[GetDayShortName Method](#)

[InputFormat Property](#)

[OutputFormat Property](#)

[SetDayLongName Method](#)

[SetDayShortName Method](#)

DTS Programming

DeleteQuery Property

The **DeleteQuery** property specifies a string of one or more parameterized SQL statements to execute at the destination as the Delete query.

Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.DeleteQuery [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Parameterized string of SQL statements

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDeleteQuery(BSTR *pRetVal);
```

```
HRESULT SetDeleteQuery(BSTR NewValue);
```

Remarks

Although the name of a data driven query property is preset, its content is not

enforced. Any of the queries may be used for any desired operation. The nomenclature is provided as a convenient means of identification, based upon the primary purpose of the operation. For example, the **DeleteQuery** property does not need to contain an Transact-SQL DELETE statement.

See Also

[DeleteQueryColumns Property](#)

[InsertQuery Property](#)

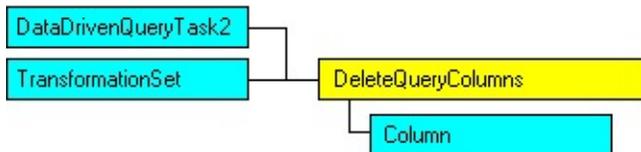
[UpdateQuery Property](#)

[UserQuery Property](#)

DTS Programming

DeleteQueryColumns Property

The **DeleteQueryColumns** property returns a reference to a collection of columns whose values are to be placed into parameters, in sequential order, for the **DeleteQuery** property.



Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.DeleteQueryColumns

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetDeleteQueryColumns(IDTSColumns **pRetVal);
```

See Also

[Column Object](#)

[DeleteQuery Property](#)

[InsertQueryColumns Property](#)

[UpdateQueryColumns Property](#)

[UserQueryColumns Property](#)

DTS Programming

Description Property

The **Description** property returns or sets the textual description of a Data Transformation Services (DTS) object.

Applies To

ActiveScriptTask Object	Package Object
BulkInsertTask Object	Package2 Object
Connection Object	PackageInfo Object
Connection2 Object	PackageLogRecord Object
CreateProcessTask Object	ParallelDataPumpTask Object
CreateProcessTask2 Object	SavedPackageInfo Object
CustomTask Object	ScriptingLanguageInfo Object
DataDrivenQueryTask Object	SendMailTask Object
DataDrivenQueryTask2 Object	Step Object
DataPumpTask Object	Step2 Object
DataPumpTask2 Object	Task Object
DTSMessagesQueueTask Object	TaskInfo Object
DynamicPropertiesTask Object	TransferObjectsTask Object
ExecutePackageTask Object	TransferObjectsTask2 Object
ExecuteSQLTask Object	TransformationInfo Object
ExecuteSQLTask2 Object	TransformationSet Object
OLEDBProviderInfo Object	

Syntax

object.**Description** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Description of the associated object

Data Type

String

Modifiable

Varies

Prototype (C/C++)

```
HRESULT GetDescription(BSTR *pRetVal);
```

```
HRESULT SetDescription(BSTR NewValue);
```

Remarks

Description is usually a read/write property. However, it is a read-only property of informational objects such as the **OLEDBProviderInfo**, **PackageInfo**, **SavedPackageInfo**, **ScriptingLanguageInfo**, **TaskInfo** and **TransformationInfo** objects.

When the associated object is a DTS class-specific task object, the **Description** property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("Description") [= value]
```

See Also

[Name Property](#)

[Properties Collection](#)

[Task Object](#)

DTS Programming

DesignerSettings Property

The **DesignerSettings** property specifies the settings that control the features available in Data Transformation Services (DTS) Designer.

Applies To

[Application Object](#)

Syntax

object.**DesignerSettings** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an Application object
<i>value</i>	Sum of values from DTSDesignerSettings constants

Data Type

[DTSDesignerSettings](#)

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetDesignerSettings(DTSDesignerSettings *pRetVal);

HRESULT SetDesignerSettings(DTSDesignerSettings NewValue);

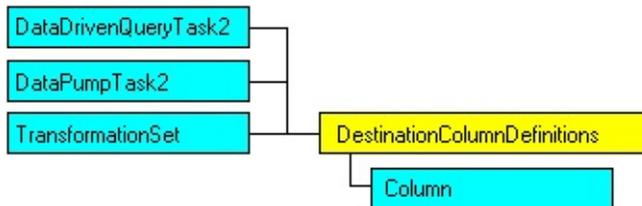
See Also

[JITDebug Property](#)

DTS Programming

DestinationColumnDefinitions Property

The **DestinationColumnDefinitions** property returns a reference to a **Columns** collection that contains the column definitions for a **DataPumpTask2**, **DataDrivenQueryTask2**, or **ParallelDataPumpTask** destination connection.



Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**DestinationColumnDefinitions** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Reference to a Columns collection that contains the destination column definitions

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetDestinationColumnDefinitions(IDTSColumns **pRetVal);
```

Remarks

This collection is constructed from the rowset obtained by querying the destination.

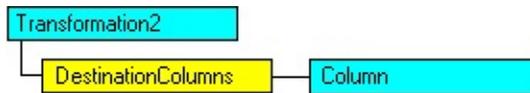
See Also

[Column Object](#)

DTS Programming

DestinationColumns Property

The **DestinationColumns** property returns a reference to a **Columns** collection that contains the definitions for the columns to which the transformation will write.



Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**DestinationColumns**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetDestinationColumns(IDTSColumns **pRetVal);
```

Remarks

The **Ordinal** property of the **Column** objects in the **DestinationColumns** collection determines the order of columns the transformation will see at the

destination. It is used to map these columns to the elements of the **SourceColumns** collection.

See Also

[Column Object](#)

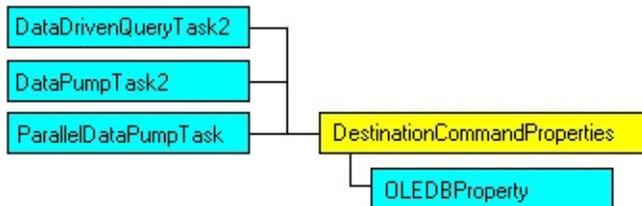
[Ordinal Property](#)

[SourceColumns Property](#)

DTS Programming

DestinationCommandProperties Property

The **DestinationCommandProperties** collection references an **OLEDBProperties** collection whose elements define the properties of the destination connection OLE DB provider.



Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**DestinationCommandProperties**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[OLEDBProperties Collection](#)

Modifiable

Read-only

Prototype (C/C++)

HRESULT GetDestinationCommandProperties(IDTSOleDbProperties

****pRetVal);**

See Also

[Connection2 Object](#)

[OLEDBProperty Object](#)

[SourceCommandProperties Property](#)

DTS Programming

DestinationConnectionID Property

The **DestinationConnectionID** property specifies the connection ID to use at the data destination.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**DestinationConnectionID** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Connection ID to use

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationConnectionID(LONG *pRetVal);
```

```
HRESULT SetDestinationConnectionID(LONG NewValue);
```

Remarks

The **DestinationConnectionID** property maps to the **ID** property of the connection.

See Also

[ID Property](#)

[SourceConnectionID Property](#)

DTS Programming

DestinationDatabase Property

The **DestinationDatabase** property specifies the name of the destination database to use when you transfer Microsoft® SQL Server™ objects.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**DestinationDatabase**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of the destination database

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationDatabase(BSTR *pRetVal);
```

```
HRESULT SetDestinationDatabase(BSTR NewValue);
```

See Also

[DestinationLogin Property](#)

[DestinationPassword Property](#)

[DestinationServer Property](#)

[DestinationUseTrustedConnection Property](#)

DTS Programming

DestinationLogin Property

The **DestinationLogin** property specifies the login ID on a destination server.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**DestinationLogin**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Login ID

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationLogin(BSTR *pRetVal);
```

```
HRESULT SetDestinationLogin(BSTR NewValue);
```

Remarks

DestinationLogin is required if an application is using SQL Server Authentication.

Note It is recommended that you connect to an instance of Microsoft® SQL

Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **DestinationUseTrustedConnection** to TRUE.

See Also

[DestinationDatabase Property](#)

[DestinationPassword Property](#)

[DestinationServer Property](#)

[DestinationUseTrustedConnection Property](#)

DTS Programming

DestinationObjectName Property

The **DestinationObjectName** property specifies the name of a data destination.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**DestinationObjectName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of a data destination

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationObjectName(BSTR *pRetVal);
```

```
HRESULT SetDestinationObjectName(BSTR NewValue);
```

Remarks

Typically, a **DestinationObjectName** is a database table or view name or a

Microsoft® Excel worksheet name.

A **DataPumpTask2** object uses the **DestinationObjectName** property to open a simple rowset on the named destination object if nothing is specified for the **DestinationSQLStatement** property. A **DataDrivenQueryTask2** object uses the **DestinationObjectName** property only to retrieve meta data and then releases the rowset.

Example

The following code illustrates how to specify a database table for the destination object name:

```
objDataPump.DestinationObjectName = "pubs.dbo.authors"
```

The following code illustrates how to specify an Excel worksheet for the destination object name:

```
objDataPump.DestinationObjectName = "DailyReport$"
```

See Also

[DestinationSQLStatement Property](#)

[SourceObjectName Property](#)

DTS Programming

DestinationPassword Property

The **DestinationPassword** property specifies the password to use on a destination server.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**DestinationPassword**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Password to use

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationPassword(BSTR *pRetVal);
```

```
HRESULT SetDestinationPassword(BSTR NewValue);
```

Remarks

DestinationPassword is required if an application is using SQL Server Authentication.

Note It is recommended that you connect to an instance of Microsoft® SQL Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **DestinationUseTrustedConnection** to TRUE.

See Also

[DestinationDatabase Property](#)

[DestinationLogin Property](#)

[DestinationServer Property](#)

[DestinationUseTrustedConnection Property](#)

DTS Programming

DestinationPropertyID Property

The **DestinationPropertyID** property sets or returns a string that defines the path through the Data Transformation Services (DTS) object model to the property to be modified by the **DynamicPropertiesTask** object.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**DestinationPropertyID** [= *objectpath*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>objectpath</i>	Path through the DTS package object model to the property to be modified

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT DestinationPropertyID(BSTR* pRetVal);

HRESULT DestinationPropertyID(BSTR pRetVal);

Remarks

The path is a list of objects, starting with the object or collection whose parent is the package, separated by a delimiter made up of three backticks (```). Objects are represented by the object names. Collections are represented by a string from the following set of Microsoft® Visual Basic® constant definitions. The following symbolic names do not need to be used, but the following case-sensitive string values must match exactly:

These are the separators for the DestinationPropertyID strings.

```
Const DTS_OBJECT_SEPARATOR = "```"
```

```
Const DTS_GLOBAL_VARIABLES = "Global Variables"
```

```
Const DTS_TASKS = "Tasks"
```

```
Const DTS_TRANSFORMATIONS = "Transformations"
```

```
Const DTS_DESTINATION_COLUMN_DEFINITIONS = _  
    "DestinationColumnDefinitions"
```

```
Const DTS_USER_QUERY_COLUMNS = "UserQueryColumns"
```

```
Const DTS_UPDATE_QUERY_COLUMNS = "UpdateQueryColumns"
```

```
Const DTS_INSERT_QUERY_COLUMNS = "InsertQueryColumns"
```

```
Const DTS_DELETE_QUERY_COLUMNS = "DeleteQueryColumns"
```

```
Const DTS_DEST_COLUMNS = "DestinationColumns"
```

```
Const DTS_SOURCE_COLUMNS = "SourceColumns"
```

```
Const DTS_LOOKUPS = "Lookups"
```

```
Const DTS_PRECEDENCE_CONSTRAINTS = "PrecedenceConstraints"
```

```
Const DTS_STEPS = "Steps"
```

```
Const DTS_CONNECTIONS = "Connections"
```

```
Const DTS_PROPERTIES = "Properties"
```

```
Const DTS_DESTINATION_COMMAND_PROPERTIES = _  
    "DestinationCommandProperties"
```

```
Const DTS_SOURCE_COMMAND_PROPERTIES = "SourceCommandProperties"
```

```
Const DTS_OLE_DB_PROPERTIES = "OLEDBProperties"
```

Do not include the package object in the string.

Example

The following example assigns a property path string to

DestinationPropertyID:

```
oAssign.DestinationPropertyID = _  
    "Connections``Pubs Authors Info``" & _  
    "OLEDBProperties``Column Lengths``Properties``Value"
```

See Also

[DynamicPropertiesTask Object](#)

DTS Programming

DestinationServer Property

The **DestinationServer** property specifies the name of the destination server when you transfer Microsoft® SQL Server™ objects.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**DestinationServer**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of the destination server

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationServer(BSTR *pRetVal);
```

```
HRESULT SetDestinationServer(BSTR NewValue);
```

See Also

[DestinationDatabase Property](#)

[DestinationLogin Property](#)

[DestinationPassword Property](#)

[DestinationUseTrustedConnection Property](#)

DTS Programming

DestinationSQLStatement Property

The **DestinationSQLStatement** property specifies an SQL statement to execute at the data destination.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**DestinationSQLStatement** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	SQL statement to execute

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationSQLStatement(BSTR *pRetVal);
```

```
HRESULT SetDestinationSQLStatement(BSTR NewValue);
```

Remarks

A **DataPumpTask** object uses the **DestinationSQLStatement** property to open a rowset on destination connection. If no **DestinationSQLStatement** is specified, a simple rowset is opened on the named destination object (for example, a table or view). The rowset must support an Insert operation on the results of a query. A **DataDrivenQueryTask** object uses the **DestinationSQLStatement** property only to retrieve meta data and then immediately closes the rowset.

See Also

[DestinationObjectName Property](#)

[SourceSQLStatement Property](#)

DTS Programming

DestinationTableName Property

The **DestinationTableName** property specifies the name of the table into which to load data.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**DestinationTableName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Name of the table

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationTableName(BSTR *pRetVal);
```

```
HRESULT SetDestinationTableName(BSTR NewValue);
```

Remarks

DestinationTableName can be fully qualified to override the database to which the application is connected (for example, [Northwind].[dbo].[Orders]).

See Also

[ConnectionID Property](#)

[Connection2 Object](#)

DTS Programming

DestinationUseTrustedConnection Property

The **DestinationUseTrustedConnection** property specifies whether Windows Authentication is used.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**DestinationUseTrustedConnection**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether Windows Authentication is used

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestinationUseTrustedConnection(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetDestinationUseTrustedConnection(VARIANT_BOOL  
NewValue);
```

Remarks

The default is FALSE.

Note It is recommended that you connect to an instance of Microsoft® SQL Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **DestinationUseTrustedConnection** to TRUE.

See Also

[DestinationDatabase Property](#)

[DestinationLogin Property](#)

[DestinationPassword Property](#)

[DestinationServer Property](#)

DTS Programming

DestSite Property

The **DestSite** property sets or returns the destination directory to which the transferred files will be moved by a file transfer protocol (FTP) task.

Applies To

[DTSFTPTask](#)

Syntax

object.**DestSite** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>value</i>	Destination directory to which the transferred files will be moved

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestSite(BSTR* pVal);
```

```
HRESULT SetDestSite(BSTR pVal);
```

Remarks

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

Set *taskprops* = *task.Properties*
taskprops("DestSite") [= *value*]

See Also

[SourceLocation Property](#)

[SourceSite Property](#)

DTS Programming

DestTranslateChar Property

The **DestTranslateChar** property sets or returns a value indicating whether translation is performed for character data on the destination server.

Applies To

[TransferObjectsTask2 Object](#)

Syntax

object.**DestTranslateChar** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a TransferObjectsTask2 object.
<i>boolean</i>	Boolean that specifies whether translation is performed for character data on the destination server.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT DestTranslateChar(VARIANT_BOOL* pRetVal);
```

```
HRESULT DestTranslateChar(VARIANT_BOOL pRetVal);
```

Remarks

The default is TRUE. Translation is performed.

See Also

[SourceTranslateChar Property](#)

DTS Programming

DestUseTransaction Property

The **DestUseTransaction** property sets or returns a value indicating whether the operations of the task are performed within a transaction on the destination server.

Applies To

[TransferObjectsTask2 Object](#)

Syntax

object.**DestUseTransaction** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a TransferObjectsTask2 object.
<i>boolean</i>	Boolean that specifies whether operations are performed within a transaction on the destination server.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDestUseTransaction(VARIANT_BOOL* pRetVal);
```

```
HRESULT SetDestUseTransaction(VARIANT_BOOL pRetVal);
```

Remarks

The default is FALSE. Operations do not use a transaction.

DTS Programming

DisableStep Property

The **DisableStep** property specifies whether a step should be executed.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**DisableStep**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether a step is executed

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDisableStep(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetDisableStep(VARIANT_BOOL NewValue);
```

Remarks

If set to TRUE, the step is not executed. This can be a useful setting when you are debugging complex Data Transformation Services (DTS) packages.

See Also

[ExecutionStatus Property](#)

DTS Programming

DropDestinationObjectsFirst Property

The **DropDestinationObjectsFirst** property specifies whether to drop objects if they already exist on the destination.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.DropDestinationObjectsFirst[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Value indicating whether to drop objects

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetDropDestinationObjectsFirst(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetDropDestinationObjectsFirst(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[CopyAllObjects Property](#)

[CopyData Property](#)

[CopySchema Property](#)

[IncludeDependencies Property](#)

[IncludeLogins Property](#)

[IncludeUsers Property](#)

DTS Programming

DTSMessageLineageID Property

The **DTSMessageLineageID** property sets or returns the globally unique identifier (GUID) of the lineage information that was saved with the Data Transformation Services (DTS) package in Microsoft® SQL Server™ 2000 Meta Data Services.

Applies To

[DTSMessageQueueTask Object](#)

Syntax

object.**DTSMessageLineageID** [= *GUID*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>GUID</i>	The GUID of the package lineage information saved in Meta Data Services

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetDTSMessageLineageID(BSTR* pVal);

HRESULT SetDTSMessageLineageID(BSTR pVal);

Remarks

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("DTSMMessageLineageID") [= GUID]
```

See Also

[DTSMMessagePackageID Property](#)

[DTSMMessageVersionID Property](#)

DTS Programming

DTSMessagePackageID Property

The **DTSMessagePackageID** property sets or returns the globally unique identifier (GUID) of the Data Transformation Services (DTS) package that is the source of the message this **DTSMessageQueueTask** object is to receive.

Applies To

[DTSMessageQueueTask Object](#)

Syntax

object.**DTSMessagePackageID** [= *GUID*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>GUID</i>	The GUID of the package that is the message source

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetDTSMessagePackageID(BSTR* pVal);

HRESULT SetDTSMessagePackageID(BSTR pVal);

Remarks

The **DTSMessagePackageID** affects only **DTSMessageQueueTask** objects that

are message receivers.

To determine the package ID of a DTS package, open the package in DTS Designer. In the **Package/Properties** dialog box, click the **General** tab.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("DTSMMessagePackageID") [= GUID]
```

Example

The following example assigns a GUID string to the **DTSMMessagePackageID** property:

```
oCustTask.DTSMMessagePackageID = _  
    "{4E078447-0EFE-11D3-8DFE-00C04FD7B78D}"
```

See Also

[DTSMMessageLineageID Property](#)

[DTSMMessageVersionID Property](#)

DTS Programming

DTSMessageVersionID Property

The **DTSMessageVersionID** property sets or returns the globally unique identifier (GUID) of the Data Transformation Services (DTS) package version that is the source of the message this **DTSMessageQueueTask** object is to receive.

Applies To

[DTSMessageQueueTask Object](#)

Syntax

object.**DTSMessageVersionID** [= *GUID*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>GUID</i>	The GUID of the package version that is the message source

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT DTSMessageVersionID(BSTR* pVal);
```

```
HRESULT DTSMessageVersionID(BSTR pVal);
```

Remarks

The **DTSMessageVersionID** affects only **DTSMessageQueueTask** objects that are message receivers.

To determine the version ID of a DTS package, open the package in DTS Designer. In the **Package/Properties** dialog box, click the **General** tab.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("DTSMessageVersionID") [= GUID]
```

Example

The following example assigns a GUID string to the **DTSMessageVersionID** property:

```
oCustTask.DTSMessageVersionID = _  
    "{4E078447-0EFE-11D3-8DFE-00C04FD7B78D}"
```

See Also

[DTSMessageLineageID Property](#)

[DTSMessagePackageID Property](#)

DTS Programming

EOF Property

The **EOF** property specifies whether all the elements have been fetched while iterating through the associated collection.

Applies To

PackageInfos Collection	StepLineages Collection
PackageLineages Collection	StepLogRecords Collection
PackageLogRecords Collection	TaskLogRecords Collection

Syntax

collection.**EOF**

Part	Description
<i>collection</i>	Expression that evaluates to a collection in the Applies To list

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetEOF(VARIANT_BOOL *pRetVal);
```

Remarks

Iterate through a collection by checking the **EOF** property after calling the **Next** method. If **EOF** is TRUE, **Next** will have returned **Nothing** and all of the elements will have been fetched. The following Microsoft® Visual Basic® code

shows this process:

```
Set object = collection.Next
Do Until collection.EOF
    ...
    Set object = collection.Next
Loop
```

The collections in the Applies To list also can be processed using **For Each ... Next** in Visual Basic:

```
For Each object In collection
    ...
Next object
```

See Also

[Next Method](#)

DTS Programming

ErrorCode Property

The **ErrorCode** property specifies the code for the error that occurred with the associated object.

Applies To

DTSTransformPhaseInfo Object	StepLogRecord Object
PackageLogRecord Object	TaskLogRecord Object
StepLineage Object	

Syntax

object.**ErrorCode**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetErrorCode(long *pRetVal);
```

Remarks

The following table explains how the interpretation of the **ErrorCode** property depends on the object with which it is associated.

--	--

Object	Description
DTSTransformPhaseInfo	Error code for the operation preceding the current transformation phase.
PackageLogRecord	Error code returned from package Execute method.
StepLineage, StepLogRecord	Error code returned from step Execute method.
TaskLogRecord	Error code associated with this log record, as determined by the logging task.

See Also

[DTSDataPumpError](#)

[DTSPackageError](#)

DTS Programming

ErrorDescription Property

The **ErrorDescription** property specifies a textual description for the error that occurred with the associated object.

Applies To

PackageLogRecord Object	StepLogRecord Object
StepLineage Object	TaskLogRecord Object

Syntax

object.**ErrorDescription**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetErrorDescription(BSTR *pRetVal);
```

Remarks

The following table explains how the interpretation of the **ErrorDescription** property depends on the object with which it is associated.

Object	Description
--------	-------------

PackageLogRecord object	Error description returned from package Execute method.
StepLineage, StepLogRecord objects	Error description returned from step Execute method.
TaskLogRecord object	Error description associated with this log record, as determined by the logging task.

See Also

[DTSDatapumpError](#)

[DTSPackageError](#)

[ErrorCode Property](#)

DTS Programming

ErrorHelpContext Property

The **ErrorHelpContext** property specifies a context ID for the error that was returned by the step **Execute** method.

Applies To

[StepLineage Object](#)

Syntax

object.**ErrorHelpContext**

Part	Description
<i>object</i>	Expression that evaluates to a StepLineage object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetErrorHelpContext(LONG *pRetVal);
```

Remarks

The context ID returned by **ErrorHelpContext** can be used to identify the topic that describes the error in the help file specified by the **ErrorHelpFile** property.

See Also

[ErrorCode Property](#)

[ErrorDescription Property](#)

[ErrorHelpFile Property](#)

[ErrorSource Property](#)

DTS Programming

ErrorHelpFile Property

The **ErrorHelpFile** property specifies the name and path of a help file that contains information relevant to the error returned by the step **Execute** method.

Applies To

[StepLineage Object](#)

Syntax

object.**ErrorHelpFile**

Part	Description
<i>object</i>	Expression that evaluates to a StepLineage object

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetErrorHelpFile(BSTR *pRetVal);
```

Remarks

The context ID returned by the **ErrorHelpContext** property can be used to identify the topic that describes the error in the help file specified by **ErrorHelpFile**.

See Also

[ErrorCode Property](#)

[ErrorDescription Property](#)

[ErrorHelpContext Property](#)

[ErrorSource Property](#)

DTS Programming

ErrorIfExists Property

The **ErrorIfExists** property specifies or returns a value indicating whether an error is raised if a destination file already exists.

Applies To

[DataPumpTransformWriteFile Object](#)

Syntax

object.**ErrorIfExists** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformWriteFile object
<i>boolean</i>	Boolean that specifies whether an error is raised if the destination file already exists

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetErrorIfExists(VARIANT_BOOL* pRetVal);
```

```
HRESULT SetErrorIfExists(VARIANT_BOOL pRetVal);
```

Remarks

If FALSE, the existing file is overwritten.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("ErrorIfFileExists") [= boolean]
```

See Also

[AppendIfFileExists Property](#)

[ErrorIfFileNotFound Property](#)

DTS Programming

ErrorIfFileNotFound Property

The **ErrorIfFileNotFound** property specifies or returns a value indicating whether an error is raised if a file named by a source column does not exist.

Applies To

[DataPumpTransformReadFile Object](#)

Syntax

object.**ErrorIfFileNotFound** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformReadFile object
<i>boolean</i>	Boolean that specifies whether an error is raised when the file named by a source column does not exist

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetErrorIfFileNotFound(VARIANT_BOOL* pRetVal);
```

```
HRESULT SetErrorIfFileNotFound(VARIANT_BOOL pRetVal);
```

Remarks

If FALSE, **NULL** is written to the destination column.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("ErrorIfFileNotFound") [= boolean]
```

See Also

[ErrorIfFileExists Property](#)

DTS Programming

ErrorIfReceiveMessageTimeout Property

The **ErrorIfReceiveMessageTimeout** property sets or returns a value indicating whether an error is raised if a message is not found in the specified queue after the specified time-out value.

Applies To

DTSMessagesQueueTask

Syntax

object.**ErrorIfReceiveMessageTimeout** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessagesQueueTask object
<i>boolean</i>	Boolean that specifies whether an error is raised if a message is not found in the queue after the time-out expires

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetErrorIfReceiveMessageTimeout(VARIANT_BOOL* pVal);
```

```
HRESULT GetErrorIfReceiveMessageTimeout(VARIANT_BOOL pVal);
```

Remarks

Default is FALSE.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("ErrorIfReceiveMessageTimeout") [= boolean]
```

See Also

[ReceiveMessageTimeout Property](#)

DTS Programming

ErrorSource Property

The **ErrorSource** property specifies the name of the component that generated the error returned by the step **Execute** method.

Applies To

[StepLineage Object](#)

Syntax

object.**ErrorSource**

Part	Description
<i>object</i>	Expression that evaluates to a StepLineage object

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

HRESULT GetErrorSource(BSTR *pRetVal);

See Also

[ErrorCode Property](#)

[ErrorDescription Property](#)

[ErrorHelpContext Property](#)

ErrorHelpFile Property

DTS Programming

ExceptionFileColumnDelimiter Property

The **ExceptionFileColumnDelimiter** property specifies the column delimiter in the exception file.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**ExceptionFileColumnDelimiter** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Column delimiter for the exception file

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetExceptionFileColumnDelimiter(BSTR *pRetVal);
```

```
HRESULT SetExceptionFileColumnDelimiter(BSTR NewValue);
```

Remarks

The default is "|".

See Also

[ExceptionFileName Property](#)

[ExceptionFileOptions Property](#)

[ExceptionFileRowDelimiter Property](#)

[ExceptionFileTextQualifier Property](#)

DTS Programming

ExceptionFileName Property

The **ExceptionFileName** property specifies the file name path where exception rows are written.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**ExceptionFileName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Exception file specification

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetExceptionFileName(BSTR *pRetVal);
```

```
HRESULT SetExceptionFileName(BSTR NewValue);
```

Remarks

After the task is complete, this file can be used to edit and resubmit the rows manually.

See Also

[ExceptionFileColumnDelimiter Property](#)

[ExceptionFileOptions Property](#)

[ExceptionFileRowDelimiter Property](#)

[ExceptionFileTextQualifier Property](#)

DTS Programming

ExceptionFileOptions Property

The **ExceptionFileOptions** property specifies how Data Transformation Services (DTS) data pump errors and exception rows are written to the exception and error files.

Applies To

DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask2 Object	

Syntax

object.**ExceptionFileOptions** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A sum of values from DTSExceptionFileOptions

Data Type

[DTSExceptionFileOptions](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetExceptionFileOptions(DTSExceptionFileOptions *pRetVal);
```

```
HRESULT SetExceptionFileOptions(DTSExceptionFileOptions NewValue);
```

See Also

[ExceptionFileColumnDelimiter Property](#)

[ExceptionFileName Property](#)

[ExceptionFileRowDelimiter Property](#)

[ExceptionFileTextQualifier Property](#)

DTS Programming

ExceptionFileRowDelimiter Property

The **ExceptionFileRowDelimiter** property specifies the row delimiter for the data in the exception file.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**ExceptionFileRowDelimiter** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Exception file row delimiter

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetExceptionFileRowDelimiter(BSTR *pRetVal);
```

```
HRESULT SetExceptionFileRowDelimiter(BSTR NewValue);
```

Remarks

The default is carriage return/line feed (CR/LF).

See Also

[ExceptionFileColumnDelimiter Property](#)

[ExceptionFileName Property](#)

[ExceptionFileOptions Property](#)

[ExceptionFileTextQualifier Property](#)

DTS Programming

ExceptionFileTextQualifier Property

The **ExceptionFileTextQualifier** property specifies the text qualifier for the data in the exception file.

Applies To

DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask2 Object	

Syntax

object.**ExceptionFileTextQualifier** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Exception file text qualifier

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetExceptionFileTextQualifier(BSTR *pRetVal);
```

```
HRESULT SetExceptionFileTextQualifier(BSTR NewValue);
```

Remarks

The default is no text qualifier.

See Also

[ExceptionFileColumnDelimiter Property](#)

[ExceptionFileName Property](#)

[ExceptionFileOptions Property](#)

[ExceptionFileRowDelimiter Property](#)

DTS Programming

ExecuteInMainThread Property

The **ExecuteInMainThread** property specifies whether the step executes in the main thread of the Data Transformation Services (DTS) package or a worker thread.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**ExecuteInMainThread** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Whether the step executes in the main thread of the package

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetExecuteInMainThread(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetExecuteInMainThread(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

If you are implementing package event handlers in Microsoft® Visual Basic®, set **ExecuteInMainThread** to TRUE for all steps in the package.

See Also

[DTS Package Events in Visual Basic](#)

DTS Programming

ExecutionDate Property

The **ExecutionDate** property specifies the date and time a Data Transformation Services (DTS) package lineage record was written.

Applies To

PackageLineage Object

Syntax

object.**ExecutionDate**

Part	Description
<i>object</i>	Expression that evaluates to a PackageLineage object

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetExecutionDate(DATE *pRetVal);
```

See Also

[ExecutionTime Property](#)

[StartTime Property](#)

DTS Programming

ExecutionResult Property

The **ExecutionResult** property returns the step execution result.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**ExecutionResult**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[DTSSStepExecResult](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetExecutionResult(DTSSStepExecResult *pRetVal);
```

Remarks

The **ExecutionResult** value indicates success or failure.

See Also

[ExecutionStatus Property](#)

StepExecutionResult Property

DTS Programming

ExecutionStatus Property

The **ExecutionStatus** property specifies the status of the step.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**ExecutionStatus**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[DTSSStepExecStatus](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetExecutionStatus(DTSSStepExecStatus *pRetVal);
```

Remarks

Use the step **ExecutionStatus** to determine whether a step:

- Is waiting to be executed.
- Is in progress.

- Has completed with success or failure.
- Has been skipped.

See Also

[ExecutionResult Property](#)

[StepExecutionStatus Property](#)

DTS Programming

ExecutionTime Property

The **ExecutionTime** property specifies the total execution time, in seconds, of the associated object.

Applies To

PackageLogRecord Object	StepLineage Object
Step Object	StepLogRecord Object
Step2 Object	

Syntax

object.**ExecutionTime**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Double

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetExecutionTime(double *pRetVal);
```

Remarks

ExecutionTime of a **PackageLogRecord** is the total execution time of the Data Transformation Services (DTS) package. For the other objects, **ExecutionTime** refers to an individual step.

See Also

[FinishTime Property](#)

[StartTime Property](#)

DTS Programming

ExplicitGlobalVariables Property

The **ExplicitGlobalVariables** property sets or returns a value indicating whether Data Transformation Services (DTS) package global variables must be explicitly added to the **GlobalVariables** collection with the **AddGlobalVariable** method before being referenced.

Applies To

[Package2 Object](#)

Syntax

object.**ExplicitGlobalVariables** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object.
<i>boolean</i>	If TRUE, package global variables must be explicitly added with AddGlobalVariable method. If FALSE, reference to global variables that do not exist cause them to be created.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT ExplicitGlobalVariables(VARIANT_BOOL* pVal);
```

```
HRESULT ExplicitGlobalVariables(VARIANT_BOOL pVal);
```

See Also

[AddGlobalVariable Method](#)

[AddGlobalVariables Property](#)

[GlobalVariables Collection](#)

DTS Programming

FailOnError Property

The **FailOnError** property specifies whether Data Transformation Services (DTS) package execution stops when there is an error in any step.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**FailOnError** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether the package execution stops when there is an error

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFailOnError(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetFailOnError(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

In Microsoft® Visual Basic®, errors will not be raised for the caller of the **Execute** method unless **FailOnError** is TRUE or **FailPackageOnError** is TRUE for the step in which the error occurs. When that happens, the error description will identify only the step that failed. To determine the reason for failure, use the **GetExecutionErrorInfo** method of the step.

See Also

[Execute Method](#)

[FailPackageOnError Property](#)

[GetExecutionErrorInfo Method](#)

DTS Programming

FailPackageOnError Property

The **FailPackageOnError** property specifies whether Data Transformation Services (DTS) package execution stops when there is an error in the step with which it is associated.

Applies To

[Step2 Object](#)

Syntax

object.**FailPackageOnError** [= *value*]

Part	Description
<i>Object</i>	Expression that evaluates to a Step2 object
<i>Value</i>	Whether the package execution stops when there is an error in the step

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFailPackageOnError(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetFailPackageOnError(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

In Microsoft® Visual Basic®, errors will not be raised for the caller of the **Execute** method unless the package **FailOnError** is TRUE or **FailPackageOnError** is TRUE for the step in which the error occurs. When that happens, the error description will identify only the step that failed. To determine the reason for failure, use the **GetExecutionErrorInfo** method of the step.

See Also

[Execute Method](#)

[FailOnError Property](#)

[GetExecutionErrorInfo Method](#)

DTS Programming

FailPackageOnLogFailure Property

The **FailPackageOnLogFailure** property sets or returns a value indicating whether the Data Transformation Services (DTS) package will fail if there is a failure during the logging of the package.

Applies To

[Package2 Object](#)

Syntax

object.**FailPackageOnLogFailure** [= *boolean*]

Part	Description
<i>Object</i>	Expression that evaluates to a Package2 object.
<i>Boolean</i>	If TRUE, the package fails if there is a failure during logging.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT FailPackageOnLogFailure(VARIANT_BOOL* pVal);
```

```
HRESULT FailPackageOnLogFailure(VARIANT_BOOL pVal);
```

Remarks

The default is FALSE.

See Also

[LogServerFlags Property](#)

[LogServerName Property](#)

[LogServerPassword Property](#)

[LogServerUserName Property](#)

[LogToSQLServer Property](#)

DTS Programming

FailPackageOnTimeout Property

The **FailPackageOnTimeout** property specifies whether the Data Transformation Services (DTS) package fails if the task is terminated by the expiration of the time-out period.

Applies To

CreateProcessTask Object	CreateProcessTask2 Object
--	---

Syntax

object.**FailPackageOnTimeout** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether the package fails if the task is terminated by the expiration of the time-out period

Data Type

Variant Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFailPackageOnTimeout(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetFailPackageOnTimeout(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE.

The **TerminateProcessAfterTimeout** property determines if the created process is terminated after the time-out occurs.

See Also

[TerminateProcessAfterTimeout Property](#)

[Timeout Property](#)

DTS Programming

FastLoadOptions Property

The **FastLoadOptions** property specifies SQLOLEDB destination connection options specific for the **UseFastLoad** property.

Applies To

DataPumpTask Object	DataPumpTask2 Object
-------------------------------------	--------------------------------------

Syntax

object.**FastLoadOptions**[= *value*]

Part	Description
<i>Object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	SQLOLEDB destination connection options

Data Type

[DTSTFastLoadOptions](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFastLoadOptions(DTSTFastLoadOptions *pRetVal);
```

```
HRESULT SetFastLoadOptions(DTSTFastLoadOptions NewValue);
```

Remarks

FastLoadOptions must be set to the sum of one or more of the DTSTFastLoadOptions values.

See Also

[InsertCommitSize Property](#)

[UseFastLoad Property](#)

DTS Programming

FetchBufferSize Property

The **FetchBufferSize** property specifies the number of rows to fetch in a single operation from the OLE DB source.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**FetchBufferSize** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Number of rows to fetch in a single operation

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFetchBufferSize(LONG *pRetVal);
```

```
HRESULT SetFetchBufferSize(LONG NewValue);
```

Remarks

Each row may be buffered by the OLE DB provider. When calling **IRowset::GetNextRows**, adjust the **FetchBufferSize** property to achieve the best balance between memory usage and reduced overhead. A value greater than 1 is ignored if the data source uses binary large object (BLOB) storage. The default is 100.

See Also

[InsertCommitSize Property](#)

DTS Programming

FieldTerminator Property

The **FieldTerminator** property specifies the field or column terminator for files that support **char** and **widechar** data types.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**FieldTerminator** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Field terminator

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFieldTerminator(BSTR *pRetVal);
```

```
HRESULT SetFieldTerminator(BSTR NewValue);
```

Remarks

Typical values are ",", ";", tab and "|". These values are also valid in the bulk copy program. The default is tab. It must not be the same as the **RowTerminator**

property.

See Also

[RowTerminator Property](#)

DTS Programming

FileAttachments Property

The **FileAttachments** property specifies the name and path of file attachments.

Applies To

[SendMailTask Object](#)

Syntax

object.**FileAttachments**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	File specifications of attachments

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetFileAttachments(BSTR *pRetVal);

HRESULT SetFileAttachments(BSTR NewValue);

Remarks

Multiple attachment file specifications are separated by semicolons.

See Also

CCLine Property

Subject Property

ToLine Property

DTS Programming

FileColumnName Property

The **FileColumnName** property specifies or returns the name of the source column that contains the name of the file to be written. It must be one of the two source columns of the transformation.

Applies To

[DataPumpTransformWriteFile Object](#)

Syntax

object.**FileColumnName** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformWriteFile object
<i>string</i>	Name of the source column that contains the name and optional path of file to be written

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetFileColumnName(BSTR* pRetVal);

HRESULT SetFileColumnName(BSTR pRetVal);

Remarks

The **FileColumnName** must be a string data type. It cannot be **NULL** or empty, and the data column to which it refers cannot be **NULL** or empty. If the file name column contains a path, it can use either a drive letter or a universal naming convention (UNC) name.

If no path is present, the **FilePath** property can be used to supply the path. However, **FilePath** is always used when it contains a value, even when the file name column contains a path.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("FileColumnName") [= string]
```

See Also

[FilePath Property](#)

DTS Programming

FileName Property

The **FileName** property sets or returns the name and path of the file that contains a Data Transformation Services (DTS) package to be run by an **Execute Package Task** object.

Applies To

[ExecutePackageTask Object](#)

Syntax

object.**FileName** [= *filespec*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object
<i>filespec</i>	Name and path of the file that contains the package

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetFileName(BSTR* pRetVal);

HRESULT SetFileName(BSTR pRetVal);

Remarks

If the **FileName** property is empty, the **ExecutePackageTask** object looks in

Microsoft® SQL Server™ 2000 Meta Data Services or in the SQL Server **msdb** database (depending on the setting of the **UseRepository** property) on the specified server to find the package to be run.

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("FileName") [= filespec]
```

Example

The following example assigns a file specification to the **FileName** property:

```
objCustTask.FileName = "C:\DTS_UE\TestPkg\WriteFileTest.dts"
```

See Also

[PackageName Property](#)

[UseRepository Property](#)

DTS Programming

FilePath Property

The **FilePath** property specifies or returns the path you want to prefix to the file name column in a custom transformation.

Applies To

DataPumpTransformReadFile Object	DataPumpTransformWriteFile Object
--	---

Syntax

object.**FilePath** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>string</i>	Path to prefix to the file name column when it contains no path

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFilePath(BSTR* pRetVal);
```

```
HRESULT SetFilePath(BSTR pRetVal);
```

Remarks

"\" will be appended to the **FilePath** property, when necessary, to separate the

path from the file name in the source column. If a value for **FilePath** is provided, the value will be prefixed to the file name even when it already includes a path.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("FilePath") [= string]
```

See Also

[FileColumnName Property](#)

DTS Programming

FinishTime Property

The **FinishTime** property specifies the date and time when the associated object completed its execution.

Applies To

PackageLogRecord Object	StepLineage Object
Step Object	StepLogRecord Object
Step2 Object	

Syntax

object.**FinishTime**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetFinishTime(DATE *pRetVal);
```

See Also

[ExecutionTime Property](#)

[StartTime Property](#)

DTS Programming

FirstRow Property

The **FirstRow** property specifies the first source row to copy.

Applies To

BulkInsertTask Object	DataPumpTask Object
DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object

Syntax

object.**FirstRow**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	First source row to copy

Data Type

Variant

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFirstRow(VARIANT *pRetVal);
```

```
HRESULT SetFirstRow(VARIANT NewValue);
```

Remarks

The default is 1, which specifies the first row.

See Also

[LastRow Property](#)

DTS Programming

Flags Property

The **Flags** property specifies the OLE DB DBCOLUMN values that describe a column.

Applies To

[Column Object](#)

Syntax

object.**Flags** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Sum of OLE DB DBCOLUMN values

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFlags(LONG *pRetVal);
```

```
HRESULT SetFlags(LONG NewValue);
```

Remarks

You can find the valid values for the **Flags** property by searching for **DBCOLUMNFLAGS_** in the include file OLEDB.h. This default location for

this file is C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\.

See Also

[Data Type Property](#)

DTS Programming

ForceBlobsInMemory Property

The **ForceBlobsInMemory** property specifies whether to store each source binary large object (BLOB) column in a transformation as a single memory allocation, even if storage objects are available from the OLE DB provider.

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**ForceBlobsInMemory** [= *value*]

Part	Description
<i>Object</i>	Expression that evaluates to an object in the Applies To list
<i>Value</i>	Boolean that specifies whether to store each source BLOB column in a transformation as a single memory allocation

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetForceBlobsInMemory(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetForceBlobsInMemory(VARIANT_BOOL NewValue);
```

Remarks

BLOBs are **image**, **ntext**, and **text** data types.

See Also

[ForceSourceBlobsBuffered Property](#)

DTS Programming

ForceSourceBlobsBuffered Property

The **ForceSourceBlobsBuffered** property specifies whether to buffer each source binary large object (BLOB) column in a transformation if storage objects are used.

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**ForceSourceBlobsBuffered** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether to buffer each source BLOB column in a transformation

Data Type

[DTSForceMode](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetForceSourceBlobsBuffered(DTSForceMode *pRetVal);
```

```
HRESULT SetForceSourceBlobsBuffered(DTSForceMode NewValue);
```

Remarks

BLOBs are **image**, **ntext**, and **text** data types.

ForceSourceBlobsBuffered must be set to one of the DTSTForceMode values.

See Also

[ForceBlobsInMemory Property](#)

DTS Programming

FormatFile Property

The **FormatFile** property specifies the name and path of a bulk copy data file to use for the load operation.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**FormatFile** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Specification for the bulk copy data file

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFormatFile(BSTR *pRetVal);
```

```
HRESULT SetFormatFile(BSTR NewValue);
```

Remarks

The format of the bulk copy data file must be acceptable to the **bcp** utility.

See Also

[DataFile Property](#)

DTS Programming

FunctionEntry Property

The **FunctionEntry** property specifies or returns the name of the script function that is to be called for the transformation.

Applies To

DataPumpTransformationScript Object	DTSTransformScriptProperties2 Object
---	--

Syntax

object.**FunctionEntry** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Function to call in the script

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFunctionEntry(*pRetVal);
```

```
HRESULT SetFunctionEntry(NewValue);
```

Remarks

In the **DTSTransformScriptProperties2** object, the **FunctionEntry** property

specifies the function name for the Transform phase. In the **DataPumpTransformScript** object, the Transform phase is the only transformation phase.

The script function specified by **FunctionEntry** has read access to the columns of the **DTSSource** collection and write access to the columns of the **DTSDestination** collection. The valid function return values are specified by the **DTSTransformStatus** constants.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("FunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[InsertFailureFunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PreSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

FunctionName Property

The **FunctionName** property specifies the function name to call in the Microsoft® ActiveX® script associated with a script task or step.

Applies To

ActiveScriptTask Object	Step2 Object
Step Object	

Syntax

object.FunctionName [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Function name that is the script entry point for the step or ActiveX Script task

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetFunctionName(BSTR *pRetVal);
```

```
HRESULT SetFunctionName(BSTR NewValue);
```

See Also

[ActiveXScript Property](#)

[ScriptLanguage Property](#)

[FunctionEntry Property](#)

DTS Programming

Get Property

The **Get** property specifies whether a property value can be read.

Applies To

[Property Object](#)

Syntax

object.**Get**

Part	Description
<i>object</i>	Expression that evaluates to a Property object

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetGet(VARIANT_BOOL *pRetVal);
```

See Also

[Set Property](#)

DTS Programming

IconFile Property

The **IconFile** property returns the name and path of the resource file that contains the icon for the task class.

Applies To

[TaskInfo Object](#)

Syntax

object.**IconFile**

Part	Description
<i>object</i>	Expression that evaluates to a TaskInfo object

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetIconFile(BSTR* pRetVal);
```

Example

The following example shows a value returned by the **IconFile** property:

```
C:\Program Files\Microsoft SQL Server\80\Tools\Binn\Resources\103.
```

See Also

[IconIndex Property](#)

DTS Programming

IconIndex Property

The **IconIndex** property returns an index that identifies the icon for the task class in the resource file that contains it.

Applies To

[TaskInfo Object](#)

Syntax

object.**IconIndex**

Part	Description
<i>object</i>	Expression that evaluates to a TaskInfo object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetIconIndex(long* pRetVal);
```

Remarks

The **IconIndex** property is zero based.

The resource file specification is available from the **IconFile** property.

See Also

IconFile Property

DTS Programming

ID Property

The **ID** property returns a unique numeric identifier assigned to a connection.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**ID** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Unique numeric identifier assigned to a connection

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetID(LONG *pRetVal);
```

```
HRESULT SetID(LONG NewValue);
```

Remarks

Other Data Transformation Services (DTS) objects that use connections use the **ID** property to link to those connections.

See Also

[ConnectionID Property](#)

[DestinationConnectionID Property](#)

[SourceConnectionID Property](#)

[SourceQueryConnectionID Property](#)

DTS Programming

ImplementationFileName Property

The **ImplementationFileName** property returns the name and path of the library file that implements the object class.

Applies To

OLEDBProviderInfo Object	TaskInfo Object
ScriptingLanguageInfo Object	TransformationInfo Object

Syntax

object.ImplementationFileName

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetImplementationFileName(BSTR* pRetVal);
```

Example

The following example shows a value returned by the **ImplementationFileName** property:

```
C:\Program Files\Common Files\System\OLE DB\sqloledb.dll
```

See Also

[ImplementationFileVersionString Property](#)

DTS Programming

ImplementationFileVersionString Property

The **ImplementationFileVersionString** property returns the version number of the library file that implements the object class.

Applies To

OLEDBProviderInfo Object	TaskInfo Object
ScriptingLanguageInfo Object	TransformationInfo Object

Syntax

object.ImplementationFileVersionString

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetImplementationFileVersionString(BSTR* pRetVal);
```

Example

The following examples show values returned by the **ImplementationFileVersionString** property. No particular syntax is enforced:

08.00.0045

2000.02.04

See Also

[ImplementationFileName Property](#)

DTS Programming

IncludeDependencies Property

The **IncludeDependencies** property specifies whether dependent objects are scripted and transferred during a transfer of Microsoft® SQL Server™ objects.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**IncludeDependencies**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether dependent objects are scripted and transferred

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetIncludeDependencies(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetIncludeDependencies(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[CopyAllObjects Property](#)

[CopyData Property](#)

[CopySchema Property](#)

[DropDestinationObjectsFirst Property](#)

[IncludeLogins Property](#)

[IncludeUsers Property](#)

DTS Programming

IncludeLogins Property

The **IncludeLogins** property specifies whether the logins on the source are scripted and transferred during a transfer of Microsoft® SQL Server™ objects.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**IncludeLogins**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether the logins on the source are scripted and transferred

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetIncludeLogins(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetIncludeLogins(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[CopyAllObjects Property](#)

[CopyData Property](#)

[CopySchema Property](#)

[DropDestinationObjectsFirst Property](#)

[IncludeDependencies Property](#)

[IncludeUsers Property](#)

DTS Programming

IncludeUsers Property

The **IncludeUsers** property specifies whether the database users on the source are scripted and transferred during the transfer of Microsoft® SQL Server™ objects.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**IncludeUsers**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether the database users on the source are scripted and transferred

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetIncludeUsers(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetIncludeUsers(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[CopyAllObjects Property](#)

[CopyData Property](#)

[CopySchema Property](#)

[DropDestinationObjectsFirst Property](#)

[IncludeDependencies Property](#)

[IncludeLogins Property](#)

DTS Programming

InMemoryBlobSize Property

The **InMemoryBlobSize** property specifies the byte size of per-column allocation for in-memory binary large objects (BLOBs) in a transformation.

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**InMemoryBlobSize** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Byte size of per-column allocation for in-memory BLOBs

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetInMemoryBlobSize(LONG *pRetVal);
```

```
HRESULT SetInMemoryBlobSize(LONG NewValue);
```

Remarks

BLOBs are **image**, **ntext**, and **text** data types.

See Also

[ForceSourceBlobsBuffered Property](#)

DTS Programming

InputFormat Property

The **InputFormat** property specifies or returns a string that defines the format of the datetime string in the source column. This format string consists of tokens and delimiters: the tokens represent components of the date and time, and the delimiters must explicitly appear in the source column.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**InputFormat** [= *formatstring*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object.
<i>formatstring</i>	String consisting of tokens and delimiters, which define the format of the source column.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT InputFormat(BSTR* pRetVal);
```

```
HRESULT InputFormat(BSTR pRetVal);
```

Remarks

For more information about token definitions, see [OutputFormat Property](#).

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("InputFormat") [= name]
```

See Also

[AMSymbol Property](#)

[Day?LongName Property](#)

[Day?ShortName Property](#)

[GetDayLongName Method](#)

[GetDayShortName Method](#)

[GetMonthLongName Method](#)

[GetMonthShortName Method](#)

[Month??LongName Property](#)

[Month??ShortName Property](#)

[PMSymbol Property](#)

[SetDayLongName Method](#)

[SetDayShortName Method](#)

[SetMonthLongName Method](#)

[SetMonthShortName Method](#)

[ShortYear2000Cutoff Property](#)

DTS Programming

InputGlobalVariableNames Property

The **InputGlobalVariableNames** property returns or specifies a list of Data Transformation Services (DTS) global variable names that are to be used as parameters in a query or created in a subpackage.

Applies To

DataDrivenQueryTask2 Object	ExecuteSQLTask2 Object
DataPumpTask2 Object	ParallelDataPumpTask Object
ExecutePackageTask Object	

Syntax

object.**InputGlobalVariableNames** [= *list*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>list</i>	List of global variable names

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetInputGlobalVariableNames(BSTR *pRetVal);
```

```
HRESULT SetInputGlobalVariableNames(BSTR NewValue);
```

Remarks

The format of the **InputGlobalVariableNames** string is a semicolon delimited, optionally double-quoted or single-quoted list. Quoting is required only when the name contains an embedded delimiter. Embedded delimiters must be doubled. A double-delimiter or a trailing delimiter indicates an empty item. For example:

"gv1";gv2 - delimiters not required here
 gv1;"gv""2" - gv"2 embedded delimiter is doubled
 gv1;;gv2; - contains an empty second and fourth item

The following table describes how the usage of the **InputGlobalVariableNames** list depends on the associated object.

Associated Object	Usage
ExecutePackageTask	Global variables from the collection of the outer DTS package are created or assigned to global variables in the target package. Empty items in the list are skipped.
DataDrivenQueryTask2 DataPumpTask2 ExecuteSQLTask2 ParallelDataPumpTask	Global variables from the collection of the package are substituted for parameters in the source query. Empty items in the list cause the corresponding parameter in the query to be bound to NULL .

See Also

[GlobalVariables Collection](#)

[SourceSQLStatement Property](#)

[SQLStatement Property](#)

DTS Programming

InsertCommitSize Property

The **InsertCommitSize** property specifies the number of rows that are inserted in a single transaction when the **FastLoad** option is being used.

Applies To

DataPumpTask Object	DataPumpTask2 Object
-------------------------------------	--------------------------------------

Syntax

object.**InsertCommitSize** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Number of rows inserted as a single transaction

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetInsertCommitSize(LONG *pRetVal);
```

```
HRESULT SetInsertCommitSize(LONG NewValue);
```

Remarks

The default is 0. This means all inserts into the table are batched in a single transaction.

See Also

[FastLoadOptions Property](#)

[UseFastLoad Property](#)

DTS Programming

InsertFailureFunctionEntry Property

The **InsertFailureFunctionEntry** property specifies or returns the name of the script function that is to be called for the OnInsertFailure transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**InsertFailureFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports the OnInsertFailure phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT InsertFailureFunctionEntry(BSTR* pRetVal);
```

```
HRESULT InsertFailureFunctionEntry(BSTR pRetVal);
```

Remarks

The OnInsertFailure phase occurs when an Insert to the destination fails in the **DataPumpTask2** or **ParallelDataPumpTask** object or after the failure of any of the queries in the **DataDrivenQueryTask2** object.

The OnInsertFailure script function has read access to the columns of the **DTSSource** collection and no access to the columns of the **DTSDestination** collection. The only valid return values are **DTSTransformStat_OK** and **DTSTransformStat_AbortPump**.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("InsertFailureFunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PreSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

InsertQuery Property

The **InsertQuery** property specifies a string of one or more parameterized SQL statements to execute at the destination as the insert query.

Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.InsertQuery [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Parameterized string of SQL statements

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetInsertQuery(BSTR *pRetVal);
```

```
HRESULT SetInsertQuery(BSTR NewValue);
```

Remarks

Although the name of a data-driven query property is preset, its content is not

enforced. Any of the queries may be used for any desired operation. The nomenclature is provided as a convenient means of identification, based upon the primary purpose of the operation. For example, the **InsertQuery** property does not need to contain an INSERT statement.

See Also

[DeleteQuery Property](#)

[DTS Query Strings in Visual Basic](#)

[InsertQueryColumns Property](#)

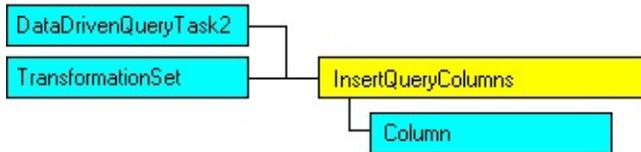
[UpdateQuery Property](#)

[UserQuery Property](#)

DTS Programming

InsertQueryColumns Property

The **InsertQueryColumns** property returns a reference to a collection of column parameters in sequential order for the **InsertQuery** parameter.



Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.InsertQueryColumns

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetInsertQueryColumns(IDTSColumns **pRetVal);
```

See Also

[Column Object](#)

[DeleteQueryColumns Property](#)

[DTS Query Strings in Visual Basic](#)

[InsertQuery Property](#)

[UpdateQueryColumns Property](#)

[UserQueryColumns Property](#)

DTS Programming

InsertSuccessFunctionEntry Property

The **InsertSuccessFunctionEntry** property specifies or returns the name of the script function that is to be called for the OnInsertSuccess transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**InsertSuccessFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports the OnInsertSuccess phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT InsertSuccessFunctionEntry(BSTR* pRetVal);
```

```
HRESULT InsertSuccessFunctionEntry(BSTR pRetVal);
```

Remarks

The OnInsertSuccess phase occurs when an Insert to the destination succeeds in the **DataPumpTask2** or **ParallelDataPumpTask** object or after the success of any of the queries in the **DataDrivenQueryTask2** object.

The OnInsertSuccess script function has read access to the columns of the **DTSSource** collection and no access to the columns of the **DTSDestination** collection. The only valid return values are **DTSTransformStat_OK** and **DTSTransformStat_AbortPump**.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("InsertSuccessFunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertFailureFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PreSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

InTransaction Property

The **InTransaction** property specifies whether the connection is included in the current Data Transformation Services (DTS) package transaction, if one exists.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**InTransaction**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetInTransaction(VARIANT_BOOL *pRetVal);
```

See Also

[JoinTransactionIfPresent Property](#)

[TransactionIsolationLevel Property](#)

[UseTransaction Property](#)

DTS Programming

InUse Property

The **InUse** property specifies whether the connection is currently in use by a task.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**InUse**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetInUse(VARIANT_BOOL *pRetVal);
```

Remarks

When an application calls the connection **AcquireConnection** method, the **InUse** property is set to TRUE.

See Also

[AcquireConnection Method](#)

[ReleaseConnection Method](#)

DTS Programming

IsDefaultValue Property

The **IsDefaultValue** property specifies whether the OLE DB property to which it refers has not been explicitly set to a value.

Applies To

[OLEDBProperty2 Object](#)

Syntax

oledbprop.**IsDefaultValue**

Part	Description
<i>oledbprop</i>	Expression that evaluates to a OLEDBProperty2 object

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetIsDefaultValue(VARIANT_BOOL *pRetVal);
```

Remarks

The **IsDefaultValue** property is TRUE if the value of the corresponding **OLEDBProperty2** object has not been explicitly set or has been set to VT_EMPTY. In the OLE DB specification, VT_EMPTY means restore the default value. If the **OLEDBProperty2** object was explicitly set to any other value, including the default value, the **IsDefaultValue** property is FALSE.

See Also

[Value Property](#)

DTS Programming

IsNTService Property

The **IsNTService** property specifies whether the caller is a Microsoft® Windows NT® 4.0 or Microsoft Windows® 2000 Service.

Applies To

[SendMailTask Object](#)

Syntax

object.**IsNTService**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	Boolean that specifies whether the caller is a Windows NT 4.0 and Windows 2000 Service

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetIsNTService(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetIsNTService(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

Set **IsNTService** to TRUE only if the program that calls the package **Execute** method is installed as a Windows NT 4.0 or Windows 2000 Service.

See Also

[Execute Method \(Package\)](#)

DTS Programming

IsOwner Property

The **IsOwner** property specifies whether the login under which the program retrieving the Data Transformation Services (DTS) package information is running is the same as the owner of the package.

Applies To

[PackageInfo Object](#)

Syntax

object.**IsOwner**

Part	Description
<i>object</i>	Expression that evaluates to a PackageInfo object

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetIsOwner(VARIANT_BOOL *pRetVal);
```

See Also

[Owner Property](#)

DTS Programming

IsPackageDSORowset Property

The **IsPackageDSORowset** property specifies whether the current step executes and returns a rowset when the Data Transformation Services (DTS) package is a rowset provider.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**IsPackageDSORowset** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>value</i>	A value indicating whether the current step executes and returns a rowset

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetIsPackageDSORowset(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetIsPackageDSORowset(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

The **IsPackageDSORowset** property cannot be set to TRUE for steps that reference the DTS flat file OLE DB provider, because that provider does not support the **IDBSchemaRowset** interface. Other providers that do not support **IDBSchemaRowset** also cannot be referenced by steps that return DSO rowsets.

See Also

[Querying a Package with OPENROWSET](#)

[Issuing Distributed Queries Against Package Data](#)

DTS Programming

IsVersionEncrypted Property

The **IsVersionEncrypted** property specifies whether a version of the Data Transformation Services (DTS) package was encrypted when saved.

Applies To

SavedPackageInfo Object

Syntax

object.**IsVersionEncrypted**

Part	Description
<i>object</i>	Expression that evaluates to a SavedPackageInfo object

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetIsVersionEncrypted(VARIANT_BOOL *pRetVal);
```

Remarks

The package version will be encrypted if either an owner password or a user password was specified when the package was saved.

See Also

[SaveToStorageFile Method](#)

[SaveToStorageFileAs Method](#)

DTS Programming

JITDebug Property

The **JITDebug** property specifies whether a run-time error in a Microsoft® ActiveX® script causes a scripting debugger session to be opened.

Applies To

[Application Object](#)

Syntax

object.**JITDebug** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an Application object.
<i>value</i>	If TRUE, the script debugger is entered.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetJITDebug(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetJITDebug(VARIANT_BOOL NewValue);
```

Remarks

If **JITDebug** is TRUE, the following behavior occurs in response to different error types.

Error Type	Behavior
Hard break (for example, a Microsoft Visual Basic® Scripting Edition (VBScript) Stop statement)	The Script Debugger is entered if you agree to debug the script.
Run-time errors and errors raised by objects	The Script Debugger is entered without prompting.

If **JITDebug** is FALSE, the following behavior occurs in response to different error types.

Error Type	Behavior
Hard break (for example, a VBScript Stop statement)	The break is ignored, and the script continues to execute.
Run-time errors	The script fails and an error message is displayed.
Errors raised by objects	An error message supplied by the object is displayed.

See Also

[DesignerSettings Property](#)

DTS Programming

JoinTransactionIfPresent Property

The **JoinTransactionIfPresent** property specifies whether a step executes within the Data Transformation Services (DTS) package transaction.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**JoinTransactionIfPresent**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether a step executes within the package transaction

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetJoinTransactionIfPresent(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetJoinTransactionIfPresent(VARIANT_BOOL NewValue);
```

Remarks

JoinTransactionIfPresent can be set to TRUE if the package **UseTransaction**

property is TRUE.

See Also

[InTransaction Property](#)

[TransactionIsolationLevel Property](#)

[UseTransaction Property](#)

DTS Programming

KeepIdentity Property

The **KeepIdentity** property indicates whether the data in the file is used for the values of identity columns.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**KeepIdentity** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	A value indicating whether the data in the file is used for the values of identity columns

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetKeepIdentity(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetKeepIdentity(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE, which causes the database to assign values to identity

columns upon insertion.

See Also

[KeepNulls Property](#)

DTS Programming

KeepNulls Property

The **KeepNulls** property returns or sets a value indicating whether **NULL** columns should keep **NULL** values even if defaults exists on destination columns.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**KeepNulls** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	A value indicating whether NULL columns keep NULL values

Remarks

The default is FALSE, which causes the database to assign the default value upon insertion.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetKeepNulls(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetKeepNulls(VARIANT_BOOL NewValue);
```

See Also

[KeepIdentity Property](#)

DTS Programming

Language Property

The **Language** property specifies the Microsoft® ActiveX® scripting language the transformation is using.

Applies To

DataPumpTransformScript Object	DTSTransformScriptProperties2 Object
--	--

Syntax

object.**Language** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Scripting language being used

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetLanguage(BSTR *pRetVal);

HRESULT SetLanguage(BSTR NewValue);

Remarks

The default is Microsoft Visual Basic® Scripting Edition (VBScript).

Script languages available on a particular system can be determined by enumerating the **ScriptingLanguageInfos** collection of the **Application** object. For more information about which scripting language to use with Data Transformation Services (DTS), see [ScriptingLanguageInfo Object](#).

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("Language") [= string]
```

See Also

[Application Object](#)

[ScriptingLanguageInfos Collection](#)

[ScriptLanguage Property](#)

[Text Property](#)

DTS Programming

LastOwnerTaskName Property

The **LastOwnerTaskName** property specifies the last task to use the connection.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.LastOwnerTaskName

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetLastOwnerTaskName(BSTR *pRetVal);
```

See Also

[Connected Property](#)

[ReleaseConnection Method](#)

DTS Programming

LastRow Property

The **LastRow** property specifies the last source row to copy.

Applies To

BulkInsertTask Object	DataPumpTask Object
DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object

Syntax

object.**LastRow**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Last source row to copy

Data Type

Variant (Long in the **BulkInsertTask** object)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetLastRow(VARIANT *pRetVal);
```

```
HRESULT SetLastRow(VARIANT NewValue);
```

Remarks

The default is 0, which specifies that all rows are to be copied.

See Also

[FirstRow Property](#)

DTS Programming

LineageFullID Property

The **LineageFullID** property specifies a globally unique identifier (GUID) that uniquely identifies the lineage record for the execution of a Data Transformation Services (DTS) package stored in Microsoft® SQL Server™ 2000 Meta Data Services.

Applies To

PackageLineage Object	StepLogRecord Object
PackageLogRecord Object	

Syntax

object.**LineageFullID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetLineageFullID(BSTR *pRetVal);
```

Remarks

If the package is not hosted in Meta Data Services, a unique **LineageFullID** is generated by DTS for each execution of the package.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits.

See Also

[GetLastExecutionLineage Method](#)

[LineageOptions Property](#)

[LineageShortID Property](#)

[RepositoryMetadataOptions Property](#)

[Using the Data Lineage Feature](#)

DTS Programming

LineageOptions Property

The **LineageOptions** property specifies how Data Transformation Services (DTS) package execution lineage is presented and recorded.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**LineageOptions**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	The way package execution lineage is presented and recorded

Data Type

[DTSLineageOptions](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetLineageOptions(DTSLineageOptions *pRetVal);
```

```
HRESULT SetLineageOptions(DTSLineageOptions NewValue);
```

Remarks

LineageOptions must be set to one of the DTSLineageOptions values.

See Also

[GetLastExecutionLineage Method](#)

[RepositoryMetadataOptions Property](#)

[Using the Data Lineage Feature](#)

DTS Programming

LineageShortID Property

The **LineageShortID** property specifies a value that uniquely identifies the lineage record for the execution of a Data Transformation Services (DTS) package that is stored in Microsoft® SQL Server™ 2000 Meta Data Services.

Applies To

PackageLineage Object	PackageLogRecord Object
---------------------------------------	---

Syntax

object.**LineageShortID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetLineageShortID(BSTR *pRetVal);
```

Remarks

If the package is not hosted in Meta Data Services, a unique **LineageShortID** property is generated by DTS for each execution of the package. Because **LineageShortID** is a long value, it is more convenient to use for indexed columns in a database than the **LineageFullID**, which is a globally unique

identifier (GUID) string.

See Also

[GetLastExecutionLineage Method](#)

[LineageFullID Property](#)

[LineageOptions Property](#)

[RepositoryMetadataOptions Property](#)

[Using the Data Lineage Feature](#)

DTS Programming

LogDate Property

The **LogDate** property specifies the date and time that the Data Transformation Services (DTS) package log record was written.

Applies To

[PackageLogRecord Object](#)

Syntax

object.**LogDate**

Part	Description
<i>object</i>	Expression that evaluates to a PackageLogRecord object

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetLogDate(DATE *pRetVal);
```

See Also

[FinishTime Property](#)

[StartTime Property](#)

DTS Programming

LogFileName Property

The **LogFileName** property specifies the name and path of the error log file.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**LogFileName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of the log file

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetLogFileName(BSTR *pRetVal);
```

```
HRESULT SetLogFileName(BSTR NewValue);
```

Remarks

If **LogFileName** is specified, Data Transformation Services (DTS) package error entries are copied to this file.

See Also

[FailOnError Property](#)

[WriteCompletionStatusToNTEventLog Property](#)

DTS Programming

LogServerFlags Property

The **LogServerFlags** property sets or returns a value indicating whether Windows Authentication is used to validate access to the log server.

Applies To

[Package2 Object](#)

Syntax

object.**LogServerFlags** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>value</i>	Code that defines the type of user authentication used when accessing the log server

Data Type

[DTSSQLServerStorageFlags](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetLogServerFlags(DTSSQLServerStorageFlags* pVal);
```

```
HRESULT SetLogServerFlags(DTSSQLServerStorageFlags pVal);
```

See Also

[FailPackageOnLogFailure Property](#)

[LogServerName Property](#)

[LogServerPassword Property](#)

[LogServerUserName Property](#)

[LogToSQLServer Property](#)

DTS Programming

LogServerName Property

The **LogServerName** property sets or returns the name of the computer running an instance of Microsoft® SQL Server™ to which package logs are written.

Applies To

[Package2 Object](#)

Syntax

object.**LogServerName** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>name</i>	Name of the server to which logs are written

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetLogServerName(BSTR* pVal);

HRESULT SetLogServerName(BSTR pVal);

See Also

[FailPackageOnLogFailure Property](#)

[LogServerFlags Property](#)

[LogServerPassword Property](#)

[LogServerUserName Property](#)

[LogToSQLServer Property](#)

DTS Programming

LogServerPassword Property

The **LogServerPassword** property sets or returns the password used to log in to the instance of Microsoft® SQL Server™ to which package logs are written.

Applies To

[Package2 Object](#)

Syntax

object.**LogServerPassword** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>string</i>	Password to log in to the instance of SQL Server to which logs are written

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetLogServerPassword(BSTR* pVal);

HRESULT SetLogServerPassword(BSTR pVal);

Remarks

This property is used only when the **LogServerFlags** has the value

DTSSQLStgFlag_Default.

See Also

[FailPackageOnLogFailure Property](#)

[LogServerFlags Property](#)

[LogServerName Property](#)

[LogServerUserName Property](#)

[LogToSQLServer Property](#)

DTS Programming

LogServerUserName Property

The **LogServerUserName** property sets or returns the user name used to log in to the instance of Microsoft® SQL Server™ to which package logs are written.

Applies To

[Package2 Object](#)

Syntax

object.**LogServerUserName** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>string</i>	User name to log in to the instance of SQL Server to which logs are written

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetLogServerUserName(BSTR* pVal);
```

```
HRESULT SetLogServerUserName(BSTR pVal);
```

Remarks

This property is used only when the **LogServerFlags** has the value

DTSSQLStgFlag_Default.

See Also

[FailPackageOnLogFailure Property](#)

[LogServerFlags Property](#)

[LogServerName Property](#)

[LogServerPassword Property](#)

[LogToSQLServer Property](#)

DTS Programming

LogToSQLServer Property

The **LogToSQLServer** property sets or returns a value indicating whether Data Transformation Services (DTS) package execution is logged to the specified Microsoft® SQL Server™ 2000 **msdb** database.

Applies To

[Package2 Object](#)

Syntax

object.**LogToSQLServer** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>boolean</i>	Boolean that specifies whether package execution is logged to the specified msdb database

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetLogToSQLServer(VARIANT_BOOL* pRetVal);
```

```
HRESULT SetLogToSQLServer(VARIANT_BOOL pRetVal);
```

Remarks

The default is FALSE, package execution is logged to a log file or the Microsoft Windows® event log. If TRUE, the **LogServerName** property specifies the instance of SQL Server to which logs are written.

See Also

[FailPackageOnLogFailure Property](#)

[LogServerFlags Property](#)

[LogServerName Property](#)

[LogServerPassword Property](#)

[LogServerUserName Property](#)

DTS Programming

LowerCaseString Property

The **LowerCaseString** property specifies or returns a value indicating whether the alphabetical characters in the source column string copied by custom transformations are converted to lowercase characters.

Applies To

DataPumpTransformMidString Object	DataPumpTransformTrimString Object
---	--

Syntax

transerver.**LowerCaseString** [= *logical*]

Part	Description
<i>transerver</i>	Expression that evaluates to a transformation object from the Applies To list
<i>logical</i>	Boolean that specifies whether the alpha characters in the source string are converted to uppercase

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT LowerCaseString(VARIANT_BOOL* pRetVal);
```

```
HRESULT LowerCaseString(VARIANT_BOOL pRetVal);
```

Remarks

The default is FALSE.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("LowerCaseString") [= logical]
```

See Also

[DataPumpTransformLowerString Object](#)

[UpperCaseString Property](#)

DTS Programming

MaxCacheRows Property

The **MaxCacheRows** property specifies the maximum number of rows to cache.

Applies To

[Lookup Object](#)

Syntax

object.**MaxCacheRows** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Lookup object
<i>value</i>	Maximum number of rows that are cached

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetMaxCacheRows(LONG *pRetVal);
```

```
HRESULT SetMaxCacheRows(LONG NewValue);
```

Remarks

The **Execute** method of the **DTSLookups** scripting object checks the cache of the Lookup object for the requested row before querying the lookup's data source. This results in significant performance improvement if the target row is

usually found in the cache.

A value of 0 means no rows are cached. The default is 100.

See Also

[ConnectionID Property](#)

[Execute Method \(DTS\)](#)

DTS Programming

MaxConcurrentSteps Property

The **MaxConcurrentSteps** property specifies the maximum number of Data Transformation Services (DTS) steps executing concurrently on separate threads.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**MaxConcurrentSteps** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Maximum number of steps executing concurrently

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetMaxConcurrentSteps(LONG *pRetVal);
```

```
HRESULT SetMaxConcurrentSteps(LONG NewValue);
```

Remarks

The default is 4.

See Also

[ExecuteInMainThread Property](#)

DTS Programming

MaximumErrorCount Property

The **MaximumErrorCount** property specifies the maximum number of error rows before the data pump terminates.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**MaximumErrorCount** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Maximum number of error rows before the data pump terminates

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetMaximumErrorCount(LONG *pRetVal);
```

```
HRESULT SetMaximumErrorCount(LONG NewValue);
```

Remarks

The default is 0.

See Also

[FailOnError Property](#)

[FailPackageOnError Property](#)

[GetExecutionErrorInfo Method](#)

DTS Programming

MaximumErrors Property

The **MaximumErrors** property specifies the maximum number of errors that can occur before the server terminates the load operation.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**MaximumErrors** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Maximum number of errors that can occur

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetMaximumErrors(LONG *pRetVal);

HRESULT SetMaximumErrors(LONG NewValue);

Remarks

The default is 10.

Transformations will fail at the first error if **Fastload** is being used as long as the

BatchSize property is 0, even if **MaximumErrors** is changed. This failure occurs because all the rows are batched in a single transaction. If you want to log more errors to the exception file, either do not use **Fastload** or set the **BatchSize** property to another value, such as 1.

See Also

[BatchSize Property](#)

DTS Programming

MessageDataFile Property

The **MessageDataFile** property sets or returns the name and path of the file that provides the data for a **DTSMQMessageQueueTask** object data file message.

Applies To

[DTSMQMessage Object](#)

Syntax

object.**MessageDataFile** [= *filespec*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMQMessage object
<i>filespec</i>	Name and path of the file that provides the data for the message

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT MessageDataFile(BSTR* pVal);

HRESULT MessageDataFile(BSTR pVal);

Remarks

The **MessageDataFile** property must be defined if the **MessageType** property is `DTSMQMessageType_DataFile`.

See Also

[DTSMQMessageType](#)

[MessageGlobalVariables Property](#)

[MessageString Property](#)

[MessageType Property](#)

DTS Programming

MessageGlobalVariables Property

The **MessageGlobalVariables** property sets or returns a list of the names of the global variables that provides the data for a **DTSMQMessageQueueTask** object global variables message.

Applies To

[DTSMQMessage Object](#)

Syntax

object.**MessageGlobalVariables** [= *list*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMQMessage object
<i>list</i>	Semicolon-separated list of the Data Transformation Services (DTS) global variables used to construct the message

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT MessageGlobalVariables(BSTR* pVal);

HRESULT MessageGlobalVariables(BSTR pVal);

Remarks

The **MessageGlobalVariables** property must be defined if the **MessageType** property is DTSMQMessageType_DTSGlobalVariables. The global variables must be defined in the package that contains the **DTSMessageQueueTask** object.

See Also

[DTSMQMessageType](#)

[MessageDataFile Property](#)

[MessageString Property](#)

[MessageType Property](#)

DTS Programming

MessageString Property

The **MessageString** property sets or returns the string used as the data for a **DTSMQueueTask** object string message.

Applies To

[DTSMQueueTask Object](#)

Syntax

object.**MessageString** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMQueueTask object
<i>string</i>	String used as the data for the message

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT MessageString(BSTR* pVal);
```

```
HRESULT MessageString(BSTR pVal);
```

Remarks

The **MessageString** property must be defined if the **MessageType** property is **DTSMQueueTaskType_String**.

See Also

[DTSMQMessageType](#)

[MessageDataFile Property](#)

[MessageGlobalVariables Property](#)

[MessageType Property](#)

DTS Programming

MessageText Property

The **MessageText** property is the body of an e-mail message.

Applies To

[SendMailTask Object](#)

Syntax

object.**MessageText**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	Text string that is the body of an e-mail message

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetMessageText(BSTR *pRetVal);

HRESULT SetMessageText(BSTR NewValue);

See Also

[CCLine Property](#)

[Subject Property](#)

ToLine Property

DTS Programming

MessageType Property

The **MessageType** property sets or returns the type of message defined by the **DTSMQMessage** object.

Applies To

[DTSMQMessage Object](#)

Syntax

object.**MessageType** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMQMessage object
<i>value</i>	Code that defines the message type

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT MessageType(long* pVal);

HRESULT MessageType(long pVal);

Remarks

The valid values for this property are defined by the **DTSMQMessageType** constants.

See Also

[DTSMQMessageType](#)

[MessageDataFile Property](#)

[MessageGlobalVariables Property](#)

[MessageString Property](#)

DTS Programming

Month??LongName Property

The **Month??LongName** property specifies or returns the string to be used for the long (full) name of the indicated month. ?? is a month number from 1 through 12.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**Month??LongName** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object.
??	Month number from 1 through 12, without a leading 0.
<i>name</i>	Long (full) name of the specified month.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT Month??LongName(BSTR* pRetVal);
```

```
HRESULT Month??LongName(BSTR pRetVal);
```

Remarks

The default value is the English month name.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("Month??LongName") [= name]
```

See Also

[GetMonthLongName Method](#)

[GetMonthShortName Method](#)

[InputFormat Property](#)

[Month??ShortName Property](#)

[OutputFormat Property](#)

[SetMonthLongName Method](#)

[SetMonthShortName Method](#)

DTS Programming

Month??ShortName Property

The **Month??ShortName** property specifies or returns the string to be used for the short (3-character abbreviation) name of the indicated month. ?? is a month number from 1 through 12.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**Month??ShortName** [= *name*]

Part	Description
<i>Object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object.
??	Month number from 1 through 12, without a leading 0.
<i>name</i>	Short (3-character abbreviation) name of the specified month.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT Month??ShortName(BSTR* pRetVal);
```

```
HRESULT Month??ShortName(BSTR pRetVal);
```

Remarks

The default value is the English month abbreviation.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("Month??ShortName") [= name]
```

See Also

[GetMonthLongName Method](#)

[GetMonthShortName Method](#)

[InputFormat Property](#)

[Month??LongName Property](#)

[OutputFormat Property](#)

[SetMonthLongName Method](#)

[SetMonthShortName Method](#)

DTS Programming

Name Property

The **Name** property specifies the name of a Data Transformation Services (DTS) object.

Applies To

ActiveScriptTask Object	OLEDBProviderInfo Object
BulkInsertTask Object	Package Object
Column Object	Package2 Object
Connection Object	PackageInfo Object
Connection2 Object	PackageLineage Object
CreateProcessTask Object	PackageLogRecord Object
CreateProcessTask2 Object	PackageRepository Object
CustomTask Object	PackageSQLServer Object
DataDrivenQueryTask Object	ParallelDataPumpTask Object
DataDrivenQueryTask2 Object	Property Object
DataPumpTask Object	ScriptingLanguageInfo Object
DataPumpTask2 Object	SendMailTask Object
DTSFTPTask Object	Step Object
DTSMessagesQueueTask Object	Step2 Object
DynamicPropertiesTask Object	StepLineage Object
ExecutePackageTask Object	StepLogRecord Object
ExecuteSQLTask Object	Task Object
ExecuteSQLTask2 Object	TaskInfo Object
GlobalVariable Object	TransferObjectsTask Object
GlobalVariable2 Object	TransferObjectsTask2 Object
Lookup Object	Transformation Object
OLEDBProperty Object	Transformation2 Object
OLEDBProperty2 Object	TransformationSet Object

Syntax

object.**Name** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of a DTS object

Data Type

Object

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetName(BSTR *pRetVal);
```

```
HRESULT SetName(BSTR NewValue);
```

Remarks

Name is a unique, user-defined text label for a **Package2** object. When saved to an instance of Microsoft® SQL Server™ or SQL Server 2000 Meta Data Services, this name is used as a primary key.

A nonempty value for the **Name** property is required for most other DTS objects, as well.

See Also

[Description Property](#)

DTS Programming

NestedExecutionLevel Property

The **NestedExecutionLevel** property specifies the number of times a Data Transformation Services (DTS) package that contains an Execute Package task recursively executes the same or another package that also contains an Execute package task.

Applies To

[Package2 Object](#)

Syntax

object.**NestedExecutionLevel** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>value</i>	Nesting depth of Execute Package invocations

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetNestedExecutionLevel(long *pVal);
```

Remarks

The innermost package in the calling sequence will fail if the **NestedExecutionLevel** property reaches a preset value (32). This property can

be used to detect excessive recursive calls by Execute Package tasks before this limit is reached.

See Also

[ExecutePackageTask Object](#)

DTS Programming

NonOverwritable Property

The **NonOverwritable** property sets or returns a value indicating whether a destination file will be overwritten if it already exists, when copied by a **DTSFTPTask** object.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**NonOverwritable** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object.
<i>boolean</i>	If TRUE, transfer of a source file is inhibited if the destination file already exists. If FALSE, the destination file is overwritten.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT NonOverwritable(VARIANT_BOOL* pVal);
```

```
HRESULT NonOverwritable(VARIANT_BOOL pVal);
```

Remarks

No error is raised when the transfer of a source file is inhibited by the **NonOverwritable** property.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("NonOverwritable") [= boolean]
```

See Also

[DestSite Property](#)

DTS Programming

Nullable Property

The **Nullable** property specifies whether a column can contain null values.

Applies To

[Column Object](#)

Syntax

object.**Nullable** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	A value indicating whether a column can contain null values

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetNullable(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetNullable(VARIANT_BOOL NewValue);
```

See Also

[DataType Property](#)

[Flags Property](#)

DTS Programming

NumericScale Property

The **NumericScale** property specifies the numeric scale of the column if it has a **decimal** or **numeric** data type.

Applies To

[Column Object](#)

Syntax

object.**NumericScale** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Numeric scale of the column

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetNumericScale(LONG *pRetVal);

HRESULT SetNumericScale(LONG NewValue);

See Also

[DataType Property](#)

[Flags Property](#)

DTS Programming

NumRetriesOnSource Property

The **NumRetriesOnSource** property sets or returns the number of times a connection to the source will be attempted before a **DTSFTPTask** object considers it failed.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**NumRetriesOnSource** [= *number*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>number</i>	Number of times connection to the source will be attempted before considered failed

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT NumRetriesOnSource(long* pVal);

HRESULT NumRetriesOnSource(long pVal);

Remarks

No error is raised when the transfer of a source file is inhibited by the **NonOverwritable** property.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("NumRetriesOnSource") [= number]
```

See Also

[SourceLocation Property](#)

[SourceSite Property](#)

DTS Programming

OEMFile Property

The **OEMFile** property specifies or returns a value indicating whether the data read from or written to files by custom transformations is translated from or to the client OEM code page.

Applies To

DataPumpTransformReadFile Object	DataPumpTransformWriteFile Object
--	---

Syntax

transerver.**OEMFile** [= *boolean*]

Part	Description
<i>transerver</i>	Expression that evaluates to an object in the Applies To list.
<i>boolean</i>	If TRUE, the file data is translated through the client OEM code page. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT OEMFile(VARIANT_BOOL* pRetVal);
```

```
HRESULT OEMFile(VARIANT_BOOL pRetVal);
```

Remarks

If the **UnicodeFile** property is TRUE, the **OEMFile** property is ignored.

For **Read File**, the file is translated through the client OEM code page to Unicode if the **OEMFile** property is TRUE. If the destination column is not Unicode, the data is translated again from Unicode to ANSI.

For **Write File**, the source column data has already been translated to Unicode, if necessary. It is then translated through the client OEM code page if the **OEMFile** property is TRUE.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("OEMFile") [= boolean]
```

See Also

[UnicodeFile Property](#)

DTS Programming

Operator Property

The **Operator** property specifies the logged-in user that ran the Data Transformation Services (DTS) package for which a lineage or log record was written.

Applies To

PackageLineage Object	PackageLogRecord Object
---------------------------------------	---

Syntax

object.**Computer**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetOperator(BSTR *pRetVal);
```

See Also

[Computer Property](#)

DTS Programming

Ordinal Property

The **Ordinal** property specifies the ordinal position of a column in a table or rowset.

Applies To

[Column Object](#)

Syntax

object.**Ordinal** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Ordinal position of a column

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetOrdinal(LONG *pRetVal);
```

```
HRESULT SetOrdinal(LONG NewValue);
```

Remarks

The value of the **Ordinal** property for the first column is 1.

See Also

[ColumnID Property](#)

DTS Programming

OutputAsRecordset Property

The **OutputAsRecordset** property returns or specifies whether the entire rowset generated by the Execute SQL task query should be stored in a global variable.

Applies To

ExecuteSQLTask2 Object
--

Syntax

object.**OutputAsRecordset** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecuteSQLTask2 object
<i>value</i>	Whether the entire rowset should be stored in a global variable

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetOutputAsRecordset(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetOutputAsRecordset(VARIANT_BOOL NewValue);
```

Remarks

The rowset is placed in the first named global variable in the **OutputGlobalVariableNames** list as a disconnected Microsoft® ActiveX®

Data Objects (ADO) recordset. The variable is set to **Nothing** if no rowset is returned from the query.

See Also

[OutputGlobalVariableNames Property](#)

[SQLStatement Property](#)

DTS Programming

OutputFormat Property

The **OutputFormat** property specifies or returns a string that defines the format of the datetime string in the destination column. This format string consists of tokens and delimiters, which define how components of the date and time are to be formatted. The delimiters are explicitly written to the destination column.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**OutputFormat** [= *formatstring*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object.
<i>formatstring</i>	String consisting of tokens and delimiters, which define the format of the source column.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT OutputFormat(BSTR* pRetVal);
```

```
HRESULT OutputFormat(BSTR pRetVal);
```

Remarks

The tokens that can be used in the **InputFormat** and **OutputFormat** properties are defined as follows.

Token	Description
yyyy	4 digit year.
yy	2 digit year.
MMMM	Month long name.
MMM	Month 3-char abbreviation.
MM	2-digit month number 01..12.
M	1- or 2-digit month number, 1..12.
dddd	Day of week long name.
ddd	Day of week 3-char abbreviation.
dd	2-digit day number 01..31.
d	1- or 2-digit day number 1..31.
hh	2-digit hours 01..12.
h	1- or 2-digit hours 1..12.
HH	2-digit hours 00..23.
H	1- or 2-digit hours 0..23.
mm	2-digit minutes 00..59.
m	1- or 2-digit minutes 0..59.
ss	2-digit seconds 00..59.
s	1- or 2-digit seconds 0..59.
f[f[f...]]	Fraction of second, in the number of digits as "f"s specified.
tt	Symbol for A.M. or P.M.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("OutputFormat") [= formatstring]
```

See Also

[AMSymbol Property](#)

[Day?LongName Property](#)

[Day?ShortName Property](#)

[GetDayLongName Method](#)

[GetDayShortName Method](#)

[GetMonthLongName Method](#)

[GetMonthShortName Method](#)

[InputFormat Property](#)

[Month??LongName Property](#)

[Month??ShortName Property](#)

[PMSymbol Property](#)

[SetDayLongName Method](#)

[SetDayShortName Method](#)

[SetMonthLongName Method](#)

[SetMonthShortName Method](#)

[ShortYear2000Cutoff Property](#)

DTS Programming

OutputGlobalVariableNames Property

The **OutputGlobalVariableNames** property returns or specifies a list of Data Transformation Services (DTS) global variable names that are to receive values from fields of a rowset or the entire rowset. The values and rowsets are generated by the Execute SQL task query.

Applies To

ExecuteSQLTask2 Object
--

Syntax

object.**OutputGlobalVariableNames** [= *list*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecuteSQLTask2 object
<i>list</i>	List of global variable names

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetOutputGlobalVariableNames(BSTR *pRetVal);

HRESULT SetOutputGlobalVariableNames(BSTR NewValue);

Remarks

The format of the **OutputGlobalVariableNames** string is a semicolon delimited, optionally double-quoted or single-quoted list. Quoting is required only when the name contains an embedded delimiter. Embedded delimiters must be doubled. A double-delimiter or a trailing delimiter indicates an empty item. For example:

"gv1";gv2 - delimiters not required here
gv1;"gv""2" - gv"2 embedded delimiter is doubled
gv1;;gv2; - contains an empty second and fourth item

Values from the first row of the rowset are placed into the global variables in the list by ordinal position. Empty items in the list cause rowset columns to be skipped. If the global variable does not already exist, one is created unless the DTS package **ExplicitGlobalVariables** property is set. In this case, an error occurs.

If the **OutputAsRecordset** property is set, the entire rowset is placed in the first named global variable as a disconnected Microsoft® ActiveX® Data Objects (ADO) recordset. The variable is set to Nothing if no rowset is returned from the query.

See Also

[GlobalVariables Collection](#)

[InputGlobalVariableNames Property](#)

[OutputAsRecordset Property](#)

[SQLStatement Property](#)

DTS Programming

Owner Property

The **Owner** property specifies the login of the owner of the Data Transformation Services (DTS) package.

Applies To

[PackageInfo Object](#)

Syntax

object.**Owner**

Part	Description
<i>object</i>	Expression that evaluates to a PackageInfo object

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetOwner(BSTR *pRetVal);
```

Remarks

The following code illustrates the value returned by the **Owner** property:

```
REDMOND\johndoe
```

See Also

IsOwner Property

DTS Programming

PackageCreationDate Property

The **PackageCreationDate** property specifies the date and time that the Data Transformation Services (DTS) package was first created.

Applies To

[SavedPackageInfo Object](#)

Syntax

[*date* =] *object*.**PackageCreationDate**

Part	Description
<i>object</i>	Expression that evaluates to a SavedPackageInfo object
<i>date</i>	Date the package version was created

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetPackageCreationDate(DATE *pRetVal);
```

See Also

[CreationDate Property](#)

DTS Programming

PackageDataSize Property

The **PackageDataSize** property specifies the size of the Data Transformation Services (DTS) package in Microsoft® SQL Server™ storage.

Applies To

[PackageInfo Object](#)

Syntax

object.**PackageDataSize**

Part	Description
<i>object</i>	Expression that evaluates to a PackageInfo object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetPackageDataSize(long *pRetVal);
```

Remarks

The **PackageDataSize** is specified in bytes. It is not available for packages stored in SQL Server 2000 Meta Data Services (0 is returned).

See Also

[EnumPackageInfos Method](#)

[PackageInfos Collection](#)

DTS Programming

PackageID Property

The **PackageID** property specifies the globally unique Data Transformation Services (DTS) package identifier, which is a string representation of a globally unique identifier (GUID).

Applies To

ExecutePackageTask Object	PackageLineage Object
Package Object	PackageLogRecord Object
Package2 Object	SavedPackageInfo Object
PackageInfo Object	

Syntax

[*guidstring* =] *object*.**PackageID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>guidstring</i>	Package ID of the referenced package

Data Type

String

Modifiable

Read-write for the **ExecutePackageTask** object. Read-only for the others.

Prototype (C/C++)

```
HRESULT GetPackageID(BSTR *pRetVal);
```

Remarks

For the **ExecutePackageTask** object, the **PackageID** does not need to be specified if sufficient other information is specified to identify the package. If the **VersionID** is not specified, the most recent version of the package is run.

To determine the package ID of a package, open the package in DTS Designer. In the **DTS Package Properties** dialog box, click the **General** tab.

The syntax of GUIDs is:

{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits.

See Also

[VersionID Property](#)

DTS Programming

PackageName Property

The **PackageName** property sets or returns the name of the Data Transformation Services (DTS) package.

Applies To

ExecutePackageTask Object	SavedPackageInfo Object
---	---

Syntax

[*name* =] *object*.**PackageName**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>name</i>	Package name

Data Type

String

Modifiable

Read/write for the **ExecutePackageTask** object. Read-only for the others.

Prototype (C/C++)

```
HRESULT GetPackageName(BSTR *pRetVal);
```

Remarks

For the **ExecutePackageTask** object, the **PackageName** property does not need to be set if either the **PackageID** or **VersionID** properties are specified or there is only one package in its containing file.

For the **ExecutePackageTask** object, this property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("PackageName") [= name]
```

See Also

[PackageID Property](#)

[VersionID Property](#)

DTS Programming

PackagePassword Property

The **PackagePassword** property sets or returns the password of the Data Transformation Services (DTS) package to be run by an **ExecutePackageTask** object.

Applies To

[ExecutePackageTask Object](#)

Syntax

object.**PackagePassword** [= *password*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object
<i>password</i>	Owner or user password of the DTS package to be run

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT PackagePassword(BSTR* pRetVal);
```

```
HRESULT PackagePassword(BSTR pRetVal);
```

Remarks

Either the owner or user password can be used to run the package.

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("PackagePassword") [= password]
```

See Also

[PackageID Property](#)

[PackageName Property](#)

DTS Programming

PackagePriorityClass Property

The **PackagePriorityClass** property specifies the Microsoft® Win32® thread priority class of the Data Transformation Services (DTS) package process.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**PackagePriorityClass** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Win32 thread priority class of the package process

Data Type

[DTSPackagePriorityClass](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetPackagePriorityClass(DTSPackagePriorityClass *pRetVal);
```

```
HRESULT SetPackagePriorityClass(DTSPackagePriorityClass NewValue);
```

Remarks

PackagePriorityClass must be set to one of the DTSPackagePriorityClass values.

See Also

[RelativePriority Property](#)

DTS Programming

PackageType Property

The **PackageType** property sets or returns a code that identifies the tool that created the Data Transformation Services (DTS) package.

Applies To

Package2 Object	PackageInfo Object
---------------------------------	------------------------------------

Syntax

object.**PackageType** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Code that identifies the tool that created the package

Data Type

[DTSPackageType](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT PackageType(DTSPackageType* pRetVal);
```

```
HRESULT PackageType(DTSPackageType pRetVal);
```

Remarks

The valid values for this property are defined by the **DTSPackageType** constants.

See Also

[CreatorComputerName Property](#)

[PackagePriorityClass Property](#)

DTS Programming

Parent Property

The **Parent** property specifies a parent object or collection.

Applies To

Nearly all objects and collections in the Data Transformation Services (DTS) hierarchy have a **Parent** property.

Syntax

object.**Parent**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[IDTSStdObject](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetParent(IDTSStdObject **pRetVal);
```

Remarks

The parent of an object is the collection of which it is a member in the DTS hierarchy. The parent of a collection is the object above it in the hierarchy. The parent of the **Package** object is itself.

Note Microsoft® Visual C++® and C applications obtain references on the parent object. The applications must release their references using the **IUnknown::Release** method.

DTS Programming

ParseName Property

The **ParseName** property returns the moniker parse name for the OLE DB data source provider class.

Applies To

[OLEDBProviderInfo Object](#)

Syntax

object.ParseName

Part	Description
<i>object</i>	Expression that evaluates to a OLEDBProviderInfo object

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetParseName(BSTR* pRetVal);
```

Remarks

Typically, the **ParseName** property returns the **ClassID** of the OLE DB provider.

See Also

[ClassID Property](#)

DTS Programming

Password Property

The **Password** property specifies the password to use when making the connection.

Applies To

Connection Object	SendMailTask Object
Connection2 Object	

Syntax

object.**Password** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Password to use

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetPassword(BSTR *pRetVal);
```

```
HRESULT SetPassword(BSTR NewValue);
```

Remarks

Use the **Password** property in conjunction with the **UserID** property to make the

connection. You must provide values for **Password** and **UserID** unless **UseTrustedConnection** is TRUE.

Note The recommended way to connect to an instance of Microsoft® SQL Server™ is to use Windows Authentication instead of SQL Server Authentication. Set **UseTrustedConnection** to TRUE to use Windows Authentication.

See Also

[UserID Property](#)

[UseTrustedConnection Property](#)

DTS Programming

PMSymbol Property

The **PMSymbol** property specifies or returns the string to be used to indicate a time format after noon when a 12-hour time format is specified.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**PMSymbol** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>string</i>	Suffix string to indicate a time format after noon for a 12-hour time format

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT PMSymbol(BSTR* pRetVal);
```

```
HRESULT PMSymbol(BSTR pRetVal);
```

Remarks

The default value is the English "PM".

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("PMSymbol") [= string]
```

See Also

[AMSymbol Property](#)

[InputFormat Property](#)

[OutputFormat Property](#)

DTS Programming

PostSourceDataFunctionEntry Property

The **PostSourceDataFunctionEntry** property specifies or returns the name of the script function that is to be called for the PostSourceData transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**PostSourceDataFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports PostSourceData phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT PostSourceDataFunctionEntry(BSTR* pRetVal);
```

```
HRESULT PostSourceDataFunctionEntry(BSTR pRetVal);
```

Remarks

The PostSourceData phase occurs after the last source row is processed in the **DataPumpTask2** or **DataDrivenQueryTask2** object or after the last row of a constituent rowset is processed in the source hierarchical rowset of the **ParallelDataPumpTask** object.

The PostSourceData script function has no access to the columns of the **DTSSource** collection and write access to the columns of the **DTSDestination** collection. Any return values that are valid during the Transform phase can be returned.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("PostSourceDataFunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertFailureFunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PreSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

PrecedenceBasis Property

The **PrecedenceBasis** property specifies whether to use the current execution status of a **Step** object or the execution results in determining whether its precedence constraint has been satisfied.

Applies To

[PrecedenceConstraint Object](#)

Syntax

object.**PrecedenceBasis** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a PrecedenceConstraint object
<i>value</i>	Value indicating whether to use current execution status of a Step object or execution results

Data Type

[DTSStepPrecedenceBasis](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetPrecedenceBasis(DTSStepPrecedenceBasis *pRetVal);
```

```
HRESULT SetPrecedenceBasis(DTSStepPrecedenceBasis NewValue);
```

Remarks

PrecedenceBasis must be set to one of the DTSSStepPrecedenceBasis values. The default is DTSSStepPrecedenceBasis_ExecResult.

See Also

[DTSSStepExecResult](#)

[DTSSStepExecStatus](#)

DTS Programming

Precision Property

The **Precision** property specifies column precision, if it has a **decimal** or **numeric** data type.

Applies To

[Column Object](#)

Syntax

object.**Precision** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Column precision

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetPrecision(LONG *pRetVal);

HRESULT SetPrecision(LONG NewValue);

See Also

[DataType Property](#)

[NumericScale Property](#)

DTS Programming

PreSourceDataFunctionEntry Property

The **PreSourceDataFunctionEntry** property specifies or returns the name of the script function that is to be called for the PreSourceData transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**PreSourceDataFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports the PreSourceData phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT PreSourceDataFunctionEntry(BSTR* pRetVal);
```

```
HRESULT PreSourceDataFunctionEntry(BSTR pRetVal);
```

Remarks

The PreSourceData phase occurs before the first source row is processed in the **DataPumpTask2** or **DataDrivenQueryTask2** objects, or before the first row of a constituent rowset is processed in the source hierarchical rowset of the **ParallelDataPumpTask** object.

The PreSourceData script function has no access to the columns of the **DTSSource** collection and write access to the columns of the **DTSDestination** collection. Any return values that are valid during the Transform phase can be returned.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("PreSourceDataFunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertFailureFunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

ProcessCommandLine Property

The **ProcessCommandLine** property specifies the universal naming convention (UNC) file name of the file to execute and any command prompt arguments.

Applies To

CreateProcessTask Object	CreateProcessTask2 Object
--	---

Syntax

object.**ProcessCommandLine** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Process command prompt arguments

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetProcessCommandLine(BSTR *pRetVal);
```

```
HRESULT SetProcessCommandLine(BSTR NewValue);
```

Remarks

The command line parameters can include environment variables. The **GetExpandedProcessCommandLine** method returns the command line with

the environment variables substituted with their values.

See Also

[GetExpandedProcessCommandLine Method](#)

DTS Programming

Profile Property

The **Profile** property specifies the profile to use when sending an e-mail message.

Applies To

[SendMailTask Object](#)

Syntax

object.**Profile**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	Profile to use

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetProfile(BSTR *pRetVal);
```

```
HRESULT SetProfile(BSTR NewValue);
```

See Also

[GetDefaultProfileName Method](#)

DTS Programming

ProgressCount Property

The **ProgressCount** property specifies the intervals (typically rows) processed during this step.

Applies To

[StepLogRecord Object](#)

Syntax

object.**ProgressCount**

Part	Description
<i>object</i>	Expression that evaluates to a StepLogRecord object

Data Type

Variant

Modifiable

Read-only

Prototype (C/C++)

HRESULT GetProgressCount(VARIANT *pRetVal);

See Also

[FirstRow Property](#)

[LastRow Property](#)

[ProgressRowCount Property](#)

DTS Programming

ProgressRowCount Property

The **ProgressRowCount** property specifies the numbers of rows that are returned between notifications to the connection point event during data pump execution.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask Object	

Syntax

object.**ProgressRowCount** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Frequency in which notifications are sent to the connection point event

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetProgressRowCount(LONG *pRetVal);
```

```
HRESULT SetProgressRowCount(LONG NewValue);
```

Remarks

The **OnProgress** event is raised after every **ProgressRowCount** row has been processed. The default is 1000 rows.

See Also

[OnProgress Event](#)

[ProgressCount Property](#)

DTS Programming

PropertyID Property

The **PropertyID** property specifies an **OLEDBProperty** object identifier (DBPROPID).

Applies To

OLEDBProperty Object	OLEDBProperty2 Object
--------------------------------------	---------------------------------------

Syntax

object.**PropertyID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetPropertyID(LONG *pRetVal);
```

See Also

[PropertySet Property](#)

DTS Programming

PropertySet Property

The **PropertySet** property specifies the globally unique identifier (GUID) of the OLE DB property set.

Applies To

OLEDBProperty Object	OLEDBProperty2 Object
--------------------------------------	---------------------------------------

Syntax

object.**PropertySet**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetPropertySet(BSTR *pRetVal);
```

Remarks

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits.

See Also

[PropertyID Property](#)

DTS Programming

ProviderID Property

The **ProviderID** property returns the program ID of the OLE DB provider.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**ProviderID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetProviderID(BSTR *pRetVal);
```

Remarks

A user-implemented OLE DB provider must support the following features if it is to be used with Data Transformation Services (DTS):

Interfaces

This is the interface support that DTS requires.

Interface	Requirement
IDBInitialize	Necessary.
IDBProperties	Necessary.
IDBCreateSession	Necessary.
ISessionProperties	Necessary.
IDBCreateCommand	Optional (for example, used for create table and queries).
IDBInfo	Necessary.
IOpenRowset	Necessary.
ICommandText	Optional (for example, used for create table and queries).
ICommandPrepare	Optional (for example, used for create table and queries).
IColumnsInfo	Necessary.
IRowset	Necessary.
IAccessor	Necessary.
ICommandWithParameters	Optional. Used for data driven queries.
IDBSchemaRowset	Necessary.

In addition to these interfaces, the DTS package requires that a provider have a **DataSource** property.

Schema

These are the schema that DTS requests:

- DBSCHEMA_TABLES
- DBSCHEMA_CATALOGS
- DBSCHEMA_PROVIDER_TYPES. This is the only one that must be supported.

- DBSCHEMA_PRIMARY_KEYS
- DBSCHEMA_FOREIGN_KEYS
- DBSCHEMA_INDEXES
- DBSCHEMA_TABLE_CONSTRAINTS
- DBSCHEMA_CHECK_CONSTRAINTS

See Also

[New \(ID\) Method](#)

DTS Programming

PumpCompleteFunctionEntry Property

The **PumpCompleteFunctionEntry** property specifies or returns the name of the script function that is to be called for the OnPumpComplete transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**PumpCompleteFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports OnPumpComplete phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT PumpCompleteFunctionEntry(BSTR* pRetVal);
```

```
HRESULT PumpCompleteFunctionEntry(BSTR pRetVal);
```

Remarks

The PostSourceData phase occurs after all other processing in the **DataPumpTask2**, **DataDrivenQueryTask2** or **ParallelDataPumpTask** objects is complete.

The OnPumpComplete script function has no access to the columns of the **DTSSource** and **DTSDestination** collections. The only valid return values are **DTSTransformStat_OK** and **DTSTransformStat_AbortPump**.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("PumpCompleteFunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertFailureFunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PreSourceDataFunctionEntry Property](#)

[TransformFailureFunctionEntry Property](#)

DTS Programming

Query Property

The **Query** property specifies a parameterized query to execute.

Applies To

[Lookup Object](#)

Syntax

object.**Query** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Lookup object
<i>value</i>	Parameterized query to execute

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetQuery(BSTR *pRetVal);
```

```
HRESULT SetQuery(BSTR NewValue);
```

See Also

[Adding DTS Lookups and Global Variables](#)

DTS Programming

QueuePath Property

The **QueuePath** property sets or returns the full path of the Message Queuing queue used to send or receive messages.

Applies To

[DTSMMessageQueueTask Object](#)

Syntax

object.**QueuePath** [= *path*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMMessageQueueTask object
<i>path</i>	Full path of the message queue

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT QueuePath(BSTR* pVal);

HRESULT QueuePath(BSTR pVal);

Remarks

The syntax of the queue path is *servername*\[PRIVATE\$]*queuename*. To refer to the local computer, use a period (.) in the *servername* field. For public queues,

nothing is coded between the back slashes (\\).

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("QueuePath") [= path]
```

Example

This example assigns a globally unique identifier (GUID) string to the **PackageID** property:

```
oCustTask.QueuePath = "DTS_SERV\\DTS_QUE"
```

See Also

[RemoveFromQueue Property](#)

DTS Programming

ReceiveMessageTimeout Property

The **ReceiveMessageTimeout** property sets or returns the time after which the **DTSMessageQueueTask** object will terminate if a message is not found in the specified queue.

Applies To

[DTSMessageQueueTask Object](#)

Syntax

object.**ReceiveMessageTimeout** [= *seconds*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>seconds</i>	Seconds after which task will terminate if message is not received

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT ReceiveMessageTimeout(long* pVal);
```

```
HRESULT ReceiveMessageTimeout(long pVal);
```

Remarks

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("ReceiveMessageTimeout") [= seconds]
```

See Also

[ErrorIfReceiveMessageTimeout Property](#)

DTS Programming

ReceiveMessageType Property

The **ReceiveMessageType** property sets or returns the type of message for which a **DTSMessageQueueTask** object that is a receiver is waiting.

Applies To

[DTSMessageQueueTask Object](#)

Syntax

object.**ReceiveMessageType** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>value</i>	Code for the message type from the DTSMQMessageType constants

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetReceiveMessageType(long* pVal);

HRESULT SetReceiveMessageType(long pVal);

Remarks

The property also can be referenced through the **Properties** collection of the

Task object with the following code:

```
Set taskprops = task.Properties  
taskprops("ReceiveMessageType") [= value]
```

See Also

[DTSMQMessageType](#)

[TaskType Property](#)

DTS Programming

RelativePriority Property

The **RelativePriority** property specifies the Microsoft® Win32® priority of the thread on which a step is running, within the priority class of the Data Transformation Services (DTS) package process.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**RelativePriority** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Win32 priority of a thread of a step

Data Type

[DTSSStepRelativePriority](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetRelativePriority(DTSSStepRelativePriority *pRetVal);
```

```
HRESULT SetRelativePriority(DTSSStepRelativePriority NewValue);
```

Remarks

RelativePriority must be set to one of the DTSSStepRelativePriority values.

See Also

[PackagePriorityClass Property](#)

DTS Programming

RemoveFromQueue Property

The **RemoveFromQueue** property sets or returns a value indicating whether a message is to be removed from the queue after it is received.

Applies To

[DTSMessagesQueueTask Object](#)

Syntax

object.**RemoveFromQueue** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessagesQueueTask object.
<i>boolean</i>	If TRUE, a message is removed from the queue after it is received. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT RemoveFromQueue(VARIANT_BOOL* pVal);
```

```
HRESULT RemoveFromQueue(VARIANT_BOOL pVal);
```

Remarks

The property also can be referenced through the **Properties** collection of the

Task object with the following code:

Set *taskprops* = *task.Properties*

taskprops("RemoveFromQueue") [= *boolean*]

See Also

[QueuePath Property](#)

DTS Programming

RepositoryDatabaseName Property

The **RepositoryDatabaseName** property sets or returns the name of the database that contains the instance of Microsoft® SQL Server™ 2000 Meta Data Services to be used by the **ExecutePackageTask** object.

Applies To

[ExecutePackageTask Object](#)

Syntax

object.**RepositoryDatabaseName** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object
<i>name</i>	Name of the database containing the instance of Meta Data Services

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT RepositoryDatabaseName(BSTR* pRetVal);
```

```
HRESULT RepositoryDatabaseName(BSTR pRetVal);
```

Remarks

If **RepositoryDatabaseName** property is not provided, the default Meta Data Services database is used. **RepositoryDatabaseName** is also not needed if the Data Transformation Services (DTS) package to be run is in SQL Server storage or a data file.

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("RepositoryDatabaseName") [= password]
```

See Also

[ServerName Property](#)

[UseRepository Property](#)

DTS Programming

RepositoryMetadataOptions Property

The **RepositoryMetadataOptions** property specifies meta data scanning and resolution options when storing a Data Transformation Services (DTS) package to Microsoft® SQL Server™ 2000 Meta Data Services.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**RepositoryMetadataOptions**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Meta data scanning and resolution options

Data Type

[DTSRepositoryMetadataOptions](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetRepositoryMetadataOptions(  
    DTSRepositoryMetadataOptions *pRetVal);
```

```
HRESULT SetRepositoryMetadataOptions(  
    DTSRepositoryMetadataOptions NewValue);
```

Remarks

RepositoryMetadataOptions must be set to one of the DTSRepositoryMetadataOptions values.

See Also

[LineageOptions Property](#)

[LoadFromRepository Method](#)

[SaveToRepository Method](#)

[SaveToRepositoryAs Method](#)

DTS Programming

Reusable Property

The **Reusable** property specifies whether a connection is reusable by multiple steps.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**Reusable** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	A value indicating whether a connection is reusable

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetReusable(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetReusable(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE.

See Also

[AcquireConnection Method](#)

[InUse Property](#)

[ReleaseConnection Method](#)

DTS Programming

RollbackFailure Property

The **RollbackFailure** property specifies whether to roll back the Data Transformation Services (DTS) package transaction if there is a step failure.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**RollbackFailure** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether to roll back the transaction

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetRollbackFailure(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetRollbackFailure(VARIANT_BOOL NewValue);
```

See Also

[CommitSuccess Property](#)

[JoinTransactionIfPresent Property](#)

UseTransaction Property

DTS Programming

RowsComplete Property

The **RowsComplete** property returns the number of source rows, including rows for which errors occurred, processed by the task or transformation set.

Applies To

DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask2 Object	

Syntax

object.RowsComplete

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Variant

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT RowsComplete(VARIANT* pRowsComplete);
```

Remarks

The property value is returned from the **IDTSDatapump::Execute** method.

See Also

RowsInError Property

DTS Programming

RowsInError Property

The **RowsInError** property returns the number of rows for which an error occurred while being processed by the Data Transformation Services (DTS) task or transformation set.

Applies To

DataDrivenQueryTask2 Object	TransformationSet Object
DataPumpTask2 Object	

Syntax

object.**RowsInError**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Variant

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT RowsInError(VARIANT* pRowsInError);
```

Remarks

The property value is returned from the **IDTSDatapump::Execute** method.

See Also

RowsComplete Property

DTS Programming

RowTerminator Property

The **RowTerminator** property specifies the row terminator for the Bulk Insert task. The same set of row terminators that apply to the bulk copy program also apply to the Bulk Insert task.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**RowTerminator** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	A row terminator

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetRowTerminator(BSTR *pRetVal);
```

```
HRESULT SetRowTerminator(BSTR NewValue);
```

Remarks

Typical values are CR, CR/LF, LF, ",", ";", tab and "|". The default is a line feed

character. It should not be the same as the **FieldTerminator** property.

See Also

[FieldTerminator Property](#)

DTS Programming

SaveDataFileName Property

The **SaveDataFileName** property sets or returns the name and path of the file into which a received data file is written by a **DTSMessageQueueTask** object.

Applies To

[DTSMessageQueueTask](#)

Syntax

object.**SaveDataFileName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>value</i>	File specification where message is to be written

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetSaveDataFileName(BSTR *pVal);

HRESULT SetSaveDataFileName(BSTR pVal);

Remarks

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

Set *taskprops* = *task.Properties*
taskprops("SaveDataFileName") [= *value*]

See Also

[ReceiveMessageType Property](#)

[TaskType Property](#)

DTS Programming

SaveMailInSentItemsFolder Property

The **SaveMailInSentItemsFolder** property specifies whether to save outgoing e-mail messages in the Sent Items folder.

Applies To

[SendMailTask Object](#)

Syntax

object.**SaveMailInSentItemsFolder**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	Boolean that specifies whether to save outgoing e-mail messages in the Sent Items folder

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSaveMailInSentItemsFolder(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetSaveMailInSentItemsFolder(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE. Outgoing e-mail messages are saved in the Sent Items

folder.

DTS Programming

ScriptFileDirectory Property

The **ScriptFileDirectory** property specifies the directory to which the script file and log files are written.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**ScriptFileDirectory**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Directory to which the script file and log files are written

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetScriptFileDirectory(BSTR *pRetVal);
```

```
HRESULT SetScriptFileDirectory(BSTR NewValue);
```

Remarks

The script file directory must exist on the computer on which the task runs.

See Also

[ScriptOption Property](#)

[ScriptOptionEx Property](#)

DTS Programming

ScriptLanguage Property

The **ScriptLanguage** property specifies the Microsoft® ActiveX® script language needed to execute a script (for example, Microsoft Visual Basic® Scripting Edition (VBScript), Microsoft JScript®, or PerlScript®).

Applies To

ActiveScriptTask Object	Step2 Object
Step Object	

Syntax

object.**ScriptLanguage** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	ActiveX script language needed to execute a script

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetScriptLanguage(BSTR *pRetVal);
```

```
HRESULT SetScriptLanguage(BSTR NewValue);
```

Remarks

The default is VBScript.

Script languages available on a particular system can be determined by enumerating the **ScriptingLanguageInfos** collection of the **Application** object. For more information about which scripting language to use with Data Transformation Services (DTS), see [ScriptingLanguageInfo Object](#).

See Also

[Application Object](#)

[ActiveXScript Property](#)

[FunctionName Property](#)

[Language Property](#)

[ScriptingLanguageInfos Collection](#)

DTS Programming

ScriptOption Property

The **ScriptOption** property specifies which scripting option to use during an object transfer operation.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**ScriptOption**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Scripting option to use

Data Type

[DTSTransfer_ScriptOption](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetScriptOption(DTSTransfer_ScriptOption *pRetVal);
```

```
HRESULT SetScriptOption(DTSTransfer_ScriptOption NewValue);
```

Remarks

ScriptOption must be set to one of the DTSTransfer_ScriptOption values. The default is DTSTransfer_Script_TransferDefault.

See Also

[ScriptFileDirectory Property](#)

[ScriptOptionEx Property](#)

DTS Programming

ScriptOptionEx Property

The **ScriptOptionEx** property specifies the extended scripting option to use during an object transfer operation.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**ScriptOptionEx**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Extended scripting option to use

Data Type

[DTSTransfer_ScriptOptionEx](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetScriptOptionEx(DTSTransfer_ScriptOptionEx *pRetVal);
```

```
HRESULT SetScriptOptionEx(DTSTransfer_ScriptOptionEx NewValue);
```

Remarks

ScriptOptionEx must be set to one of the DTSTransfer_ScriptOptionEx values. The default is DTSTransfer_ScriptEx_TransferDefault.

See Also

[ScriptFileDirectory Property](#)

[ScriptOption Property](#)

DTS Programming

SequenceID Property

The **SequenceID** property specifies a sequence number for the task log record.

Applies To

[TaskLogRecord Object](#)

Syntax

object.**SequenceID**

Part	Description
<i>object</i>	Expression that evaluates to a TaskLogRecord object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetSequenceID(long *pRetVal);
```

Remarks

Task log records are written by custom tasks if they are implemented to do so. The Data Transformation Services (DTS) task classes supplied with Microsoft® SQL Server™ 2000 do not write task log records.

See Also

ErrorCode Property

ErrorDescription Property

DTS Programming

ServerName Property

The **ServerName** property sets or returns the name of the server on which the Data Transformation Services (DTS) package to be run by an **ExecutePackageTask** object is located.

Applies To

[ExecutePackageTask Object](#)

Syntax

object.**ServerName** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object
<i>name</i>	Name of the server on which the DTS package to be run is located

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT ServerName(BSTR* pRetVal);
```

```
HRESULT ServerName(BSTR pRetVal);
```

Remarks

If **ServerName** property is not provided, the local computer is used; you can

also specify "(local)".

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("PackagePassword") [= password]
```

See Also

[ServerPassword Property](#)

[ServerUserName Property](#)

[UseTrustedConnection Property](#)

DTS Programming

ServerPassword Property

The **ServerPassword** property sets or returns the login password for the instance of Microsoft® SQL Server™ that contains the Data Transformation Services (DTS) package to be run by an **ExecutePackageTask** object.

Applies To

[ExecutePackageTask Object](#)

Syntax

object.**ServerPassword** [= *password*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object
<i>password</i>	Login password for the instance of SQL Server that contains the DTS package to be run

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT ServerPassword(BSTR* pRetVal);
```

```
HRESULT ServerPassword(BSTR pRetVal);
```

Remarks

The **ServerPassword** and **ServerUserName** properties must be provided unless **UseTrustedConnection** is TRUE or the DTS package is contained in a storage file.

Note It is recommended that you connect to an instance of SQL Server using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **UseTrustedConnection** to TRUE.

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("ServerPassword") [= password]
```

See Also

[ServerName Property](#)

[ServerUserName Property](#)

[UseTrustedConnection Property](#)

DTS Programming

ServerUserName Property

The **ServerUserName** property sets or returns the login user name for the instance of Microsoft® SQL Server™ containing the Data Transformation Services (DTS) package to be run by an **ExecutePackageTask** object.

Applies To

[ExecutePackageTask Object](#)

Syntax

object.**ServerUserName** [= *username*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object
<i>username</i>	Login ID for the instance of SQL Server containing the DTS package to be run

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT ServerUserName(BSTR* pRetVal);
```

```
HRESULT ServerUserName(BSTR pRetVal);
```

Remarks

The **ServerUserName** and **ServerPassword** properties must be provided unless **UseTrustedConnection** is TRUE or the DTS package is contained in a storage file.

Note It is recommended that you connect to an instance of SQL Server using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **UseTrustedConnection** to TRUE.

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("ServerUserName") [= username]
```

See Also

[ServerName Property](#)

[ServerPassword Property](#)

[UseTrustedConnection Property](#)

DTS Programming

Set Property

The **Set** property returns TRUE when the referenced object property is read/write rather than read-only.

Applies To

[Property Object](#)

Syntax

object.Set

Part	Description
<i>object</i>	Expression that evaluates to a Property object

Data Type

Boolean

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetSet(VARIANT_BOOL *pRetVal);
```

Remarks

When TRUE, the property referenced is read/write or write-only. However, an application attempt to change the property value is not guaranteed to succeed.

When FALSE, the property referenced is read-only.

See Also

[Get Property](#)

DTS Programming

ShortYear2000Cutoff Property

The **ShortYear2000Cutoff** property specifies or returns the two-digit year below which the year is assumed to be 20yy. If the two-digit year is equal to or above the **ShortYear2000Cutoff** property, the year is 19yy.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**ShortYear2000Cutoff** [= *yy*]

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>yy</i>	Two-digit number specifying the lowest year that is to be considered 19yy

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT ShortYear2000Cutoff(long* pRetVal);
```

```
HRESULT ShortYear2000Cutoff(long pRetVal);
```

Remarks

The default value is 30.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("ShortYear2000Cutoff") [= yy]
```

See Also

[InputFormat Property](#)

[OutputFormat Property](#)

DTS Programming

Size Property

The **Size** property specifies the maximum size of the column.

Applies To

[Column Object](#)

Syntax

object.**Size** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Column object
<i>value</i>	Maximum size of the column

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSize(LONG *pRetVal);
```

```
HRESULT SetSize(LONG NewValue);
```

Remarks

The **Size** property is ignored for columns that have fixed-length data types.

See Also

[DataType Property](#)

[NumericScale Property](#)

[Precision Property](#)

DTS Programming

SortedData Property

The **SortedData** property specifies a string that corresponds to the ORDER clause in the Transact-SQL BULK INSERT statement.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**SortedData** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	String that corresponds to the ORDER clause in the Transact-SQL BULK INSERT statement

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSortedData(BSTR *pRetVal);
```

```
HRESULT SetSortedData(BSTR NewValue);
```

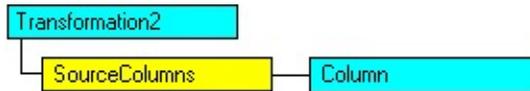
See Also

[BULK INSERT](#)

DTS Programming

SourceColumns Property

The **SourceColumns** property returns a reference to a **Columns** collection that contains the source columns the transformation uses.



Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

[Set *columns* =] *object*.**SourceColumns**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>columns</i>	Reference to collection containing source columns

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetSourceColumns(IDTSColumns **pRetVal);
```

Remarks

For this collection, either the name or the ordinal can be used to reference individual elements.

See Also

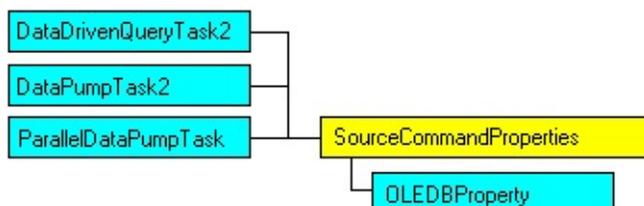
[Column Object](#)

[DestinationColumns Property](#)

DTS Programming

SourceCommandProperties Property

The **SourceCommandProperties** property specifies an **OLEDBProperties** collection of properties of the OLE DB provider used by the source connection.



Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**SourceCommandProperties**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[OLEDBProperties Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetSourceCommandProperties(IDTSOLEDBProperties **pRetVal);
```

See Also

[Connection2 Object](#)

[DestinationCommandProperties Property](#)

[OLEDBProperty Object](#)

DTS Programming

SourceConnectionID Property

The **SourceConnectionID** property specifies the ID of the source connection.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**SourceConnectionID** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Source connection ID

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceConnectionID(LONG *pRetVal);
```

```
HRESULT SetSourceConnectionID(LONG NewValue);
```

Remarks

This property is the linkage between a data pump task and the source

Connection2 object.

See Also

[Connection2 Object](#)

[DestinationConnectionID Property](#)

DTS Programming

SourceConstantValue Property

The **SourceConstantValue** property sets or returns the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_Constant.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceConstantValue** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>string</i>	Value to which a DTS package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT SourceConstantValue(BSTR* pRetVal);

HRESULT SourceConstantValue(BSTR pRetVal);

Example

The following example sets the **SourceConstantValue** property to a string:

```
oAssign.SourceConstantValue = "C:\DTS_UE\TestData\PubsAuthors.t
```

See Also

[DynamicPropertiesTask Object](#)

[SourceType Property](#)

DTS Programming

SourceDatabase Property

The **SourceDatabase** property specifies the name of the source database.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**SourceDatabase**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of the source database

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceDatabase(BSTR *pRetVal);
```

```
HRESULT SetSourceDatabase(BSTR NewValue);
```

See Also

[DestinationDatabase Property](#)

[SourceLogin Property](#)

[SourceServer Property](#)

[SourcePassword Property](#)

[SourceUseTrustedConnection Property](#)

DTS Programming

SourceDataFileFileName Property

The **SourceDataFileFileName** property sets or returns a string that is the name and path of a file that contains the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_DataFile.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceDataFileFileName** [= *filespec*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>filespec</i>	Name and path of file containing value to which a DTS package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceDataFileFileName(BSTR* pRetVal);
```

```
HRESULT SourceDataFileFileName(BSTR pRetVal);
```

Example

The following example sets the **SourceDataFileName** property to a file specification:

```
oAssign.SourceDataFileFileName = "C:\DTS_UE\TestData\PubsAuth
```

See Also

[DynamicPropertiesTask Object](#)

[SourceType Property](#)

DTS Programming

SourceEnvironmentVariable Property

The **SourceEnvironmentVariable** property sets or returns the name of an environment variable that contains the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_EnvironmentVariable.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceEnvironmentVariable** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>name</i>	Name of environment variable containing value to which a DTS package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceEnvironmentVariable(BSTR* pRetVal);
```

```
HRESULT SourceEnvironmentVariable(BSTR pRetVal);
```

Remarks

Only system environment variables can be referenced with this property.

Example

The following example sets the **SourceEnvironmentVariable** property to a name:

```
oAssign.SourceEnvironmentVariable = "DTS_UE_Env"
```

See Also

[DynamicPropertiesTask Object](#)

[SourceType Property](#)

DTS Programming

SourceFilename Property

The **SourceFilename** property sets or returns a list of files, with path and size, to be transferred from the source by a **DTSFTPTask** object.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**SourceFilename** [= '*name*';'*path*';'*size*';'*name*';'*path*';'*size*'; ...]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>name</i>	Name of file to be transferred from the source
<i>path</i>	Path of file specified by <i>name</i>
<i>size</i>	Size of file specified by <i>name</i> and <i>path</i>

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT SourceFilename(BSTR* pVal);

HRESULT SourceFilename(BSTR pVal);

Remarks

If a path or site is specified by the **SourceSite** property, the *path* from the **SourceFilename** list is appended to the **SourceSite** property, and this value is used as the full path.

The *size* field is used by Data Transformation Services (DTS) Designer. It is not necessary to provide a *size* value when referencing the **SourceFilename** property programmatically. However, the enclosing apostrophes should still be coded.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("SourceFilename") [= list]
```

Example

The following example sets the **SourceFilename** property to a list of two file names:

```
oCustTask.SourceFilename = _  
    "File3.dat';";'123';'NWProdWiz.xls';";'458240';"
```

See Also

[SourceLocation Property](#)

[SourcePassword \(DTSFTPTask\) Property](#)

[SourceSite Property](#)

[SourceUsername Property](#)

DTS Programming

SourceGlobalVariable Property

The **SourceGlobalVariable** property sets or returns the name of a Data Transformation Services (DTS) package global variable that contains the value to which a package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_GlobalVariable.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceGlobalVariable** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>name</i>	Name of a package global variable containing value to which a package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceGlobalVariable(BSTR* pRetVal);
```

```
HRESULT SourceGlobalVariable(BSTR pRetVal);
```

See Also

[DynamicPropertiesTask Object](#)

[SourceType Property](#)

DTS Programming

SourceIniFileFileName Property

The **SourceIniFileFileName** property sets or returns a string that is the name and path of an .ini file, which contains the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_IniFile.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceIniFileFileName** [= *filespec*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>filespec</i>	Name and path of .ini file containing value to which a package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceIniFileFileName(BSTR* pRetVal);
```

```
HRESULT SourceIniFileFileName(BSTR pRetVal);
```

Example

The following code sets the **SourceIniFileName** property:

```
oAssign.SourceIniFileName = "C:\DTS_UE\TestData\DynProp.ini"
oAssign.SourceIniFileSection = "FlatFile"
oAssign.SourceIniFileKey = "Lengths"
```

See Also

[DynamicPropertiesTask Object](#)

[SourceIniFileKey Property](#)

[SourceIniFileSection Property](#)

[SourceType Property](#)

DTS Programming

SourceIniFileKey Property

The **SourceIniFileKey** property sets or returns the name of a key within an .ini file that identifies the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_IniFile.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceIniFileKey** [= *key*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>key</i>	Name of key within an .ini file identifying value to which a package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceIniFileKey(BSTR* pRetVal);
```

```
HRESULT SourceIniFileKey(BSTR pRetVal);
```

Remarks

Do not include the equal sign (=) that follows the key name in the .ini file with the key name when setting this property.

The specified key must appear in the section specified by the **SourceIniFileSection** property, which in turn must appear within the .ini file specified by the **SourceIniFileFileName** property.

Example

The following code sets the **SourceIniFileKey** property:

```
oAssign.SourceIniFileFileName = "C:\DTS_UE\TestData\DynProp.ini  
oAssign.SourceIniFileSection = "FlatFile"  
oAssign.SourceIniFileKey = "Lengths"
```

See Also

[DynamicPropertiesTask Object](#)

[SourceIniFileFileName Property](#)

[SourceIniFileSection Property](#)

[SourceType Property](#)

DTS Programming

SourceIniFileSection Property

The **SourceIniFileSection** property sets or returns the name of a section within an .ini file that contains the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_IniFile.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceIniFileSection** [= *section*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>section</i>	Name of section within an .ini file containing value to which a package object property will be set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceIniFileSection(BSTR* pRetVal);
```

```
HRESULT SourceIniFileSection(BSTR pRetVal);
```

Remarks

Do not include the square brackets "[]" that enclose the section name in the .ini file with the section name when setting this property.

The specified section must appear within the .ini file specified by the **SourceIniFileFileName** property.

Example

The following code sets the **SourceIniFileSection** property:

```
oAssign.SourceIniFileFileName = "C:\DTS_UE\TestData\DynProp.ini"
oAssign.SourceIniFileSection = "FlatFile"
oAssign.SourceIniFileKey = "Lengths"
```

See Also

[DynamicPropertiesTask Object](#)

[SourceIniFileFileName Property](#)

[SourceIniFileKey Property](#)

[SourceType Property](#)

DTS Programming

SourceLocation Property

The **SourceLocation** property sets or returns the source location type, an Internet site, or a network directory to be used by a **DTSFTPTask** object.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**SourceLocation** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>value</i>	Code that defines the source location type

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

HRESULT SourceLocation(long* pVal);

HRESULT SourceLocation(long pVal);

Remarks

The valid values for this property are defined by the **DTSFTPSourceLocation** enumeration in the **DTSCustTasks** library (CustTask.dll).

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("SourceLocation") [= list]
```

See Also

[DTSFTPSourceLocation](#)

[SourceFilename Property](#)

[SourcePassword \(DTSFTPTask\) Property](#)

[SourceSite Property](#)

[SourceUsername Property](#)

DTS Programming

SourceLogin Property

The **SourceLogin** property specifies the login ID on the source server.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**SourceLogin**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Login ID

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceLogin(BSTR *pRetVal);
```

```
HRESULT SetSourceLogin(BSTR NewValue);
```

Remarks

SourceLogin is required if an application is using SQL Server Authentication security mode.

Note It is recommended that you connect to an instance of Microsoft® SQL

Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **SourceUseTrustedConnection** to TRUE.

See Also

[DestinationLogin Property](#)

[SourceDatabase Property](#)

[SourceServer Property](#)

[SourcePassword Property](#)

[SourceUseTrustedConnection Property](#)

DTS Programming

SourceObjectName Property

The **SourceObjectName** property specifies the source object name if no value for the **SourceSQLStatement** property is specified.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.**SourceObjectName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Source object name

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceObjectName(BSTR *pRetVal);
```

```
HRESULT SetSourceObjectName(BSTR NewValue);
```

Remarks

The **SourceObjectName** property is typically a database table name or worksheet name.

Example

The following code sets the **SourceObjectName** property:

```
objDataPump.SourceObjectName = "pubs..authors"
```

See Also

[DestinationObjectName Property](#)

[SourceSQLStatement Property](#)

DTS Programming

SourcePassword Property

The **SourcePassword** property specifies the password on the source server.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**SourcePassword**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Password on the source server

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetSourcePassword(BSTR *pRetVal);

HRESULT SetSourcePassword(BSTR NewValue);

Remarks

Note It is recommended that you connect to an instance of Microsoft® SQL Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **SourceUseTrustedConnection** to TRUE.

See Also

[DestinationPassword Property](#)

[SourceDatabase Property](#)

[SourceServer Property](#)

[SourceLogin Property](#)

[SourceUseTrustedConnection Property](#)

DTS Programming

SourcePassword (DTSFTPTask) Property

The **SourcePassword** property sets or returns the password that will be used to connect to the Internet File Transfer Protocol (FTP) site by a **DTSFTPTask** object.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**SourcePassword** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>value</i>	Password that will be used to connect to the Internet FTP site

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT SourcePassword(BSTR* pVal);

HRESULT SourcePassword(BSTR pVal);

Remarks

Typically, if "anonymous" is used for the user name when connecting to the FTP

site, the e-mail address of the user is used as the password.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("SourcePassword") [= value]
```

Example

This example sets the **SourcePassword** property to a string.

```
oCustTask.SourcePassword = "user@microsoft.com"
```

See Also

[SourceFilename Property](#)

[SourceLocation Property](#)

[SourceSite Property](#)

[SourceUsername Property](#)

DTS Programming

SourceQueryConnectionID Property

The **SourceQueryConnectionID** property sets or returns the connection ID of the connection against which a query will be run. The query provides the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_Query.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceQueryConnectionID** [= *number*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>number</i>	Connection ID against which a query will be run to provide the value to which a DTS package object property will be set

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceQueryConnectionID(long* pRetVal);
```

```
HRESULT SourceQueryConnectionID(long pRetVal);
```

Remarks

The SQL query to be run is specified by the **SourceQuerySQL** property.

See Also

[SourceQuerySQL Property](#)

[SourceType Property](#)

DTS Programming

SourceQuerySQL Property

The **SourceQuerySQL** property sets or returns a string that is an SQL query. The query provides the value to which a Data Transformation Services (DTS) package object property is set by the **DynamicPropertiesTask** object, when the **SourceType** property is DTSDynamicPropertiesSourceType_Query.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceQuerySQL** [= *query*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>query</i>	SQL query that provides the value to which a DTS package object property is set

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceQuerySQL(BSTR* pRetVal);
```

```
HRESULT SourceQuerySQL(BSTR pRetVal);
```

Remarks

In general, it is recommended that you program the SQL query to return a single row containing a single field. Only the first field of the first row is used to set the specified package object property.

The query is run against the connection identified by the **SourceQueryConnectionID** property.

See Also

[SourceQueryConnectionID Property](#)

[SourceType Property](#)

DTS Programming

SourceServer Property

The **SourceServer** property specifies the name of the source server.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**SourceServer**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of the source server

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceServer(BSTR *pRetVal);
```

```
HRESULT SetSourceServer(BSTR NewValue);
```

See Also

[DestinationServer Property](#)

[SourceDatabase Property](#)

[SourceLogin Property](#)

[SourcePassword Property](#)

[SourceUseTrustedConnection Property](#)

DTS Programming

SourceSite Property

The **SourceSite** property sets or returns the location from which the files will be transferred by a **DTSFTPTask** object.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**SourceSite** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>string</i>	Location from which the files will be transferred, either an Internet FTP site or a network directory

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT SourceSite(BSTR* pVal);

HRESULT SourceSite(BSTR pVal);

Remarks

If the **SourceLocation** property specifies an Internet site, the **SourceSite**

property must be defined.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("SourceSite") [= string]
```

Example

The following example sets the **SourceSite** property to a File Transfer Protocol (FTP) site name:

```
oCustTask.SourceSite = "ftp.microsoft.com"
```

See Also

[SourceFilename Property](#)

[SourceLocation Property](#)

[SourcePassword \(DTSFTPTask\) Property](#)

[SourceUsername Property](#)

DTS Programming

SourceSQLStatement Property

The **SourceSQLStatement** property specifies the SQL statement used to execute on the source rowset.

Applies To

DataDrivenQueryTask Object	DataPumpTask2 Object
DataDrivenQueryTask2 Object	ParallelDataPumpTask Object
DataPumpTask Object	

Syntax

object.SourceSQLStatement [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	SQL statement used to execute on the source rowset

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceSQLStatement(BSTR *pRetVal);
```

```
HRESULT SetSourceSQLStatement(BSTR NewValue);
```

Remarks

The SQL statement specified by this property can contain ? tokens that are replaced by the values of global variables named by the **InputGlobalVariableNames** property.

See Also

[DestinationSQLStatement Property](#)

[InputGlobalVariableNames Property](#)

DTS Programming

SourceTranslateChar Property

The **SourceTranslateChar** property sets or returns a value indicating whether translation is performed for character data on the source server.

Applies To

[TransferObjectsTask2 Object](#)

Syntax

object.**SourceTranslateChar** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a TransferObjectsTask2 object.
<i>boolean</i>	Boolean that specifies whether translation is performed for character data on the source server.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceTranslateChar(VARIANT_BOOL* pRetVal);
```

```
HRESULT SourceTranslateChar(VARIANT_BOOL pRetVal);
```

Remarks

The default is TRUE; translation is performed.

See Also

[DestTranslateChar Property](#)

DTS Programming

SourceType Property

The **SourceType** property sets or returns a code for the type of source object that provides the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object.

Applies To

[DynamicPropertiesTaskAssignment Object](#)

Syntax

object.**SourceType** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DynamicPropertiesTaskAssignment object
<i>value</i>	Code for the type of source object that provides the value to which a DTS package object property will be set

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT SourceType(long* pRetVal);
```

```
HRESULT SourceType(long pRetVal);
```

Remarks

The valid values for this property are defined by the **DynamicPropertiesTaskSourceType** constants.

Example

The following example sets the **SourceType** property to a value from a DTS enumeration:

```
oAssign.SourceType = DTSDynamicPropertiesSourceType_IniFile
```

See Also

[DynamicPropertiesTaskSourceType](#)

[SourceConstantValue Property](#)

[SourceDataFileFileName Property](#)

[SourceEnvironmentVariable Property](#)

[SourceGlobalVariable Property](#)

[SourceIniFileFileName Property](#)

[SourceIniFileKey Property](#)

[SourceIniFileSection Property](#)

[SourceQueryConnectionID Property](#)

[SourceQuerySQL Property](#)

DTS Programming

SourceUsername Property

The **SourceUsername** property sets or returns the user name that will be used to connect to the Internet File Transfer Protocol (FTP) site by a **DTSFTPTask** object.

Applies To

[DTSFTPTask Object](#)

Syntax

object.**SourceUsername** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSFTPTask object
<i>value</i>	User name that will be used to connect to the Internet FTP site

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT SourceUsername(BSTR* pVal);

HRESULT SourceUsername(BSTR pVal);

Remarks

Typically, "anonymous" can be used for the user name when establishing a read-

only connection to an Internet FTP site, such as by the **DTSFTPTask**.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("SourceUsername") [= value]
```

Example

The following example sets the **SourceUsername** property to a string:

```
oCustTask.SourceUsername = "anonymous"
```

See Also

[SourceFilename Property](#)

[SourceLocation Property](#)

[SourcePassword \(DTSFTPTask\) Property](#)

[SourceSite Property](#)

DTS Programming

SourceUseTrustedConnection Property

The **SourceUseTrustedConnection** property specifies whether the Windows Authentication security mode is used.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**SourceUseTrustedConnection**[= *value*]

Part	Description
<i>object</i>	An expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether the Windows Authentication security mode is to be used

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSourceUseTrustedConnection(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetSourceUseTrustedConnection(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

Note It is recommended that you connect to an instance of Microsoft® SQL Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **SourceUseTrustedConnection** to TRUE.

See Also

[DestinationUseTrustedConnection Property](#)

[SourceDatabase Property](#)

[SourceLogin Property](#)

[SourcePassword Property](#)

[SourceServer Property](#)

DTS Programming

SQLStatement Property

The **SQLStatement** property specifies a sequence of one or more SQL statements or stored procedure references to be executed.

Applies To

ExecuteSQLTask Object	ExecuteSQLTask2 Object
---------------------------------------	--

Syntax

object.**SQLStatement** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Query that is a sequence of one or more SQL statements or stored procedure references

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSQLStatement(BSTR *pRetVal);
```

```
HRESULT SetSQLStatement(BSTR NewValue);
```

Remarks

In the **ExecuteSQLTask2** object, the **SQLStatement** query can contain ? tokens

that are replaced by the values of global variables specified by the **InputGlobalVariableNames** property. Fields from the first row of the rowset generated by the query are assigned to the global variables specified by the **OutputGlobalVariableNames** property. The entire rowset can be written to a global variable as a disconnected Microsoft® ActiveX® Data Objects (ADO) recordset if the **OutputAsRecordset** property is set.

See Also

[InputGlobalVariableNames Property](#)

[OutputAsRecordset Property](#)

[OutputGlobalVariableNames Property](#)

DTS Programming

StartTime Property

The **StartTime** property specifies when the Data Transformation Services (DTS) package or step execution started.

Applies To

PackageLogRecord Object	StepLineage Object
Step Object	StepLogRecord Object
Step2 Object	

Syntax

object.**StartTime**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetStartTime(DATE *pRetVal);
```

See Also

[ExecutionTime Property](#)

[FinishTime Property](#)

DTS Programming

StepExecutionID Property

The **StepExecutionID** property specifies a sequence number for the step log record.

Applies To

[StepLogRecord Object](#)

Syntax

object.**StepExecutionID**

Part	Description
<i>object</i>	Expression that evaluates to a StepLogRecord object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetStepExecutionID(long *pRetVal);
```

Remarks

StepExecutionID can be specified as a parameter when enumerating or removing step log records and task log records.

See Also

[EnumStepLogRecords Method](#)

[EnumTaskLogRecords Method](#)

[RemoveStepLogRecords Method](#)

[RemoveTaskLogRecords Method](#)

[StepExecutionResult Property](#)

[StepExecutionStatus Property](#)

DTS Programming

StepExecutionResult Property

The **StepExecutionResult** property returns the result of the logged step execution.

Applies To

StepLineage Object	StepLogRecord Object
------------------------------------	--------------------------------------

Syntax

object.**StepExecutionResult**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[DTSStepExecResult](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetStepExecutionResult(DTSStepExecResult *pRetVal);
```

Remarks

The **StepExecutionResult** value indicates success or failure of the logged step.

See Also

[ExecutionResult Property](#)

[ExecutionStatus Property](#)

[StepExecutionStatus Property](#)

DTS Programming

StepExecutionStatus Property

The **StepExecutionStatus** property returns the status of the logged step execution.

Applies To

StepLineage Object	StepLogRecord Object
------------------------------------	--------------------------------------

Syntax

object.**StepExecutionStatus**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[DTSStepExecStatus](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetStepExecutionStatus(DTSStepExecStatus *pRetVal);
```

Remarks

Typically, you will see DTSStepExecStat_Completed (4) for **StepExecutionStatus** logged step execution records.

See Also

[ExecutionResult Property](#)

[ExecutionStatus Property](#)

[StepExecutionResult Property](#)

DTS Programming

StepName Property

The **StepName** property specifies the name of the step whose status or result is evaluated when determining if this constraint is satisfied.

Applies To

[PrecedenceConstraint Object](#)

Syntax

object.**StepName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a PrecedenceConstraint object
<i>value</i>	Name of the step that is evaluated

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetStepName(BSTR *pRetVal);
```

```
HRESULT SetStepName(THIS_ DTS_IN BSTR NewValue);
```

See Also

[PrecedenceBasis Property](#)

[Step2 Object](#)

DTS Programming

StringCompareType Property

The **StringCompareType** property sets or returns the type of comparison to be performed on a received string message.

Applies To

[DTSMMessageQueueTask Object](#)

Syntax

object.**StringCompareType** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMMessageQueueTask object
<i>value</i>	Code that defines the string comparison type

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT StringCompareType(long* pVal);
```

```
HRESULT StringCompareType(long pVal);
```

Remarks

The comparison is performed between the received message and the value of the **StringCompareValue** property.

The valid values for this property are defined by the **DTSMQStringMessageCompare** constants.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("StringCompareType") [= value]
```

See Also

[DTSMQStringMessageCompare](#)

[StringCompareValue Property](#)

DTS Programming

StringCompareValue Property

The **StringCompareValue** property sets or returns the string to be compared with a received string message.

Applies To

[DTSMessagesQueueTask Object](#)

Syntax

object.**StringCompareValue** [= *string*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMessagesQueueTask object
<i>string</i>	String to be compared with a received string message

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT StringCompareValue(BSTR* pVal);

HRESULT StringCompareValue(BSTR pVal);

Remarks

The type of comparison is specified by the **StringCompareType** property. The **StringCompareValue** property must be defined unless the comparison type is

None.

The property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("StringCompareValue") [= string]
```

See Also

[StringCompareType Property](#)

DTS Programming

Subject Property

The **Subject** property specifies the Subject: line of an e-mail message.

Applies To

[SendMailTask Object](#)

Syntax

object.**Subject**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	Subject line of an e-mail message

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetSubject(BSTR *pRetVal);

HRESULT SetSubject(BSTR NewValue);

See Also

[CCLine Property](#)

[MessageText Property](#)

ToLine Property

DTS Programming

SuccessReturnCode Property

The **SuccessReturnCode** property specifies a return code that indicates whether the task completed successfully.

Applies To

CreateProcessTask Object	CreateProcessTask2 Object
--	---

Syntax

object.**SuccessReturnCode** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Return code from the process

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetSuccessReturnCode(LONG *pRetVal);
```

```
HRESULT SetSuccessReturnCode(LONG NewValue);
```

Remarks

If the process exit code does not match the value specified by **SuccessReturnCode**, the task fails. The default value is 0.

See Also

[FailPackageOnTimeout Property](#)

[TerminateProcessAfterTimeout Property](#)

[Timeout Property](#)

DTS Programming

TableLock Property

The **TableLock** property indicates whether an entire table is locked during a load operation.

Applies To

[BulkInsertTask Object](#)

Syntax

object.**TableLock** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a BulkInsertTask object
<i>value</i>	Boolean that specifies whether an entire table is locked

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTableLock(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetTableLock(VARIANT_BOOL NewValue);
```

Remarks

The default is FALSE.

See Also

[DestinationTableName Property](#)

DTS Programming

TaskName Property

The **TaskName** property specifies the name of the task to execute.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**TaskName** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Name of the task to execute

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTaskName(BSTR *pRetVal);
```

```
HRESULT SetTaskName(BSTR NewValue);
```

Remarks

The **TaskName** property is the link between the **Task** object and the step that contains it. Typically, set:

```
Step2.TaskName = Task.Name
```

See Also

[Task Object](#)

[Tasks Collection](#)

DTS Programming

TaskType Property

The **TaskType** property sets or returns the type of the **DTSMessageQueueTask** object, sender, or receiver.

Applies To

[DTSMessageQueueTask Object](#)

Syntax

object.**TaskType** [= *value*]

Part	Description
<i>Object</i>	Expression that evaluates to a DTSMessageQueueTask object
<i>Value</i>	Code that defines the task type, sender, or receiver

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT TaskType(long* pVal);
```

```
HRESULT TaskType(long pVal);
```

Remarks

The valid values for this property are defined by the **DTSMQType** constants.

The property also can be referenced through the **Properties** collection of the

Task object with the following code:

```
Set taskprops = task.Properties  
taskprops("TaskType") [= value]
```

See Also

[DTSMQType](#)

DTS Programming

TerminateProcessAfterTimeout Property

The **TerminateProcessAfterTimeout** property specifies whether to terminate the process after the time-out period has expired.

Applies To

CreateProcessTask Object	CreateProcessTask2 Object
--	---

Syntax

object.**TerminateProcessAfterTimeout** [= *value*]

Part	Description
<i>Object</i>	Expression that evaluates to an object in the Applies To list
<i>Value</i>	Boolean that specifies whether to terminate the process

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTerminateProcessAfterTimeout(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetTerminateProcessAfterTimeout(VARIANT_BOOL NewValue);
```

Remarks

If **TerminateProcessAfterTimeout** is FALSE (the default), the task fails after the time-out without terminating the created process. The

FailPackageOnTimeout property determines if the entire package is terminated after the time-out occurs.

See Also

[FailPackageOnTimeout Property](#)

[Timeout Property](#)

DTS Programming

Text Property

The **Text** property specifies or returns the text of a Microsoft® ActiveX® script.

Applies To

DataPumpTransformScript Object	DTSTransformScriptProperties2 Object
--	--

Syntax

object.**Text** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	ActiveX script text

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetText(BSTR *pRetVal);
```

```
HRESULT SetText(BSTR NewValue);
```

Remarks

The scripting language is specified by the **Language** property.

The property also can be referenced through the **TransformServerProperties**

collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("Text") [= string]
```

See Also

[FunctionEntry Property](#)

[Language Property](#)

DTS Programming

Timeout Property

The **Timeout** property specifies the number of seconds in the time-out period.

Applies To

CreateProcessTask Object	CreateProcessTask2 Object
--	---

Syntax

object.**Timeout** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Number of seconds in the time-out period

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTimeout(LONG *pRetVal);
```

```
HRESULT SetTimeout(LONG NewValue);
```

Remarks

A value of 0 (default) indicates no time-out period. Otherwise, the task fails after the time-out period elapses. If the **FailPackageOnTimeout** property is set, the entire package fails. If the **TerminateProcessAfterTimeout** property is set, the

created process is failed.

See Also

[FailPackageOnTimeout Property](#)

[TerminateProcessAfterTimeout Property](#)

DTS Programming

ToLine Property

The **ToLine** property specifies e-mail addresses to include on the To: line.

Applies To

[SendMailTask Object](#)

Syntax

object.**ToLine**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>value</i>	E-mail addresses to which the mail message is sent

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

HRESULT GetToLine(BSTR *pRetVal);

HRESULT SetToLine(BSTR NewValue);

Remarks

The e-mail addresses must be separated by semicolons.

See Also

[CCLine Property](#)

[MessageText Property](#)

[Subject Property](#)

DTS Programming

TransactionIsolationLevel Property

The **TransactionIsolationLevel** property specifies the isolation level at which a **Package2** object transaction executes if the **UseTransaction** property is set to TRUE.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**TransactionIsolationLevel**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Isolation level at which a Package2 object transaction executes

Data Type

[DTSIsoationLevel](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTransactionIsolationLevel(DTSIsolationLevel *pRetVal);
```

```
HRESULT SetTransactionIsolationLevel(DTSIsolationLevel NewValue);
```

Remarks

The default is ReadCommitted.

See Also

[AutoCommitTransaction Property](#)

[InTransaction Property](#)

[JoinTransactionIfPresent Property](#)

[UseTransaction Property](#)

DTS Programming

TransformationSetOptions Property

The **TransformationSetOptions** property returns or sets the mode in which a **ParallelDataPumpTask** object operates.

Applies To

[ParallelDataPumpTask Object](#)

Syntax

oParallelDP.**TransformationSetOptions** [= *value*]

Part	Description
<i>oParallelDP</i>	Expression that evaluates to a ParallelDataPumpTask object
<i>value</i>	DTSTransformationSetOptions constant that specifies the ParallelDataPumpTask operating mode

Data Type

[DTSTransformationSetOptions Constants](#)

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTransformationSetOptions(DTSTransformationSetOptions* pRetVal);
```

```
HRESULT SetTransformationSetOptions(DTSTransformationSetOptions NewValue);
```

Remarks

The property must be set to one of the DTSTransformationSetOptions values.
The default is DTSTranSetOpt_Flattened.

See Also

[TransformationSet Object](#)

DTS Programming

TransformFailureFunctionEntry Property

The **TransformFailureFunctionEntry** property specifies or returns the name of the script function that is to be called for the OnTransformFailure transformation phase.

Applies To

[DTSTransformScriptProperties2 Object](#)

Syntax

object.**TransformFailureFunctionEntry** [= *name*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSTransformScriptProperties2 object
<i>name</i>	Name of the script function that supports the OnTransformFailure phase

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT TransformFailureFunctionEntry(BSTR* pRetVal);
```

```
HRESULT TransformFailureFunctionEntry(BSTR pRetVal);
```

Remarks

The OnTransformFailure phase occurs after the Transform phase returns **DTSTransformStat_Error** or **DTSTransformStat_ExceptionRow**, in the **DataPumpTask2**, **DataDrivenQueryTask2** or **ParallelDataPumpTask** objects.

The OnTransformFailure script function has read access to the columns of the **DTSSource** collection and write access to the columns of the **DTSDestination** collection. Any return values that are valid during the Transform phase can be returned.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("TransformFailureFunctionEntry") [= string]
```

See Also

[BatchCompleteFunctionEntry Property](#)

[DataDrivenQueryTask2 Object](#)

[DataPumpTask2 Object](#)

[DTSTransformStatus](#)

[FunctionEntry Property](#)

[InsertFailureFunctionEntry Property](#)

[InsertSuccessFunctionEntry Property](#)

[ParallelDataPumpTask Object](#)

[PostSourceDataFunctionEntry Property](#)

[PreSourceDataFunctionEntry Property](#)

[PumpCompleteFunctionEntry Property](#)

DTS Programming

TransformFlags Property

The **TransformFlags** property specifies transformation flags that indicate characteristics of a transformation.

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**TransformFlags** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Flags that indicate characteristics of the transformation. Use a sum of codes from the DTSTransformFlags constants.

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTransformFlags(LONG *pRetVal);
```

```
HRESULT SetTransformFlags(LONG NewValue);
```

Remarks

The **TransformFlags** property controls the types of conversions that are

considered either valid or invalid (for example, possible overflow, possible loss of sign when converting signed to unsigned, possible string truncation). The test is made at the beginning of the transformation, not row by row.

See Also

[DTSTransformFlags](#)

DTS Programming

TransformPhases Property

The **TransformPhases** property returns or sets the transform phases that this transformation supports.

Applies To

[Transformation2 Object](#)

Syntax

object.**TransformPhases** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to a Transformation2 object
<i>value</i>	Codes that indicate the phases this transformation supports. Must be a sum of values from the DTSTransformPhaseEnum constants.

Data Type

Long

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTransformPhases(LONG *pRetVal);
```

```
HRESULT SetTransformPhases(LONG NewValue);
```

Remarks

These are the transformation phases that a transformation can support.

Phase	Description
PreSourceData	Occurs before the first row is fetched from source connection.
Transform	Occurs after each source row is fetched, before the destination row is written.
OnTransformFailure	Occurs after a failure in the Transform phase, indicated by the return of DTSTransformStat_Error or DTSTransformStat_ExceptionRow . Typically, this phase is caused by conversion errors.
OnInsertSuccess	Occurs after each data row is written successfully to the destination connection.
OnInsertFailure	Occurs after each attempt to write a data row to the destination connection failed (for example, by attempting to write a duplicate value to a primary key field, or a NULL to a NOT NULL field).
OnBatchComplete	Occurs in DataPumpTask2 if you select the FastLoad check box after each batch is written, successfully or unsuccessfully.
PostSourceData	Occurs after the last row is written to the destination connection.
OnPumpComplete	Occurs at the end of the execution of the task.

See Also

[CurrentPhase Property](#)

[DTSTransformPhaseEnum](#)

DTS Programming

TransformServer Property

The **TransformServer** property returns a reference to the transform server object (the class-specific transformation object) through which the properties of that object can be directly accessed.

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

[Set *transform* =] *object*.**TransformServer**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>transform</i>	Reference to the transform server object

Data Type

Object

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTransformServer(IDispatch **pRetVal);
```

Remarks

You can access the properties of the class-specific transformation object directly or through the **TransformServerProperties** collection of the **Transformation2** object.

Example

The following code creates a **DataPumpTransformMidString** transformation, and then references the transform server object and uses it to set the **CharacterStart** property:

```
Dim objDataPump As DTS.DataPumpTask
Dim objTransform As DTS.Transformation
Dim objMidString As DTSPump.DataPumpTransformMidString
...
Set objTransform = objDataPump.Transformations. _
    New("DTSPump.DataPumpTransformMidString")
Set objMidString = objTransform.TransformServer
objMidString.CharacterStart = 5
```

See Also

[TransformServerID Property](#)

[TransformServerParameter Property](#)

[TransformServerProperties Property](#)

DTS Programming

TransformServerID Property

The **TransformServerID** property returns the programmatic identifier (ProgID) or class identifier (CLSID) of the transform server object (the class-specific transformation object).

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**TransformServerID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetTransformServerID(BSTR *pRetVal);
```

Remarks

The ProgID or CLSID returned by the **TransformServerID** property is the identifier that was used with the **New** method of the **Transformations** collection to create the **Transformation2** object.

See Also

[New \(ID\) Method](#)

[Transformations Collection](#)

[TransformServer Property](#)

[TransformServerParameter Property](#)

[TransformServerProperties Property](#)

DTS Programming

TransformServerParameter Property

The **TransformServerParameter** property specifies an initialization parameter for the transform server object (class-specific transformation object), if required.

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**TransformServerParameter** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Transform server's initialization parameter

Data Type

Variant

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetTransformServerParameter(VARIANT *pRetVal);
```

```
HRESULT SetTransformServerParameter(VARIANT NewValue) ;
```

Remarks

Because the **TransformServerParameter** property has a variant data type, it may be either a scalar value or an array. Some custom transformation servers

may expose an alternate interface on their **IDispatch** transformation server objects to specify complex parameters.

See Also

[TransformServer Property](#)

[TransformServerID Property](#)

[TransformServerProperties Property](#)

DTS Programming

TransformServerProperties Property

The **TransformServerProperties** property returns a reference to a **Properties** collection containing the properties of the transform server object (the class-specific transformation object).

Applies To

Transformation Object	Transformation2 Object
---------------------------------------	--

Syntax

object.**TransformServerProperties**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Properties

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetTransformServerProperties(IDTSProperties **pRetVal);
```

Remarks

You can use the **TransformServerProperties** property to access the properties of the class-specific transformation object without creating a reference to that object.

Example

The following code creates a **DataPumpTransformMidString** transformation, and then references the **CharacterStart** property through the **TransformServerProperties** property:

```
Dim objDataPump As DTS.DataPumpTask
Dim objTransform As DTS.Transformation
...
Set objTransform = objDataPump.Transformations. _
    New("DTSPump.DataPumpTransformMidString")
objTransform.TransformServerProperties("CharacterStart").Value =
```

See Also

[Properties Collection](#)

[TransformServer Property](#)

[TransformServerID Property](#)

[TransformServerParameter Property](#)

DTS Programming

TrimEmbeddedWhiteSpace Property

The **TrimEmbeddedWhiteSpace** property specifies or returns a value indicating whether embedded white-space characters are removed from the source column string copied by custom transformations.

Applies To

DataPumpTransformMidString Object	DataPumpTransformTrimString Object
---	--

Syntax

transerver.**TrimEmbeddedWhiteSpace** [= *boolean*]

Part	Description
<i>transerver</i>	Expression that evaluates to an object from the Applies To list.
<i>boolean</i>	If TRUE, embedded white-space characters are removed from the source string. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

HRESULT TrimEmbeddedWhiteSpace(VARIANT_BOOL* pRetVal);

HRESULT TrimEmbeddedWhiteSpace(VARIANT_BOOL pRetVal);

Remarks

White-space characters are spaces, tabs, carriage returns and linefeeds. Embedded white-space characters are those that appear between the first and last character that is not a white-space character.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("TrimEmbeddedWhiteSpace") [= boolean]
```

See Also

[TrimLeadingWhiteSpace Property](#)

[TrimTrailingWhiteSpace Property](#)

DTS Programming

TrimLeadingWhiteSpace Property

The **TrimLeadingWhiteSpace** property specifies or returns a value indicating whether leading white-space characters are removed from the source column string copied by custom transformations.

Applies To

DataPumpTransformMidString Object	DataPumpTransformTrimString Object
---	--

Syntax

transerver.**TrimLeadingWhiteSpace** [= *boolean*]

Part	Description
<i>transerver</i>	Expression that evaluates to an object from the Applies To list.
<i>boolean</i>	If TRUE, leading white-space characters are removed from the source string. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT TrimLeadingWhiteSpace(VARIANT_BOOL* pRetVal);
```

```
HRESULT TrimLeadingWhiteSpace(VARIANT_BOOL pRetVal);
```

Remarks

White-space characters are spaces, tabs, carriage returns and linefeeds. Leading white-space characters are those that appear ahead of the first character that is not a white-space character.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("TrimLeadingWhiteSpace") [= boolean]
```

See Also

[TrimEmbeddedWhiteSpace Property](#)

[TrimTrailingWhiteSpace Property](#)

DTS Programming

TrimTrailingWhiteSpace Property

The **TrimTrailingWhiteSpace** property specifies or returns a value indicating whether trailing white-space characters are removed from the source column string copied by custom transformations.

Applies To

DataPumpTransformMidString Object	DataPumpTransformTrimString Object
---	--

Syntax

transerver.**TrimTrailingWhiteSpace** [= *boolean*]

Part	Description
<i>transerver</i>	Expression that evaluates to an object from the Applies To list.
<i>boolean</i>	If TRUE, trailing white-space characters are removed from the source string. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT TrimTrailingWhiteSpace(VARIANT_BOOL* pRetVal);
```

```
HRESULT TrimTrailingWhiteSpace(VARIANT_BOOL pRetVal);
```

Remarks

White-space characters are spaces, tabs, carriage returns and linefeeds. Trailing white-space characters are those that appear after the last character that is not a white-space character.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("TrimTrailingWhiteSpace") [= boolean]
```

See Also

[TrimEmbeddedWhiteSpace Property](#)

[TrimLeadingWhiteSpace Property](#)

DTS Programming

Type Property

The **Type** property specifies the data type of the value of a **Property** object.

Applies To

[Property Object](#)

Syntax

object.Type

Part	Description
<i>object</i>	Expression that evaluates to a Property object

Data Type

Long

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetType(long *pRetVal);
```

See Also

[Get Property](#)

[Set Property](#)

[Value Property](#)

DTS Programming

UDLPath Property

The **UDLPath** property specifies the name and path of a Microsoft® Data Link file used to create a connection.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.UDLPath [= *path*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>path</i>	Name and path of the data link file

Data Type

String

Modifiable

Read/write for the **Connection2** object. Read-only for the **Connection** object.

Prototype (C/C++)

```
HRESULT GetUDLPath(BSTR *pRetVal);
```

Remarks

When a data link file is used, the **ConnectionProperties** collection is not available.

See Also

[ConnectionProperties Property](#)

DTS Programming

UnicodeFile Property

The **UnicodeFile** property specifies or returns a value indicating whether the data read from or written to files by transformations is translated from or to Unicode.

Applies To

DataPumpTransformReadFile Object	DataPumpTransformWriteFile Object
--	---

Syntax

transerver.**UnicodeFile** [= *boolean*]

Part	Description
<i>transerver</i>	Expression that evaluates to an object from the Applies To list.
<i>boolean</i>	If TRUE, the file data is translated from or to Unicode. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT UnicodeFile(VARIANT_BOOL* pRetVal);
```

```
HRESULT UnicodeFile(VARIANT_BOOL pRetVal);
```

Remarks

For the **DataPumpTransformReadFile** object, the file is translated from Unicode to ANSI if the destination column is not Unicode and the **UnicodeFile** property is TRUE.

For the **DataPumpTransformWriteFile** object, the source column data has already been translated to from ANSI to Unicode, if necessary. If the **UnicodeFile** property is TRUE, the Unicode header bytes 0xFFFFE are written to the file, unless the **AppendIfFileExists** property is TRUE and the header is already in the file.

If the **UnicodeFile** property is TRUE, the **OEMFile** property is ignored.

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("UnicodeFile") [= boolean]
```

See Also

[AppendIfFileExists Property](#)

[OEMFile Property](#)

DTS Programming

UpdateQuery Property

The **UpdateQuery** property specifies a string of one or more parameterized SQL statements to execute at the destination as the Update query.

Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.UpdateQuery [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Parameterized string of SQL statements

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUpdateQuery(BSTR *pRetVal);
```

```
HRESULT SetUpdateQuery(BSTR NewValue);
```

Remarks

Although the name of a data driven query property is preset, its content is not

enforced. Any of the queries may be used for any desired operation. The nomenclature is provided as a convenient means of identification. The **UpdateQuery** property does not need to contain a Transact-SQL UPDATE statement.

The values of the columns specified by the **UpdateQueryColumns** property replace the parameters placeholders in the **UpdateQuery** in the order in which the columns were added to the collection.

See Also

[Adding DTS Query Strings](#)

[DeleteQuery Property](#)

[InsertQuery Property](#)

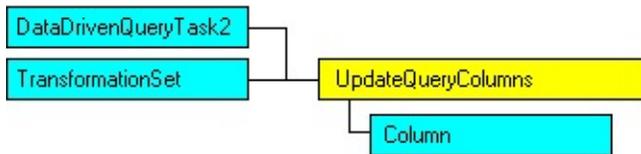
[UpdateQueryColumns Property](#)

[UserQuery Property](#)

DTS Programming

UpdateQueryColumns Property

The **UpdateQueryColumns** property returns a reference to a collection of **Column** objects that serve as parameters for the query specified by the **UpdateQuery** property.



Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.UpdateQueryColumns

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetUpdateQueryColumns(IDTSColumns **pRetVal);
```

Remarks

The values of the columns specified by the **UpdateQueryColumns** property are substituted for the parameters in the **UpdateQuery** in the order in which the columns were added to the collection.

See Also

[Adding DTS Query Strings](#)

[Column Object](#)

[DeleteQueryColumns Property](#)

[InsertQueryColumns Property](#)

[UpdateQuery Property](#)

[UserQueryColumns Property](#)

DTS Programming

UpperCaseString Property

The **UpperCaseString** property specifies or returns a value indicating whether the alphabetical characters in the source column string copied by transformations are all converted to uppercase characters.

Applies To

DataPumpTransformMidString Object	DataPumpTransformTrimString Object
---	--

Syntax

transerver.**UpperCaseString** [= *boolean*]

Part	Description
<i>transerver</i>	Expression that evaluates to an object from the Applies To list.
<i>boolean</i>	Boolean that specifies whether the alpha characters in the source string are converted to uppercase. Default is FALSE.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT UpperCaseString(VARIANT_BOOL* pRetVal);
```

```
HRESULT UpperCaseString(VARIANT_BOOL pRetVal);
```

Remarks

The property also can be referenced through the **TransformServerProperties** collection with the following code:

```
Set transprops = transform.TransformServerProperties  
transprops("UpperCaseString") [= boolean]
```

See Also

[LowerCaseString Property](#)

DTS Programming

UseCache Property

The **UseCache** property returns or sets whether cached information is used when enumerating the associated collection.

Applies To

OLEDBProviderInfos Collection	TaskInfos Collection
ScriptingLanguageInfos Collection	TransformationInfos Collection

Syntax

object.UseCache [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>value</i>	Determines whether cached information is used to enumerate the associated collection.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUseCache(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetUseCache(VARIANT_BOOL NewValue);
```

Remarks

When **UseCache** is FALSE, the information about the components of the associated collection is obtained from the operating system registry. This requires all registered classes to be examined and may take a significant amount of time, depending on the amount of software installed on the computer.

Data Transformation Services (DTS) maintains a cache in the system registry of the components of each collection in the Applies To list and examines the appropriate cache, rather than all registered classes, when **UseCache** is TRUE. Use the **Refresh** method to refresh the appropriate cache from the system registry.

See Also

[Refresh Method](#)

DTS Programming

UseCollation Property

The **UseCollation** property specifies whether column-level collation settings on the source table are used when transferring data between computers running instances of Microsoft® SQL Server™ 2000.

Applies To

[TransferObjectsTask2 Object](#)

Syntax

transfobj.**UseCollation** [= *value*]

Part	Description
<i>transfobj</i>	Expression that evaluates to a TransferObjectsTask2 object.
<i>value</i>	Determines whether column-level collation settings are used when transferring data.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUseCollation(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetUseCollation(VARIANT_BOOL NewValue);
```

Remarks

The default for this property is TRUE. The **UseCollation** property of the **TransferObjectsTask2** object exposes the **UseCollation** property of the SQL-DMO **Transfer2** object.

If **UseCollation** is set to TRUE and the destination table is dropped during transfer, the new destination table will be created with column-level collations identical to those of the source table.

If **UseCollation** is set to FALSE and the destination table is dropped during transfer, the new destination table will be created with column-level collations identical to those of the destination server default code page. Note that if the column-level collation of the source data does not match the source server default code page, mistranslation will occur.

See Also

[DestTranslateChar Property](#)

[SourceTranslateChar Property](#)

[Transfer2 Object](#)

[UseCollation Property](#)

DTS Programming

UseFastLoad Property

The **UseFastLoad** property specifies whether to use the **FastLoad** option, where rows are processed in batches under a single transaction commit.

Applies To

DataPumpTask Object	DataPumpTask2 Object
-------------------------------------	--------------------------------------

Syntax

object.**UseFastLoad** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>value</i>	Boolean that specifies whether to use the FastLoad option, if available.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUseFastLoad(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetUseFastLoad(VARIANT_BOOL NewValue);
```

Remarks

The batch size is controlled by the **InsertCommitSize** property.

The **FastLoad** option is available if the OLE DB provider supports the **IRowsetFastLoad** interface.

See Also

[InsertCommitSize Property](#)

DTS Programming

UseOLEDBServiceComponents Property

The **UseOLEDBServiceComponents** property specifies whether to use OLE DB service components when initializing data sources.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.UseOLEDBServiceComponents[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether to use OLE DB service components

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUseOLEDBServiceComponents(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetUseOLEDBServiceComponents(VARIANT_BOOL NewValue);
```

Remarks

The default is TRUE.

See Also

[ConnectionProperties Property](#)

DTS Programming

UseRepository Property

The **UseRepository** property sets or returns a value indicating whether Microsoft® SQL Server™ 2000 Meta Data Services should be used as the source of the Data Transformation Services (DTS) package to be run by an **ExecutePackageTask** object.

Applies To

[ExecutePackageTask](#)

Syntax

object.**UseRepository** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to an ExecutePackageTask object.
<i>boolean</i>	If TRUE, Meta Data Services is searched for the DTS package to be run.

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT UseRepository(VARIANT_BOOL* pRetVal);
```

```
HRESULT UseRepository(VARIANT_BOOL pRetVal);
```

Remarks

If **UseRepository** is FALSE, the DTS package is sought in the SQL Server **msdb** database, unless a storage file specification is provided in the **FileName** property.

This property also can be referenced through the **Properties** collection of the **Task** object with the following code:

```
Set taskprops = task.Properties  
taskprops("UseRepository") [= boolean]
```

See Also

[FileName Property](#)

[RepositoryDatabaseName Property](#)

DTS Programming

UserID Property

The **UserID** property specifies a user ID or name to use when making a connection.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

object.**UserID** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	User ID or name to use

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUserID(BSTR *pRetVal);
```

```
HRESULT SetUserID(BSTR NewValue);
```

Remarks

You need to specify values for the **UserID** and **Password** properties if **UseTrustedConnection** is FALSE.

Note It is recommended that you connect to an instance of Microsoft® SQL Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **UseTrustedConnection** to TRUE.

See Also

[Password Property](#)

[UseTrustedConnection Property](#)

DTS Programming

UserQuery Property

The **UserQuery** property specifies a string of one or more parameterized SQL statements to execute at the destination as the user query.

Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.**UserQuery** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Parameterized string of SQL statements

Data Type

String

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUserQuery(BSTR *pRetVal);
```

```
HRESULT SetUserQuery(BSTR NewValue);
```

Remarks

Although the name of a data driven query property is preset, its content is not

enforced. Any of the queries may be used for any desired operation. The nomenclature is provided as a convenient means of identification. Typically, the **UserQuery** property contains a stored procedure reference.

The values of the columns specified by the **UserQueryColumns** property replace the parameter placeholders in the **UserQuery** in the order in which the columns were added to the collection.

See Also

[Adding DTS Query Strings](#)

[DeleteQuery Property](#)

[InsertQuery Property](#)

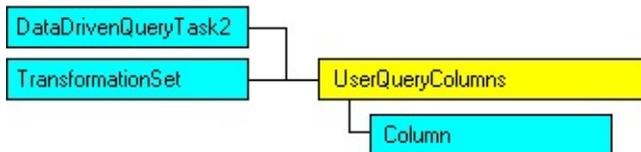
[UpdateQuery Property](#)

[UserQueryColumns Property](#)

DTS Programming

UserQueryColumns Property

The **UserQueryColumns** property returns a reference to a collection of **Column** objects that serve as parameters for the query specified by the **UserQuery** property.



Applies To

DataDrivenQueryTask Object	TransformationSet Object
DataDrivenQueryTask2 Object	

Syntax

object.**UserQueryColumns**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

[Columns Collection](#)

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetUserQueryColumns(IDTSColumns **pRetVal);
```

Remarks

The values of the columns specified by the **UserQueryColumns** property replace the parameter placeholders in the **UserQuery** in the order in which the columns were added to the collection.

See Also

[Adding DTS Query Strings](#)

[Column Object](#)

[DeleteQueryColumns Property](#)

[InsertQueryColumns Property](#)

[UpdateQueryColumns Property](#)

[UserQuery Property](#)

DTS Programming

UseTransaction Property

The **UseTransaction** property specifies whether the **Package2** object creates a transaction for supporting Data Transformation Services (DTS) tasks.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**UseTransaction**[= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>value</i>	Determines whether the Package2 object creates a transaction.

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUseTransaction(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetUseTransaction(VARIANT_BOOL NewValue);
```

See Also

[InTransaction Property](#)

[JoinTransactionIfPresent Property](#)

TransactionIsolationLevel Property

DTS Programming

UseTransaction (DTSMQMessage) Property

The **UseTransaction** property sets or returns a value indicating whether a transaction is used to send the message defined by the **DTSMQMessage** object.

Applies To

[DTSMQMessage Object](#)

Syntax

object.**UseTransaction** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMQMessage object
<i>boolean</i>	Boolean that specifies whether the message is sent as part of a transaction

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT UseTransaction(VARIANT_BOOL* pVal);
```

```
HRESULT UseTransaction(VARIANT_BOOL pVal);
```

Remarks

All the transacted messages sent by a **DTSMessageQueueTask** object use the

same transaction.

See Also

[WaitForAcknowledgement Property](#)

DTS Programming

UseTrustedConnection Property

The **UseTrustedConnection** property specifies whether the connection connects to the data source using Windows Authentication security mode.

Applies To

Connection Object	ExecutePackageTask Object
Connection2 Object	

Syntax

object.**UseTrustedConnection** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Value indicating whether to connect using Windows Authentication

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetUseTrustedConnection(VARIANT_BOOL *pRetVal);
```

```
HRESULT SetUseTrustedConnection(VARIANT_BOOL NewValue);
```

Remarks

You need to specify values for the **UserID** and **Password** properties if **UseTrustedConnection** is FALSE.

Note It is recommended that you connect to an instance of Microsoft® SQL Server™ using Windows Authentication instead of SQL Server Authentication. To use Windows Authentication, set **UseTrustedConnection** to TRUE.

See Also

[Password Property](#)

[UserID Property](#)

DTS Programming

Value Property

The **Value** property specifies the value of a **GlobalVariable**, **OLEDBProperty**, or **PrecedenceConstraint** object.

Applies To

GlobalVariable Object	OLEDBProperty2 Object
GlobalVariable2 Object	PrecedenceConstraint Object
OLEDBProperty Object	Property Object

Syntax

object.Value

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

Variant

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetValue(VARIANT *pRetVal);
```

```
HRESULT SetValue(VARIANT NewValue);
```

Remarks

For the **PrecedenceConstraint** object, the **Value** property must be from either

the **DTStepExecResult** or **DTStepExecStatus** constants. For the other objects, the **Value** property can be any **Variant**, including references to COM objects, such as a Data Transformation Services (DTS) **Package2** object or a Microsoft® ActiveX® Data Objects (ADO) disconnected recordset.

See Also

[DTStepExecResult](#)

[DTStepExecStatus](#)

DTS Programming

VersionID Property

The **VersionID** property specifies the globally unique identifier (GUID) of this version of the Data Transformation Services (DTS) package.

Applies To

ExecutePackageTask Object	PackageLineage Object
Package Object	PackageLogRecord Object
Package2 Object	SavedPackageInfo Object
PackageInfo Object	

Syntax

object.**VersionID**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Data Type

String

Modifiable

Read-write for the **ExecutePackageTask** object. Read-only for the other objects.

Prototype (C/C++)

```
HRESULT GetVersionID(BSTR *pRetVal);
```

Remarks

For the **ExecutePackageTask** object, if the **VersionID** is not specified, the most recent version of the package is run. The **PackageID** does not need to be

specified if the **VersionID** is specified because **VersionID** uniquely identifies both package and version.

To determine the version ID of a package, open the package in DTS Designer, and then in the **Package/Properties** dialog box, click the **General** tab.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where X represents hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded.

See Also

[PackageID Property](#)

DTS Programming

VersionSaveDate Property

The **VersionSaveDate** property specifies the date and time a version of a Data Transformation Services (DTS) package was saved.

Applies To

[SavedPackageInfo Object](#)

Syntax

object.VersionSaveDate

Part	Description
<i>object</i>	Expression that evaluates to a SavedPackageInfo object

Data Type

Date

Modifiable

Read-only

Prototype (C/C++)

```
HRESULT GetVersionSaveDate(*pRetVal);
```

See Also

[PackageCreationDate Property](#)

DTS Programming

WaitForAcknowledgement Property

The **WaitForAcknowledgement** property returns or sets a value indicating whether a **DTSMessageQueueTask** waits for an acknowledgement after sending the message defined by the **DTSMQMessage** object.

Applies To

[DTSMQMessage Object](#)

Syntax

object.**WaitForAcknowledgement** [= *boolean*]

Part	Description
<i>object</i>	Expression that evaluates to a DTSMQMessage object
<i>boolean</i>	Boolean that specifies whether the task sending the message waits for an acknowledgement

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetWaitForAcknowledgement(VARIANT_BOOL *pVal);
```

```
HRESULT SetWaitForAcknowledgement(VARIANT_BOOL pVal);
```

Remarks

The sending task does not complete until the acknowledgement is received. If a transaction is used, it is not committed until the acknowledgement is received.

All the transacted messages sent by a **DTSMessageQueueTask** object use the same transaction.

See Also

[UseTransaction Property](#)

DTS Programming

WriteCompletionStatusToNTEventLog Property

The **WriteCompletionStatusToNTEventLog** property specifies whether the completion status of the Data Transformation Services (DTS) package is written to the Microsoft® Windows® application log.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**WriteCompletionStatusToNTEventLog** [= *value*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>value</i>	Boolean that specifies whether package status is written to the Windows application log

Data Type

Boolean

Modifiable

Read/write

Prototype (C/C++)

```
HRESULT GetWriteCompletionStatusToNTEventLog(VARIANT_BOOL  
*pRetVal);
```

```
HRESULT SetWriteCompletionStatusToNTEventLog(VARIANT_BOOL  
NewValue);
```

Remarks

The default is TRUE.

See Also

[FailOnError Property](#)

[FailPackageOnLogFailure Property](#)

DTS Programming

Methods

This section defines the methods of the Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) objects and collections. The methods control the operation of the DTS packages and other objects.

DTS Programming

AcquireConnection Method

The **AcquireConnection** method allows a task to acquire exclusive ownership of a connection to an OLE DB service provider.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

[*Value* =] *Connection*.**AcquireConnection**(*TaskName*)

Part	Description
<i>Connection</i>	Expression that evaluates to an object in the Applies To list
<i>TaskName</i>	Name of the task that is to acquire the connection
<i>Value</i>	Boolean that indicates whether the connection was acquired

Remarks

When an application calls **Connection.AcquireConnection**, the **InUse** property is set to **TRUE**. The call fails if another task owns the connection. A task releases a connection by calling the **ReleaseConnection** method. This method is required only if you are creating a custom task.

All tasks must include calls to **AcquireConnection** and **ReleaseConnection**. The tasks supplied with Microsoft® SQL Server™ 2000 already include them.

Prototype (C/C++)

```
HRESULT AcquireConnection(  
    BSTR TaskName,  
    LPUNKNOWN *pRetVal);
```

See Also

[Creating DTS Package Objects and Connections](#)

[InUse Property](#)

[ReleaseConnection Method](#)

DTS Programming

Add Method

The **Add** method adds an object to a collection.

Applies To

Columns Collection	PrecedenceConstraints Collection
Connections Collection	Steps Collection
DTSMQMessages Collection	Tasks Collection
DynamicPropertiesTaskAssignments Collection	Transformations Collection
GlobalVariables Collection	TransformationSets Collection
Lookups Collection	

Syntax

Collection.**Add** *Object*

Part	Description
<i>Collection</i>	Expression that evaluates to an object in the Applies To list
<i>Object</i>	Object to add to the collection

Remarks

The required type of *Object* depends on the collection to which it is being added.

Collection	Object
Columns	Column Object
Connections	Connection Object Connection2 Object
DTSMQMessages	DTSMQMessage Object
DynamicPropertiesTaskAssignments	DynamicPropertiesTaskAssignment

	Object
GlobalVariables	GlobalVariable Object
Lookups	Lookup Object
PrecedenceConstraints	PrecedenceConstraint Object
Steps	Step Object Step2 Object
Tasks	Task Object
Transformations	Transformation Object Transformation2 Object
TransformationSets	TransformationSet Object

The **Add** method does not create an object; it adds an existing object to its collection. Typically, you create the object with the **New** method of the collection.

Prototype (C/C++)

```
HRESULT Add(ObjectInterface *Object);
```

See Also

[New Method](#)

[New \(Columns\) Method](#)

[New \(ID\) Method](#)

[New \(Name\) Method](#)

DTS Programming

AddColumn Method

The **AddColumn** method creates a **Column** object by name and ordinal position and adds it to the collection.

Applies To

[Columns Collection](#)

Syntax

Columns.**AddColumn** *Name,Ordinal*

Part	Description
<i>Columns</i>	Expression that evaluates to a Columns collection
<i>Name</i>	String that is the name of the column
<i>Ordinal</i>	Long that is the ordinal position of the column

Remarks

The **AddColumn** method does not return a reference to the column object it creates. If you need to set **Column** properties other than *Name* and *Ordinal*, you can create the **Column** object with the **New** method, set the properties, and then add it to the collection with the **Add** method. Or, you can access the **Column** object from the collection after creating it with **AddColumn**.

Prototype (C/C++)

```
HRESULT AddColumn(  
    BSTR Name,  
    long Ordinal);
```

Example

This Microsoft® Visual Basic® sample accesses a **Column** object from the collection after creating it with **AddColumn**. The expression **objTransform.SourceColumns** evaluates to a **Columns** collection:

```
Dim objTransform As DTS.Transformation2
```

```
Dim objColumn As DTS.Column
```

```
...
```

```
objTransform.SourceColumns.AddColumn "LastName", 2
```

```
Set objColumn = objTransform.SourceColumns( "LastName" )
```

See Also

[Add Method](#)

[Adding DTS Column Objects](#)

[New \(Columns\) Method](#)

DTS Programming

AddConstraint Method

The **AddConstraint** method adds a **PrecedenceConstraint** object to a **Step2** object.

Applies To

PrecedenceConstraints Collection
--

Syntax

Constraints.**AddConstraint** *StepName*

Part	Description
<i>Constraints</i>	Expression that evaluates to a PrecedenceConstraints collection
<i>StepName</i>	Name of the step referenced by the precedence constraint

Remarks

StepName is not the name of the **Step2** object to which the **PrecedenceConstraint** object is being added. It is the name of the step whose status or result determines when the **Step2** object can be released for execution.

The **AddConstraint** method does not return a reference to the **PrecedenceConstraint** object it creates. You may need to set **PrecedenceConstraint** properties, **PrecedenceBasis**, or **Value** to other than their defaults. In this case, you can create the **PrecedenceConstraint** object with the **New** method, set the properties, and then add it to the collection with the **Add** method. Or, you can access the **PrecedenceConstraint** object from the collection after creating it with **AddConstraint**.

Prototype (C/C++)

```
HRESULT AddConstraint(BSTR StepName);
```

Example

This Microsoft® Visual Basic® sample accesses a **PrecedenceConstraint** object from the collection after creating it with **AddConstraint**:

```
objStep.PrecedenceConstraints.AddConstraint "StartStep"  
Set objConstraint = objStep.PrecedenceConstraints( "StartStep" )
```

See Also

[Add Method](#)

[Creating DTS Package Workflow and Tasks](#)

[PrecedenceBasis Property](#)

[PrecedenceConstraint Object](#)

[New \(Name\) Method](#)

[Value Property](#)

DTS Programming

AddGlobalVariable Method

The **AddGlobalVariable** method adds a **GlobalVariable** object to the collection by name.

Applies To

[GlobalVariables Collection](#)

Syntax

GlobalVariables.**AddGlobalVariable** *Name*, *Value*

Part	Description
<i>GlobalVariables</i>	Expression that evaluates to a GlobalVariables collection
<i>Name</i>	String that is the name of the global variable
<i>Value</i>	Variant that is the initial value of the global variable

Remarks

The **AddGlobalVariable** method does not return a reference to the **GlobalVariable** object it creates. You also can create the **GlobalVariable** object with the **New** method and then add it to the collection with the **Add** method.

Prototype (C/C++)

```
HRESULT AddGlobalVariable(  
    BSTR Name,  
    VARIANT Value);
```

See Also

[Add Method](#)

[Adding DTS Lookups and Global Variables](#)

[GlobalVariable Object](#)

[New \(Name\) Method](#)

DTS Programming

AddLookup Method

The **AddLookup** method adds a parameterized query to the **Lookups** collection.

Applies To

[Lookups Collection](#)

Syntax

object.**AddLookup** *Name*, *Query*, *ConnectionID*, [*MaxCacheRows*]

Part	Description
<i>object</i>	Expression that evaluates to a Lookups collection
<i>Name</i>	String that is the lookup query name
<i>Query</i>	String that is the SQL query
<i>ConnectionID</i>	Long that is the connection ID
<i>MaxCacheRows</i>	Long that is the maximum number of rows to cache for reuse

Remarks

Typically, a **Lookup** object is referenced in a Microsoft® ActiveX® script. The *Query* is executed against the data source specified by the *ConnectionID*. The first row of the returned rowset is used. *MaxCacheRows* specifies the number of rows cached locally so that subsequent lookups that target that row do not need to reissue the *query* against the data source.

The **AddLookup** method does not return a reference to the **Lookup** object it creates. You also can create the **Lookup** object with the **New** method, set its properties, and then add it to the collection with the **Add** method.

Prototype (C/C++)

HRESULT AddLookup(BSTR Name,

```
BSTR Query,  
long ConnectionID,  
long MaxCacheRows);
```

Example

This Microsoft Visual Basic® sample creates and adds a **Lookup** object to the collection:

```
objDataPump.Lookups.AddLookup "ExpandState", _  
    "SELECT StateName FROM StateInfo WHERE POCode = ?", _  
    2, 50
```

See Also

[Add Method](#)

[Adding DTS Lookups and Global Variables](#)

[Lookup Object](#)

[New \(Name\) Method](#)

DTS Programming

AddObjectForTransfer Method

The **AddObjectForTransfer** method adds an object to the list of Microsoft® SQL Server™ objects to be transferred.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**AddObjectForTransfer** *ObjectName*, *OwnerName*, *Type*

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>ObjectName</i>	Name of the object to transfer
<i>OwnerName</i>	Object owner name
<i>Type</i>	Type of object to transfer, from the DTSSQLObjectType constants

Prototype (C/C++)

```
HRESULT AddObjectForTransfer(  
    BSTR ObjectName,  
    BSTR OwnerName,  
    DTSSQLObjectType Type);
```

See Also

[DTSSQLObjectType](#)

[GetObjectForTransfer Method](#)

[ResetObjectsList Method](#)

DTS Programming

BeginAcquireMultipleConnections Method

The **BeginAcquireMultipleConnections** method acquires ownership of the **Connections** collection's synchronization object. This is to serialize the acquisition of multiple connections by a Data Transformation Services (DTS) task.

Applies To

Connections Collection
--

Syntax

Connections.**BeginAcquireMultipleConnections**

Part	Description
<i>Connections</i>	Expression that evaluates to a Connections collection

Remarks

When implementing a custom task that needs to acquire more than one connection, do the following to avoid deadlocks:

1. Call **BeginAcquireMultipleConnections**.
2. For each *connection* to be acquired:
 - Verify that *connection.InUse* is FALSE.
 - Call *connection.AcquireConnection*.
3. Call **EndAcquireMultipleConnections**.

Prototype (C/C++)

HRESULT BeginAcquireMultipleConnections());

See Also

[AcquireConnection Method](#)

[Connection Object](#)

[Creating DTS Package Objects and Connections](#)

[EndAcquireMultipleConnections Method](#)

[InUse Property](#)

DTS Programming

CancelExecution Method

The **CancelExecution** method cancels execution of the task.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**CancelExecution**

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Remarks

CancelExecution must be called on a thread different from the execution thread.

Prototype (C/C++)

```
HRESULT CancelExecution();
```

DTS Programming

CheckSyntax Method

The **CheckSyntax** method evaluates the script specified by the **ActiveXScript** property for correct syntax.

Applies To

[ActiveScriptTask Object](#)

Syntax

object.**CheckSyntax()**

Part	Description
<i>Object</i>	Expression that evaluates to an object in the Applies To list

Prototype (C/C++)

HRESULT CheckSyntax();

See Also

[ActiveXScript Property](#)

[Adding DTS ActiveX Scripts](#)

[ScriptLanguage Property](#)

DTS Programming

CreateCustomToolTip Method

The **CreateCustomToolTip** method creates a ToolTip window for a Data Transformation Services (DTS) custom task.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

```
Sub CustomTaskUI_CreateCustomToolTip(  
    ByVal hwndParent As Long,  
    ByVal x As Long,  
    ByVal y As Long,  
    plTipWindow As Long )
```

Part	Description
<i>CustomTaskUI</i>	Expression that evaluates to a CustomTaskUI interface
<i>HwndParent</i>	Handle of window in which ToolTip is to be displayed
<i>X</i>	Horizontal coordinate of ToolTip window
<i>Y</i>	Vertical coordinate of ToolTip window
<i>PlTipWindow</i>	Handle of ToolTip window generated by method

Remarks

The **CreateCustomToolTip** method must be implemented by a custom task that is to show custom ToolTips in its user interface. It is called by DTS at the time the ToolTip is to be displayed. **CreateCustomToolTip** draws the ToolTip window and returns its window handle to DTS in the *plTipWindow* parameter.

Prototype (C/C++)

```
HRESULT CreateCustomToolTip(
```

```
long hwndParent,  
long x,  
long y,  
long *plTipWindow);
```

See Also

[Edit Method](#)

[Delete Method](#)

[New \(CustomTaskUI\) Method](#)

DTS Programming

Delete Method

The **Delete** method of the **CustomTaskUI** interface must be supplied by the implementer of a Data Transformation Services (DTS) custom task that has a user interface. It is called by DTS when an instance of the custom task is to be deleted.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

```
Sub CustomTaskUI_Delete(ByVal hwndParent As Long )
```

Part	Description
<i>hwndParent</i>	Handle to the window to be the parent of the custom task user interface

Remarks

The window that is parent to the custom task's user interface window typically would be the DTS Designer design sheet or the Microsoft® Visual Studio® development environment desktop.

Prototype (C/C++)

```
HRESULT Delete(long hwndParent);
```

See Also

[CreateCustomToolTip Method](#)

[Edit Method](#)

[New \(CustomTaskUI\) Method](#)

DTS Programming

Edit Method

The **Edit** method of the **CustomTaskUI** interface must be supplied by the implementer of a Data Transformation Services (DTS) custom task that has a user interface. It is called by DTS when a user wants to edit properties of the **CustomTask** object.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

```
Sub CustomTaskUI_Edit(ByVal hwndParent As Long )
```

Part	Description
<i>hwndParent</i>	Handle to the window to be the parent of the custom task user interface

Remarks

The **Edit** method should display the user interface for the custom task, typically a dialog box or property sheet.

The window that is parent to the custom task's user interface window typically is the DTS Designer design sheet or the Microsoft® Visual Studio® development environment desktop.

Prototype (C/C++)

```
HRESULT Edit(long hwndParent);
```

See Also

[CreateCustomToolTip Method](#)

[Delete Method](#)

[New \(CustomTaskUI\) Method](#)

DTS Programming

EndAcquireMultipleConnections Method

The **EndAcquireMultipleConnections** method releases ownership of the **Connections** collection's synchronization object. This is after serializing the acquisition of multiple connections by a Data Transformation Services (DTS) task.

Applies To

Connections Collection
--

Syntax

Connections.**EndAcquireMultipleConnections**

Part	Description
<i>Connections</i>	Expression that evaluates to a Connections collection

Remarks

When implementing a custom task that needs to acquire more than one connection, do the following to avoid deadlocks:

1. Call **BeginAcquireMultipleConnections**.
2. For each *connection* to be acquired:
 - Verify that *connection.InUse* is FALSE.
 - Call *connection.AcquireConnection*.
3. Call **EndAcquireMultipleConnections**.

Prototype (C/C++)

HRESULT EndAcquireMultipleConnections();

See Also

[AcquireConnection Method](#)

[BeginAcquireMultipleConnections Method](#)

[Connection Object](#)

[Creating DTS Package Objects and Connections](#)

[InUse Property](#)

DTS Programming

EnumPackageInfos Method

The **EnumPackageInfos** method returns a **PackageInfos** collection containing information about all the packages stored in Microsoft® SQL Server™ 2000 Meta Data Services or in SQL Server storage that satisfy the criteria of the input parameters.

Applies To

PackageRepository Object	PackageSQLServer Object
--	---

Syntax

```
Set collection = object.EnumPackageInfos(  
    PackageName,  
    ReturnLatest,  
    PackageID )
```

Part	Description
<i>collection</i>	Expression that evaluates to a PackageInfos collection
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>PackageName</i>	A string specifying the package name to which the collection is to be limited
<i>ReturnLatest</i>	A Boolean specifying whether only the latest version of each package is to be included in the collection
<i>PackageID</i>	A globally unique identifier (GUID) string specifying the package ID to which the collection is to be limited

Remarks

All parameters are required. To return information on all packages in Meta Data Services or in SQL Server storage, code the empty string "" for *PackageName* and *PackageID*, and FALSE for *ReturnLatest*.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where the X's are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

Prototype (C/C++)

```
HRESULT EnumPackageInfos(  
    BSTR PackageName,  
    VARIANT_BOOL ReturnLatest,  
    BSTR PackageID,  
    IDTSPackageInfos **pRetVal );
```

See Also

[PackageID Property](#)

[PackageInfos Collection](#)

[PackageName Property](#)

[Retrieving DTS System, Package, and Log Data](#)

DTS Programming

EnumPackageLineages Method

The **EnumPackageLineages** method returns a **PackageLineages** collection containing data from the package lineage records stored in Microsoft® SQL Server™ 2000 Meta Data Services that satisfy the criteria of the input parameters.

Applies To

[PackageRepository Object](#)

Syntax

```
Set collection = object.EnumPackageLineages(  
    PackageVersionID,  
    LineageFullID,  
    LineageShortID )
```

Part	Description
<i>collection</i>	Expression that evaluates to a PackageLineages collection
<i>object</i>	Expression that evaluates to a PackageRepository object
<i>PackageVersionID</i>	A globally unique identifier (GUID) string specifying the version ID of the package version to which the collection is to be limited
<i>LineageFullID</i>	A GUID string specifying the lineage full ID to which the collection is to be limited
<i>LineageShortID</i>	A long specifying the lineage short ID to which the collection is to be limited

Remarks

A package lineage record is written each time a package stored in Meta Data

Services is executed, if the package **LineageOptions** property specifies this to be done. Each such record has a unique lineage full ID and lineage short ID.

A package ID or the empty string cannot be specified for *PackageVersionID*. An error will be raised if there is no package lineage record in Meta Data Services with the specified version ID.

All parameters are required. To return all package lineage records for a particular package version, code the empty string "" for *LineageFullID* and 0 for *LineageShortID*. To return all package lineage records in Meta Data Services, iterate through the Meta Data Services packages with the **EnumPackageInfos** method, and then call **EnumPackageLineages** for each package version.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X's are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded.

Prototype (C/C++)

```
HRESULT EnumPackageLineages(  
    BSTR PackageVersionID,  
    BSTR LineageFullID,  
    long LineageShortID,  
    IDTSPackageLineages **pRetVal );
```

See Also

[EnumPackageInfos Method](#)

[LineageFullID Property](#)

[LineageOptions Property](#)

[LineageShortID Property](#)

[PackageID Property](#)

[PackageLineages Collection](#)

[RemovePackageLineages Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[VersionID Property](#)

DTS Programming

EnumPackageLogRecords Method

The **EnumPackageLogRecords** method returns a **PackageLogRecords** collection containing data from the package log records in the database. These package log records satisfy the criteria of the input parameters.

Applies To

[PackageSQLServer Object](#)

Syntax

```
Set collection = object.EnumPackageLogRecords(  
    PackageName,  
    ReturnLatest,  
    PackageID,  
    VersionID,  
    LineageFullID )
```

Part	Description
<i>collection</i>	Expression that evaluates to a PackageLogRecords collection.
<i>object</i>	Expression that evaluates to a PackageSQLServer object.
<i>PackageName</i>	A string specifying the package name to which log records in the collection are to be limited.
<i>ReturnLatest</i>	A Boolean specifying whether log records for only the latest version of each package are to be included in the collection.
<i>PackageID</i>	A globally unique identifier (GUID) string specifying the package ID to which log records in the collection are to be limited.
<i>VersionID</i>	A GUID string specifying the version ID of the package version to which log records in the collection are to be limited.
<i>LineageFullID</i>	A GUID string specifying the lineage full ID to which the collection is to be limited.

Remarks

A package log record is written to the **msdb** database on the server specified by the package **LogServerName** property each time a Data Transformation Services (DTS) package is executed, if the package **LogToSQLServer** property has been set. For Microsoft® SQL Server™ 2000 Meta Data Services packages, the log record's lineage full ID links to the package lineage record. For other packages, a GUID is created.

All parameters are required. To prevent a parameter from participating in the filter process, code the empty string "" for *PackageName*, *PackageID*, *VersionID* and *LineageFullID*, and *FALSE* for *ReturnLatest*. Coding all parameters this way will cause all package log records in the database to be returned, which may result in an unmanageably large collection.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X's are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded.

Prototype (C/C++)

```
HRESULT EnumPackageLogRecords(  
    BSTR PackageName,  
    VARIANT_BOOL ReturnLatest,  
    BSTR PackageID,  
    BSTR VersionID,  
    BSTR LineageFullID,  
    IDTSPackageLogRecords **pRetVal );
```

See Also

[LineageFullID Property](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageID Property](#)

[PackageLineages Collection](#)

[PackageLogRecords Collection](#)

[PackageName Property](#)

[RemovePackageLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[VersionID Property](#)

DTS Programming

EnumStepLineages Method

The **EnumStepLineages** method returns a **StepLineages** collection containing information about all the step lineage records stored in Microsoft® SQL Server™ 2000 Meta Data Services that satisfy the criteria of the input parameter.

Applies To

[PackageRepository Object](#)

Syntax

Set *collection* = *object*.**EnumStepLineages**(*LineageFullID*)

Part	Description
<i>collection</i>	Expression that evaluates to a StepLineages collection.
<i>object</i>	Expression that evaluates to a PackageRepository object.
<i>LineageFullID</i>	A globally unique identifier (GUID) string specifying the lineage full ID to which the collection is to be limited.

Remarks

A step lineage record is written each time a step in a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this to be done. A unique lineage full ID is created each time the package is executed (not for each step).

The parameter is required. You cannot code the empty string "" for *LineageFullID*. If there is no lineage record in Meta Data Services with the specified *LineageFullID*, an error occurs. To return all step lineage records in Meta Data Services, iterate through the Meta Data Services packages with the **EnumPackageInfos** method, then call **EnumPackageLineages** for each package version, and finally call **EnumStepLineages** for each package lineage. There may be many step lineage records in Meta Data Services.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X's are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

Prototype (C/C++)

```
HRESULT EnumPackageInfos(  
    BSTR LineageFullID,  
    IDTSStepLineages **pRetVal);
```

See Also

[EnumPackageInfos Method](#)

[EnumPackageLineages Method](#)

[LineageFullID Property](#)

[LineageOptions Property](#)

[RemovePackageLineages Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[StepLineages Collection](#)

DTS Programming

EnumStepLogRecords Method

The **EnumStepLogRecords** method returns a **StepLogRecords** collection containing data from the step log records in the database that satisfy the criteria of the input parameters.

Applies To

[PackageSQLServer Object](#)

Syntax

```
Set collection = object.EnumStepLogRecords(  
    LineageFullID,  
    StepExecutionID )
```

Part	Description
<i>collection</i>	Expression that evaluates to a StepLogRecords collection.
<i>object</i>	Expression that evaluates to a PackageSQLServer object.
<i>LineageFullID</i>	A globally unique identifier (GUID) string specifying the lineage full ID to which the collection is to be limited.
<i>StepExecutionID</i>	A variant specifying the step execution ID to which the collection is to be limited.

Remarks

A step log record is written to the Microsoft® SQL Server™ **msdb** database on the server specified by the package **LogServerName** property each time a step in a Data Transformation Services (DTS) package is executed, if the package **LogToSQLServer** property has been set. For SQL Server 2000 Meta Data Services packages, the log record's lineage full ID links to the package lineage record. For other packages, a GUID is created.

Both parameters are required. To prevent a parameter from participating in the filter process, code the empty string "" for *LineageFullID*, and NULL for *StepExecutionID*. Coding all parameters this way will cause all step log records in the database to be returned, which may result in an unmanageably large collection.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X's are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

Prototype (C/C++)

```
HRESULT EnumStepLogRecords(  
    BSTR LineageFullID,  
    VARIANT StepExecutionID,  
    IDTSStepLogRecords **pRetVal );
```

See Also

[LineageFullID Property](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageLineages Collection](#)

[RemoveStepLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[StepExecutionID Property](#)

[StepLogRecords Collection](#)

DTS Programming

EnumTaskLogRecords Method

The **EnumTaskLogRecords** method returns a **TaskLogRecords** collection containing data from the task log records in the database that satisfy the criteria of the input parameters.

Applies To

[PackageSQLServer Object](#)

Syntax

```
Set collection = object.EnumTaskLogRecords(  
    StepExecutionID,  
    SequenceID )
```

Part	Description
<i>collection</i>	Expression that evaluates to a TaskLogRecords collection
<i>object</i>	Expression that evaluates to a PackageSQLServer object
<i>StepExecutionID</i>	A variant specifying the step execution ID to which the collection is to be limited
<i>SequenceID</i>	A long specifying the sequence number of the desired task log record

Remarks

Task log records are not automatically written by the task classes supplied with Microsoft® SQL Server™ 2000, but the **PackageLog** interface is available so that a custom task or the script of an **ActiveScriptTask** object can write them. They are written to the **msdb** database in the instance of SQL Server specified by the package **LogServerName** property each time a task in a Data Transformation Services (DTS) package that has been implemented to write them is executed, if the package **LogToSQLServer** property has been set.

SequenceID is a unique sequence number for each task log record.

Both parameters are required. To prevent a parameter from participating in the filter process, code NULL for *StepExecutionID* and 0 for *SequenceID*. Coding all parameters this way will cause all task log records in the database to be returned, which could result in an unmanageably large collection.

Prototype (C/C++)

```
HRESULT EnumTaskLogRecords(  
    VARIANT StepExecutionID,  
    long SequenceID,  
    IDTSTaskLogRecords **pRetVal );
```

See Also

[ActiveScriptTask Object](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageLog Object](#)

[RemoveTaskLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[SequenceID Property](#)

[StepExecutionID Property](#)

[TaskLogRecords Collection](#)

DTS Programming

Execute Method

The **Execute** method executes a Data Transformation Services (DTS) task object.

Applies To

ActiveScriptTask Object	DynamicPropertiesTask Object
BulkInsertTask Object	ExecutePackageTask Object
CreateProcessTask Object	ExecuteSQLTask Object
CreateProcessTask2 Object	ExecuteSQLTask2 Object
CustomTask Object	ParallelDataPumpTask Object
DataDrivenQueryTask Object	SendMailTask Object
DataPumpTask Object	Task Object
DTSFTPTask Object	TransferObjectsTask Object
DTSMessagesQueueTask Object	

Syntax

object.**Execute** *pPackage*, *pPackageEvents*, *pPackageLog*, *pTaskResult*

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>pPackage</i>	Reference to the Package2 object
<i>pPackageEvents</i>	Reference to the events of the Package2 object
<i>pPackageLog</i>	Reference to a PackageLog object
<i>pTaskResult</i>	Return code from the DTSTaskExecResult constants

Remarks

An application does not have to call the **Execute** method for each step or task. Instead the **Package2** object calls the **Execute** method to launch each step or

task after the application calls **Package2.Execute**.

A reference to the **Package2** object is passed as a parameter to allow access to the objects in the hierarchy for the package. But *pPackage* and all objects in its hierarchy must not be saved or referenced after the **Execute** method returns. Check *pPackageEvents* and *pPackageLog* for NULL/Nothing before using them.

Prototype (C/C++)

```
HRESULT Execute(  
    IDispatch.*pPackage,  
    IDispatch.*pPackageEvents,  
    IDispatch.*pPackageLog,  
    long.*pTaskResult);
```

See Also

[DTSTaskExecResult](#)

[Execute \(Package\) Method](#)

DTS Programming

Execute (Package) Method

The **Execute** method executes a Data Transformation Services (DTS) package or step.

Applies To

Package Object	Step Object
Package2 Object	

Syntax

object.Execute

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Remarks

When used with the **Package2** and **Step2** objects, the **Execute** method takes no parameters. An application need not call the **Execute** method for each step or task. Instead the **Package2** object calls the **Execute** method to launch each step or task after the application calls **Package2.Execute**.

Prototype (C/C++)

```
HRESULT Execute( );
```

See Also

[Execute Method](#)

[Managing DTS Package Programs](#)

DTS Programming

GetDayLongName Method

The **GetDayLongName** method returns the long (full) name for the specified day of the week.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

[*string* =] *object*.**GetDayLongName**(*daynumber*)

Part	Description
<i>string</i>	Long (full) name of the specified day of week
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>daynumber</i>	Number of the day of the week

Remarks

The valid day numbers are from 1 through 7.

Prototype (C/C++)

```
HRESULT GetDayLongName(  
    long DayNumber,  
    BSTR* pRetVal);
```

See Also

[Adding DTS Transformations](#)

[Day?LongName Property](#)

[GetDayShortName Method](#)

[SetDayLongName Method](#)

DTS Programming

GetDayShortName Method

The **GetDayShortName** method returns the short name (3-character abbreviation) for the specified day of the week.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

[*string* =] *object*.**GetDayShortName**(*daynumber*)

Part	Description
<i>string</i>	Short name (3-character abbreviation) of the specified day of week
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>daynumber</i>	Number of the day of the week

Remarks

The valid day numbers are from 1 through 7.

Prototype (C/C++)

```
HRESULT GetDayShortName(  
    long DayNumber,  
    BSTR* pRetVal);
```

See Also

[Adding DTS Transformations](#)

[Day?ShortName Property](#)

[GetDayLongName Method](#)

[SetDayShortName Method](#)

DTS Programming

GetDefaultProfileName Method

The **GetDefaultProfileName** method returns the default profile name.

Applies To

[SendMailTask Object](#)

Syntax

[*string* =] *object*.**GetDefaultProfileName**

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>string</i>	String to receive the default profile name

Prototype (C/C++)

```
HRESULT GetDefaultProfileName(BSTR *DefaultProfile);
```

See Also

[Profile Property](#)

DTS Programming

GetDTSVersionInfo Method

The **GetDTSVersionInfo** method retrieves version information for Data Transformation Services (DTS).

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**GetDTSVersionInfo** [*VersionMajor*], [*VersionMinor*], _
[*VersionBuild*], [*VersionComments*])

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>VersionMajor</i>	Major version number of Microsoft® SQL Server™
<i>VersionMinor</i>	Incremental version number of SQL Server
<i>VersionBuild</i>	SQL Server build number
<i>VersionComments</i>	SQL Server product name including version number

Prototype (C/C++)

```
HRESULT GetDTSVersionInfo(long *VersionMajor,  
long *VersionMinor,  
long *VersionBuild,  
BSTR *VersionComments);
```

DTS Programming

GetExecutionErrorInfo Method

The **GetExecutionErrorInfo** method returns details about step execution failure.

Applies To

Step Object	Step2 Object
-----------------------------	------------------------------

Syntax

object.**GetExecutionErrorInfo** *pErrorCode*, [*pbstrSource*], [*pbstrDescription*],
—
[*pbstrHelpFile*], [*pHelpContext*], [*pbstrIDofInterfaceWithError*]

Part	Description
<i>Step</i>	Expression that evaluates to an object in the Applies To list
<i>pErrorCode</i>	Error code of the failure
<i>pbstrSource</i>	Source of error
<i>pbstrDescription</i>	Description property of the error
<i>pbstrHelpFile</i>	Help file name
<i>pHelpContext</i>	Help context ID
<i>pbstrIDofInterfaceWithError</i>	ID of the interface returning the error

Remarks

Step2 object failure is separate from **Package2** object failure. Therefore, error information for each step is unavailable from the COM **IErrorInfo** object or the Microsoft® Visual Basic® **Err** object.

When a step fails, the package will not fail unless the package **FailOnError** property or the step **FailPackageOnError** property is set. The error that is

raised will only specify the step that failed, not the error that occurred. Use **GetExecutionErrorInfo** on the step that failed to get information about the error.

If you do not set either **FailOnError** or **FailPackageOnError** for any step, you can use **GetExecutionErrorInfo** on all steps in the **Steps** collection when package execution completes. **GetExecutionErrorInfo** returns meaningful information only when the step **ExecutionStatus** is **DTSSStepExecStat_Completed** and **ExecutionResult** is **DTSSStepExecResult_Failure**.

Prototype (C/C++)

```
HRESULT GetExecutionErrorInfo(  
    long *pErrorCode,  
    BSTR *pbstrSource,  
    BSTR *pbstrDescription,  
    BSTR *pbstrHelpFile,  
    long *pHelpContext,  
    BSTR *pbstrIDofInterfaceWithError);
```

See Also

[ExecutionResult Property](#)

[ExecutionStatus Property](#)

[FailOnError Property](#)

[FailPackageOnError Property](#)

[Handling DTS Events and Errors](#)

[Steps Collection](#)

DTS Programming

GetExpandedProcessCommandLine Method

The **GetExpandedProcessCommandLine** method returns the process command line parameter string with the environment variables expanded.

Applies To

[CreateProcessTask2 Object](#)

Syntax

[*string* =] *object*.**GetExpandedProcessCommandLine**

Part	Description
<i>object</i>	Expression that evaluates to a CreateProcessTask2 object
<i>string</i>	String to receive the expanded command line

Prototype (C/C++)

HRESULT GetExpandedProcessCommandLine(BSTR* pRetVal)

See Also

[ProcessCommandLine Property](#)

DTS Programming

GetLastExecutionLineage Method

The **GetLastExecutionLineage** method retrieves lineage information written to Microsoft® SQL Server™ 2000 Meta Data Services during the last execution of the Data Transformation Services (DTS) package.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

[*string* =] *object*.**GetLastExecutionLineage**([*LineageShort*])

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>LineageShort</i>	Package version checksum value
<i>string</i>	String to receive the lineage data

Prototype (C/C++)

```
HRESULT GetLastExecutionLineage(  
long *LineageShort,  
BSTR *pRetVal);
```

See Also

[EnumPackageLineages Method](#)

[EnumStepLineages Method](#)

[LineageShortID Property](#)

[PackageLineages Collection](#)

[Recording Data Lineage in DTS](#)

[Retrieving DTS System, Package, and Log Data](#)
[StepLineages Collection](#)

DTS Programming

GetMonthLongName Method

The **GetMonthLongName** method returns the long (full) name for the specified month.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

[*string* =] *object*.**GetMonthLongName**(*monthnumber*)

Part	Description
<i>string</i>	Long (full) name of the specified month
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>monthnumber</i>	Number of the month

Remarks

The valid month numbers are from 1 through 12.

Prototype (C/C++)

```
HRESULT GetMonthLongName(  
long MonthNumber,  
BSTR* pRetVal);
```

See Also

[Adding DTS Transformations](#)

[GetMonthShortName Method](#)

[Month??LongName Property](#)

[SetMonthLongName Method](#)

DTS Programming

GetMonthShortName Method

The **GetMonthShortName** method returns the short name (3-character abbreviation) for the specified month.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

[*string* =] *object*.**GetMonthShortName**(*monthnumber*)

Part	Description
<i>string</i>	Short name (3-character abbreviation) of the specified month
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>monthnumber</i>	Number of the month

Remarks

The valid month numbers are from 1 through 12.

Prototype (C/C++)

```
HRESULT GetMonthShortName(  
long MonthNumber,  
BSTR* pRetVal);
```

See Also

[Adding DTS Transformations](#)

[GetMonthLongName Method](#)

[Month.ShortName Property](#)

[SetMonthShortName Method](#)

DTS Programming

GetObjectForTransfer Method

The **GetObjectForTransfer** method iterates objects on the list of objects to be transferred.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.**GetObjectForTransfer** *Index*, *ObjectName*, *OwnerName*, *Type*

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>Index</i>	Index into the list of objects to be transferred.
<i>ObjectName</i>	Object name.
<i>OwnerName</i>	Object owner name.
<i>Type</i>	Object type, from the DTSSQLObjectType constants.

Remarks

The index of the first object in the list of objects to be transferred is 0.

Prototype (C/C++)

```
HRESULT GetObjectForTransfer(LONG Index,  
BSTR *ObjectName,  
BSTR *OwnerName,  
DTSSQLObjectType *Type);
```

See Also

[AddObjectForTransfer Method](#)

[DTSSQLObjectType](#)

[ResetObjectsList Method](#)

DTS Programming

GetPackageRepository Method

The **GetPackageRepository** method returns a **PackageRepository** object for the server and database specified by the input parameters.

Applies To

[Application Object](#)

Syntax

```
Set object = appobject.GetPackageRepository(  
    ServerName,  
    DatabaseName,  
    UserName ,  
    Password ,  
    ConnectionFlags )
```

Part	Description
<i>object</i>	Expression that evaluates to a PackageRepository object
<i>appobject</i>	Expression that evaluates to an Application object
<i>ServerName</i>	A string specifying the name of the server on which the Microsoft® SQL Server™ 2000 Meta Data Services instance is hosted
<i>DatabaseName</i>	A string specifying the name of the database in which the instance of Meta Data Services is located
<i>UserName</i>	A string specifying the logon name used to access the server specified by <i>ServerName</i>
<i>Password</i>	A string specifying the password used to access the server specified by <i>ServerName</i>
<i>ConnectionFlags</i>	A value from the DTSRepositoryStorageFlags constants that specifies the type of user authentication used to access the server specified by <i>ServerName</i>

Remarks

The **PackageRepository** object provides access to the Data Transformation Services (DTS) packages and lineage data stored on the associated instance of Meta Data Services.

UserName and *Password* need be specified (nonempty string) only if *ConnectionFlags* has the value `DTSReposFlag_Default`, which implies database authentication.

Prototype (C/C++)

```
HRESULT GetPackageRepository(  
    BSTR ServerName,  
    BSTR DatabaseName,  
    BSTR Username,  
    BSTR Password,  
    DTSRepositoryStorageFlags ConnectionFlags,  
    IDTSPackageRepository **pRetVal );
```

See Also

[DTSRepositoryStorageFlags](#)

[GetPackageSQLServer Method](#)

[PackageRepository Object](#)

[Retrieving DTS System, Package, and Log Data](#)

DTS Programming

GetPackageSQLServer Method

The **GetPackageSQLServer** method returns a **PackageSQLServer** object for the server specified by the input parameters.

Applies To

[Application Object](#)

Syntax

```
Set object = appobject.GetPackageSQLServer(  
    ServerName,  
    UserName ,  
    Password ,  
    ConnectionFlags )
```

Part	Description
<i>object</i>	Expression that evaluates to a PackageSQLServer object.
<i>appobject</i>	Expression that evaluates to an Application object.
<i>ServerName</i>	A string specifying the name of the server to which to connect.
<i>UserName</i>	A string specifying the logon name used to access the server specified by <i>ServerName</i> .
<i>Password</i>	A string specifying the password used to access the server specified by <i>ServerName</i> .
<i>ConnectionFlags</i>	A value from the DTSSQLServerStorageFlags constants that specifies the type of user authentication used to access the server specified by <i>ServerName</i> .

Remarks

The **PackageSQLServer** object provides access to the Data Transformation

Services (DTS) packages and log data stored on the associated server, which must be running an instance of Microsoft® SQL Server™.

UserName and *Password* need be specified (nonempty string) only if *ConnectionFlags* has the value *DTSSQLStgFlag_Default*, which implies database authentication.

Prototype (C/C++)

```
HRESULT GetPackageSQLServer(  
    BSTR ServerName,  
    BSTR Username,  
    BSTR Password,  
    DTSSQLServerStorageFlags ConnectionFlags,  
    IDTSPackageSQLServer **pRetVal );
```

See Also

[DTSSQLServerStorageFlags](#)

[GetPackageRepository Method](#)

[PackageSQLServer Object](#)

[Retrieving DTS System, Package, and Log Data](#)

DTS Programming

GetPropertiesForObject Method

The **GetPropertiesForObject** method returns a Data Transformation Services (DTS) **Properties** collection from an object implementing a custom task.

Applies To

PropertiesProvider Object

Syntax

[Set *colProperties* =] *object*.**GetPropertiesForObject**(*pObject*)

Part	Description
<i>object</i>	Expression that evaluates to a PropertiesProvider object
<i>pObject</i>	Expression that evaluates to an object for which a Properties collection is to be retrieved
<i>colProperties</i>	Object variable of type Properties

Remarks

Use **GetPropertiesForObject** and the **Properties** collection it returns to determine, without causing an error, if an object supports a property.

Prototype (C/C++)

```
HRESULT GetPropertiesForObject(  
    IDispatch *pObject,  
    IDTSProperties **pRetVal);
```

See Also

[Properties Collection](#)

DTS Programming

GetSavedPackageInfos Method

The **GetSavedPackageInfos** method retrieves a list of versions in this storage location.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

[Set *colSavedInfos* =] *Package*.**GetSavedPackageInfos**(*UNCFile*)

Part	Description
<i>Package</i>	Expression that evaluates to an object in the Applies To list
<i>UNCFile</i>	Data Transformation Services (DTS) package storage file from which a list of versions is to be retrieved
<i>colSavedInfos</i>	Object variable of type SavedPackageInfos

Prototype (C/C++)

```
HRESULT GetSavedPackageInfos(  
    BSTR UNCFile,  
    IDTSSavedPackageInfos **pRetVal);
```

See Also

[Retrieving DTS System, Package, and Log Data](#)

[SavedPackageInfos Collection](#)

DTS Programming

GetUIInfo Method

The **GetUIInfo** method returns top-level user interface information for a Data Transformation Services (DTS) custom task user interface element to its caller. It must be supplied by the implementer of a custom task that has a user interface.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

```
Sub CustomTaskUI_GetUIInfo( pbstrToolTip As String, pbstrDescription As String, _  
    plVersion As Long, pFlags As DTSCustomTaskUIFlags)
```

Part	Description
<i>pbstrToolTip</i>	ToolTip text
<i>PbstrDescription</i>	Description of user interface element
<i>PlVersion</i>	Reserved for future use
<i>PFlags</i>	Value from the DTSCustomTaskUIFlags constants that indicates whether the user interface element has a custom ToolTip

Prototype (C/C++)

```
HRESULT GetUIInfo(  
    BSTR *pbstrToolTip,  
    BSTR *pbstrDescription,  
    long *plVersion,  
    DTSCustomTaskUIFlags *pFlags);
```

See Also

[CreateCustomToolTip Method](#)

[DTSCustomTaskUIFlags](#)

DTS Programming

Help Method

The **Help** method of the **CustomTaskUI** interface must be supplied by the implementer of a Data Transformation Services (DTS) custom task that has a user interface. It is called by DTS when Help has been invoked for the custom task.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

Sub CustomTaskUI_Help(ByVal *hwndParent* As Long)

Part	Description
<i>hwndParent</i>	Handle to the window to be the parent of the custom task user interface

Remarks

The window that is parent to the custom task's user interface window typically would be the DTS Designer design sheet or the Microsoft® Visual Studio® development environment desktop.

Prototype (C/C++)

HRESULT Help(long hwndParent);

See Also

[CustomTask Object](#)

[GetUIInfo Method](#)

DTS Programming

Initialize Method

The **Initialize** method of the **CustomTaskUI** interface must be supplied by the implementer of a Data Transformation Services (DTS) custom task that has a user interface. It is called by DTS whenever the custom task is opened in the design environment.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

```
Sub CustomTaskUI_Initialize(ByVal pTask As Task )
```

Part	Description
<i>pTask</i>	Task object for the custom task

Remarks

Typically, the **Initialize** method provides initial values for user interface elements.

Prototype (C/C++)

```
HRESULT Initialize(IDTSTask *pTask);
```

See Also

[Edit Method](#)

[GetUIInfo Method](#)

[New \(CustomTaskUI\) Method](#)

DTS Programming

InitializeMAPI Method

The **InitializeMAPI** method initializes the MAPI provider.

Applies To

[SendMailTask Object](#)

Syntax

object.**InitializeMAPI**

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object

Prototype (C/C++)

HRESULT InitializeMAPI();

See Also

[UnitializeMAPI Method](#)

DTS Programming

Insert Method

The **Insert** method inserts an object in an ordinal position in a collection.

Applies To

Columns Collection	Steps Collection
Connections Collection	Tasks Collection
GlobalVariables Collection	Transformations Collection
Lookups Collection	TransformationSets Collection
PrecedenceConstraints Collection	

Syntax

collection.**Insert** *Position*, *Object*

Part	Description
<i>collection</i>	Expression that evaluates to an collection in the Applies To list
<i>Position</i>	Ordinal position of the object to insert
<i>Object</i>	Object to be inserted

Remarks

The required type of *Object* depends on the collection to which it is being inserted.

Collection	Object
Columns	Column Object
Connections	Connection Object Connection2 Object
GlobalVariables	GlobalVariable Object
Lookups	Lookup Object
PrecedenceConstraints	PrecedenceConstraint Object

Steps	Step Object Step2 Object
Tasks	Task Object
Transformations	Transformation Object Transformation2 Object
TransformationSets	TransformationSet Object

The **Insert** method does not create an object. It adds an existing object to its collection. Typically, you create the object with the **New** method of the collection.

Prototype (C/C++)

```
HRESULT Insert(  
    VARIANT Position,  
    Interface *Object);
```

See Also

[Add Method](#)

[New Method](#)

[New \(Columns\) Method](#)

[New \(ID\) Method](#)

[New \(Name\) Method](#)

DTS Programming

Item Method

The **Item** method retrieves an object from a collection.

Applies To

Columns Collection	Properties Collection
Connections Collection	SavedPackageInfos Collection
DTSMQMessages Collection	ScriptingLanguageInfos Collection
DynamicPropertiesTaskAssignments Collection	Steps Collection
GlobalVariables Collection	TaskInfos Collection
Lookups Collection	Tasks Collection
OLEDBProperties Collection	TransformationInfos Collection
OLEDBProviderInfos Collection	Transformations Collection
PrecedenceConstraints Collection	

Syntax

[Set *object* =] *collection*.**Item**(*Index*)

Part	Description
<i>object</i>	Object to be retrieved from collection
<i>collection</i>	Expression that evaluates to an collection in the Applies To list
<i>Index</i>	Object name or ordinal position

Remarks

Index is the object name or the numeric position within the collection.

The required type of *Object* depends on the collection from which it is being retrieved.

Collection	Object
------------	--------

Columns	Column Object
Connections	Connection Object Connection2 Object
DTSMQMessages	DTSMQMessage Object
DynamicPropertiesTaskAssignments	DynamicPropertiesTaskAssignment Object
GlobalVariables	GlobalVariable Object
Lookups	Lookup Object
OLEDBProperties	OLEDBProperty Object
OLEDBProviderInfos	OLEDBProviderInfo Object
PrecedenceConstraints	PrecedenceConstraint Object
Properties	Property Object
SavedPackageInfos	SavedPackageInfo Object
ScriptingLanguageInfos	ScriptingLanguageInfo Object
Steps	Step Object Step2 Object
TaskInfos	TaskInfo Object
Tasks	Task Object
TransformationInfos	TransformationInfo Object
Transformations	Transformation Object Transformation2 Object
TransformationSets	TransformationSet Object

You can iterate through these collection using the **Item** method and **Count** property. In Microsoft® Visual Basic® this looks like:

```

For Index = 1 To collection.Count
    Set object = collection.Item(Index)
    ...
Next Index

```

However, it is faster to iterate through these collections using **For Each ... Next** in Visual Basic:

For Each object In collection

...

Next object

Prototype (C/C++)

```
HRESULT Item(  
    VARIANT Index,  
    interface **pRetVal);
```

See Also

[Add Method](#)

[Count Property](#)

[Insert Method](#)

DTS Programming

Load Method

The **Load** method fills the container object using persisted property values. The **Load** method is called by Data Transformation Services (DTS) when a DTS package is loaded.

Applies To

[PersistPropertyBag Object](#)

Syntax

object.**Load** *PropertyBag*

Part	Description
<i>object</i>	Expression that evaluates to a PersistPropertyBag object
<i>PropertyBag</i>	PropertyBag object to be loaded

Remarks

Generic binary large objects (BLOBs) or objects are not supported as property values.

Prototype (C/C++)

```
HRESULT Load(IDTSPropertyBag *IDTSPropertyBag);
```

See Also

[PropertyBag Object](#)

[Save Method](#)

DTS Programming

LoadFromRepository Method

The **LoadFromRepository** method loads the Data Transformation Services (DTS) package from the specified instance of Microsoft® SQL Server™ 2000 Meta Data Services. This includes information held in the objects and collections that belong to the **Package2** object.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

Package.**LoadFromRepository** *RepositoryServerName*,
RepositoryDatabaseName, _
RepositoryUserName, *RepositoryUserPassword*, *PackageID*, [*VersionID*], _
[*PackageName*], [*Flags*], [*pVarPersistStgOfHost*]

Part	Description
<i>Package</i>	Expression that evaluates to an object in the Applies To list.
<i>RepositoryServerName</i>	Meta Data Services server name.
<i>RepositoryDatabaseName</i>	Meta Data Services database name or data source name (DSN).
<i>RepositoryUserName</i>	Meta Data Services user name.
<i>RepositoryUserPassword</i>	Meta Data Services user password.
<i>PackageID</i>	Package identifier, which is a string representation of a globally unique identifier (GUID).
<i>VersionID</i>	Version identifier, which is a string representation of a GUID.
<i>PackageName</i>	Name of package to be loaded.
<i>Flags</i>	Value from the DTSRepositoryStorageFlags constants indicating user authentication type.

<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use only).
-----------------------------	--

Remarks

RepositoryDatabaseName is evaluated as an ODBC DSN if *RepositoryServerName* is empty or NULL. Otherwise, *RepositoryServerName* and *RepositoryDatabaseName* are used to create a connection without a DSN.

If *VersionID* is not specified or is blank, the most recent version of the package is retrieved.

Prototype (C/C++)

```
HRESULT LoadFromRepository(  
    BSTR RepositoryServerName,  
    BSTR RepositoryDatabaseName  
    BSTR RepositoryUserName,  
    BSTR RepositoryUserPassword,  
    BSTR PackageID,  
    BSTR VersionID CPPDEFAULT(= NULL)  
    BSTR PackageName CPPDEFAULT(= NULL)  
    DTSRepositoryStorageFlags Flags CPPDEFAULT(=  
DTSReposFlag_Default),  
    VARIANT *pUnkPersistStgOfHost CPPDEFAULT(= NULL));
```

See Also

[DTSRepositoryStorageFlags](#)

[LoadFromSQLServer Method](#)

[LoadFromStorageFile Method](#)

[Managing DTS Package Programs](#)

[RemoveFromRepository Method](#)

[SaveToRepository Method](#)

[SaveToRepositoryAs Method](#)

DTS Programming

LoadFromSQLServer Method

The **LoadFromSQLServer** method loads the Data Transformation Services (DTS) package from the specified server running an instance of Microsoft® SQL Server™. This includes information held in the objects and collections that belong to the **Package2** object.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

Package.**LoadFromSQLServer** *ServerName*, [*ServerUserName*], [*ServerPassword*], _
 [*Flags*], [*PackagePassword*], [*PackageGuid*], [*PackageVersionGuid*], _
 [*PackageName*], [*pVarPersistStgOfHost*]

Part	Description
<i>Package</i>	Expression that evaluates to an object in the Applies To list.
<i>ServerName</i>	Server name.
<i>ServerUserName</i>	Server user name.
<i>ServerPassword</i>	Server user password.
<i>Flags</i>	Value from the DTSSQLServerStorageFlags constants indicating user authentication type.
<i>PackagePassword</i>	Package password if the package is encrypted.
<i>PackageGuid</i>	Package identifier, which is a string representation of a globally unique identifier (GUID).
<i>PackageVersionGuid</i>	Version identifier which is a string representation of a GUID.
<i>PackageName</i>	Package name.
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use only).

Remarks

If *PackageVersionGUID* is not specified or is blank, the most recent version of the package is retrieved.

Prototype (C/C++)

```
HRESULT LoadFromSQLServer(  
    BSTR ServerName,  
    BSTR ServerUserName,  
    BSTR ServerPassword,  
    DTSSQLServerStorageFlags Flags CPPDEFAULT(=  
DTSSQLStgFlag_Default),  
    BSTR PackagePassword CPPDEFAULT(= NULL),  
    BSTR PackageGuid CPPDEFAULT(= NULL),  
    BSTR PackageVersionGuid CPPDEFAULT(= NULL),  
    BSTR PackageName CPPDEFAULT(= NULL),  
    VARIANT *pUnkPersistStgOfHost CPPDEFAULT(= NULL) );
```

See Also

[DTSSQLServerStorageFlags](#)

[LoadFromRepository Method](#)

[LoadFromStorageFile Method](#)

[Managing DTS Package Programs](#)

[RemoveFromSQLServer Method](#)

[SaveToSQLServer Method](#)

[SaveToSQLServerAs Method](#)

DTS Programming

LoadFromStorageFile Method

The **LoadFromStorageFile** method loads the Data Transformation Services (DTS) package from the specified structured storage file. This includes information held in the objects and collections that belong to the **Package** object.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**LoadFromStorageFile** *UNCFile*, *Password*, [*PackageID*], _
[*VersionID*], [*Name*], [*pVarPersistStgOfHost*]

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>UNCFile</i>	File specification of DTS package storage file.
<i>Password</i>	Package password if the package is encrypted.
<i>PackageID</i>	Package identifier, which is a string representation of a globally unique identifier (GUID).
<i>VersionID</i>	Version identifier, which is a string representation of a GUID.
<i>Name</i>	Package name.
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use only).

Remarks

If *VersionID* is not specified or is blank, the most recent version of the package is retrieved.

Prototype (C/C++)

```
HRESULT LoadFromStorageFile(  
    BSTR UNCFFile,  
    BSTR Password,  
    BSTR PackageID,  
    BSTR VersionID CPPDEFAULT(= NULL),  
    BSTR Name CPPDEFAULT(= NULL),  
    VARIANT *pUnkPersistStgOfHost CPPDEFAULT(= NULL));
```

See Also

[LoadFromRepository Method](#)

[LoadFromSQLServer Method](#)

[Managing DTS Package Programs](#)

[SaveToStorageFile Method](#)

[SaveToStorageFileAs Method](#)

DTS Programming

Lock Method

The **Lock** method locks a **GlobalVariable2** object for exclusive use.

Applies To

[GlobalVariable2 Object](#)

Syntax

globalvar.**Lock** *timeout*

Part	Description
<i>globalvar</i>	Expression that evaluates to a GlobalVariable2 object
<i>timeout</i>	Time-out value, in milliseconds

Remarks

If the global variable is already locked, the **Lock** method waits until the holder of the lock unlocks the global variable, or until the time-out occurs. When the time-out period elapses, an error occurs.

If 0 is specified for the time-out value, an error occurs immediately if the global variable is already locked.

Prototype (C/C++)

```
HRESULT Lock( long TimeOut );
```

See Also

[Adding DTS Lookups and Global Variables](#)

[Unlock Method](#)

DTS Programming

Logoff Method

The **Logoff** method ends a MAPI session.

Applies To

[SendMailTask Object](#)

Syntax

object.**Logoff**()

Part	Description
<i>Object</i>	Expression that evaluates to a SendMailTask object

Prototype (C/C++)

HRESULT Logoff();

See Also

[Logon Method](#)

DTS Programming

Logon Method

The **Logon** method creates a MAPI session.

Applies To

[SendMailTask Object](#)

Syntax

[*errorstring* =] *object*.**Logon**()

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>errorstring</i>	String describing error if Logon fails

Prototype (C/C++)

```
HRESULT Logon(BSTR *ErrorString);
```

See Also

[Logoff Method](#)

DTS Programming

Messages Method

The **Messages** method returns a reference to the **DTSMQMessages** collection.

Applies To

[DTSMMessageQueueTask Object](#)

Syntax

[Set *oMsgColl* =] *object*.**Messages**

Part	Description
<i>object</i>	Expression that evaluates to a DTSMMessageQueueTask object
<i>oMsgColl</i>	Object variable to receive a reference to a DTSMQMessages collection

Prototype (C/C++)

HRESULT Messages(DTSMQMessages **pVal)

See Also

[DTSMQMessages Collection](#)

DTS Programming

New Method

The **New** method creates a new, unnamed object for a collection.

Applies To

DTSMQMessages Collection	Steps Collection
DynamicPropertiesTaskAssignments Collection	

Syntax

[Set *object* =] *Collection*.**New**

Part	Description
<i>Collection</i>	Expression that evaluates to an object in the Applies To list
<i>object</i>	Object variable to receive reference to created object

Remarks

The required type of *object* depends on the collection to which it is being added:

Collection	Object
DTSMQMessages	DTSMQMessage Object
DynamicPropertiesTaskAssignments	DynamicPropertiesTaskAssignment Object
Steps	Step Object Step2 Object

The **New** method for the collections in the Applies To list creates a new object based on the collection type. The **New** methods for other collections use different parameters and syntax.

Prototype (C/C++)

```
HRESULT New(  
    ObjectInterface **RetVal);
```

Example

To create a new **Step** object using Microsoft® Visual Basic®, use:

```
Set objStep = objPackage.Steps.New
```

See Also

[Add Method](#)

[New \(Columns\) Method](#)

[New \(ID\) Method](#)

[New \(Name\) Method](#)

DTS Programming

New (Columns) Method

The **New** method creates a new **Column** object with a specified name and ordinal position.

Applies To

[Columns Collection](#)

Syntax

[Set *object* =] *Collection*.**New**(*Name*, *Ordinal*)

Part	Description
<i>Columns</i>	Expression that evaluates to a Columns collection
<i>object</i>	Expression that evaluates to a Column object
<i>Name</i>	String that is the name of the column
<i>Ordinal</i>	Long that is the ordinal position of the column

Remarks

The **New** method for the Columns collection creates a new object based on the specified *Name* and *Ordinal*. The **New** methods for other collections use different parameters and syntax.

Prototype (C/C++)

```
HRESULT New(  
    BSTR Name,  
    long Ordinal,  
    IDTSColumn **RetVal);
```

Examples

This Microsoft® Visual Basic® example creates a new **Column** object named **LastName** that is to be the third column in the source columns collection:

```
Set objColumn = objTransform.SourceColumns.New( "LastName", 3
```

See Also

[Add Method](#)

[Adding DTS Column Objects](#)

[New Method](#)

[New \(ID\) Method](#)

[New \(Name\) Method](#)

DTS Programming

New (CustomTaskUI) Method

The **New** method of the **CustomTaskUI** interface must be supplied by the implementer of a Data Transformation Services (DTS) custom task that has a user interface. It is called by DTS when a new instance of the custom task is to be created.

Applies To

[CustomTaskUI Object](#)

Syntax (Visual Basic)

```
Sub CustomTaskUI_New(ByVal hwndParent As Long )
```

Part	Description
<i>hwndParent</i>	Handle to the window to be the parent of the custom task user interface

Remarks

The window that is parent to the custom task's user interface window typically would be the DTS Designer design sheet or the Microsoft® Visual Studio® development environment desktop.

The **New** methods for various collections use different parameters and syntax than the **New** method of the **CustomTaskUI** interface.

Prototype (C/C++)

```
HRESULT New(  
    long hwndParent );
```

See Also

[CreateCustomToolTip Method](#)

[Delete Method](#)

[Edit Method](#)

[New Method](#)

[New \(Columns\) Method](#)

[New \(ID\) Method](#)

[New \(Name\) Method](#)

DTS Programming

New (ID) Method

The **New** method creates a new object for a collection from a ProgID or CLSID.

Applies To

Connections Collection	Transformations Collection
Tasks Collection	

Syntax

[Set *object* =] *Collection*.**New**(*ID*)

Part	Description
<i>Collection</i>	Expression that evaluates to an object in the Applies To list
<i>object</i>	Object variable to receive reference to created object
<i>ID</i>	ProgID or CLSID of object to be created

Remarks

The required type of *object* depends on the collection to which it is being added:

Collection	Object
Connections	Connection Object Connection2 Object
Tasks	Task Object
Transformations	Transformation Object Transformation2 Object

The **New** method for the collections in the Applies To list creates a new object based on the ProgID or CLSID of the corresponding class. The **New** methods for other collections use different parameters and syntax.

There are version-dependent and version-independent forms of a ProgID. Typically, the version-dependent form looks like *name.version*. The version-independent form typically has *name* only, without the *.version*, although that form is not required. The version-independent form should be used unless you need a feature of a particular version.

Prototype (C/C++)

```
HRESULT New(  
    BSTR bstrID,  
    ObjectInterface **RetVal);
```

Examples

To create a new connection using the Microsoft® OLE DB Provider for ODBC drivers in Microsoft Visual Basic®, use one of the following forms.

By version-independent ProgID:

```
Set objConnection = objPackage.Connections.New( "MSDASQL" )
```

By version-dependent ProgID:

```
Set objConnection = objPackage.Connections.New( "MSDASQL.1"
```

By CLSID:

```
Set objConnection = _  
    objPackage.Connections.New( "{C8B522CB-5CF3-11CE-ADE5-C
```

See Also

[Add Method](#)

[New Method](#)

[New \(Columns\) Method](#)

[New \(Name\) Method](#)

DTS Programming

New (Name) Method

The **New** method creates a new object for a collection with a specified name.

Applies To

GlobalVariables Collection	PrecedenceConstraints Collection
Lookups Collection	

Syntax

[Set *object* =] *Collection*.**New**(*Name*)

Part	Description
<i>Collection</i>	Expression that evaluates to an object in the Applies To list
<i>object</i>	Object variable to receive reference to created object
<i>Name</i>	String that is to be the name of the created object

Remarks

The required type of *object* depends on the collection to which it is being added:

Collection	Object
GlobalVariables	GlobalVariable Object
Lookups	Lookup Object
PrecedenceConstraints	PrecedenceConstraint Object
TransformationSets	TransformationSet Object

The **New** method for the collections in the Applies To list creates a new object with the specified name. The **New** methods for other collections use different parameters and syntax.

Prototype (C/C++)

```
HRESULT New(  
    BSTR Name,  
    ObjectInterface **RetVal);
```

Examples

This sample statement creates a new **TransformationSet** object named **Customers**, in Microsoft® Visual Basic®:

```
Set objTransSet = objParPumpTask.TransformationSets.New( "Custo
```

See Also

[Add Method](#)

[New Method](#)

[New \(Columns\) Method](#)

[New \(ID\) Method](#)

DTS Programming

NewDataLink Method

The **NewDataLink** method gets a new **Connection** object using Microsoft® Data Link files.

Applies To

[Connections Collection](#)

Syntax

[Set *object* =] *collection*.**NewDataLink**(*UDLPath*)

Part	Description
<i>collection</i>	Expression that evaluates to a Connections collection
<i>UDLPath</i>	File specification of the UDL file
<i>object</i>	Object Variable of type Connection

Remarks

NewDataLink is similar to **New**, except you supply a UDL path instead of a provider ID.

Prototype (C/C++)

```
HRESULT NewDataLink (  
    BSTR UDLPath,  
    IDTSConnection **pRetVal);
```

See Also

[Connection Object](#)

[New \(ID\) Method](#)

DTS Programming

Next Method

The **Next** method fetches the next object while iterating through the associated collection.

Applies To

PackageInfos Collection	StepLineages Collection
PackageLineages Collection	StepLogRecords Collection
PackageLogRecords Collection	

Syntax

Set *object* = *collection*.**Next**

Part	Description
<i>collection</i>	Expression that evaluates to a collection in the Applies To list
<i>object</i>	Reference to next object in <i>collection</i>

Remarks

The *object* associated with each *collection* type is specified in this table.

Collection	Associated object	C/C++ interface
PackageInfos Collection	PackageInfo Object	IDTSPackageInfo
PackageLineages Collection	PackageLineage Object	IDTSPackageLineage
PackageLogRecords	PackageLogRecord Object	IDTSPackageLogRecord
StepLineages Collection	StepLineage Object	IDTSSStepLineage
StepLogRecords Collection	StepLogRecord Object	IDTSSStepLogRecord

TaskLogRecords Collection	TaskLogRecord Object	IDTSTaskLogRecord
-------------------------------------	--------------------------------------	-------------------

Iterate through a collection by checking the **EOF** property after calling the **Next** method. If **EOF** is True, **Next** will have returned **Nothing** and all of the elements will have been fetched. In Microsoft® Visual Basic® this looks like:

```

Set object = collection.Next
Do Until collection.EOF
    ...
    Set object = collection.Next
Loop

```

The collections in the Applies To list can also be processed using **For Each ... Next** in Visual Basic:

```

For Each object In collection
    ...
Next object

```

Prototype (C/C++)

```

HRESULT Next(interface **pRetVal);

```

interface is as defined in the above table.

See Also

[EOF Property](#)

[Retrieving DTS System, Package, and Log Data](#)

DTS Programming

Read Method

The **Read** method retrieves a property value.

Applies To

[PropertyBag Object](#)

Syntax

[*value* =] *object*.**Read**(*bstrPropertyName*)

Part	Description
<i>Object</i>	Expression that evaluates to a PropertyBag object
<i>BstrPropertyName</i>	String identifying an exposed property by name
<i>Value</i>	Variant that receives the property value

Remarks

Generic BLOBs or objects are not supported as property values.

Prototype (C/C++)

```
HRESULT Read(  
    BSTR bstrPropertyName,  
    VARIANT *pValue);
```

See Also

[PersistPropertyBag Object](#)

[Write Method](#)

DTS Programming

Refresh Method

The **Refresh** method updates the cached information for the associated collection by scanning the registered classes in the operating system registry.

Applies To

OLEDBProviderInfos Collection	TaskInfos Collection
ScriptingLanguageInfos Collection	

Syntax

collection.**Refresh**

Part	Description
<i>collection</i>	Expression that evaluates to a collection in the Applies To list

Remarks

Data Transformation Services (DTS) maintains a cache (in the registry) of the components of each collection in the Applies To list. When iterating through any of the collections the appropriate cache is examined, rather than all registered classes, when the **UseCache** property is True. Use the **Refresh** method to refresh the appropriate cache from the system registry.

Prototype (C/C++)

```
HRESULT Refresh( );
```

See Also

[Retrieving DTS System, Package, and Log Data](#)

[UseCache Property](#)

DTS Programming

ReleaseConnection Method

The **ReleaseConnection** method releases ownership of the connection and makes it available to other tasks.

Applies To

Connection Object	Connection2 Object
-----------------------------------	------------------------------------

Syntax

Connection.**ReleaseConnection()**

Part	Description
<i>Connection</i>	Expression that evaluates to an object in the Applies To list

Remarks

The **InUse** property is set to False. This method is required only if you are creating a custom task. All tasks must include calls to **AcquireConnection** and **ReleaseConnection**. The tasks supplied with Microsoft® SQL Server™ 2000 already include them.

Rowsets are always released at the **Task** object level.

Prototype (C/C++)

```
HRESULT ReleaseConnection();
```

See Also

[AcquireConnection Method](#)

[Creating DTS Package Objects and Connections](#)

[InUse Property](#)

DTS Programming

Remove Method

The **Remove** method removes an object from a collection.

Applies To

Columns Collection	PrecedenceConstraints Collection
Connections Collection	Steps Collection
DTSMQMessages Collection	Tasks Collection
DynamicPropertiesTaskAssignments Collection	Transformations Collection
GlobalVariables Collection	TransformationSets Collection
Lookups Collection	

Syntax

collection.**Remove** *Index*

Part	Description
<i>collection</i>	Expression that evaluates to a collection in the Applies To list
<i>Index</i>	Object name or ordinal position

Remarks

Index is the object name or the numeric position within the collection.

The types of object contained in each collection type is as follows.

Collection	Object
Columns	Column Object
Connections	Connection Object Connection2 Object
DTSMQMessages	DTSMQMessage Object
DynamicPropertiesTaskAssignments	DynamicPropertiesTaskAssignment

	Object
GlobalVariables	GlobalVariable Object
Lookups	Lookup Object
PrecedenceConstraints	PrecedenceConstraint Object
Steps	Step Object Step2 Object
Tasks	Task Object
Transformations	Transformation Object Transformation2 Object
TransformationSets	TransformationSet Object

Prototype (C/C++)

HRESULT Remove(VARIANT Index);

See Also

[Add Method](#)

[Insert Method](#)

[Item Method](#)

DTS Programming

RemoveAllLogRecords Method

The **RemoveAllLogRecords** method removes all package, step, and task log records from the database associated with the **PackageSQLServer** object.

Applies To

[PackageSQLServer Object](#)

Syntax

object.**RemoveAllLogRecords**()

Part	Description
<i>object</i>	Expression that evaluates to a PackageSQLServer object

Remarks

Log records are written to the **msdb** database on the instance of Microsoft® SQL Server™ specified by the package **LogServerName** property each time a Data Transformation Services (DTS) package is executed, if the package **LogToSQLServer** property has been set. The **RemoveAllLogRecords** method removes these records for all packages from the database.

Prototype (C/C++)

HRESULT RemovePackageLogRecords();

See Also

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageLogRecords Collection](#)

[RemovePackageLogRecords Method](#)

[RemoveStepLogRecords Method](#)

[RemoveTaskLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[StepLogRecords Collection](#)

[TaskLogRecords Collection](#)

DTS Programming

RemoveFromRepository Method

The **RemoveFromRepository** method removes the package from the specified instance of Microsoft® SQL Server™ 2000 Meta Data Services.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

Package.**RemoveFromRepository** *RepositoryServerName*, _
RepositoryDatabaseName, *RepositoryUserName*, *RepositoryUserPassword*, _
PackageID, [*VersionID*], [*PackageName*], [*Flags*]

Part	Description
<i>Package</i>	Expression that evaluates to an object in the Applies To list.
<i>RepositoryServerName</i>	Meta Data Services server name.
<i>RepositoryDatabaseName</i>	Meta Data Services database name or datasource name (DSN).
<i>RepositoryUsername</i>	Meta Data Services user name.
<i>RepositoryUserPassword</i>	Meta Data Services user password.
<i>PackageID</i>	Package identifier, which is a string representation of a globally unique identifier (GUID).
<i>VersionID</i>	Version identifier, which is a string representation of a GUID.
<i>PackageName</i>	Package name.
<i>Flags</i>	Value from the DTSRepositoryStorageFlags constants, which specified the user authentication type.

Remarks

RepositoryDatabaseName is evaluated as an ODBC DSN if *RepositoryServerName* is empty or NULL. Otherwise, *RepositoryServerName* and *RepositoryDatabaseName* are used to create a connection without a DSN.

If *VersionID* is not specified or is blank, the most recent version of the package is removed.

Prototype (C/C++)

```
HRESULT RemoveFromRepository(  
    BSTR RepositoryServerName,  
    BSTR RepositoryDatabaseName,  
    BSTR RepositoryUserName,  
    BSTR RepositoryUserPassword,  
    BSTR PackageID,  
    BSTR VersionID CPPDEFAULT(= NULL)  
    BSTR PackageName CPPDEFAULT(= NULL)  
    DTSRepositoryStorageFlags Flags CPPDEFAULT(= DTSReposFlag_Default)  
);
```

See Also

[DTSRepositoryStorageFlags](#)

[LoadFromRepository Method](#)

[Managing DTS Package Programs](#)

[SaveToRepository Method](#)

[SaveToRepositoryAs Method](#)

DTS Programming

RemoveFromSQLServer Method

The **RemoveFromSQLServer** method removes the package from the specified server running an instance of Microsoft® SQL Server™.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

Package.**RemoveFromSQLServer** *ServerName*, [*ServerUserName*], _
[*ServerPassword*], [*Flags*], [*PackageGuid*], _
[*PackageVersionGuid*], [*PackageName*]

Part	Description
<i>Package</i>	Expression that evaluates to an object in the Applies To list.
<i>ServerName</i>	Server name.
<i>ServerUserName</i>	Server user name.
<i>ServerPassword</i>	Server user password.
<i>Flags</i>	Value from the DTSSQLServerStorageFlags constants indicating user authentication type.
<i>PackageGuid</i>	Package identifier, which is a string representation of a globally unique identifier (GUID).
<i>PackageVersionGuid</i>	Version identifier, which is a string representation of a GUID.
<i>PackageName</i>	Package name.

Remarks

If *PackageVersionGUID* is not specified or is blank, the most recent version of the package is removed.

Prototype (C/C++)

```
HRESULT RemoveFromSQLServer(  
    BSTR ServerName,  
    BSTR ServerUserName,  
    BSTR ServerPassword,  
    DTSSQLServerStorageFlags Flags CPPDEFAULT(=  
DTSSQLStgFlag_Default),  
    BSTR PackageGuid CPPDEFAULT(= NULL),  
    BSTR PackageVersionGuid CPPDEFAULT(= NULL),  
    BSTR PackageName CPPDEFAULT(= NULL) );
```

See Also

[DTSSQLServerStorageFlags](#)

[LoadFromSQLServer Method](#)

[Managing DTS Package Programs](#)

[SaveToSQLServer Method](#)

[SaveToSQLServerAs Method](#)

DTS Programming

RemovePackageLineages Method

The **RemovePackageLineages** method removes the package and step lineage records from Microsoft® SQL Server™ 2000 Meta Data Services that satisfy the criteria of the input parameters.

Applies To

[PackageRepository Object](#)

Syntax

object.**RemovePackageLineages**(*PackageVersionID*, *KeepLatest*, *LineageFullID*, *LineageShortID*)

Part	Description
<i>Object</i>	Expression that evaluates to a PackageRepository object.
<i>PackageVersionID</i>	A globally unique identifier (GUID) string specifying the version ID of the package version for which lineage records are to be removed.
<i>KeepLatest</i>	A Boolean which specifies whether the lineage records for the most recent execution are to be retained.
<i>LineageFullID</i>	A GUID string specifying the lineage full ID of the lineage records to be removed.
<i>LineageShortID</i>	A long specifying the lineage short ID of the lineage records to be removed.

Remarks

A package lineage record and step lineage records are written each time a package stored in Meta Data Services is executed, if the package **LineageOptions** property specifies this to be done. **RemovePackageLineages** removes these records from Meta Data Services.

A package ID or the empty string cannot be specified for *PackageVersionID*. An error will be raised if there is no package lineage record in Meta Data Services with the specified version ID.

All parameters are required. To remove all package lineage records for a particular package version, code `False` for *KeepLatest*, the empty string "" for *LineageFullID* and 0 for *LineageShortID*. To remove all package lineage records in Meta Data Services, iterate through the Meta Data Services packages with the **EnumPackageInfos** method, then call **RemovePackageLineages** for each package version.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

Prototype (C/C++)

```
HRESULT RemovePackageLineages(  
    BSTR PackageVersionID,  
    VARIANT_BOOL KeepLatest,  
    BSTR LineageFullID,  
    long LineageShortID );
```

See Also

[EnumPackageLineages Method](#)

[LineageFullID Property](#)

[LineageOptions Property](#)

[LineageShortID Property](#)

[PackageID Property](#)

[Retrieving DTS System, Package, and Log Data](#)

[VersionID Property](#)

DTS Programming

RemovePackageLogRecords Method

The **RemovePackageLogRecords** method removes the package log records that satisfy the criteria of the input parameters from the database.

Applies To

[PackageSQLServer Object](#)

Syntax

```
object.RemovePackageLogRecords(  
    PackageName,  
    KeepLatest,  
    PackageID,  
    VersionID,  
    LineageFullID )
```

Part	Description
<i>object</i>	Expression that evaluates to a PackageSQLServer object.
<i>PackageName</i>	A string specifying the package name for which log records are to be removed.
<i>KeepLatest</i>	A Boolean which specifies whether the log records for the most recent execution are to be retained.
<i>PackageID</i>	A globally unique identifier (GUID) string specifying the package ID for which log records are to be removed.
<i>VersionID</i>	A GUID string specifying the version ID of the package version for which log records are to be removed.
<i>LineageFullID</i>	A GUID string specifying the lineage full ID for which log records are to be removed.

Remarks

A package log record is written to the **msdb** database on the instance of

Microsoft® SQL Server™ specified by the package **LogServerName** property each time a Data Transformation Services (DTS) package is executed, if the package **LogToSQLServer** property has been set. The **RemovePackageLogRecords** method removes these records from the database.

All parameters are required. To prevent a parameter from participating in the filter process, code the empty string "" for *PackageName*, *PackageID*, *VersionID* and *LineageFullID*, and False for *KeepLatest*. Coding all parameters this way will cause all package log records in the database to be removed.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

Prototype (C/C++)

```
HRESULT RemovePackageLogRecords(  
    BSTR PackageName,  
    VARIANT_BOOL KeepLatest,  
    BSTR PackageID,  
    BSTR VersionID,  
    BSTR LineageFullID );
```

See Also

[EnumPackageLogRecords Method](#)

[LineageFullID Property](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageID Property](#)

[PackageLogRecords Collection](#)

[PackageName Property](#)

[RemoveAllLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[VersionID Property](#)

DTS Programming

RemoveStepLogRecords Method

The **RemoveStepLogRecords** method removes the step log records that satisfy the criteria of the input parameters from the database.

Applies To

[PackageSQLServer Object](#)

Syntax

```
object.RemoveStepLogRecords(  
    LineageFullID,  
    StepExecutionID )
```

Part	Description
<i>object</i>	Expression that evaluates to a PackageSQLServer object.
<i>LineageFullID</i>	A globally unique identifier (GUID) string specifying the lineage full ID for which log records are to be removed.
<i>StepExecutionID</i>	A variant specifying the step execution ID for which log records are to be removed.

Remarks

A step log record is written to the **msdb** database on the instance of Microsoft® SQL Server™ specified by the package **LogServerName** property each time a step in a Data Transformation Services (DTS) package is executed, if the package **LogToSQLServer** property has been set. The **RemoveStepLogRecords** method removes these records from the database.

Both parameters are required. To prevent a parameter from participating in the filter process, code the empty string "" for *LineageFullID*, and Null for *StepExecutionID*. Coding all parameters this way will cause all step log records to be removed.

The syntax of GUIDs is:

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

where the X are hexadecimal digits. The groupings are 8, 4, 4, 4, and 12 digits. The curly brackets are required. Spaces cannot be embedded for readability.

Prototype (C/C++)

```
HRESULT RemoveStepLogRecords(  
    BSTR LineageFullID,  
    VARIANT StepExecutionID );
```

See Also

[EnumStepLogRecords Method](#)

[LineageFullID Property](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[RemoveAllLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[StepExecutionID Property](#)

[StepLogRecords Collection](#)

DTS Programming

RemoveTaskLogRecords Method

The **RemoveTaskLogRecords** method removes the task log records that satisfy the criteria of the input parameters from the database.

Applies To

[PackageSQLServer Object](#)

Syntax

```
object.RemoveTaskLogRecords(  
    StepExecutionID,  
    SequenceID )
```

Part	Description
<i>object</i>	Expression that evaluates to a PackageSQLServer object
<i>StepExecutionID</i>	A variant specifying the step execution ID for which log records are to be removed
<i>SequenceID</i>	A long specifying the sequence number of the log record to be removed

Remarks

Task log records are not written by the task classes supplied with Microsoft® SQL Server™ 2000, but the **PackageLog** interface is available so that custom tasks can write them. They are written to the **msdb** database on the server specified by the package **LogServerName** property, if the package **LogToSQLServer** property has been set. *SequenceID* is a unique sequence number for each task log record. The **RemoveTaskLogRecords** method removes these records from the database.

Both parameters are required. To prevent a parameter from participating in the filter process, code Null for *StepExecutionID* and 0 for *SequenceID*. Coding all parameters this way will cause all task log records in the database to be

removed.

Prototype (C/C++)

```
HRESULT RemoveTaskLogRecords(  
    VARIANT StepExecutionID,  
    long SequenceID );
```

See Also

[EnumTaskLogRecords Method](#)

[LogServerName Property](#)

[LogToSQLServer Property](#)

[PackageLog Object](#)

[RemoveAllLogRecords Method](#)

[Retrieving DTS System, Package, and Log Data](#)

[SequenceID Property](#)

[StepExecutionID Property](#)

[TaskLogRecords Collection](#)

DTS Programming

Reset Method

The **Reset** method resets all values in the target object to their defaults.

Applies To

DTSMQMessage Object	DynamicPropertiesTaskAssignment Object
-------------------------------------	--

Syntax

object.Reset

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Prototype (C/C++)

HRESULT Reset()

DTS Programming

ResetObjectsList Method

The **ResetObjectsList** method clears the list of objects to be transferred.

Applies To

TransferObjectsTask Object	TransferObjectsTask2 Object
--	---

Syntax

object.ResetObjectsList

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list

Prototype (C/C++)

```
HRESULT ResetObjectsList();
```

See Also

[AddObjectForTransfer Method](#)

[GetObjectForTransfer Method](#)

DTS Programming

ResolveName Method

The **ResolveName** method resolves an e-mail address.

Applies To

[SendMailTask Object](#)

Syntax

[string =] *object*.**ResolveName**(*Address*)

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>Address</i>	E-mail addresses to be resolved
<i>string</i>	String to receive the list of resolved names

Remarks

E-mail addresses are separated by semicolons.

Prototype (C/C++)

```
HRESULT ResolveName(  
    BSTR Address,  
    BSTR *ErrorString);
```

DTS Programming

Save Method

The **Save** method instructs an object implementing a custom task to perform custom property storage into a **PropertyBag** object. The **Save** method is called by Data Transformation Services (DTS) when a DTS package is stored.

Applies To

[PersistPropertyBag Object](#)

Syntax

object.**Save** *PropertyBag*

Part	Description
<i>object</i>	Expression that evaluates to a PersistPropertyBag object
<i>PropertyBag</i>	PropertyBag object to receive the persisted properties

Remarks

Generic BLOBs or objects are not supported as property values.

Prototype (C/C++)

```
HRESULT Save(IDTSPropertyBag *IDTSPropertyBag);
```

See Also

[Load Method](#)

[PropertyBag Object](#)

DTS Programming

SaveAs Method

The **SaveAs** method creates a new Data Transformation Services (DTS) package ID and assigns the new name while preserving all other properties.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

object.**SaveAs** *NewName*

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>NewName</i>	New package name

Remarks

The package is given a new name and package ID, but nothing is saved to persistent storage.

Prototype (C/C++)

```
HRESULT SaveAs( BSTR NewName );
```

See Also

[Managing DTS Package Programs](#)

[SaveToRepository Method](#)

[SaveToRepositoryAs Method](#)

[SaveToSQLServer Method](#)

[SaveToSQLServerAs Method](#)

[SaveToStorageFile Method](#)

[SaveToStorageFileAs Method](#)

DTS Programming

SaveToRepository Method

The **SaveToRepository** method saves information being held in the **Package2** object and its subordinate objects and collections to the specified instance of Microsoft® SQL Server™ 2000 Meta Data Services.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

```
object.SaveToRepository(  
    RepositoryServerName,  
    RepositoryDatabaseName,  
    RepositoryUserName,  
    RepositoryUserPassword,  
    [Flags],  
    [CategoryID],  
    [pVarPersistStgOfHost])
```

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>RepositoryServerName</i>	A string specifying the name of the server on which the instance of Meta Data Services is hosted.
<i>RepositoryDatabaseName</i>	A string specifying the name of the database in which the instance of Meta Data Services is located.
<i>RepositoryUsername</i>	A string specifying the logon name used to access the server specified by <i>ServerName</i> .
<i>RepositoryUserPassword</i>	A string specifying the password used to access the server specified by <i>ServerName</i> .
<i>Flags</i>	A value from the DTSRepositoryStorageFlags

	constants that specifies the type of user authentication used to access the server specified by <i>RepositoryServerName</i> .
<i>CategoryID</i>	Reserved.
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use).

Remarks

RepositoryDatabaseName is evaluated as an ODBC datasource name (DSN) if *RepositoryServerName* is empty or NULL. Otherwise, *RepositoryServerName* and *RepositoryDatabaseName* are used to create a DSN-less connection.

A new *VersionID* value is always generated when the package is saved.

Prototype (C/C++)

```
HRESULT SaveToRepository(
    BSTR RepositoryServerName,
    BSTR RepositoryDatabaseName,
    BSTR RepositoryUserName,
    BSTR RepositoryUserPassword,
    DTSRepositoryStorageFlags Flags,
    BSTR CategoryID,
    VARIANT pVarPersistStgOfHost);
```

See Also

[DTSRepositoryStorageFlags](#)

[Managing DTS Package Programs](#)

[SaveAs Method](#)

[SaveToRepositoryAs Method](#)

[SaveToSQLServer Method](#)

[SaveToStorageFile Method](#)

DTS Programming

SaveToRepositoryAs Method

The **SaveToRepositoryAs** method saves information being held in the **Package** object and its subordinate objects and collections to the specified instance of Microsoft® SQL Server™ 2000 Meta Data Services. The new name and a new Data Transformation Services (DTS) package ID are assigned.

Applies To

[Package2 Object](#)

Syntax

```
object.SaveToRepository(  
    NewName,  
    RepositoryServerName,  
    RepositoryDatabaseName,  
    RepositoryUserName,  
    RepositoryUserPassword,  
    [Flags],  
    [CategoryID],  
    [pVarPersistStgOfHost])
```

Part	Description
<i>Object</i>	Expression that evaluates to a Package2 object.
<i>NewName</i>	A string specifying the new name for the package.
<i>RepositoryServerName</i>	A string specifying the name of the server on which the instance of Meta Data Services is hosted.
<i>RepositoryDatabaseName</i>	A string specifying the name of the database in which the instance of Meta Data Services is located.
<i>RepositoryUsername</i>	A string specifying the logon name used to access the server specified by <i>ServerName</i> .

<i>RepositoryUserPassword</i>	A string specifying the password used to access the server specified by <i>ServerName</i> .
<i>Flags</i>	A value from the <i>DTSRepositoryStorageFlags</i> constants that specifies the type of user authentication used to access the server specified by <i>RepositoryServerName</i> .
<i>CategoryID</i>	Reserved.
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use).

Remarks

RepositoryDatabaseName is evaluated as an ODBC data source name (DSN) if *RepositoryServerName* is empty or NULL. Otherwise, *RepositoryServerName* and *RepositoryDatabaseName* are used to create a connection without a DSN.

New package ID and version ID values are generated when the package is saved.

Prototype (C/C++)

```
HRESULT SaveToRepositoryAs(
    BSTR NewName,
    BSTR RepositoryServerName,
    BSTR RepositoryDatabaseName,
    BSTR RepositoryUserName,
    BSTR RepositoryUserPassword,
    DTSRepositoryStorageFlags Flags,
    BSTR CategoryID,
    VARIANT pVarPersistStgOfHost);
```

See Also

[DTSRepositoryStorageFlags](#)

[Managing DTS Package Programs](#)

[SaveAs Method](#)

[SaveToRepository Method](#)

[SaveToSQLServerAs Method](#)

[SaveToStorageFileAs Method](#)

DTS Programming

SaveToSQLServer Method

The **SaveToSQLServer** method saves information being held in the **Package2** object and its subordinate objects and collections to the specified server running an instance of Microsoft® SQL Server™.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

```
object.SaveToSQLServer(  
    ServerName,  
    [ServerUserName],  
    [ServerPassword],  
    [Flags],  
    [PackageOwnerPassword],  
    [PackageOperatorPassword],  
    [PackageCategoryID],  
    [pVarPersistStgOfHost],  
    [bReusePasswords])
```

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list.
<i>ServerName</i>	A string specifying the name of the server to which to connect.
<i>ServerUserName</i>	A string specifying the logon name used to access the server specified by <i>ServerName</i> .
<i>ServerPassword</i>	A string specifying the password used to access the server specified by <i>ServerName</i> .
<i>Flags</i>	A value from the DTSSQLServerStorageFlags constants that specifies the type of user authentication used to access the server

	specified by <i>ServerName</i> .
<i>PackageOwnerPassword</i>	A string specifying the package owner password if the package is encrypted.
<i>PackageOperatorPassword</i>	A string specifying the package user password if the package is encrypted.
<i>PackageCategoryID</i>	A string specifying the package category (reserved).
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use).
<i>bReusePasswords</i>	A Boolean specifying whether to reuse package passwords.

Remarks

A new version ID value is always generated when the package is saved.

The default for *bReusePasswords* is True.

Prototype (C/C++)

```
HRESULT SaveToSqlServer(
    BSTR ServerName,
    BSTR ServerUserName,
    BSTR ServerPassword,
    DTSSQLServerStorageFlags Flags,
    BSTR PackageOwnerPassword,
    BSTR PackageOperatorPassword,
    BSTR PackageCategoryID,
    VARIANT pVarPersistStgOfHost),
    VARIANT_BOOL bReusePasswords);
```

See Also

[DTSSQLServerStorageFlags](#)

[Managing DTS Package Programs](#)

[SaveAs Method](#)

[SaveToRepository Method](#)

[SaveToSQLServerAs Method](#)

[SaveToStorageFile Method](#)

DTS Programming

SaveToSQLServerAs Method

The **SaveToSQLServer** method saves information being held in the **Package2** object and its subordinate objects and collections to the specified server running an instance of Microsoft® SQL Server™. The new name and a new package ID are assigned.

Applies To

[Package2 Object](#)

Syntax

```
object.SaveToSQLServerAs(  
    NewName,  
    ServerName,  
    [ServerUserName],  
    [ServerPassword],  
    [Flags],  
    [PackageOwnerPassword],  
    [PackageOperatorPassword],  
    [PackageCategoryID],  
    [pVarPersistStgOfHost],  
    [bReusePasswords])
```

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object.
<i>NewName</i>	A string specifying the new name for the package.
<i>ServerName</i>	A string specifying the name of the server to which to connect.
<i>ServerUserName</i>	A string specifying the logon name used to access the server specified by <i>ServerName</i> .
<i>ServerPassword</i>	A string specifying the password used to access

	the server specified by <i>ServerName</i> .
<i>Flags</i>	A value from the DTSSQLServerStorageFlags constants that specifies the type of user authentication used to access the server specified by <i>ServerName</i> .
<i>PackageOwnerPassword</i>	A string specifying the package owner password if the package is encrypted.
<i>PackageOperatorPassword</i>	A string specifying the package user password if the package is encrypted.
<i>PackageCategoryID</i>	A string specifying the package category (reserved).
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use).
<i>bReusePasswords</i>	A Boolean specifying whether to reuse package passwords.

Remarks

New package ID and version ID values are generated when the package is saved, and the new name is assigned.

The default for *bReusePasswords* is True.

Prototype (C/C++)

```
HRESULT SaveToSQLServerAs(
    BSTR NewName,
    BSTR ServerName,
    BSTR ServerUserName,
    BSTR ServerPassword,
    DTSSQLServerStorageFlags Flags,
    BSTR PackageOwnerPassword,
    BSTR PackageOperatorPassword,
    BSTR PackageCategoryID,
    VARIANT pVarPersistStgOfHost),
    VARIANT_BOOL bReusePasswords);
```

See Also

[DTSSQLServerStorageFlags](#)

[Managing DTS Package Programs](#)

[SaveAs Method](#)

[SaveToRepositoryAs Method](#)

[SaveToSQLServer Method](#)

[SaveToStorageFileAs Method](#)

DTS Programming

SaveToStorageFile Method

The **SaveToStorageFile** method saves the information being held in the **Package2** object and its subordinate objects and collections to a structured storage file.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

```
object.SaveToStorageFile(  
    [UNCFFile],  
    [OwnerPassword],  
    [OperatorPassword],  
    [pVarPersistStgOfHost],  
    [bReusePasswords])
```

Part	Description
<i>object</i>	Expression that evaluates to an object in the Applies To list
<i>UNCFFile</i>	File specification to which package is to be written
<i>OwnerPassword</i>	A string specifying the package owner password if the package is encrypted
<i>OperatorPassword</i>	A string specifying the package user password if the package is encrypted
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use)
<i>bReusePasswords</i>	A Boolean specifying whether to reuse package passwords

Remarks

A new version ID value is always generated when the package is saved.
The default for *bReusePasswords* is True.

Prototype (C/C++)

```
HRESULT SaveToStorageFile(  
    BSTR UNCFFile,  
    BSTR OwnerPassword  
    BSTR OperatorPassword  
    VARIANT *pVarPersistStgOfHost  
    VARIANT_BOOL bReusePasswords );
```

See Also

[Managing DTS Package Programs](#)

[SaveAs Method](#)

[SaveToRepository Method](#)

[SaveToSQLServer Method](#)

[SaveToStorageFileAs Method](#)

DTS Programming

SaveToStorageFileAs Method

The **SaveToStorageFileAs** method saves the information being held in the **Package2** object and its subordinate objects and collections to a structured storage file. The new name and a new package ID are assigned.

Applies To

[Package2 Object](#)

Syntax

```
object.SaveToStorageFileAs(  
    NewName,  
    [UNCFile],  
    [OwnerPassword],  
    [OperatorPassword],  
    [pVarPersistStgOfHost],  
    [bReusePasswords])
```

Part	Description
<i>object</i>	Expression that evaluates to a Package2 object
<i>NewName</i>	A string specifying the new name for the package
<i>UNCFile</i>	File specification to which package is to be written
<i>OwnerPassword</i>	A string specifying the package owner password if the package is encrypted
<i>OperatorPassword</i>	A string specifying the package user password if the package is encrypted
<i>pVarPersistStgOfHost</i>	Screen layout information associated with a package (for internal use)
<i>bReusePasswords</i>	A Boolean specifying whether to reuse package passwords

Remarks

New package ID and version ID values are generated when the package is saved, and the new name is assigned.

The default for *bReusePasswords* is True.

Prototype (C/C++)

```
HRESULT SaveToStorageFileAs(  
    BSTR UNCFFile,  
    BSTR OwnerPassword  
    BSTR OperatorPassword  
    VARIANT *pVarPersistStgOfHost  
    VARIANT_BOOL bReusePasswords );
```

See Also

[Managing DTS Package Programs](#)

[SaveAs Method](#)

[SaveToRepositoryAs Method](#)

[SaveToSQLServerAs Method](#)

[SaveToStorageFile Method](#)

DTS Programming

SetDayLongName Method

The **SetDayLongName** method sets the long (full) name for the specified day of the week.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**SetDayLongName**(*daynumber*) = *string*

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>daynumber</i>	Number of the day of the week
<i>string</i>	Long (full) name of the specified day of week

Remarks

The valid day numbers are from 1 through 7.

Prototype (C/C++)

```
HRESULT SetDayLongName(  
long DayNumber,  
BSTR NewValue);
```

See Also

[Adding DTS Transformations](#)

[Day?LongName Property](#)

[GetDayLongName Method](#)

[SetDayShortName Method](#)

DTS Programming

SetDayShortName Method

The **SetDayShortName** method sets the short name (3-character abbreviation) for the specified day of the week.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**SetDayShortName**(*daynumber*) = *string*

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>daynumber</i>	Number of the day of the week
<i>string</i>	Short name (3-character abbreviation) of the specified day of week

Remarks

The valid day numbers are from 1 through 7.

Prototype (C/C++)

```
HRESULT SetDayShortName(  
long DayNumber,  
BSTR NewValue);
```

See Also

[Adding DTS Transformations](#)

[Day?ShortName Property](#)

[GetDayShortName Method](#)

[SetDayLongName Method](#)

DTS Programming

SetMonthLongName Method

The **SetMonthLongName** method sets the long (full) name for the specified month.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**SetMonthLongName**(*monthnumber*) = *string*

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>monthnumber</i>	Number of the month
<i>string</i>	Long (full) name of the specified month

Remarks

The valid month numbers are from 1 through 12.

Prototype (C/C++)

```
HRESULT SetMonthLongName(  
long MonthNumber,  
BSTR NewValue);
```

See Also

[Adding DTS Transformations](#)

[GetMonthLongName Method](#)

[Month??LongName Property](#)

[SetMonthShortName Method](#)

DTS Programming

SetMonthShortName Method

The **SetMonthShortName** method sets the short name (3-character abbreviation) for the specified month.

Applies To

[DataPumpTransformDateTimeString Object](#)

Syntax

object.**SetMonthShortName**(*monthnumber*) = *string*

Part	Description
<i>object</i>	Expression that evaluates to a DataPumpTransformDateTimeString object
<i>monthnumber</i>	Number of the month
<i>string</i>	Short name (3-character abbreviation) of the specified month

Remarks

The valid month numbers are from 1 through 12.

Prototype (C/C++)

```
HRESULT SetMonthShortName(  
long MonthNumber,  
BSTR NewValue);
```

See Also

[Adding DTS Transformations](#)

[GetMonthShortName Method](#)

[Month.ShortName Property](#)

[SetMonthLongName Method](#)

DTS Programming

ShowAddressBook Method

The **ShowAddressBook** method displays the address book user interface.

Applies To

[SendMailTask Object](#)

Syntax

[*Address* =] *object*.**ShowAddressBook**(*hwndParent*)

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object
<i>hwndParent</i>	Window handle of the parent window
<i>Address</i>	String to receive the address selected by the user

Remarks

hwndParent is the handle of the window on which the address book is to be displayed.

Prototype (C/C++)

```
HRESULT ShowAddressBook(  
    long hwndParent,  
    BSTR *Address);
```

DTS Programming

Uninitialize Method

The **Uninitialize** method clears all state information and releases all related objects, allowing the **Package** object to be reused.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax

Package.**Uninitialize**()

Part	Description
<i>Package</i>	Expression that evaluates to an object in the Applies To list

Remarks

The **Uninitialize** method clears the execution-related state of the **Package2** object, including any event handlers registered or any active sessions made by a **Connection** object. You must call **Uninitialize** before the **Package2** object is final-released because event handlers may contain a circular reference. All child objects of the **Package2** object must be released before you call **Uninitialize**.

In Microsoft® Visual Basic®, release object references by setting the corresponding object variables to **Nothing** or allow the variables to go out of scope. If you have multiple **Package** or **Package2** object variables, release all but one of them and execute the **Uninitialize** method against the one remaining reference.

This method does not affect the presence of any structural elements of the package, such as removing items that have been added to collections. To implement a new **Package** object, simply release all references to a prior one or its children, and then create a new one.

Prototype (C/C++)

```
HRESULT UnInitialize();
```

See Also

[Managing DTS Package Programs](#)

DTS Programming

UninitializeMAPI Method

The **UninitializeMAPI** method uninitializes the MAPI provider.

Applies To

[SendMailTask Object](#)

Syntax

object.**UninitializeMAPI**

Part	Description
<i>object</i>	Expression that evaluates to a SendMailTask object

Prototype (C/C++)

```
HRESULT UnInitializeMAPI();
```

See Also

[InitializeMAPI Method](#)

DTS Programming

Unlock Method

The **Unlock** method unlocks a **GlobalVariable2** object that had been previously locked with the **Lock** method.

Applies To

GlobalVariable2 Object
--

Syntax

globalvar.**Unlock**

Part	Description
<i>globalvar</i>	Expression that evaluates to a GlobalVariable2 object

Remarks

If the global variable is not already locked, the **Unlock** method does nothing, and no error occurs.

If the global variable was locked from a different thread than that which issued the **Unlock** method, an error occurs and the global variable is not unlocked.

Prototype (C/C++)

```
HRESULT Unlock( void );
```

See Also

[Adding DTS Lookups and Global Variables](#)

[Lock Method](#)

DTS Programming

Write Method

The **Write** method updates a value in a **PropertyBag** collection.

Applies To

[PropertyBag Object](#)

Syntax

object.**Write** *bstrPropertyName*, *Value*

Part	Description
<i>object</i>	Expression that evaluates to a PropertyBag object
<i>bstrPropertyName</i>	String identifying an exposed property by name
<i>Value</i>	Value of the property to be saved

Remarks

Generic BLOBs or objects are not supported as property values.

Prototype (C/C++)

```
HRESULT Write(  
    BSTR bstrPropertyName,  
    VARIANT Value);
```

See Also

[PersistPropertyBag Object](#)

[Read Method](#)

DTS Programming

WriteStringToLog Method

The **WriteStringToLog** adds a string to the log record that is being written for the step.

Applies To

[PackageLog Object](#)

Syntax

object.**WriteStringToLog** *LogString*

Part	Description
<i>object</i>	Expression that evaluates to a PackageLog object reference
<i>LogString</i>	String containing the log message that is to be appended to the log message for the step

Remarks

A log string is accumulated for each step. **WriteStringToLog** adds a new string to the log message, separated by a Newline character from the previously accumulated message. **WriteStringToLog** can be called any number of times during task execution.

The **WriteStringToLog** method is available to custom tasks, where the reference to the **PackageLog** object is a parameter of the task **Execute** method, which in a custom task, you must implement. It is also available in the Data Transformation Services (DTS) Microsoft® ActiveX® Script task, through the **DTSPackageLog** object.

Prototype (C/C++)

```
HRESULT WriteStringToLog( BSTR LogString );
```

See Also

[Retrieving DTS System, Package, and Log Data](#)

[WriteTaskRecord Method](#)

DTS Programming

WriteTaskRecord Method

The **WriteTaskRecord** adds a record to the server log table for the current task execution, and formats it for WriteStringToLog to write it to the log file.

Applies To

[PackageLog Object](#)

Syntax

object.**WriteTaskRecord** *ErrorCode*, *Description*

Part	Description
<i>object</i>	Expression that evaluates to a PackageLog object reference
<i>ErrorCode</i>	A long integer that can be an error code or other numeric information to be logged
<i>Description</i>	A string that can be an error description or other textual information to be logged

Remarks

The **WriteTaskRecord** method is available to custom tasks, where the reference to the **PackageLog** object is a parameter of the task **Execute** method, which in a custom task, you must implement. It is also available in the Data Transformation Services (DTS) Microsoft® ActiveX® Script task, through the **DTSPackageLog** object.

Prototype (C/C++)

```
HRESULT WriteTaskRecord(  
    long ErrorCode,  
    BSTR Description );
```

See Also

[Retrieving DTS System, Package, and Log Data](#)

[WriteStringToLog Method](#)

DTS Programming

Events

This section describes package events, which return information about the status of a Data Transformation Services (DTS) package execution.

DTS Programming

OnError Event

The **OnError** event indicates and provides information about an error. It allows the event handler to cancel task execution.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax (Visual Basic)

```
Sub objPackage_OnError(ByVal EventSource As String, ByVal ErrorCode As Long, _  
    ByVal Source As String, ByVal Description As String, ByVal HelpFile As String, _  
    ByVal HelpContext As Long, ByVal IDOfInterfaceWithError As String, _  
    ByVal pbCancel As Boolean)
```

Part	Description
<i>objPackage</i>	Object variable of a type specified in the Applies To list.
<i>EventSource</i>	Source of event being executed (for example, a step name).
<i>ErrorCode</i>	Error code of the failure.
<i>Source</i>	Source of error message (for example, an OLE DB provider description).
<i>Description</i>	Description of the error.
<i>HelpFile</i>	Help file name.
<i>HelpContext</i>	Help context ID.
<i>IDOfInterfaceWithError</i>	ID of the user interface returning the error, a globally unique identifier (GUID).
<i>pbCancel</i>	Boolean that specifies whether to cancel task execution.

Remarks

The **OnError** event occurs during the **Package2** object **Execute** method. An error can be raised by a Data Transformation Services (DTS) component or by a custom task. When an error occurs in execution, the **OnError** event is raised before return from the **Execute** method.

Use the *pbCancel* argument to direct package execution on error. When the **OnError** handler sets *Cancel* to TRUE, execution of the package is terminated on return from the error handler. When *Cancel* is FALSE, DTS package execution continues.

The **OnError** event is raised on each error occurring in a task or step. A step does not necessarily terminate after an error. For example, the **MaximumErrorCount** property of the **DataPumpTask2** object specifies the number of errors that can occur before the task is terminated.

For task types defined by DTS, the *EventSource* argument specifies a step name. Custom tasks choose what to return for *EventSource*. When an error is not caused by or associated with a step, the *EventSource* argument is empty.

If you need to raise this event from a custom task implemented in Microsoft® Visual Basic®, use this syntax:

```
pPackageEvents.OnError EventSource, ErrorCode, Source, Description, _  
    HelpFile, HelpContext, IDOfInterfaceWithError, pbCancel
```

pPackageEvents is a reference to the package events object, passed in as a parameter of the custom task **Execute** method that you have implemented. When the event returns, terminate task execution if *pbCancel* is TRUE.

Prototype (C/C++)

```
HRESULT OnError(  
    BSTR EventSource,  
    long ErrorCode,  
    BSTR Source,  
    BSTR Description,  
    BSTR HelpFile,  
    long HelpContext,  
    BSTR IDOfInterfaceWithError,
```

BOOL *pbCancel);

See Also

[Execute Method](#)

[Handling DTS Events and Errors](#)

DTS Programming

OnFinish Event

The **OnFinish** event indicates completion of a Data Transformation Services (DTS) task or step.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax (Visual Basic)

Sub *objPackage*_**OnFinish**(ByVal *EventSource* As String)

Part	Description
<i>objPackage</i>	Object variable of a type specified in the Applies To list.
<i>EventSource</i>	Source of event being executed (for example, a step name).

Remarks

For task types defined by DTS, the *EventSource* argument specifies a step name. Custom tasks choose what to return for *EventSource*. When an event is not caused by or associated with a step, the *EventSource* argument is empty.

If you need to raise this event from a custom task implemented in Microsoft® Visual Basic®, use this syntax:

pPackageEvents.**OnFinish** *EventSource*

pPackageEvents is a reference to the package events object, passed in as a parameter of the custom task **Execute** method that you have implemented.

Prototype (C/C++)

HRESULT OnFinish(BSTR EventSource);

See Also

[Execute Method](#)

[Handling DTS Events and Errors](#)

DTS Programming

OnProgress Event

The **OnProgress** event provides information about the progress of a Data Transformation Services (DTS) task.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax (Visual Basic)

```
Sub objPackage_OnProgress(ByVal EventSource As String, _  
    ByVal ProgressDescription As String, ByVal PercentComplete As Long, _  
    ByVal ProgressCountLow As Long, ByVal ProgressCountHigh As Long)
```

Part	Description
<i>objPackage</i>	Object variable of a type specified in the Applies To list.
<i>EventSource</i>	Source of event being executed (for example, a step name).
<i>ProgressDescription</i>	Description of task progress.
<i>PercentComplete</i>	Percent of task completed.
<i>ProgressCountLow</i>	Low 32 bits of units (for example, rows) completed.
<i>ProgressCountHigh</i>	High 32 bits of units (for example, rows) completed.

Remarks

If the percent completed cannot be reported, 0 is returned.

For task types defined by DTS, the *EventSource* argument specifies a step name. Custom tasks choose what to return for *EventSource*. When an event is not caused by or associated with a step, the *EventSource* argument is empty.

If you need to raise this event from a custom task implemented in Microsoft® Visual Basic®, use this syntax:

pPackageEvents.OnProgress EventSource, ProgressDescription, _
PercentComplete, ProgressCountLow, ProgressCountHigh

pPackageEvents is a reference to the package events object, passed in as a parameter of the custom task **Execute** method that you have implemented.

Prototype (C/C++)

```
HRESULT OnProgress(  
    BSTR EventSource,  
    BSTR ProgressDescription,  
    long PercentComplete,  
    long ProgressCountLow,  
    long ProgressCountHigh);
```

See Also

[Execute Method](#)

[Handling DTS Events and Errors](#)

DTS Programming

OnQueryCancel Event

The **OnQueryCancel** event terminates tasks. A Data Transformation Services (DTS) package raises this event only when it is safe to stop execution of the task. The event handler determines whether task execution should be terminated.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax (Visual Basic)

```
Sub objPackage_OnQueryCancel(ByVal EventSource As String, _  
    ByVal pbCancel As Boolean)
```

Part	Description
<i>objPackage</i>	Object variable of a type specified in the Applies To list.
<i>EventSource</i>	Source of event being executed (for example, a step name).
<i>pbCancel</i>	Boolean that specifies whether to cancel task execution.

Remarks

If *pbCancel* is set to TRUE by the event handler, DTS stops task execution and fails with error. This event may not occur if execution of the step or task completes quickly.

For task types defined by DTS, the *EventSource* argument specifies a step name. Custom tasks choose what to return for *EventSource*. When an event is not caused by or associated with a step, the *EventSource* argument is empty.

If you need to raise this event from a custom task implemented in Microsoft® Visual Basic®, use this syntax:

```
pPackageEvents.OnQueryCancel EventSource, pbCancel
```

pPackageEvents is a reference to the package events object, passed in as a parameter of the custom task **Execute** method that you have implemented. When the event returns, terminate task execution if *pbCancel* is TRUE.

Prototype (C/C++)

```
HRESULT OnQueryCancel(BSTR EventSource,  
BOOL *pbCancel);
```

See Also

[Execute Method](#)

[Handling DTS Events and Errors](#)

[KILL](#)

DTS Programming

OnStart Event

The **OnStart** event indicates the start of a Data Transformation Services (DTS) task or step.

Applies To

Package Object	Package2 Object
--------------------------------	---------------------------------

Syntax (Visual Basic)

Sub *objPackage*_**OnStart**(ByVal *EventSource* As String)

Part	Description
<i>objPackage</i>	Object variable of a type specified in the Applies To list.
<i>EventSource</i>	Source of event being executed (for example, a step name).

Remarks

For task types defined by DTS, the *EventSource* argument specifies a step name. Custom tasks choose what to return for *EventSource*. When an event is not caused by or associated with a step, the *EventSource* argument is empty.

If you need to raise this event from a custom task implemented in Microsoft® Visual Basic®, use this syntax:

```
pPackageEvents.OnStart EventSource
```

pPackageEvents is a reference to the package events object, passed in as a parameter of the custom task **Execute** method that you have implemented.

Prototype (C/C++)

```
HRESULT OnStart(BSTR EventSource);
```

See Also

[Execute Method](#)

[Handling DTS Events and Errors](#)

DTS Programming

Constants

This section describes a Data Transformation Services (DTS) package constants, which are enumerated data types. These constants are used as parameters and return values in DTS package properties and methods.

DTS Programming

DTSBulkInsert_DataFileType

The **DTSBulkInsert_DataFileType** constants specify the type of data file used in Bulk Insert operations.

Constant	Value	Description
DTSBulkInsert_DataFileType_Char	0	Char data file type
DTSBulkInsert_DataFileType_Native	1	Native data file type
DTSBulkInsert_DataFileType_WideChar	2	WideChar data file type
DTSBulkInsert_DataFileType_WideNative	3	WideNative data file type

See Also

[BulkInsertTask Object](#)

[DataFileType Property](#)

DTS Programming

DTSCustomTaskUIFlags

The **DTSCustomTaskUIFlags** constants specify flags indicating the type of user interface supported by the database custom task.

Constant	Value	Description
DTSCustomTaskUIFlags_Default	0	Default
DTSCustomTaskUIFlags_DoesCustomToolTip	1	Custom task supports custom tooltips

See Also

[GetUIInfo Method](#)

[CustomTaskUI Object](#)

DTS Programming

DTSDataPumpError

The **DTSDataPumpError** constants specify error ranges for Data Transformation Services (DTS) data pump execution.

Constant	Value	Description
DTSDataPump_E_AutoBufferInterfaceNotSupported	8273 (x2051)	A transform defining the specified source interface is not ISILockBytes .
DTSDataPump_E_AxScript_AbortPumpReturned	8502 (x2136)	A transformation DTSTransformState .
DTSDataPump_E_AxScript_BadTransformFunction	8259 (x2043)	Microsoft® ActiveX function was not found.
DTSDataPump_E_AxScript_CantChangeSrcCols	8260 (x2044)	ActiveX scripting script to change source columns.
DTSDataPump_E_AxScript_CantInitializeEngine	8262 (x2046)	ActiveX scripting engine cannot initialize the script engine.
DTSDataPump_E_AxScript_CantResetAfterInitialize	8261 (x2045)	The scripting engine cannot be reset after the engine is instantiated.
DTSDataPump_E_AxScript_InvalidPhaseColumnAccess	8500 (x2134)	The script attempted to access a column access phase that is not supported.
DTSDataPump_E_AxScript_NoPhaseFunction	8503 (x2137)	A script function is not supported for this transformation phase.
DTSDataPump_E_AxScript_ParseError	8258 (x2042)	ActiveX script parser error. ValidateSchema .
DTSDataPump_E_AxScript_RequiredParams	8256 (x2040)	ActiveX scripting engine requires the Text, Language, Parameters parameters to be specified in the IDTSDataPump interface.
DTSDataPump_E_AxScript_RunTimeError	8263 (x2047)	ActiveX scripting engine run-time error during script execution.

DTSDataPump_E_ AxScript_ValidateSchemaError	8257 (x2041)	A transformation : User-defined data types are not supp
DTSDataPump_E_BadTransformFlag	8210 (x2012)	Invalid or incomple value(s).
DTSDataPump_E_BadTransformPhase	8496 (x2130)	An invalid transfo via the Transform
DTSDataPump_E_ BadTransformStatusReturned	8211 (x2013)	Transform server DTSTransformS script did not retu pump task will be
DTSDataPump_E_CannotRebindColumn	8213 (x2015)	Binding informati has been specifiec cannot be respecif
DTSDataPump_E_ CannotTransformChapterColumns	8219 (x201B)	A column in a hie chapters (child ro transformed.
DTSDataPump_E_ColCountButNoCols	8198 (x2006)	A nonzero column column specificat
DTSDataPump_E_ColumnNameNotFound	8200 (x2008)	Column name not
DTSDataPump_E_ColumnOutOfRange	8201 (x2009)	Column ordinal w
DTSDataPump_E_Convert_BadBindInfo	8231 (x2027)	Incorrect binding
DTSDataPump_E_ Convert_BlobStorageNoInterface	8238 (x202E)	The required sour interface does not
DTSDataPump_E_Convert_ConversionFailed	8236 (x202C)	General conversio
DTSDataPump_E_Convert_ConversionInvalid	8235 (x202B)	Conversion invali pair.
DTSDataPump_E_Convert_DestNotNull	8230 (x2026)	Destination does n pair.
DTSDataPump_E_Convert_DestOverflow	8232 (x2028)	Destination overfl

DTSDataPump_E_ Convert_ProviderOwnedTypeMismatch	8237 (x202D)	DBMEMOWNEE mismatch for an a pair.
DTSDataPump_E_Convert_SourceInvalidLength	8234 (x202A)	Source data lengtl conversion to dest
DTSDataPump_E_Convert_SourceInvalidVariant	8233 (x2029)	Source variant inv
DTSDataPump_E_ Copy_NeedSrcAndDestColumns	8228 (x2024)	A DTSTransform specify no column source and destina
DTSDataPump_E_Copy_ValidateSchemaError	8229 (x2025)	ValidateSchema error information description. For m OnError Event an Method .
DTSDataPump_E_DataPumpNotReentrant	8207 (x200F)	Data pump imple operation and is tl
DTSDataPump_E_ DDQ_BadTransformStatusContext	8293 (x2065)	Transform status 1 DataDrivenQuer DataDrivenQuer
DTSDataPump_E_DDQ_DestColumnNeedsLength	8295 (x2067)	One or more desti variable-length an column size to be
DTSDataPump_E_ DDQ_DestColumnNotTransformed	8294 (x2066)	One or more desti had no transform
DTSDataPump_E_ DDQ_DestDoesNotSupportSQL	8296 (x2068)	The destination O implement the ID which is needed to
DTSDataPump_E_ DDQ_InformationNotSet	8293 (x2065)	Attempted to get c information when DataDrivenQuer
DTSDataPump_E_DDQ_NeedDeleteQuery	8291 (x2063)	Transform status 1 but no DeleteQue
DTSDataPump_E_DDQ_NeedInsertQuery	8289 (x2061)	Transform status 1 but no InsertQue

DTSDataPump_E_DDQ_NeedTransformStatus	8288 (x2060)	No query specific status.
DTSDataPump_E_DDQ_NeedUpdateQuery	8290 (x2062)	Transform status is not a valid value, but no UpdateQuery specified.
DTSDataPump_E_DDQ_NeedUserQuery	8292 (x2064)	Transform status is not a valid value but no UserQuery specified.
DTSDataPump_E_DDQ_TransformStatusForced	8297 (x2069)	Transformation is not a valid type.
DTSDataPump_E_DestColumnAlreadySpecified	8202 (x200A)	One or more destination columns already specified for transformation.
DTSDataPump_E_DestColumnReadOnly	8208 (x2010)	One or more destination columns are read-only; you must delete them or not include them.
DTSDataPump_E_DestinationBlobBinding	8224 (x2020)	In-memory destination blob binding (DBCOLUMNFLATTEN) must specify a non-zero value.
DTSDataPump_E_DestRowsetNotSupplied	8195 (x2003)	Destination column specified, but no rowset supplied.
DTSDataPump_E_IDataConvertRequired	8203 (x200B)	Transformation requires data conversion to be available on the destination from an OLE DB provider.
DTSDataPump_E_InvalidDataPumpOption	8499 (x2133)	DataPumpOptions property has an invalid value.
DTSDataPump_E_InvalidDTSBindMode	8215 (x2017)	Invalid or incompatible DTSBindMode value(s), or DTSBindMode incompatible with DestinationType .
DTSDataPump_E_InvalidFetchBufferSize	8214 (x2016)	FetchBufferSize property is nonzero.
DTSDataPump_E_InvalidSpecifyBlobDefaults	8273 (x2051)	Invalid DTSBlobDefault value(s).
DTSDataPump_E_InvalidStatusForPhase	8497 (x2131)	The transformation status is not a valid value that is invalid for the current phase.

DTSDataPump_E_ LastRowCantBeLessThanFirst	8275 (x2053)	The LastRow pro FirstRow propert
DTSDataPump_E_LineageVariableNotFound	8274 (x2052)	A global variable found.
DTSDataPump_E_LookupDupName	8218 (x201A)	A lookup name du exists.
DTSDataPump_E_MismatchColOrdAndName	8199 (x2007)	A column ordinal column name. Or distinguish betwe same name.
DTSDataPump_E_ MustSpecifyDestOrTransform	8209 (x2011)	A destination or o must be specified.
DTSDataPump_E_NonBlobStorageBind	8217 (x2019)	Non-BLOB (DBC columns cannot b
DTSDataPump_E_NotImplemented	8192 (x2000)	Method or proper
DTSDataPump_E_NotReentrant	8205 (x200D)	Data pump imple operation and is th
DTSDataPump_E_NullVariantIUnknown	8206 (x200C)	Passed variant is c contains a NULL
DTSDataPump_E_RowFailuresExceedLimit	8298 (x206A)	The number of er MaximumError
DTSDataPump_E_RowsetChangeMustInsert	8206 (x200E)	Destination IRow InsertRow (DBPF
DTSDataPump_E_RowsetsAlreadySet	8193 (x2001)	Can be set only be DataDrivenQuer
DTSDataPump_E_RowsetsNotSupplied	8196 (x2004)	NonNULL source destination rowse
DTSDataPump_E_SourceBlobBinding	8216 (x2018)	In-Memory Sourc (DBCOLUMNFL cannot be bound v and must specify . <i>cbInMemoryBlob</i> .
DTSDataPump_E_SourceColumnsRequired	8197 (x2005)	Source columns a transformations.

DTSDataPump_E_TransformDateTimeString_ErrorParsingInputData	8342 (x2096)	Error parsing the
DTSDataPump_E_TransformDateTimeString_ErrorParsingInputFormat	8340 (x2094)	Error parsing the property.
DTSDataPump_E_TransformDateTimeString_ErrorParsingOutputFormat	8341 (x2095)	Error parsing the property.
DTSDataPump_E_TransformDateTimeString_IndexOutOfRange	8337 (x2091)	The index used to names or month n
DTSDataPump_E_TransformDateTimeString_InvalidFormatString	8343 (x2097)	A string did not h
DTSDataPump_E_TransformDateTimeString_InvalidNameOrFormat	8338 (x2092)	A required proper to an empty string
DTSDataPump_E_TransformDateTimeString_InvalidY2KCutoff	8339 (x2093)	The ShortYear20 a value outside of
DTSDataPump_E_TransformDateTimeString_NeedSrcAndDestColumns	8336 (x2090)	Something other t column and exact was specified.
DTSDataPump_E_TransformReadFile_FileNotFound	8480 (x2120)	The file from whi read could not be
DTSDataPump_E_TransformReadFile_InvalidDestSchema	8482 (x2122)	The destination co DBTYPE_STR, I DBTYPE_BYTE;
DTSDataPump_E_TransformReadFile_InvalidSourceSchema	8481 (x2121)	The source colum DBTYPE_STR, I DBTYPE_BSTR.
DTSDataPump_E_TransformReadFile_NeedSrcAndDestColumns	8322 (x2082)	Something other t column and exact were specified.
DTSDataPump_E_TransformsAlreadySet	8194 (x2002)	Reserved.
DTSDataPump_E_TransformServerException	8212 (x2014)	Transform server
DTSDataPump_E_TransformString_ColumnCount	8321 (x2081)	A Trim String or l transformation sp column.

DTSDataPump_E_TransformString_ DestStringTrunc	8324 (x2084)	Destination trunc Lowercase String, String or Middle c when DTSTransformFla was not set.
DTSDataPump_E_TransformString_ NeedSrcAndDestColumns	8320 (x2080)	The number of so the number of des Lowercase String, String or Middle c
DTSDataPump_E_TransformString_ ValidateSchemaError	8323 (x2083)	The conversion fr column cannot be within the parame property, in a Low String, Trim Strin transformation.
DTSDataPump_E_TransformWriteFile_ FetchedNullFileName	8454 (x2106)	The source colum specification is N
DTSDataPump_E_TransformWriteFile_ FileAlreadyExists	8449 (x2101)	The file that is to ErrorIfFileExists
DTSDataPump_E_TransformWriteFile_ FileColumnNameNotFound	8450 (x2102)	The column name property could no
DTSDataPump_E_TransformWriteFile_ InvalidDestSchema	8452 (x2104)	The source colum has type other tha DBTYPE_WSTR
DTSDataPump_E_TransformWriteFile_ InvalidFileColumnName	8453 (x2105)	The FileColumn NULL or an empt
DTSDataPump_E_TransformWriteFile_ InvalidSourceSchema	8451 (x2103)	The source colum specification has t DBTYPE_STR, I DBTYPE_BYTE
DTSDataPump_E_TransformWriteFile_ NeedSrcAndDestColumns	8448 (x2100)	Something other t columns and no d specified for a Wr
DTSDataPump_E_UnsupportedPhase	8498 (x2132)	A Copy, Uppercas Trim String, Midd

	String, Read File had a phase other
--	--

See Also

[Handling DTS Events and Errors](#)

DTS Programming

DTSDesignerSettings

The **DTSDesignerSettings** constants specify settings that control whether features are visible or hidden for Data Transformation Services (DTS).

Constant	Value	Description
DTSDesigner_Default	0	Multiphase transformation features are not shown in DTS Designer.
DTSDesigner_ShowMultiPhaseTransforms	1	Multiphase transformation features are visible in DTS Designer.

See Also

[DesignerSettings Property](#)

DTS Programming

DTSExceptionFileOptions

The **DTSExceptionFileOptions** constants specify how Data Transformation Services (DTS) DataPump errors and exception rows are to be written to files.

Constant	Value	Description
DTSExceptionFile_AbortOnRowLogFailure	8192 (x2000)	Terminate the data pump if execution logging fails.
DTSExceptionFile_Ansi	256 (x0100)	File type is ANSI (uses ANSI code page).
DTSExceptionFile_DestRowFile	8	Destination exception rows are written to the destination exception file.
DTSExceptionFile_ErrorFile	2	Error rows are written to the error file.
DTSExceptionFile_OEM	512 (x0200)	File type is OEM (uses OEM code page).
DTSExceptionFile_Overwrite	4096 (x1000)	Data is overwritten, rather than appended, to file.
DTSExceptionFile_SingleFile70	1	Errors, source, and destination exception rows are all written to a single ANSI file.
DTSExceptionFile_SourceRowFile	4	Source exception rows are written to the source exception file.

DTSExceptionFile_Unicode	1024 (x0400)	File type is Unicode.
--------------------------	-----------------	-----------------------

See Also

[ExceptionFileOptions Property](#)

DTS Programming

DTSExecuteStatus

The **DTSExecuteStatus** constants return values (**int** or **long**) from data pump execution.

Constant	Value	Description
DTSTransformExec_AbortPump	4100 (x1004)	Pump terminated due to transform request or ErrorSink return.
DTSTransformExec_Error	4096 (x1000)	Indicates the status code is an error. Added to other items in this table.
DTSTransformExec_ErrorCountExceeded	4098 (x1002)	Pump terminated because too many rows had errors.
DTSTransformExec_OK	0	All rows copied (or skipped) without error.
DTSTransformExec_OKErrors	4097 (x1001)	Pump continued to completion, but encountered errors.

See Also

[IDTSDataPumpErrorSink](#)

DTS Programming

DTSFastLoadOptions

The **DTSFastLoadOptions** constants specify **FastLoad** options for the **DataPumpTask FastLoadOptions** property.

Constant	Value	Description
DTSFastLoad_CheckConstraints	2	Check constraints (default).
DTSFastLoad_Default	2	Specifies the default, same as check constraints
DTSFastLoad_KeepNulls	1	Keep NULLs.
DTSFastLoad_NoOptions	0	No options.
DTSFastLoad_TableLock	4	Lock table.

See Also

[DataPumpTask2 Object](#)

[FastLoadOptions Property](#)

DTS Programming

DTSForceMode

The **DTSForceMode** constants override the default handling of associated properties.

Constant	Value	Description
DTSForceMode_Always	1	Data Transformation Services (DTS) always overrides default handling of property.
DTSForceMode_Default	0	DTS can choose to override default handling of property.
DTSForceMode_Never	2	DTS never overrides default handling of property.

See Also

[ForceSourceBlobsBuffered Property](#)

DTS Programming

DTSFTPError

The **DTSFTPError** constants specify codes used to report errors in **DTSFTPTask** object execution.

Constant	Value	Description
DTSFTP_E_CancelExecution	1007 (x03EF)	A cancel execution request was received from the OnQueryCancel event.
DTSFTP_E_CopyFileError	1011 (x03F3)	An error occurred copying one of the specified files.
DTSFTP_E_ExceedeMaximumStringSize	1001 (x03E9)	A string property value exceeded the maximum allowed size (usually 256 characters).
DTSFTP_E_FTPExecutionError	1006 (x03EE)	A reference to the IDTSPackageEvents interface could not be obtained.
DTSFTP_E_IncorrectOverwriteCBSelect	1003 (x03EB)	An invalid value for the NonOverwritable property was specified.
DTSFTP_E_IncorrectRetryTimes	1004 (x03EC)	An invalid value for the NumRetriesOnSource property was specified.
DTSFTP_E_InternetConnectionError	1008 (x03F0)	Connection to the internet using Microsoft® Internet Explorer as agent failed.

DTSFTP_E_InvalidFileNameProperty	1012 (x03F4)	An error occurred parsing the SourceFilename property.
DTSFTP_E_InvalidSourceLocation	1005 (x03ED0)	An invalid value for the SourceLocation property was specified.
DTSFTP_E_OutOfMemory	1010 (x03F2)	A memory allocation for character string data failed.
DTSFTP_E_SiteConnectionError	1009 (x03F1)	Connection to the destination site failed, after connection to the internet succeeded.
DTSFTP_E_TooManyFilesSelected	1002 (x03EA)	The value used to set the SourceFilename property exceeded the allowed maximum (2000 characters).

See Also

[DTSFTPTask Object](#)

DTS Programming

DTSFTPSourceLocation

The **DTSFTPSourceLocation** constants are used with the **SourceLocation** property to specify the source location type for a **DTSFTPTask** object.

Symbol	Value	Description
DTSFTPSourceLocation_Directory	1	Source is a network directory.
DTSFTPSourceLocation_InternetSite	0	Source is an Internet site (default).

See Also

[SourceLocation Property](#)

DTS Programming

DTSIsolationLevel

The **DTSIsolationLevel** constants specify isolation levels for the **Package TransactionIsolationLevel** property.

Constant	Value	Description
DTSIsoLevel_Browse	256 (x0100)	Browse level
DTSIsoLevel_Chaos	16 (x0010)	Chaos level
DTSIsoLevel_CursorStability	4096 (x1000)	Cursor stability level
DTSIsoLevel_Isolated	1048576 (x00100000)	Isolated level
DTSIsoLevel_ReadCommitted	4096 (x1000)	Read committed level
DTSIsoLevel_ReadUncommitted	256 (x0100)	Read uncommitted level
DTSIsoLevel_RepeatableRead	65536 (x00010000)	Repeatable read level
DTSIsoLevel_Serializable	1048576 (x00100000)	Serializable level

See Also

[Package2 Object](#)

[TransactionIsolationLevel Property](#)

DTS Programming

DTSLineageOptions

The **DTSLineageOptions** constants specify Microsoft® SQL Server™ 2000 Meta Data Services lineage options for the **Package LineageOptions** property.

Constant	Value	Description
DTSLineage_AddLineageVariables	1	Add lineage variables.
DTSLineage_None	0	Provide no lineage (default).
DTSLineage_WriteToReposIfAvailable	2	Write to Meta Data Services if available.
DTSLineage_WriteToReposRequired	3	Write to Meta Data Services (required).

See Also

[LineageOptions Property](#)

[Package2 Object](#)

DTS Programming

DTSMQMessageType

The **DTSMQMessageType** constants are used with the **MessageType** property to specify the type of message defined by a **DTSMQMessage** object.

Symbol	Value	Description
DTSMQMessageType_DataFile	1	Message consists of the contents of a data file.
DTSMQMessageType_GlobalVariables	2	Message consists of the names and values of one or more Data Transformation Services (DTS) package global variables.
DTSMQMessageType_String	0	Message is a text string.

See Also

[MessageType Property](#)

DTS Programming

DTSMQStringMessageCompare

The **DTSMQStringMessageCompare** constants are used with the **StringCompareType** property to specify the type of comparison to be performed on a received string message by a **DTSMessageQueueTask** object. The **Description** column in the table below specifies the condition for successful comparison.

Symbol	Value	Description
DTSMQStringMessageCompare_Contains	3	Received message contains the comparison string.
DTSMQStringMessageCompare_Exact	1	Received message matches comparison string exactly, including case of letters.
DTSMQStringMessageCompare_IgnoreCase	2	Received message matches comparison string, ignoring case of letters.
DTSMQStringMessageCompare_None	0	No comparison is performed (default).

See Also

[StringCompareType Property](#)

DTS Programming

DTSMQType

The **DTSMQType** constants are used with the **TaskType** property to specify the type of the **DTSMessageQueueTask** object.

Symbol	Value	Description
DTSMQType_Receiver	1	Task object is to receive a single message.
DTSMQType_Sender	0	Task object is to send one or more messages.

See Also

[DTSMessageQueueTask Object](#)

[TaskType Property](#)

DTS Programming

DTSMQMError

The **DTSMQMError** constants specify codes used to report errors in **DTSMQueueTask** object execution.

Constant	Value	Description
DTSMQM_E_AssignmentIndexOutOfRange	1006 (x03EE)	The index for the Remove method of the DTSMQueueMessageCollection was out of range.
DTSMQM_E_CancelExecution	1023 (x03FF)	A cancel execution request was received from the OnQueryCancel event.
DTSMQM_E_CannotGetMessageQueueInfo	1015 (x03F7)	An error occurred while retrieving the label or the label ID of the Message Queueing message.
DTSMQM_E_CannotGetPackageInfo	1019 (x03FB)	An error occurred while retrieving the package name or version ID of the Transformation (DTS) package.
DTSMQM_E_CannotOpenMessageQueue	1014 (x03F6)	The queue specified in the QueuePath property could not be opened.
DTSMQM_E_CannotSendMessage	1017 (x03F9)	An error occurred while sending a message to the queue after it was successfully opened.
DTSMQM_E_DataFileSizeError	1009 (x03F1)	A data file message exceeds the maximum allowed size, 4 MB (MB).

DTSMSMQ_E_ErrorAccessMessageCollections	1016 (x03F8)	An error occurred accessing the data message to be saved.
DTSMSMQ_E_ErrorOpeningDataFile	1008 (x03F0)	An error occurred opening the file source of a data message.
DTSMSMQ_E_ErrorReadingDataFile	1010 (x03F2)	An error occurred reading the file that is the source of a data file message.
DTSMSMQ_E_ErrorSavingToDataFile	1027 (x0403)	An error occurred saving a data file message to a file.
DTSMSMQ_E_ErrorWritingDataFile	1021 (x03FD)	An error occurred writing a data file message to a receiving file.
DTSMSMQ_E_ExceededMaximumStringSize	1001 (x03E9)	A string property exceeds the maximum allowed size (in characters).
DTSMSMQ_E_IncorrectDataFileMessageRead	1020 (x03FC)	The length of a data file message differs from the expected length.
DTSMSMQ_E_IncorrectGlobalVariablesMessageRead	1025 (x0401)	An error occurred reading a global variable message that is not a global variable message.
DTSMSMQ_E_IncorrectMSMQMessageType	1007 (x03EF)	The message type specified by the MessageType or ReceiveMessage property is not valid.
DTSMSMQ_E_IncorrectStringCompareType	1003 (x03EB)	The comparison type specified by the CompareType property is not valid.

		StringCompare property is not v
DTSMSMQ_E_IncorrectTaskType	1002 (x03EA)	The task type sp the TaskType p not valid.
DTSMSMQ_E_IncorrectTimeoutValue	1004 (x03EC)	The timeout val specified by the ReceiveMessag property is not v (negative).
DTSMSMQ_E_InvalidAssignmentIndexVariantType	1005 (x03ED)	The index speci the Item or Ren method of the DTSMQMessa collection is not
DTSMSMQ_E_InvalidGlobalVariablesProperties	1024 (x0400)	An error occurre formatting the g variables messa sent.
DTSMSMQ_E_InvalidTaskProperties	1022 (x03FE)	A property requ the type of mess received was no specified.
DTSMSMQ_E_MessageQueueObjectsNotSupported	1012 (x03F4)	An error occurre creating a Mess Queuing object. Queuing was pr not properly ins
DTSMSMQ_E_NoMessageCollectionsFound	1011 (x03F3)	The DTSMQM collection does or contains zero
DTSMSMQ_E_NoMessageQueuePathSpecified	1013 (x03F5)	The required Qu property was no specified.
DTSMSMQ_E_OutOfMemory	1018 (x03FA)	A memory alloc character string

		failed.
DTSMSMQ_E_ReceiveMessageTimeout	1026 (x0402)	The timeout value specified by the ReceiveMessageTimeout property has elapsed and the task is being failed.

See Also

[DTSMessageQueueTask Object](#)

DTS Programming

DTSPackageError

The **DTSPackageError** constants specify codes used to report errors in Data Transformation Services (DTS) package creation and execution.

Constant	Value	Description
DTSPackage_E_AbandonedRowQueueDest	1076 (x0434)	This Step2 of DataPumpTask RowQueue destination. A corresponding object with a DataPumpTask RowQueue source not found or was skipped.
DTSPackage_E_AxScript_BadFunctionName	1020 (x03FC)	Microsoft® ActiveX scripting: Function not found.
DTSPackage_E_AxScript_CantAddGlobals	1021	ActiveX scripting: Cannot add global variables to ActiveX script.
DTSPackage_E_AxScript_CantInitializeEngine	1017	ActiveX scripting: Unable to initialize script execution engine.
DTSPackage_E_AxScript_ParseError	1019	ActiveX scripting: Parsing script.
DTSPackage_E_AxScript_RequiredParams	1018	ActiveX scripting: Language, and FunctionEntry required to be.
DTSPackage_E_AxScript_RunTimeError	1022	ActiveX scripting: Encountered a error during the execution of the.

DTSPackage_E_BadForceMode	1065	Invalid DTSE value.
DTSPackage_E_BadGUIDValue	1038	Invalid global identifier (GUID) specified.
DTSPackage_E_BadPackageDSORowsetTask	1075	PackageDSORowsetTask object is not a Step2 object or DataPumpTask .
DTSPackage_E_BadPrecedenceBasis	1026	Precedence basis is not a valid step status.
DTSPackage_E_BadPrecedenceStep	1036	Step specified precedence code was not found.
DTSPackage_E_BadPriorityClass	1042	Invalid priority class specified for the package.
DTSPackage_E_BadRelativePriority	1043	Invalid relative priority specified for a package.
DTSPackage_E_BadStepResultValue	1024	Invalid step result value.
DTSPackage_E_BadStepStatusValue	1023	Invalid step status value.
DTSPackage_E_BadStepTask	1027	Cannot find task associated with step.
DTSPackage_E_BadTaskResultValue	1025	Invalid task result value.
DTSPackage_E_CannotFindConnection	1031	Connection specified for task was not found.
DTSPackage_E_CannotPersistProperty	1062	Cannot store property values in file repository if the repository is empty, null, or contains objects.
DTSPackage_E_CantChangeLoadedPkgName	1060	Cannot change package name of a package that has been loaded from a file or Microsoft Server™ 2000.

		Data Services
DTSPackage_E_CantFindPackageInStg	1044	Cannot find sp package in the location speci
DTSPackage_E_CantFindVersionInStg	1040	Cannot find sp version of pac storage locati specified.
DTSPackage_E_CantSetCommandProps	1041	Cannot set co properties spe
DTSPackage_E_ColumnNeedsNameOrOrdinal	1007	Columns coll be indexed on or ordinal.
DTSPackage_E_ColumnsNotDescribed	1014	Unable to fetc meta data.
DTSPackage_E_ConnectionInUse	1030	Connection is being used by connection ca closed or reus
DTSPackage_E_ConnectionRequiresValidTaskName	1029	Acquiring a c requires a vali name.
DTSPackage_E_CreateProcTask_Timeout	1037	Process create did not termin the time speci
DTSPackage_E_DataDrivenQueryTask_RequireXforms	1077	DataDrivenQ object require transformation specified.
DTSPackage_E_DatatypeNotFound	1009	Invalid colum
DTSPackage_E_DescribeNeedsQuery	1013	Custom imple object's query must be set be columns can b described.
DTSPackage_E_DSO_CantRelaunchPackage	1073	Package assoc

		this OLE DB cannot be rela
DTSPackage_E_DSO_OnlyOneCommand	1074	This OLE DB supports only active comma
DTSPackage_E_DSO_OnlyOneRowset	1071	This OLE DB supports only active rowset.
DTSPackage_E_DSO_OnlyOneSession	1070	This OLE DB supports only active session
DTSPackage_E_DSO_ProviderStringRequired	1072	This OLE DB requires a PROVIDERS DATASOURC initialization p be set.
DTSPackage_E_EncryptStg_CantCreateOrWrite	1045	Cannot create elements or w streams while encrypted pac
DTSPackage_E_EncryptStg_CantOpenOrRead	1046	Cannot open s elements or re streams while encrypted pac
DTSPackage_E_EncryptStg_HandsOnStg	1047	Encrypted sto be released, c or reverted wh elements in st not been rela
DTSPackage_E_EncryptStg_PasswordNotMatching	1049	Password spe not match the operator passy
DTSPackage_E_EncryptStg_RequirePassword	1050	Cannot load e package withc password.

DTSPackage_E_EncryptStg_StreamTooLarge	1051	Encrypted stream cannot exceed 128 KB.
DTSPackage_E_EncryptStg_UnsupportedFlags	1048	Encrypted stream not support the specified to or create a storage
DTSPackage_E_ExecutionCanceled	1063	Execution was by user.
DTSPackage_E_FailedOnStepError	1064	Package failed step failed.
DTSPackage_E_MultiPackageStgNeedsID	1066	Specified storage contains multiple packages; load requires a name package ID.
DTSPackage_E_NameDup	1004	Object of the name already this object collection new object of name cannot be
DTSPackage_E_NameMustBeUniqueInStgFile	1067	Specified storage already contains package of this with a different ID.
DTSPackage_E_NameNotFound	1003	Object of specified was not found object collection
DTSPackage_E_NeedConnectionInfo	1010	Connection information was not specified custom implementation child of this object
DTSPackage_E_NeedDataDrivenQueries	1078	DataDrivenCollection object require one query (an

		associated col be specified.
DTSPackage_E_NeedDataDrivenQueryAndColumns	1079	Data-driven q specify the tex parameterized identify any c the associated collection) ne in the parame
DTSPackage_E_NeedDestinationColumnDefinitions	1069	Required colu definitions we supplied by th application.
DTSPackage_E_NoPackageDataFromServer	1068	No data for th package was r from the spec running an ins SQL Server.
DTSPackage_E_NoStepsDefined	1005	No steps have defined for th transformatio
DTSPackage_E_NoStepsToExecute	1012	No steps have added.
DTSPackage_E_NotImplemented	1001	Method or pro yet implemen
DTSPackage_E_NoXformDispatch	1061	Transformatio does not supp properties thro automation in
DTSPackage_E_ODBC_NeedConnectionInfo	1011	ODBC connec requires either source name c and driver nar
DTSPackage_E_OrdOutofRange	1002	Index value is range for this collection.

DTSPackage_E_PropertyStringTooLong	1059	This property hold a string l 255 characters
DTSPackage_E_PumpTask_RequireRowsetDataSrcInfo	1034	Data source or SQL statement required to obtain rowset.
DTSPackage_E_PumpTask_RequireSrcAndDestColumns	1033	Source and destination columns are not a Transformation object.
DTSPackage_E_PumpTask_RequireXformServer	1035	Transformation server transform server be provided for pump task transformation
DTSPackage_E_PumpTask_RequireXforms	1032	Data pump task transformation specified.
DTSPackage_E_RequireColumnNameAndOrdinal	1058	Specify a valid ordinal value column.
DTSPackage_E_RequireConnectionProperties	1028	Required Connection object property not been specified. Connection object
DTSPackage_E_RequireConnectionID	1057	Specify a valid connection.
DTSPackage_E_RequireNameForExecOrStg	1039	Cannot load, save or execute the package some objects or a name. Specify for these objects
DTSPackage_E_Security_InvalidPassword	1052	Password specified invalid. Specify password that

		characters in l
DTSPackage_E_Security_OperatorNotPrivileged	1054	Operator is pr only to load th and execute it
DTSPackage_E_Security_RequireBothPasswords	1053	Specify both o operator pass save a packag encrypted stor
DTSPackage_E_SQLTask_RequireSQL	1055	ExecuteSQL object require statements to specified.
DTSPackage_E_Step_CyclicDependency	1015	Step cannot b predecessor o
DTSPackage_E_UnknownOleDBProperty	1056	OLE DB prop specified is no by this OLE I provider.
DTSPackage_E_UsageBeforeDescribeOnly	1016	Reserved.
DTSPackage_E_WrongCollection	1008	Collection me only be added under the sam from which it acquired.

See Also

[Handling DTS Events and Errors](#)

[Package2 Object](#)

DTS Programming

DTSPackagePriorityClass

The **DTSPackagePriorityClass** constants specify the Microsoft® Win32® process priority class to be used when the Data Transformation Services (DTS) package is executed.

Constant	Value	Description
DTSPackagePriorityClass_High	3	High package priority
DTSPackagePriorityClass_Low	1	Low package priority
DTSPackagePriorityClass_Normal	2	Normal package priority

See Also

[PackagePriorityClass Property](#)

DTS Programming

DTSPackageType

The **DTSPackageType** constants are used with the **PackageType** property to identify the tool that created the database package.

Name	Value	Package Created By
DTSPkgType_ActiveDirectory	4	Microsoft® Active Directory™, the directory service included with Microsoft Windows® 2000.
DTSPkgType_Default	0	Custom program (or not set).
DTSPkgType_DTSDesigner	2	Data Transformation Services (DTS) Designer.
DTSPkgType_DTSWizard	1	The DTS Import/Export Wizard.
DTSPkgType_SQLReplication	3	Microsoft SQL Server™ 2000 replication function.

See Also

[PackageType Property](#)

DTS Programming

DTSRepositoryMetadataOptions

The **DTSRepositoryMetadataOptions** constants specify scanning and resolution options to use when storing a Data Transformation Services (DTS) package to Microsoft® SQL Server™ 2000 Meta Data Services.

Constant	Value	Description
DTSReposMetadata_Default	0	Package performs no scanner resolution.
DTSReposMetadata_RequireScannedCatalog	1	Package requires that any database objects must have been scanned into Meta Data Services.
DTSReposMetadata_ScanCatalogAlways	8	Package will scan all catalogs referenced, rescanning if already scanned.
DTSReposMetadata_ScanCatalogIfNotFound	4	Package will issue a scan on all catalogs that are not found already scanned.
DTSReposMetadata_UseScannedCatalogIfPresent	2	Package will use any scanned objects found; nonscanned references will create local objects.

See Also

[RepositoryMetadataOptions Property](#)

DTS Programming

DTSRepositoryStorageFlags

The **DTSRepositoryStorageFlags** constants specify Microsoft® SQL Server™ 2000 Meta Data Services options to use when saving or loading a Data Transformation Services (DTS) package.

Constant	Value	Description
DTSReposFlag_Default	0	Use database authentication to connect to Meta Data Services on an instance of SQL Server.
DTSReposFlag_UseTrustedConnection	256	Use Windows Authentication to connect to Meta Data Services on an instance of SQL Server.

See Also

[LoadFromRepository Method](#)

[RemoveFromRepository Method](#)

[SaveToRepository Method](#)

[SaveToRepositoryAs Method](#)

DTS Programming

DTSSQLObjectType

The **DTSSQLObjectType** constants specify object copying options for the **TransferObjectsTask**, **AddObjectForTransfer**, and **GetObjectForTransfer** methods.

Constant	Value	Description
DTSSQLObj_AllDatabaseObjects	4607 (x11FF)	System and database objects
DTSSQLObj_AllDatabaseUserObjects	4605 (x11FD)	User database objects
DTSSQLObj_Default	64 (x0040)	Defaults
DTSSQLObj_Rule	128 (x0080)	Rules
DTSSQLObj_StoredProcedure	16 (x0010)	Stored procedures
DTSSQLObj_SystemTable	2	System tables
DTSSQLObj_Trigger	256 (x0100)	Triggers
DTSSQLObj_UserDefinedDatatype	1	User-defined data types
DTSSQLObj_UserDefinedFunction	4096 (x1000)	User-defined functions
DTSSQLObj_UserTable	8	User tables
DTSSQLObj_View	4	Views

See Also

[AddObjectForTransfer Method](#)

[GetObjectForTransfer Method](#)

DTS Programming

DTSSQLServerStorageFlags

The **DTSSQLServerStorageFlags** constants specify Microsoft® SQL Server™ 2000 options to use when saving or loading a Data Transformation Services (DTS) package.

Constant	Value	Description
DTSSQLStgFlag_Default	0	Use SQL Server Authentication to connect to an instance of SQL Server.
DTSSQLStgFlag_UseTrustedConnection	256	Use Windows Authentication to connect to an instance of SQL Server.

See Also

[LoadFromSQLServer Method](#)

[LogServerFlags Property](#)

[RemoveFromSQLServer Method](#)

[SaveToSQLServer Method](#)

DTS Programming

DTStepExecResult

The **DTStepExecResult** constants specify the results from the execution of a step.

Constant	Value	Description
DTStepExecResult_Failure	1	Step execution failed.
DTStepExecResult_Success	0	Step execution succeeded.

See Also

[ExecutionResult Property](#)

DTS Programming

DTSStepExecStatus

The **DTSStepExecStatus** constants specify status codes that indicates the current step status.

Constant	Value	Description
DTSStepExecStat_Completed	4	Step execution is completed.
DTSStepExecStat_Inactive	3	Step execution is inactive.
DTSStepExecStat_InProgress	2	Step execution is in progress.
DTSStepExecStat_Waiting	1	Step is waiting to execute.

See Also

[ExecutionStatus Property](#)

DTS Programming

DTSSStepPrecedenceBasis

Steps can be executed after the precedence constraint is satisfied. The precedence constraint is based on either the execution status or execution result of another step. The **PrecedenceBasis** property indicates whether to use the step result or step status.

Constant	Value	Description
DTSSStepPrecedenceBasis_ExecResult	1	PrecedenceBasis based on execution result
DTSSStepPrecedenceBasis_ExecStatus	0	PrecedenceBasis based on execution status

See Also

[PrecedenceBasis Property](#)

DTS Programming

DTSStepRelativePriority

The **DTSStepRelativePriority** constants specify the Microsoft® Win32® thread priority to be used when a step is executed.

Constant	Value	Description
DTSStepRelativePriority_AboveNormal	4	Above normal thread priority
DTSStepRelativePriority_BelowNormal	2	Below normal thread priority
DTSStepRelativePriority_Highest	5	Highest thread priority
DTSStepRelativePriority_Lowest	1	Lowest thread priority
DTSStepRelativePriority_Normal	3	Normal thread priority

See Also

[RelativePriority Property](#)

DTS Programming

DTSScriptResult

The **DTSScriptResult** constants specify return codes to be used from the Microsoft® ActiveX® scripts associated with package steps. They should not be returned from the scripts associated with an **ActiveXScriptTask** object or **DataPumpTransformScript** or **DTSTransformScriptProperties2** transformations.

Constant	Value	Description
DTSScriptResult_DontExecuteTask	1	Do not execute task.
DTSScriptResult_ExecuteTask	0	Execute task.
DTSScriptResult_RetryLater	2	Retry execution later.

See Also

[ActiveXScript Property](#)

[Step2 Object](#)

DTS Programming

DTSTaskExecResult

The **DTSTaskExecResult** constants specify the result from the execution of a task.

Constant	Value	Description
DTSTaskExecResult_Failure	1	Task execution failed.
DTSTaskExecResult_RetryStep	2	Task execution is to be repeated.
DTSTaskExecResult_Success	0	Task execution succeeded.

See Also

[Execute Method](#)

DTS Programming

DTSTransfer_CopyDataOption

The **DTSTransfer_CopyDataOption** constants specify data copying options for the **TransferObjectsTask CopyData** property.

Constant	Value	Description
DTSTransfer_AppendData	2	Data that is copied is appended to existing tables.
DTSTransfer_DontCopyData	0	Schema only is copied.
DTSTransfer_ReplaceData	1	Data that is copied replaces existing data.

See Also

[CopyData Property](#)

DTS Programming

DTSTransfer_ScriptOption

The **DTSSQLServerStorageFlags** constants specify extended scripting options. They are equivalent to **SQLDMO_SCRIPT2_TYPE** constants used by SQL-DMO.

The **DTSTransfer_ScriptOption** constants specify scripting options for the **TransferObjectsTask ScriptOption** property.

Constant	Value	Description
DTSTransfer_Script_Aliases	16384	For users, script aliases.
DTSTransfer_Script_AppendToFile	256	Append to output file if i
DTSTransfer_Script_Bindings	128	Include rule/default bind
DTSTransfer_Script_ClusteredIndexes	8	Include clustered index (only).
DTSTransfer_Script_DatabasePermissions	32	Database (statement) per
DTSTransfer_Script_Default	4	Object creation only.
DTSTransfer_Script_DRI_All	532676608	All the foregoing (specif PrimaryObject gets just l
DTSTransfer_Script_DRI_AllConstraints	520093696	Bitmask of all constraint key, foreign key, unique,
DTSTransfer_Script_DRI_AllKeys	469762048	Bitmask of all key types foreign key, unique).
DTSTransfer_Script_DRI_Checks	16777216	Generated script creates CHECK constraints. Dir when declarative referen establishes dependency r Applies only when scrip Microsoft SQL Server™
DTSTransfer_Script_DRI_Clustered	8388608	Generated script creates Directs scripting when d referential integrity estat dependency relationship when scripting reference table.

DTSTransfer_Script_DRI_Defaults	33554432	Generated script include specified defaults. Direct declarative referential in dependency relationship when scripting reference table.
DTSTransfer_Script_DRI_ForeignKeys	134217728	Generated script creates constraints. Directs script declarative referential in dependency relationship when scripting reference table.
DTSTransfer_Script_DRI_NonClustered	4194304	Generated script creates indexes. Directs scripting referential integrity establish dependency relationship when scripting reference table.
DTSTransfer_Script_DRI_PrimaryKey	268435456	Generated script creates constraints. Directs script declarative referential in dependency relationship when scripting reference table.
DTSTransfer_Script_DRI_UniqueKeys	67108864	Generated script creates defined using a unique in scripting when declarative integrity establishes dependencies. Applies or references a SQL Server
DTSTransfer_Script_DRIIndexes	65536	Script DRI-generated in NoDRI is specified.
DTSTransfer_Script_DRIWithNoCheck	536870912	Script DRI with nocheck (DRI_All).
DTSTransfer_Script_Drops	1	Include object drops.
DTSTransfer_Script_IncludeHeaders	131072	Include descriptive head

		object script output.
DTSTransfer_Script_IncludeIfNotExists	4096	Include "if not exists" or
DTSTransfer_Script_Indexes	73736	Include all index creation
DTSTransfer_Script_NoCommandTerm	32768	Do not append "GO" to c
DTSTransfer_Script_NoDRI	512	Do not include DRI (use targeting a Microsoft SQ 6.0 or earlier installation
DTSTransfer_Script_NoIdentity	1073741824	Script with no IDENTIT as for replication).
DTSTransfer_Script_NonClusteredIndexes	8192	Includes nonclustered in only).
DTSTransfer_Script_ObjectPermissions	2	Includes object creation.
DTSTransfer_Script_OwnerQualify	262144	Owner-qualify DROP sta CREATE where possible
DTSTransfer_Script_Permissions	34	Both database and objec scripting users).
DTSTransfer_Script_PrimaryObject	4	Generate Transact-SQL c referenced component.
DTSTransfer_Script_SortedData	1048576	If the index or constraint append sorted_data.
DTSTransfer_Script_SortedDataReorg	2097152	Same as DTSTransfer_S but DTSTransfer_Script Reorg is used to preserve
DTSTransfer_Script_TimestampToBinary	524288	Converts timestamp col (for replication, and so o
DTSTransfer_Script_ToFileOnly	64	If not set, a string is retu nonnull, both are done).
DTSTransfer_Script_TransferDefault	2147061505	The default script type fo is combination of follow DTSTransfer_Script_Pri DTSTransfer_Script_Bir DTSTransfer_Script_Clu DTSTransfer_Script_No DTSTransfer_Script_Tri

		DTSTransfer_Script_To DTSTransfer_Script_Per DTSTransfer_Script_Inc DTSTransfer_Script_Ali DTSTransfer_Script_Inc DTSTransfer_Script_Ow DRI-restrictive flags. Th combined with PrimaryC These are used to includ specific DRI component
DTSTransfer_Script_Triggers	16	Include trigger creation (
DTSTransfer_Script_UDDTsToBaseType	1024	Converts user-defined da type when creating color
DTSTransfer_Script_UseQuotedIdentifiers	2147483648	Scripts with quoted iden cause the Transfer objec QUOTED_IDENTIFIEF destination.

See Also

[ScriptOption Property](#)

DTS Programming

DTSTransfer_ScriptOptionEx

The **DTSTransfer_ScriptOptionEx** constants specify extended scripting options. They are equivalent to **SQLDMO_SCRIPT2_TYPE** constants used by SQL-DMO.

Constant	Value	Description
DTSTransfer_ScriptEx_70Only	16777216 (x01000000)	Script the transfer of Microsoft® SQL Server™ 7.0 objects only.
DTSTransfer_ScriptEx_AgentAlertJob	2048 (x0800)	Include job in alert scripting.
DTSTransfer_ScriptEx_AgentNotify	1024 (x0400)	Script notification for SQLServerAgent alert object.
DTSTransfer_ScriptEx_AnsiFile	2	Generate ANSI output file.
DTSTransfer_ScriptEx_AnsiPadding	1	Explicitly SET ANSI PADDING on or off before the CREATE TABLE statement.
DTSTransfer_ScriptEx_EncryptPWD	128 (x0080)	Script encrypted password for logins.
DTSTransfer_ScriptEx_ExtendedProperty	4194304 (x00400000)	Include extended property scripting as part of object scripting
DTSTransfer_ScriptEx_FullTextCat	2097152 (x00200000)	Include full-text catalog scripting.
DTSTransfer_ScriptEx_FullTextIndex	524288 (x00080000)	Include full-text index scripting

		(table only).
DTSTransfer_ScriptEx_JobDisable	33554432 (x02000000)	Script Transact-SQL to disable the job at the end of job creation.
DTSTransfer_ScriptEx_LoginSID	1048576 (x00100000)	Include the security identification number (SID) for standard SQL Server logins.
DTSTransfer_ScriptEx_MarkTriggers	32 (x0020)	Mark system triggers. For replication, single table script only.
DTSTransfer_ScriptEx_NoCollatin	8388608 (x00800000)	Do not script the collation clause if source is an instance of SQL Server 2000.
DTSTransfer_ScriptEx_NoFG	16 (x0010)	Do not generate ON <filegroup>. For replication.
DTSTransfer_ScriptEx_NonStop	8	When error occurs during script file generation, log error and continue.
DTSTransfer_ScriptEx_NoWhatIfIndexes	512 (x0200)	Do not script What-If indexes (default: script out).
DTSTransfer_ScriptEx_OnlyUserTriggers	64 (x0040)	Only script user triggers. For replication, single table script only.

DTSTransfer_ScriptEx_SeparateXPs	256 (x0100)	Script XP to a separate file (Convert).
DTSTransfer_ScriptEx_TransferDefault	4112 (x1010)	Default.
DTSTransfer_ScriptEx_UnicodeFile	4	Generate UNICODE output file.

See Also

[ScriptOptionEx Property](#)

DTS Programming

DTSTransformationSetOptions

The **DTSTransformationSetOptions** constants specify the operating mode of a **Parallel Data Pump Task** object.

Constant	Value	Description
DTSTranSetOpt_Flattened	0	Transforms component rowsets independently without reference to chapter subsets (default).
DTSTranSetOpt_Hierarchical	1	Transforms component rowsets row at a time, using chapter subsets to transform child rowsets.
DTSTranSetOpt_DataDrivenQueries	4	Transforms component rowset as in flattened-mode, user queries to save data.

See Also

[Parallel Data Pump Task](#)

[TransformationSetOptions Property](#)

DTS Programming

DTSTransformFlags

The **DTSTransformFlags** constants specify the flags controlling transformation. They are used to set the **TransformFlags** property of the **Transformation** object. These values are used during schema validation, which occurs before any rows are transformed.

Constant	Value	Description
DTSTransformFlag_AllowDemotion	1	Allows the transfer to process potential overflows. Overflows occur during transformation and can be exceptioned. This value is set when the source values are within the range of the destination.
DTSTransformFlag_AllowLosslessConversion	512 (x0200)	Allows all conversions for which lossless conversion is possible (for example, Promotion, non-NULL to NULL, unsigned -> signed with no loss).
DTSTransformFlag_AllowNullChange	16 (x0010)	Allows the transfer to process source column allows NULL and destination column does not. If the source column containing NULL is exceptioned.
DTSTransformFlag_AllowNumericTruncation	8	Allows the transfer to process numeric truncation is possible. If the source is a floating-point numeric/decimal type and the destination is an integral type. Loss of precision without error, but integer truncation is an error.
DTSTransformFlag_AllowPromotion	2	Allows the transfer to process promotion in the data range. For example, from the source to the destination as I2->I4 or I4->float/double.
DTSTransformFlag_AllowSignChange	32 (x0020)	Allows the transfer to process sign change event that the source and destination have different sign conventions.

		signed versus unsigned numeric data. DTSTransformFlag_AllowStringTruncation may occur during a transformation.
DTSTransformFlag_AllowStringTruncation	4	Allows column (wide) character data to be truncated silently (for example, data from a char(60) to a char(50)).
DTSTransformFlag_Default	63 (x003F)	Includes the default flags: DTSTransformFlag_AllowStringTruncation, DTSTransformFlag_AllowOverflow, DTSTransformFlag_AllowConversion, DTSTransformFlag_AllowRoundtripLoss, DTSTransformFlag_AllowRoundtripLoss, DTSTransformFlag_AllowRoundtripLoss.
DTSTransformFlag_ForceConvert	128 (x0080)	Allows the conversion to occur even when the source and destination data types are fundamentally different. This flag is only effective when no other conversion flags are set.
DTSTransformFlag_PreserveDestRows	256 (x0100)	Causes the data pump to preserve destination row storage after processing. This allows table values to be reused by the data pump.
DTSTransformFlag_RequireExactType	64 (x0040)	Requires that the data type of the destination column be exactly the same as the source column (including precision and scale, fixed length, sign, and nullability).
DTSTransformFlag_Strict	0	Specifies no flags; the conversion is strict between exact types, although conversions between string and nonstring types are allowed and may cause errors. This flag can be overridden by DTSTransformFlag_RequireExactType, which is even stricter.

See Also

[Transformation2 Object](#)

[TransformFlags Property](#)

DTS Programming

DTSTransformPhaseEnum

The **DTSTransformPhaseEnum** constants specify the available Data Transformation Services (DTS) data pump transformation phases.

Constant	Value	Description
DTSTransformPhase_All	255 (x00FF)	Bitmask for all transform phases.
DTSTransformPhase_None	0	Specifies no phases.
DTSTransformPhase_OnBatchComplete	64 (x0040)	Occurs after a fast load batch completes, on success or failure.
DTSTransformPhase_OnInsertFailure	32 (x0020)	Occurs after an Insert operation or a data driven query fails.
DTSTransformPhase_OnInsertSuccess	16 (x0010)	Occurs after an Insert operation or a data driven query succeeds.
DTSTransformPhase_OnPumpComplete	128 (x0080)	Occurs once at end of Data Transformation Services (DTS) data pump operation.
DTSTransformPhase_OnTransformFailure	8	Occurs after transformation fails (for example, a conversion error).
DTSTransformPhase_PostSourceData	2	Occurs after all source rows processed.
DTSTransformPhase_PreSourceData	1	Occurs before first source row processed.
DTSTransformPhase_Transform	4	Occurs after source row is fetched,

	performs the primary transformation processing.
--	---

See Also

[CurrentPhase Property](#)

[TransformPhases Property](#)

DTS Programming

DTSTransformStatus

The **DTSTransformStatus** constants return values (**int** or **long**) from the **ActiveXScriptTask** object transformation.

Constant	Value	Description
DTSTransformStat_AbortPump	16384 (x4000)	Processing is terminated with the current row and DTSTransformExec_AbortPump returned from IDTSDataPump::Execute .
DTSTransformStat_DeleteQuery	64 (x0040)	The executed DELETE statement is the SetRowsetAndQueries property of the current rowset, with values from the current transformed destination row.
DTSTransformStat_DestDataNotSet	512 (x0200)	The current row is not written to the destination only if all transformations return this value. If present in the returned value from a transformation, it is removed before being passed to the next transformation through the DTSTransformPhaseInfo.TransformDataNotSet property.
DTSTransformStat_Error	8192 (x2000)	Indicates the transformation encountered an error.
DTSTransformStat_ErrorSkipRow	8194 (x2002)	Terminate further processing of this row due to error and call the error sink, but do not write to exception file.
DTSTransformStat_ExceptionRow	8448 (x2100)	Terminate further processing of this row due to exception and call the error sink, and write this row to exception file.
DTSTransformStat_Info	4096 (x1000)	Success with additional information. The application can process further based on its <i>pvTransformUserData</i> value (if present) and that knowledge with the transformation through OLE DB error records.
DTSTransformStat_InsertQuery	16 (x0010)	Executes the INSERT statement passed in the SetRowsetAndQueries property of the current rowset, with values from the current transformed destination row.

		transformed destination row.
DTSTransformStat_NoMoreRows	32768 (x8000)	The current row is the last to be processed. The current row is processed as specified by other transformation status values. This differs from DTSTransformStat_Abort in that no error is raised.
DTSTransformStat_OK	1	Default conversions (if any) succeed. Write the row to destination if specified, and call any error handlers.
DTSTransformStat_OKInfo	4097 (x1001)	Write row if destination specified; also call ErrorSink with information.
DTSTransformStat_SkipFetch	4	Do not fetch the next row; reexecute transforms against the current source and destination rows.
DTSTransformStat_SkipInsert	8	Do not write the current row to the destination.
DTSTransformStat_SkipRow	2	Terminate further processing of this row for nonerror reasons.
DTSTransformStat_SkipRowInfo	4098 (x1002)	Terminate further processing of this row; call ErrorSink with information.
DTSTransformStat_UpdateQuery	32 (x0020)	Executes the UPDATE statement part of the SetRowsetAndQueries property on the destination, with values from the currently transformed destination row.
DTSTransformStat_UserQuery	128 (x0080)	Executes the user query statement part of the SetRowsetAndQueries on the destination with values from the currently transformed destination row.

See Also

[IDTSDataPumpErrorSink](#)

DTS Programming

DynamicPropertiesTaskError

The **DynamicPropertiesTaskError** constants specify codes used to report errors in **DynamicPropertiesTask** object execution.

Constant	Value	Descri
DTSDynamicProperties_E_AssignmentIndexOutOfRange	1005 (x03ED)	An ind the Assign collect out of
DTSDynamicProperties_E_ConnectionIdNotFoundInPackage	1009 (x03F1)	No Conne object specifi Conne was fo
DTSDynamicProperties_E_CouldNotOpenFileForReading	1012 (x03F4)	The da specifi the pro value s could n openec readin;
DTSDynamicProperties_E_EnvironmentVariableNotFound	1015 (x03F7)	The enviro variabl specifi the pro value s could n found.
DTSDynamicProperties_E_ExceededMaximumPropertySize	1016 (x03F8)	The le a string proper

		exceed maxim (256 charac
DTSDynamicProperties_E_GlobalVariableNotFoundInPackage	1013 (x03F5)	The gl variabl specifi the pro value s does n in the packag
DTSDynamicProperties_E_InvalidAssignmentIndexVariantType	1018 (x03FA)	The Va specifi an Assign collect index o have a valid f purpos
DTSDynamicProperties_E_InvalidSourceType	1002 (x03EA)	The va specifi the Source proper define
DTSDynamicProperties_E_KeyNotFound	1006 (x03EE)	The ke specifi the pro value s could n found .ini file
DTSDynamicProperties_E_NoEnvironmentVariableProvided	1014 (x03F6)	No enviro

		variabl specifi the pro value s althou was re
DTSDynamicProperties_E_NoFileNameProvided	1003 (x03EB)	No dat name v specifi the pro value s althou was re
DTSDynamicProperties_E_NoGlobalVariableProvided	1008 (x03F0)	No glo variabl specifi the pro value s althou was re
DTSDynamicProperties_E_NoKeyProvided	1007 (x03EF)	No .ini key wa specifi the pro value s althou was re
DTSDynamicProperties_E_NoPropertyValueProvided	1011 (x03F3)	No pro value c retriev the .ini
DTSDynamicProperties_E_NoRowsReturnedFromQuery	1010 (x03F2)	No rov returne the que was th proper

		source
DTSDynamicProperties_E_NoSectionProvided	1004 (x03EC)	No .ini section specified although was re
DTSDynamicProperties_E_PackagePropertyNotFound	1001 (x03E9)	The sp package proper whose was to change defined
DTSDynamicProperties_E_SectionNotFound	1005 (x03ED)	The sp section not be in the .

See Also

[DynamicPropertiesTask Object](#)

DTS Programming

DynamicPropertiesTaskSourceType

The **DynamicPropertiesTaskSourceType** constants are used with the **SourceType** property to specify the type of source object that provides the value to which a Data Transformation Services (DTS) package object property will be set by the **DynamicPropertiesTask** object.

Symbol	Value	Description
DTSDynamicPropertiesSourceType_Constant	4	Source is a constant.
DTSDynamicPropertiesSourceType_DataFile	5	Source is the contents of a data file.
DTSDynamicPropertiesSourceType_EnvironmentVariable	3	Source is the value of a system environment variable.
DTSDynamicPropertiesSourceType_GlobalVariable	2	Source is the value of a Data Transformation Services (DTS) global variable within the package.
DTSDynamicPropertiesSourceType_IniFile	0	Source is the value of a key within an .ini file.
DTSDynamicPropertiesSourceType_Query	1	Source is a value returned by an SQL query.

See Also

SourceType Property

DTS Programming

Data Pump Interfaces

The Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) data pump is an OLE DB provider that provides the interfaces and methods to import, export, and transform data from an OLE DB data source to an OLE DB destination. The DTS data pump is the engine of the Transform Data task, Data Driven Query task, and Parallel Data Pump task, which is accessed through the **ParallelDataPumpTask** object.

These tasks create rowsets on the source and destination connections, then create an instance of the data pump to move rows between the source and destination rowsets. They also add instances of transformations to the data pump instance, transforming individual row data as it is moved from source to destination.

The data pump provides an extensible COM-based architecture that allows custom applications to perform complex data validations and transformations as data moves from source to destination. Third-party applications can also extend the data pump by creating custom COM objects that implement the **IDTSDataPumpTransform** interface to make use of the full power of Microsoft Win32® and COM. This allows an application to avoid the overhead of converting native data types to OLE DB variant data types and then converting them back again.

Data pump interfaces are implemented using Microsoft Visual C++® applications that include Dtspump.h.

Using the Data Pump Interfaces

An application must call **IDTSDataPump** interface methods in order. For example, these methods must be called in the following order:

1. **SetRowsets**
2. **AddTransform**
3. **Execute**

Other methods on the interface may be called in any order prior to a call to the

Execute method. Because the data pump represents a single execute operation, the data pump is not reentrant within calls. It returns an error if it is called as though it were reentrant.

Calls to **IDTSDDataPump** result in calls to the **IDTSDDataPumpTransform** interfaces in the following sequence:

1. Create an instance of the transformation server object and associated properties, if necessary.
2. Call **IDTSDDataPump::AddTransform**.
3. Call **IPersistPropertyBag::Load**, if **IPropertyBag** is specified.
4. Call **IDTSDDataPumpTransform::Initialize**.
5. Call **IDTSDDataPumpTransform::AddVariable** ("DTSErrorRecords").
6. Call **IDTSDDataPumpTransform::ValidateSchema**.
7. Call **IDTSDDataPump::Execute**.
8. Call **IDTSDDataPumpTransform::AddVariable**, including any object except **DTSErrorRecords**.
9. Call **IDTSDDataPumpTransform::Execute**.

DTS Programming

IDTSDataPump

The **IDTSDataPump** interface is used to specify OLE DB source and destination rowsets to be used, add transformations to the data pump, and execute the transfer of data by the data pump.

The **AddTransform** method returns an error if **IDTSDataPump::SetRowsets** has not been called, and **IDTSDataPump::SetRowsets** returns an error if any transformations have been added due to the **IDTSDataPumpTransform::ValidateSchema** method having been executed against them previously. **IDTSDataPump::InitNew** reinitializes the **IDTSDataPump** object.

DTS Programming

IDTSDataPump::AddTransform

The **AddTransform** method adds a transformation to the data pump.

Syntax

```
HRESULT AddTransform (  
LPBYTE pvUserData,  
LPCDTSTransformColumnsSpecification pColumns,  
DTSGuid ServerClsid,  
VARIANT ServerParameters,  
DTSTransformFlags dwFlags,  
IStorage *pIStorage );
```

Argument	Description
<i>pvUserData</i> [in]	Data that is passed to the event sink if an event occurs during a transformation.
<i>pColumns</i> [in]	Source and destination columns.
<i>ServerClsid</i> [in]	COM server ProgID, CLSID, or Iunknown.
<i>ServerParameters</i> [in]	Server parameters for the current transformation.
<i>dwFlags</i> [in]	Transformation column-validation flags.
* <i>pIStorage</i> [in]	Pointer to persistent storage of transformation properties.

Remarks

The data pump calls **CoCreateInstance** on the transformation object specified by *ServerClsid*. The transformation specified must support the **IDTSDataPumpTransform** interface and is responsible for verifying that the source and destination column values can be converted as specified.

The variant *ServerParameters* may be of type VT_UNKNOWN. If *ServerParameters* has the type VT_UNKNOWN, then the VT_UNKNOWN pointer is used to access the **IDTSDataPumpTransform** interface using

QueryInterface. This allows the data pump to use custom COM objects that have not been previously registered on the system, making it easier to distribute custom transformations. This is also used when providing a transformation server whose properties must be set programmatically by the data pump consumer prior to adding the transformation. In this case, the consumer creates an instance of the object, calls **IUnknown::QueryInterface**, and sets the properties.

If an **IUnknown** pointer is passed to the data pump, the data pump determines whether the pointer has previously been passed, and if so, appropriately handles calling **OnNextRow** only one time, regardless of how many times that pointer has been passed as a transformation server. In this case, *ServerParameters* specifies the shared object to handle. This is useful for objects that need to perform specific row-by-row aggregations without potentially conflicting with named variables added by **AddTransform**.

The transformation validation is controlled by the **DTSTransformFlags** constants specified in the parameter list. If different validations are required for different columns of the row, then a separate transformation must be added.

ServerClsid allows specification of the CLSID of the transformation server as a CLSID, ProgID, or as an existing **IUnknown** object implementing the **IDTSDataPumpTransform** interface.

pIStorage is used to set the properties of the transformation server, after an instance of the transformation is created, if necessary.

pvUserData allows information about a specific transformation to be passed to an event sink. This pointer is not passed to the transformation server; *ServerParameters* performs that task.

Transformations are always invoked in the order they are added.

See Also

[DTSTransformFlags](#)

DTS Programming

IDTSDataPump::AddTransformVariable

The **AddTransformVariable** method allows the data pump consumer to pass its global variables through to the executing transformation server.

Syntax

```
HRESULT AddTransformVariable( LPCOLESTR pwzName,  
BOOL bGlobal,  
VARIANT Variable );
```

Argument	Description
<i>pwzName</i> [in]	Variable name
<i>bGlobal</i> [in]	For Microsoft® ActiveX® scripts, indicates whether this variable's methods must be qualified by the object name
<i>Variable</i> [in]	Variable value, passed to and updatable by the transformation

Remarks

This method results in a call to **IDTSDataPumpTransform::AddVariable** during **IDTSDataPump::Execute** initialization, before transformations are actually executed. The data pump itself does not operate on these variables, but simply passes them through. If *bGlobal* is TRUE for an ActiveX script, then the methods of this variable are considered global and can be called directly, without qualifying by object name.

DTS Programming

IDTSDataPump::Execute

The **Execute** method executes the data pump and any transformations that have been defined.

Syntax

```
HRESULT Execute (  
LPBYTE pvUserData,  
ULARGE_INTEGER *puliRowsComplete,  
ULONG *pulErrorRows,  
LPDTSExecuteStatus pExecStatus );
```

Argument	Description
<i>pvUserData</i> [in]	User data passed back to event and error sinks
<i>*puliRowsComplete</i> [out]	Total number of source rows processed, including those skipped
<i>*pulErrorRows</i> [out]	Total number of error rows encountered
<i>pExecStatus</i> [out]	Pump return status

Remarks

Control is not returned to the caller until the last row has been processed or the data pump fails. Calls to the **Execute** method of each transformation server are made for each row in the order in which the transformations were added. Structured exception handling is placed around each call. If the called server returns an exception (for example, an access violation), an error is reported. The data pump reports all errors through the **IDTSDataPumpErrorSink::OnTransformError** event.

The **Execute** method returns DTSExecuteStatus constant values. Execute returns E_FAIL if it is terminated; DB_E_ERRORSOCURRED if the maximum error count is exceeded; DB_S_ERRORSOCURRED if errors occurred, but the maximum error count is not exceeded; and NOERROR if it completes with no errors.

See Also

[DTSExecuteStatus](#)

[IDTSDataPumpErrorSink::OnTransformError](#)

DTS Programming

IDTSDataPump::GetRowsets

The **GetRowsets** method is used to return interface pointers to the current source rowset and destination rowsetchange object.

Syntax

```
HRESULT GetRowsets ( IRowset **ppSrcRowset,  
IRowsetChange **ppDestRowsetChange );
```

Argument	Description
**ppSrcRowset [out]	Variable to receive a pointer to the source rowset
**ppDestRowsetChange [out]	Variable to receive a pointer to the destination rowsetchange object

Remarks

GetRowsets allows a consumer application to reuse an instance of the data pump.

DTS Programming

IDTSDDataPump::InitNew

The **InitNew** method is used to reset the data pump between successive executions.

Syntax

```
HRESULT InitNew();
```

DTS Programming

IDTSDataPump::SetFetchBufferSize

The **SetFetchBufferSize** method specifies the size of the buffer that the data pump uses to hold rows fetched from the source using the OLE DB **IRowset::GetNextRows** method.

Syntax

HRESULT SetFetchBufferSize (ULONG *cIn*);

Argument	Description
<i>cIn</i> [in]	Size of the GetNextRows HROW buffer

Remarks

The default buffer size is 1.

DTS Programming

IDTSDataPump::SetInsertCommitSize

The **SetInsertCommitSize** method specifies the number of rows inserted at a data destination between commit operations.

Syntax

HRESULT SetInsertCommitSize (ULONG *cIn*);

Argument	Description
<i>cIn</i> [in]	Number of successful rows inserted between commit operations, if supported by the OLE DB provider.

Remarks

The **SetInsertCommitSize** default setting is 0.

DTS Programming

IDTSDataPump::SetMaximumRowCount

The **SetMaximumRowCount** method specifies the maximum number of errors that can occur before the data pump fails.

Syntax

```
HRESULT SetMaximumRowCount ( ULONG cIn );
```

Argument	Description
<i>cIn</i> [in]	Maximum number of allowable error rows

Remarks

By default, the data pump fails on the first error.

DTS Programming

IDTSDataPump::SetProgressRowCount

The **SetProgressRowCount** method specifies how often notifications should be sent using the connection point.

Syntax

```
HRESULT SetProgressRowCount (  
    ULONG cIn );
```

Argument	Description
<i>cIn</i> [in]	Number of rows inserted between progress notifications

Remarks

If no event sink has been created, then no notifications are sent. If an event sink exists and **SetProgressRowCount** is not called, progress notifications are sent every 1000 rows.

DTS Programming

IDTSDataPump::SetRowsets

The **SetRowsets** method specifies the source and destination rowsets to be used by the data pump.

Syntax

```
HRESULT SetRowsets ( IRowset *pSrcRowset,  
IRowsetChange *pDestRowsetChange );
```

Argument	Description
<i>*pSrcRowset</i> [in]	Pointer to the source rowset
<i>*pDestRowsetChange</i> [in]	Pointer to the destination rowset

Remarks

The consumer application must open the rowsets before the data pump can use them.

The OLE DB provider must support DBPROP_UPDATABILITY:
DPBROP_UP_INSERT.

DTS Programming

IDTSDataPump2

The **IDTSDataPump2** interface extends the **IDTSDataPump** interface and supports features added to the Data Transformation Services (DTS) data pump for Microsoft® SQL Server™ 2000. The new features supported by **IDTSDataPump2** are:

- Support for execution of a single row or set of rows.
- Support for processing rows on a different thread from which previous rows in the rowset were processed.
- Support for multiphased operation of transformation servers

DTS Programming

IDTSDDataPump2::AddTransform2

The **AddTransform** method adds a new Transform with extended multiphase capability to the data pump.

Syntax

```
HRESULT AddTransform2(  
    LPCOLESTR pwzName,  
    LPBYTE pvUserData,  
    LPCDTSTransformColumnsSpecification pColumns,  
    DTSGuid ServerClsid,  
    VARIANT ServerParameters,  
    DTSTransformFlags dwFlags,  
    DTSTransformPhaseEnum ePhases,  
    IStorage *pIStorage );
```

Argument	Description
<i>pwzName</i> [in]	Transformation name
<i>pvUserData</i> [in]	Data that is passed to the event sink if an event occurs during a transformation
<i>pColumns</i> [in]	Structure specifying source and destination columns
<i>ServerClsid</i> [in]	ProgID or CLSID of transformation, which can be Unknown
<i>ServerParameters</i> [in]	Server parameters for the current transformation
<i>dwFlags</i> [in]	Transformation column-validation flags
<i>ePhases</i> [in]	Phases for which the transformation will be called
<i>pIStorage</i> [in]	Pointer to persistent storage of transformation properties

Remarks

The data pump calls **CoCreateInstance** on the transformation object specified by *ServerClsid*. The transformation specified must support the

IDTSDataPumpTransform2 interface and, by inheritance, **IDTSDataPumpTransform**. It must respond to **QueryInterface** for both **IDTSDataPumpTransform2** and **IDTSDataPumpTransform**.

See Also

[IDTSDataPump::AddTransform](#)

[IDTSDataPumpTransform2 Interface](#)

DTS Programming

IDTSDataPump2::ExecuteComplete

The **ExecuteComplete** method terminates a sequence of single-row data pump executions.

Syntax

```
HRESULT ExecuteComplete (  
    LPDTSExecuteInfo pExecInfo );
```

Argument	Description
<i>pExecInfo</i> [in/out]	Execution info to be passed into and returned from the data pump

Remarks

To execute one or more rows of the source rowset, the data pump caller first calls **ExecuteInit**, calls **ExecuteRow** for each row to be processed, and then calls **ExecuteComplete**. If **ExecuteComplete** is not called, Insert batches may not be completed successfully and transformation server processing may not be correctly cleaned up.

See Also

[IDTSDataPump2::ExecuteInit](#)

[IDTSDataPump2::ExecuteRow](#)

[IDTSDataPump2::GetExecuteInfo](#)

DTS Programming

IDTSDataPump2::ExecuteInit

The **ExecuteInit** method initiates a sequence of single-row data pump executions.

Syntax

```
HRESULT ExecuteInit (  
    LPBYTE pvUserData,  
    BOOL *pbEndOfRowset,  
    LPDTSExecuteInfo pExecInfo );
```

Argument	Description
<i>pvUserData</i> [in]	Data that is passed to an event sink if an event occurs
<i>pbEndOfRowset</i> [out]	TRUE if at end of source rowset
<i>pExecInfo</i> [in/out]	Execution info to be passed into and returned from the data pump

Remarks

To execute one or more rows of the source rowset, the data pump caller first calls **ExecuteInit**, calls **ExecuteRow** one or more times, and then calls **ExecuteComplete**. **ExecuteInit** returns TRUE in *pbEndOfRowset* if the end of the rowset has already been reached.

See Also

[IDTSDataPump2::ExecuteComplete](#)

[IDTSDataPump2::ExecuteRow](#)

[IDTSDataPump2::GetExecuteInfo](#)

DTS Programming

IDTSDataPump2::ExecuteRow

The **ExecuteRow** method causes the data pump to process a single-row of the source rowset.

Syntax

```
HRESULT ExecuteRow (  
    BOOL *pbEndOfRowset,  
    LPDTSExecuteInfo pExecInfo );
```

Argument	Description
<i>pbEndOfRowset</i> [out]	TRUE if at end of source rowset
<i>pExecInfo</i> [in/out]	Execution information to be passed into and returned from the data pump

Remarks

To execute one or more rows of the source rowset, the data pump caller calls **ExecuteInit**, calls **ExecuteRow** for each row to be processed, and then calls **ExecuteComplete**. **ExecuteRow** returns TRUE in *pbEndOfRowset* if the end of the rowset has been reached.

If **MaximumErrorRowCount** is greater than 0, **ExecuteRow** continues until a row is successfully processed or **MaximumErrorRowCount** is exceeded.

See Also

[IDTSDataPump2::ExecuteComplete](#)

[IDTSDataPump2::ExecuteInit](#)

[IDTSDataPump2::GetExecuteInfo](#)

DTS Programming

IDTSDDataPump2::GetExecuteInfo

The **GetExecuteInfo** returns information about data pump execution.

Syntax

```
HRESULT GetExecuteInfo(  
    LPDTSExecuteInfo pExecInfo );
```

Argument	Description
<i>pExecInfo</i> [in/out]	Execution information to be passed into and returned from the data pump

Remarks

GetExecuteInfo may be called at any point after **ExecuteInit**.

pExecInfo is a pointer to a **DTSExecuteInfo** structure. This structure is used to pass the following information to and from the data pump:

- Total number of source rows processed.
- Total number of error rows encountered.
- Data pump execution status.
- Transform status, used to select the data driven query to be executed.
- Handles to input source and destination chapters, used to specify the chapters where hierarchical rowset processing occurs.
- Handles to output source and destination rows, returned by the data pump to enable caller to specify where rowset processing should resume

at next **ExecuteRow** call.

See Also

[IDTSDataPump2::ExecuteRow](#)

DTS Programming

IDTSDataPump2::GetOptions

The **GetOptions** method returns extended data pump processing options.

Syntax

```
HRESULT GetOptions(  
    LPDTSDataPumpOptions peOptions );
```

Argument	Description
<i>peOptions</i> [out]	Extended data pump processing options

Remarks

For more information about the extended data pump processing options, see [IDTSDataPump2::SetOptions](#).

DTS Programming

IDTSDataPump2::SetExecuteThreadComplete

The **SetExecuteThreadComplete** method completes processing that must be done on the current thread, prior to the data pump being called on another thread.

Syntax

HRESULT SetExecuteThreadComplete();

Remarks

Some transformations may generate a thread-specific state that requires cleanup to be executed on that thread. For example, if **ExecuteInit** or **ExecuteRow** is called on a worker thread that is not the same as that on which **ExecuteComplete** is called, **SetExecuteThreadComplete** must be called on the worker thread when it has completed its operation (prior to **ExecuteComplete** being called).

Currently this is specific to the Microsoft® ActiveX® Script transformation, due to the requirement that **IActiveScript** execution state cannot be transferred across threads. In ActiveX Script transformations, any variables in the script, including global variables outside function scope, will be independent across multiple threads calling **ExecuteRow**. Calling this method causes **IDTSDataPumpTransform2::SetExecuteThreadComplete** to be called for all transformations supporting the **IDTSDataPumpTransform2** interface.

See Also

[IDTSDataPump2::ExecuteComplete](#)

[IDTSDataPump2::ExecuteInit](#)

[IDTSDataPump2::ExecuteRow](#)

DTS Programming

IDTSDataPump2::SetOptions

The **SetOptions** method sets extended data pump processing options.

Syntax

```
HRESULT SetOptions(  
    DTSDataPumpOptions eOptions );
```

Argument	Description
<i>eOptions</i> [in]	Extended data pump processing options

Remarks

The extended data pump processing options are:

Symbol	Value	Description
DTSDataPumpOpt_Default	0	Normal processing occurs.
DTSDataPumpOpt_AlwaysCommitFinalBatch	1	The final batch is committed even if the data pump fails. This is to support restartability.

See Also

[IDTSDataPump2::GetOptions](#)

DTS Programming

IDTSDataPumpErrorSink

The **IDTSDataPumpErrorSink** interface provides optional error sinks for the source, transformations, or destination to data pump consumers. Only one of the error sink methods is called for each row, and the method called indicates where the error was encountered.

The error sinks determine the appropriate response to the error. The response is one of the following:

- Terminate the data pump.
- Increment an error count and continue if the error count does not exceed the value of the **MaximumErrorCount** property.
- Continue the data pump operation.

The error information is written to the exception file and the package **OnError** event is raised if a handler has been provided.

See Also

[Handling DTS Events and Errors](#)

[MaximumErrorCount Property](#)

[OnError Event](#)

DTS Programming

IDTSDDataPumpErrorSink::OnBindingError

The **OnBindingError** method indicates that a binding error occurred in a call to the OLE DB **IAccessor::CreateAccessor** method.

Syntax

```
HRESULT OnBindingError (  
    LPBYTE pvExecUserData,  
    HRESULT hrError,  
    LPCDTSTransformColumnInfo pSourceRow,  
    const DBBINDSTATUS *pSourceDBBindStatus,  
    LPCDTSTransformColumnInfo pDestinationRow,  
    const DBBINDSTATUS *pDestinationDBBindStatus );
```

Argument	Description
<i>pvExecUserData</i> [in]	User data pointer passed to IDTSDDataPump::Execute .
<i>hrError</i> [in]	Error code from CreateAccessor : specifies the destination if <i>pDestinationRow</i> , the source if <i>pSourceRow</i> .
<i>pSourceRow</i> [in]	Pointer to the source row and binding information.
<i>pSourceDBBindStatus</i> [in]	Pointer to source binding status returns.
<i>pDestinationRow</i> [in]	Pointer to the destination row and binding information; NULL if from a source binding error.
<i>pDestinationDBBindStatus</i> [in]	Pointer to destination binding status returns; NULL if from a source binding error.

Remarks

OnBindingError can be useful when designing custom transformation servers. It is primarily used to diagnose binary large object (BLOB)-related problems.

DTS Programming

IDTSDataPumpErrorSink::OnDestinationError

The **OnDestinationError** method indicates that an error occurred during **InsertRow**.

Syntax

```
HRESULT OnDestinationError ( LPBYTE pvExecUserData,  
    LPDTSTransformColumnInfo pSourceRow,  
    LPDTSTransformColumnInfo pDestinationRow,  
    HRESULT hrError,  
    ULARGE_INTEGER uliRow,  
    ULONG cErrors,  
    BOOL *pbAbort );
```

Argument	Description
<i>pvExecUserData</i> [in]	User data pointer passed to IDTSDataPump::Execute .
<i>pSourceRow</i> [in]	Pointer to the source row and binding information; NULL if GetNextRows or GetData failed.
<i>PDestinationRow</i> [in]	Pointer to the destination row and binding information; NULL if prior to the execution of the transformation.
<i>hrError</i> [in]	Error code from the OLE DB or system call.
<i>uliRow</i> [in]	Number of the row that failed.
<i>cErrors</i> [in]	Number of error rows encountered during the transformation, including the current row.
<i>pbAbort</i> [out]	Set to TRUE by the error sink if this error should terminate IDTSDataPump::Execute . Otherwise, Execute continues until MaximumErrorRowCount is exceeded.

Remarks

OnDestinationError is called when an error is encountered sending a row to the

destination using **IRowsetChange::InsertRow**. Operation of the data pump continues unless the *pbAbort* flag is set to TRUE or the maximum allowable error count is exceeded.

DTS Programming

IDTSDataPumpErrorSink::OnSourceError

The **OnSourceError** method indicates that an error occurred during a **GetNextRows** or **GetData** operation.

Syntax

```
HRESULT OnSourceError (  
    LPBYTE pvExecUserData,  
    LPDTSTransformColumnInfo pSourceRow,  
    HRESULT hrError,  
    ULARGE_INTEGER uliRow,  
    ULONG cErrors,  
    BOOL *pbAbort );
```

Argument	Description
<i>pvExecUserData</i> [in]	User data pointer passed to IDTSDataPump::Execute .
<i>pSourceRow</i> [in]	Pointer to the source row and binding information; NULL if GetNextRows or GetData failed.
<i>hrError</i> [in]	Error code from the OLE DB or system call.
<i>uliRow</i> [in]	Number of the row that failed.
<i>cErrors</i> [in]	Number of error rows encountered during the transformation, including the current row.
<i>pbAbort</i> [out]	Set to TRUE by the error sink if this error should terminate IDTSDataPump::Execute . Otherwise, Execute continues until MaximumErrorRowCount is exceeded.

Remarks

OnSourceError is called when an error is encountered acquiring a source row. This is generally considered to be a fatal data pump error, and the *pbAbort* flag is set to TRUE.

DTS Programming

IDTSDataPumpErrorSink::OnTransformError

The **OnTransformError** method indicates that an error occurred during one or more transformations.

Syntax

```
HRESULT OnTransformError ( LPBYTE pvExecUserData,  
    LPBYTE pvTransformUserData,  
    IDTSDataPumpTransform *pTransformServer,  
    LPDTSTransformColumnInfo pSourceRow,  
    DTSTransformStatus TransformStatus,  
    HRESULT hrTransform,  
    ULARGE_INTEGER uliRow,  
    ULONG cErrors,  
    BOOL *pbAbort );
```

Argument	Description
<i>pvExecUserData</i> [in]	User data pointer passed to IDTSDataPump::Execute .
<i>pvTransformUserData</i> [in]	User data pointer passed to IDTSDataPump::AddTransform in DTSTransformColumnsSpecification parameter.
<i>pTransformServer</i> [in]	Pointer to the transformation server returning the error.
<i>pSourceRow</i> [in]	Pointer to the source row and binding information.
<i>TransformStatus</i> [in]	Transformation returned status code.
<i>hrTransform</i> [in]	DTSDataPumpTransform::Execute or DTSDataPumpTransform2::ProcessPhase HRESULT return code.
<i>uliRow</i> [in]	Number of the row that failed.
<i>cErrors</i> [in]	Number of error rows encountered during the transformation, including the current row.

<i>pbAbort</i> [out]	Set to TRUE by the error sink if this error should terminate IDTSDataPump::Execute . Otherwise, Execute continues until MaximumErrorRowCount is exceeded.
----------------------	--

Remarks

OnTransformError is called when an error is encountered transforming a row. Such an error may be encountered in one of the following ways:

- The transformation server encounters data that cannot be transformed. It reports this by returning an error transform status. This is considered normal and operation of the data pump continues unless the *pbAbort* flag is set to TRUE or the maximum allowable error count is exceeded.
- The transformation server returns an error in the HRESULT from **DTSDDataPumpTransform::Execute**. This is considered a fatal data pump error.
- The transformation server returns an exception. Any such exception is reported by the data pump as a normal row error, and operation continues.

DTS Programming

IDTSDataPumpProgressSink

The **IDTSDataPumpProgressSink** interface exposes the **OnIntervalComplete** method, which is a progress indicator event sink.

DTS Programming

IDTSDataPumpProgressSink::OnIntervalComplete

The **OnIntervalComplete** method is a progress indicator event sink that a custom application can use to indicate progress during a transformation.

Syntax

```
HRESULT OnIntervalComplete (  
LPBYTE pvExecUserData,  
ULARGE_INTEGER uliRowsComplete,  
BOOL *pbAbort );
```

Argument	Description
<i>pvExecUserData</i> [in]	User data pointer passed to IDTSDataPump::Execute .
<i>uliRowsComplete</i> [in]	Total source rows processed during the current transformation, including those skipped.
<i>*pbAbort</i> [out]	Indicates whether to terminate transformation execution; set to TRUE by the event sink to terminate IDTSDataPump::Execute .

DTS Programming

IDTSDataPumpTransform

The **IDTSDataPumpTransform** interface is retrieved and its methods are called by the data pump to perform individual transformations. All custom transformation COM objects must support the **IDTSDataPumpTransform** interface.

An instance of the **IDTSDataPumpTransform** interface is created by the *ServerClsid* being passed to **IDTSDataPump::AddTransform**. *ServerParameters* allows the data pump consumer to create a single instance of an **IDTSDataPumpTransform** implementation and pass it to multiple **IDTSDataPump::AddTransform** calls. This is used by custom transformation servers and can be used for internal operations as performing aggregations. The custom transformation server should track this using the *pvTransformServerData* parameter to optimize operations such as **AddVariable**, **OnRowComplete**, and so on.

DTS Programming

IDTSDataPumpTransform::AddVariable

The **AddVariable** method adds a variable to the execution context of a transformation server. **AddVariable** is always called one time before **ValidateSchema** to add the **IDTSErrorRecords** object.

Syntax

```
HRESULT AddVariable (  
LPBYTE pvTransformServerData,  
LPCOLESTR pwzName,  
BOOL bGlobal,  
VARIANT Variable );
```

Argument	Description
<i>pvTransformServerData</i> [in]	Transform server state data
<i>pwzName</i> [in]	Variable name
<i>bGlobal</i> [in]	For Microsoft® ActiveX® scripts, indicates whether the methods of this variable must be qualified by the object name
<i>Variable</i> [in]	Variable value; passed to and updatable by the transformation server

Remarks

AddVariable allows an application variable to be passed through to the transformation process. The data pump always calls this method to add the **DTSErrorRecords** object immediately after a call to **IDTSDataPumpTransform::Initialize** and before a call to **ValidateSchema**. This allows **ValidateSchema** to add errors to the OLE DB error records collection for the thread if columns are found to be in error.

For objects to be used in ActiveX scripts, the variable should be ignored if the variant is not of type VT_DISPATCH. This allows all variables to be passed to

all transformation servers. The *bGlobal* parameter indicates whether the properties and methods of the variable are to be added to the global namespace of the script; if so, the method can be called directly, instead of requiring qualification using the object name. Data pump-generated variables (for example, **DTSErrorInfo**, and the **DTSSource** and **DTSDestination** column collections) are always added with this flag set to FALSE to minimize the likelihood of name conflict.

DTS Programming

IDTSDataPumpTransform::Execute

The **Execute** method executes the transformation for a single row.

Syntax

```
HRESULT Execute ( LPBYTE pvTransformServerData,  
LPCDTSTransformColumnInfo pSrcColumnInfo,  
LPDTSTransformColumnInfo pDestColumnInfo,  
IDTSDataConvert *pIDTSDataConvert,  
LPDTSTransformStatus peTransformStatus );
```

Argument	Description
<i>pvTransformServerData</i> [in]	Transformation server state data
<i>pSrcColumnInfo</i> [in]	Source column and row data
<i>pDestColumnInfo</i> [in/out]	Destination column and row data
<i>pIDTSDataConvert</i> [in]	Pointer to the data conversion interface
<i>peTransformStatus</i> [out]	Result of the transformation

Remarks

This function is called by the data pump and executes the specified transformation. The **IDTSDataConvert** interface is supplied to provide a DTS-compatible conversion utility. For more information, search include file dtspump.h for **IDTSDataConvert**. dtspump.h is installed by default to X:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\.

Execute returns NO_ERROR except when a fatal failure occurs. When **Execute** returns an error, the data pump terminates. Normal errors such as data violations should be handled as a returned *peTransformStatus*, which results in a call to **IDTSDataPumpErrorSink::OnTransformError**. Because some OLE DB providers may have restrictions on the number of storage objects that may be open, **Execute** should release any BLOB storage objects that do not need to be

held during the destination insert upon completion, instead of waiting for **OnRowComplete**.

Column data should be written in the buffer referenced by **DTSColumnData.pvData** of the element for the destination column, at the offset specified by **DBBINDING.obValue**. You can instead store a pointer to the data at this location if you add the DBTYPE_BYREF flag to **DBBINDING.wType**. For more information and a description of these fields, see [Column Information Structures in DTS Transformations](#).

Transformations that implement both **IDTSDDataPumpTransform** and **IDTSDDataPumpTransform2** typically implement **Execute** by calling **IDTSDDataPumpTransform2::ProcessPhase** with NULL for the *pPhaseInfo* parameter.

See Also

[IDTSDDataPumpTransform2::ProcessPhase](#)

DTS Programming

IDTSDataPumpTransform::Initialize

The **Initialize** method allows the transformation server to initialize its state for the current transformation.

Syntax

```
HRESULT Initialize (  
    LPCOLESTR pwzName,  
    VARIANT ServerParameters,  
    LPBYTE *ppvTransformServerData);
```

Argument	Description
<i>pwzName</i> [in]	Transformation name
<i>ServerParameters</i> [in]	Parameters to server for this transformation
<i>ppvTransformServerData</i> [out]	Transformation server state data

Remarks

An instance of the transformation server object is created by *ServerClsid* and is passed to **IDTSDataPump::AddTransform**, unless this is an **IUnknown** object.

ServerParameters allows the Data Transformation Services (DTS) data pump to create an instance of an **IDTSDataPumpTransform** implementation and pass it to multiple **IDTSDataPump::AddTransform** calls. This is only used by custom transformation servers and can be helpful for internal operations such as aggregations. The transformation server should track this in the *pvTransformServerData* parameter to optimize operations such as **AddVariable**, **OnRowComplete**, and so on.

The output *ppvTransformServerData* is passed to all subsequent methods, so a single instance of a transformation server can be used to implement multiple separate transformations. Generally, you will need to allocate the storage whose

reference you return in *ppvTransformServerData*. You need to release the storage in **OnTransformComplete**, or in **ValidateSchema**, if **ValidateSchema** returns an error.

See Also

[IDTSDataPump::AddTransform](#)

[IDTSDataPumpTransform::AddVariable](#)

[IDTSDataPumpTransform::OnRowComplete](#)

[IDTSDataPumpTransform::OnTransformComplete](#)

[IDTSDataPumpTransform::ValidateSchema](#)

DTS Programming

IDTSDataPumpTransform::OnRowComplete

The **OnRowComplete** method is called after every successful fetch operation, allowing the transformation server to free per-row allocations and client-owned data in both source and destination rows.

Syntax

```
HRESULT OnRowComplete ( LPBYTE pvTransformServerData,  
LPDTSTransformColumnInfo pSrcColumnInfo,  
LPDTSTransformColumnInfo pDestColumnInfo,  
IDTSDataConvert *pIDTSDataConvert,  
DTSTransformStatus eTransformStatus,  
HRESULT hrInsert );
```

Argument	Description
<i>pvTransformServerData</i> [in]	Transform server state data
<i>pSrcColumnInfo</i> [in/out]	Source column and row data
<i>pDestColumnInfo</i> [in/out]	Destination column and row data
<i>pIDTSDataConvert</i> [in]	Pointer to the data conversion interface
<i>eTransformStatus</i> [in]	Result of Execute
<i>hrInsert</i> [in]	Result of IRowsetChange::InsertRow

Remarks

After a successful fetch operation, the data pump calls **Execute** and attempts to insert the row into the destination using **IRowsetChange::InsertRow**, if specified. **OnRowComplete** is called for every successful fetch operation, regardless of whether **Execute** or **InsertRow** succeeded or failed. If **OnRowComplete** returns FAILED, the data pump terminates.

hrInsert indicates whether **IRowsetChange::InsertRow** succeeded.

eTransformStatus indicates whether errors occurred that resulted in no call to **InsertRow**. OLE DB requires that **IRowsetChange::InsertRow** release any storage objects contained in the row; therefore, the transformation server must be careful not to call *pIDTSDataConvert->ClearBindingData* on a storage object that has already been released.

DTS Programming

IDTSDataPumpTransform::OnTransformComplete

After all rows have been transformed or the data pump has been terminated due to errors, the **OnTransformComplete** method is called to allow the transformation server to release all allocations made for the transformation.

Syntax

```
HRESULT OnTransformComplete ( LPBYTE pvTransformServerData,  
LPDTSTransformColumnInfo pSrcColumnInfo,  
LPDTSTransformColumnInfo pDestColumnInfo,  
IDTSDataConvert *pIDTSDataConvert );
```

Argument	Description
<i>pvTransformServerData</i> [in]	Transform server state data
<i>pSrcColumnInfo</i> [in/out]	Source column and row data
<i>pDestColumnInfo</i> [in/out]	Destination column and row data
* <i>pIDTSDataConvert</i> [in]	Pointer to the data conversion interface

Remarks

OnTransformComplete is called only if **ValidateSchema** completes successfully. **OnTransformComplete** is not called if the transformation implements **IDTSDataPumpTransform2** and **IDTSDataPumpTransform2::GetTransformServerInfo** indicates the transformation supports **DTSTransformPhase_OnPumpComplete** phase.

See Also

[IDTSDataPumpTransform2::GetTransformServerInfo](#)

[IDTSDataPumpTransform2::ProcessPhase](#)

DTS Programming

IDTSDDataPumpTransform::ValidateSchema

The **ValidateSchema** method validates the schema of the source and destination columns to be transformed.

Syntax

```
HRESULT ValidateSchema ( LPBYTE pvTransformServerData,  
LPCDTSTransformColumnInfo pSrcColumnInfo,  
LPCDTSTransformColumnInfo pDestColumnInfo,  
IDTSDDataConvert *pIDTSDDataConvert,  
DTSTransformFlags eTransformFlags );
```

Argument	Description
<i>pvTransformServerData</i> [in]	Transform server state data
<i>pSrcColumnInfo</i> [in/out]	Source column and row data
<i>pDestColumnInfo</i> [in/out]	Destination column and row data
* <i>pIDTSDDataConvert</i> [in]	Pointer to the data conversion interface
<i>eTransformFlags</i> [in]	Input flags for transformation validation and execution

Remarks

Pointers to source and destination column information structures are passed to **ValidateSchema**. The transformation can then validate data types, preventing unintended row transfers if the types are incompatible. If the transformation does not validate at this time (for example, due to a complex conversion performed on the row data), then it simply returns **NO_ERROR**. A validation failure should return an appropriate failure code, such as **DB_E_SCHEMAVIOLATION**, or one defined by the transformation server because the data pump cannot proceed.

The pointer to the **IDTSDDataConvert** interface is passed in to allow the destination to verify that conversion between the source and destination data types is possible and to indicate any special conditions that may be encountered.

The transformation flags define the data type promotion, demotion, and conversions that are allowed. Additional custom properties may be defined through a custom interface, in which case the object (such as an **IUnknown** object) must be created and edited prior to the object being passed to **IDTSDataPump::AddTransform**.

DTS Programming

IDTSDataPumpTransform2

The **IDTSDataPumpTransform2** interface must be implemented by all transformations that are to run with Microsoft® SQL Server™ 2000. It supports multi-phase transform operations. Transformations that implement **IDTSDataPumpTransform2** must still implement **IDTSDataPumpTransform**.

DTS Programming

IDTSDataPumpTransform2::GetTransformServerInfo

The **GetTransformServerInfo** method returns information that describes the functionality of the transformation.

Syntax

```
HRESULT GetTransformServerInfo (   BSTR *pstrHelpString,  
    LPDTSTransformPhaseEnum peSupportedPhases );
```

Argument	Description
<i>pstrHelpString</i> [out]	Description of the transformation implementation
<i>peSupportedPhases</i> [out]	Phases supported by this transformation

Remarks

This method is called before **IDTSDataPumpTransform2::Initialize**. It returns a textual description that can be used as a help string and a bitmask that contains flags for the supported phases. These flags are values from **DTSTransformPhaseEnum**.

Any output parameter can be NULL if that information is not desired by the caller. **GetTransformServerInfo** must be prepared to handle these.

IDTSDataPumpTransform2::ProcessPhase is not called for a phase specified by **GetTransformServerInfo** unless the value specified for the **TransformPhases** property of the **Transformation2** object also specifies the phase. For packages built in Data Transformation Services (DTS) Designer, specify phases on the **Phases** tab of the **Transformation Options** dialog box.

See Also

[IDTSDataPumpTransform2::Initialize](#)

[IDTSDataPumpTransform2::OnTransformComplete](#)

[IDTSDDataPumpTransform2::ProcessPhase](#)

[TransformPhases Property](#)

DTS Programming

IDTSDataPumpTransform2::PreValidateSchema

The **PreValidateSchema** method validates the schema of the source and destination columns to be transformed. It is called from Data Transformation Services (DTS) Designer at the time the transformation is being configured.

Syntax

```
HRESULT PreValidateSchema ( LPCDTSTransformColumnMetadata  
pSrcMetadata,  
    LPCDTSTransformColumnMetadata pDestMetaData,  
    DTSTransformFlags eTransformFlags,  
    DTSTransformPhaseEnum ePhases );
```

Argument	Description
<i>pSrcMetadata</i> [in]	Source column meta data
<i>pDestMetaData</i> [in]	Destination column meta data
<i>eTransformFlags</i> [in]	Input flags for transformation validation and execution
<i>ePhases</i> [in]	Phases this transformation is expected to support

Remarks

PreValidateSchema provides validation at the time the package is built, rather than when it is executed. It is called from DTS Designer when the user selects the **Only Show Valid Transformations** check box in the **Create New Transformation** dialog box and when the **Transformation Options** dialog box is closed. It is passed the meta data for the source and destination columns, the transformation flags and a bitmask defining the phases the transformation is expected (by the caller) to support.

Any level of validation can be provided, up to and including that done by **IDTSDataPumpTransform::ValidateSchema**. However, **Only Show Valid Transformations** simply removes the transformation name from the **Create New Transformation** dialog box, so complex validations may leave the user

wondering why the transformation is not valid.

The transformation flags, which use values from **DTSTransformFlags**, define the data type promotion, demotion and conversions that are allowed. However, **PreValidateSchema** is not called when they are changed from the **Transformation Flags** dialog box, so you may want to defer validation involving the flags to **ValidateSchema**.

The source and destination column meta data structure **DTSTransformColumnMetadata** has these fields.

Field	Description
cColumns	Count of source or destination columns.
rgDBCColumnInfo	Pointer to an array of OLE DB DBCOLUMNINFO structures. There is one array element for each column.

For more information about the **DBCOLUMNINFO** structure, see [Column Information Structures in DTS Transformations](#).

See Also

[DTSTransformFlags](#)

[IDTSDataPumpTransform::ValidateSchema](#)

DTS Programming

IDTSDDataPumpTransform2::ProcessPhase

The **ProcessPhase** method executes a phase of the transformation for a single row.

Syntax

```
HRESULT ProcessPhase ( LPBYTE pvTransformServerData,  
    LPCDTSTransformColumnInfo pSrcColumnInfo,  
    LPDTSTransformColumnInfo pDestColumnInfo,  
    IDTSDDataConvert *pIDTSDDataConvert,  
    LPCDTSTransformPhaseInfo pPhaseInfo,  
    LPDTSTransformStatus peTransformStatus );
```

Argument	Description
<i>pvTransformServerData</i> [in]	Transformation server state data
<i>pSrcColumnInfo</i> [in]	Source column and row data
<i>pDestColumnInfo</i> [in/out]	Destination column and row data
<i>pIDTSDDataConvert</i> [in]	Pointer to the data conversion interface
<i>pPhaseInfo</i> [in]	Transform phase information structure
<i>peTransformStatus</i> [out]	Result of the transformation

Remarks

This function is called by the data pump and executes a phase of the transformation. The phase is identified by a code from **DTSTransformPhaseEnum** in a field of *pPhaseInfo*, a **DTSTransformPhaseInfo** structure. The following table lists the transformation phases that are available.

Phase	Description
DTSTransformPhase_PreSourceData	Called before first fetch of source data; destination codes can be returned in <i>p</i> to write a destination row.

	DTSTransformStat_SkipFetch can be and write multiple rows.
DTSTransformPhase_Transform	Transforms source columns to destination. Performs same function as IDTSDDataPumpTransform::Execute
DTSTransformPhase_OnTransformFailure	Called when DTSTransformPhase_Transform returns DTSTransformStat_Error or DTSTransformStat_ExceptionRow. The <i>peTransformStatus</i> overrides that from DTSTransformPhase_Transform. The and data driven queries (DDQs) will be <i>peTransformStatus</i> so directs.
DTSTransformPhase_OnInsertSuccess	Called on success of the insert/DDQ. operations cannot be specified in <i>peTransformStatus</i> if the destination row has already been v
DTSTransformPhase_OnInsertFailure	Called on failure of the insert operation query. Destination operations cannot be <i>peTransformStatus</i> (for example, the i and data driven query cannot be retrie
DTSTransformPhase_OnBatchComplete	Called after the success or failure of a The batch size is defined by the Insert property.
DTSTransformPhase_PostSourceData	Called after last row of source data has and transformed. Valid destination code returned in <i>peTransformStatus</i> to write row, but no source data is available. DTSTransformStat_SkipFetch can be and write multiple rows.
DTSTransformPhase_OnPumpComplete	Called when data pump operation completes success or failure, if the transformation phase. It is called in place of IDTSDDataPumpTransform::OnTransform . The <i>pPhaseInfo</i> structure is available.

ProcessPhase is only called for the phases that are returned in the

peSupportedPhases parameter of

IDTSDDataPumpTransform2::GetTransformServerInfo. When there are multiple transformations, each phase is executed for all the transformations, in the order in which they were created, before moving on to the next phase.

Note **ProcessPhase** is not called for a phase specified by **GetTransformServerInfo** unless the value specified for the **TransformPhases** property of the **Transformation2** object also specifies the phase. For packages built in Data Transformation Services (DTS) Designer, specify phases on the **Phases** tab of the **Transformation Options** dialog box. The one exception is **DTSTransformPhase_OnPumpComplete**, for which **ProcessPhase** called (in place of **OnTransformComplete**) whenever **GetTransformServerInfo** specifies the phase is supported.

The source and destination column data is described by the **DTSTransformColumnInfo** structures referenced by *pSrcColumnInfo* and *pDestColumnInfo*. Column data should be written in the buffer referenced by **DTSColumnData.pvData** of the element for the destination column, at the offset specified by **DBBINDING.obValue**. You can instead store a pointer to the data at this location if you add the **DBTYPE_BYREF** flag to **DBBINDING.wType**. For more information and a description of these fields, see [Column Information Structures in DTS Transformations](#).

Transformations that implement both **IDTSDDataPumpTransform** and **IDTSDDataPumpTransform2** typically implement **Execute** by calling **IDTSDDataPumpTransform2::ProcessPhase** with **NULL** for the *pPhaseInfo* parameter.

The **IDTSDDataConvert** interface provides a DTS-compatible conversion utility. For more information, search include file **dtspump.h** for **IDTSDDataConvert**. **dtspump.h** is installed by default to `c:\Program Files\Microsoft SQL Server\80\Tools\DevTools\include\`.

See Also

[DTSTransformPhaseEnum](#)

[DTSTransformPhaseInfo Object](#)

[IDTSDDataPumpTransform::Execute](#)

[IDTSDDataPumpTransform::OnTransformComplete](#)

[IDTSDDataPumpTransform2::GetTransformServerInfo](#)

[InsertCommitSize Property](#)

[TransformPhases Property](#)

DTS Programming

IDTSDataPumpTransform2::SetExecuteThreadComp

The **SetExecuteThreadComplete** method performs any deallocations and cleanup that might be necessary because the Data Transformation Services (DTS) data pump instance (and thus the transformation) is going to continue execution on another thread. It is called when **IDTSDataPump2::SetExecuteThreadComplete** is called.

Syntax

HRESULT SetExecuteThreadComplete();

The method has no parameters.

Remarks

Data pump execution can be segmented so that it processes one or more rows on a thread when **IDTSDataPump2::ExecuteRow** is called, then processes additional rows when **ExecuteRow** is called on another thread. The DTS tasks supplied with Microsoft® SQL Server™ 2000 do not cross threads like this, but custom tasks can do so. Custom tasks doing this must call **IDTSDataPump2::SetExecuteThreadComplete** on the original thread before calling **ExecuteRow** on the new thread. **IDTSDataPump2::SetExecuteThreadComplete** calls **IDTSDataPumpTransform2::SetExecuteThreadComplete** for each transformation.

In most cases, the transformation does not need to do anything. Only when the transformation has invoked a thread-affinitive process, it may need to close that process before re-opening it on the new thread. For example, if the transformation uses the ActiveScripting engine, it needs to make a separate copy of the scripting engine instance, then call **IActiveScript::Close** on the old thread, then initialize the scripting engine instance copy on the new thread.

DTS Programming

IDTSDataPumpTransform2::SetExtendedInfo

The **SetExtendedInfo** method is reserved for future use.

Syntax

```
HRESULT SetExtendedInfo ( IUnknown *pUnkExtendedInfo );
```

Argument	Description
<i>pUnkExtendedInfo</i> [in]	Object supplying extended information

Remarks

If you implement **IDTSDataPumpTransform2**, you must implement **SetExtendedInfo** even though it is not called. Simply return **NO_ERROR**.

When implemented, **SetExtendedInfo** will be called prior to **IDTSDataPumpTransform2::ValidateSchema**. *pUnkExtendedInfo* will be a pointer to an object that will QueryInterface to one or more interfaces that supply extended information. These will need to be released in or prior to **IDTSDataPumpTransform2::OnTransformComplete**.

See Also

[IDTSDataPumpTransform2::OnTransformComplete](#)

[IDTSDataPumpTransform2::ValidateSchema](#)

DTS Programming

Transform Status Enumerations

Dtspump.h and Dtspump.dll expose these constant enumerations.

Enumeration	Description
DTSDataPumpError	Error codes generated by the data pump.
DTSExecuteStatus	Return values from data pump execution.
DTSTransformFlags	Flags that control the conversions performed by transformations.
DTSTransformPhaseEnum	Codes that denote the transformation phases.
DTSTransformStatus	Return codes that Microsoft® ActiveX® Script Transformations can generate.

DTS Programming

DTS Programming Samples

The following samples illustrate Microsoft® SQL Server™ 2000 Data Transformation Services (DTS) application development in Microsoft Visual C++® and Microsoft Visual Basic®. They also include DTS packages.

Sample	Description
CustomTaskNoUI	Active Template Library (ATL) template. Implements the framework for a DTS custom task that does not support a user interface.
CustomTaskWithUI	ATL template. Implements the framework for a DTS custom task that supports a user interface.
CustomTransform	ATL template. Implements the framework for a DTS custom transformation.
DTS Custom Transformation Sample	C++ sample. Concatenates a number of ANSI source columns into a single destination column.
DTS Custom Task	C++ sample. Creates and registers a DTS custom task that is similar to the CreateProcessTask object.
DTSCopy	C++ sample. Copies a single source column into a destination column of the same type.
DTSStrings	C++ sample. Reformats two source columns that are assumed to be a first and last name into a single destination column.
DTSTskGVUpdate	C++ sample. Displays and updates the value of a global variable through a user interface.
DTSTskPropIcon	C++ sample. Displays a message.
Packages	DTS package sample. Demonstrates how to create and execute packages and how to solve typical business problems.
Complex Transformation Sample from SQL Server to Excel	Visual Basic sample. Copies data from the pubs database to a Microsoft Excel spreadsheet, transforming it into a Microsoft PivotTable®.
DTS Package Sample	Visual Basic sample. Creates a package from a

Supporting Multiple Source and Destination Providers	variety of data sources and destinations.
DTSActiveScriptTask	Visual Basic sample. Demonstrates how to run a Microsoft Visual Basic Scripting Edition (VBScript) script as part of a DTS Task.
DTSApplicationObject	Visual Basic sample. Illustrates some of the information that can be obtained from the DTS Application object.
DTSAppObject	Visual Basic sample. Demonstrates use of the DTS Application object on the local server.
DTSBulkInsertTask	Visual Basic sample. Demonstrates how to use DTS to perform a bulk insert operation from a flat text file.
DTSCopyDatabase	Visual Basic sample. Demonstrates how to use the DTS TransferObjectsTask object to copy a database.
DTSExecProcess	Visual Basic sample. Demonstrates how to run a Win32 application from a DTS task.
DTSExecSQLTask	Visual Basic sample. Demonstrates how to execute an SQL statement while running a DTS package.
DTSExecutePackage	Visual Basic sample. Demonstrates how to execute programmatically a DTS package that has been saved in .dts format to a structured storage file.
DTSFTPTask	Visual Basic sample. Demonstrates how to use DTS to copy non-SQL Server files from a source to a destination.
DTSPackageInfo	Visual Basic sample. Illustrates some of the information that can be obtained from the DTS Application.GetPackageInfos method.
DTSTransferObjectsTask	Visual Basic sample. Demonstrates how to use DTS to transfer various types of SQL Server objects from the pubs database to the pubs2 database.

FoodMart2000	Visual Basic sample. Demonstrates how to use the Visual Basic file output from the DTS import/export wizard to convert the FoodMart 2000.mdb database to SQL Server 2000.
Pub2Pubs	Visual Basic sample. Copies the Authors table from the pubs database to the pubs2 database while performing several operations on various fields.
Simple DTS Package Sample Using Visual Basic	Visual Basic sample. Demonstrates how to build and execute a DTS package.
Simple Transformation Sample Between Two SQL Server Tables	Visual Basic sample. Copies data from a source table to a destination table after a Transact-SQL script is used to create a sample table in the pubs database.

To install the samples during SQL Server installation

1. On the **Setup Type** page, select **Custom**.
2. On the **Select Components** page, under **Components**, select **Code Samples**.

Samples are installed as a self-extracting file. To extract the samples, double-click Unzip_dts.exe, located at C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\Dts. All samples include a project file applicable to the language used.

Prerequisites

C++ samples require Visual C++ version 6.0. Visual Basic samples require Visual Basic version 6.0.

See Also

[Samples](#)

DTS Programming

CustomTaskNoUI

This sample is an Active Template Library (ATL) template for a Data Transformation Services (DTS) custom task that does not support a custom user interface, but instead uses the default property grid in DTS Designer.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\DevTools\Samples\Dts\ATLTemplates\CustomTaskNoUI

Running the Sample

To install this ATL template, do the following:

1. Copy all the files in the C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\Dts\ATLTemplates\CustomTaskNoUI directory except DTSCuTsk.reg to the ATL template directory.

The default location for this directory is C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template\ATL\.

2. Run DTSCuTsk.reg.

For more information about how to implement a custom task framework using this template, see [Building a Custom Task from the ATL Custom Task Basic Template](#).

See Also

[Implementing and Testing a DTS Custom Task](#)

[DTS Programming Samples](#)

DTS Programming

CustomTaskWithUI

This sample is an Active Template Library (ATL) template for a Data Transformation Services (DTS) custom task that supports a custom user interface.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\ATLTemplates\CustomTaskWithUI

Running the Sample

To install this ATL template, do the following:

1. Copy all the files in the C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\Dts\ATLTemplates\CustomTaskWithUI directory except DTSCuTskUI.reg to the ATL template directory.

The default location for this directory is C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template\ATL\.

2. Run DTSCuTskUI.reg.

For more information about how to implement a custom task framework using this template, see [Building a Custom Task with User Interface from the ATL Custom Task Templates](#).

See Also

[Implementing and Testing a DTS Custom Task](#)

[DTS Programming Samples](#)

DTS Programming

CustomTransform

This sample is an Active Template Library (ATL) template for a Data Transformation Services (DTS) custom transformation.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\ATLTemplates\CustomTransform

Running the Sample

To install this ATL template, do the following:

1. Copy all the files in the C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\ATLTemplates\CustomTransform
directory, except DTSCuXFmUI.reg, to the ATL template directory.

The default location for this directory is C:\Program Files\Microsoft
Visual Studio\Common\MSDev98\Template\ATL\
2. Run DTSCuXFmUI.reg.

For more information about how to implement a custom transformation framework using this template, see [Building a Custom Transformation from the ATL Custom Transformation Template](#).

See Also

[Implementing and Testing a DTS Custom Transformation](#)

[DTS Programming Samples](#)

DTS Programming

DTS Custom Transformation Sample

This Microsoft® Visual C++® sample creates and registers a custom transformation. It concatenates a number of ANSI string source columns into a single destination column.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\CustomTransforms\DTSXForm

Running the Sample

1. In the Visual C++ development environment, open and run DTSXForm.dsw.
2. On the **Build** menu, click **Build DTSXForm.dll** to build the project.
3. In Microsoft SQL Server™ Enterprise Manager, right-click **Data Transformation Services**, click **Properties**, and then click **Refresh Cache**, if caching is active.
4. Include the custom transformation in a Transform Data task in DTS Designer.

If you are compiling the example transform for use under Microsoft Windows® 98, be sure you compile using a non-UNICODE configuration. To set the configuration, use the **Set Active Configuration** option on the **Build** menu.

See Also

[Implementing and Testing a DTS Custom Transformation](#)

[DTS Programming Samples](#)

DTS Programming

DTS Custom Task

This Microsoft® Visual C++® sample creates and registers a Data Transformation Services (DTS) custom task that is similar to the **CreateProcessTask** object.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\CustomTasks\DTStask

Running the Sample

1. In the Visual C++ development environment, open and run Dtstask.dsw.
2. On the **Build** menu, click **Build dtstask.dll** to build the project.
3. In Microsoft SQL Server™ Enterprise Manager, right-click **Data Transformation Services**, click **Properties**, and then click **Refresh Cache**, if caching is active.
4. Include the custom task in a DTS package.

See Also

[Implementing and Testing a DTS Custom Task](#)

[DTS Programming Samples](#)

DTS Programming

DTSCopy

This Microsoft® Visual C++® sample creates and registers a simple custom transformation. It copies a single source column of any simple type into a destination column of the same type.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\CustomTransforms\DTSCopy

Running the Sample

1. In the Visual C++ development environment, open and run DTSXForm.dsw.
2. On the **Build** menu, click **Build DTSXForm.dll** to build the project.
3. In Microsoft SQL Server™ Enterprise Manager, right-click **Data Transformation Services**, click **Properties**, and then click **Refresh Cache**, if caching is active.
4. Include the custom transformation in a Transform Data task in DTS Designer.

For more information about how this example is implemented, see [DTS Custom Transformation Example: Copy One Column](#).

See Also

[Implementing and Testing a DTS Custom Transformation](#)

[DTS Programming Samples](#)

DTS Programming

DTSStrings

This Microsoft® Visual C++® sample creates and registers a custom transformation. It reformats two source columns that are assumed to be a first and last name into the form of Last, First in a single destination column. The source and destination columns can be ANSI or wide-character strings. Conversion between ANSI and wide characters is performed, as necessary.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\CustomTransforms\DTSStrings

Running the Sample

1. In the Visual C++ development environment, open and run DTSStrings.dsw.
2. On the **Build** menu, click **Build DTSStrings.dll** to build the project.
3. In Microsoft SQL Server™ Enterprise Manager, right-click **Data Transformation Services**, click **Properties**, and then click **Refresh Cache**, if caching is active.
4. Include the custom transformation in a Transform Data task in DTS Designer.

For more information about how this example is implemented, see [DTS Custom Transformation Example: Format Names](#).

See Also

[Implementing and Testing a DTS Custom Transformation](#)

[DTS Programming Samples](#)

DTS Programming

DTSTskGVUpdate

This Microsoft® Visual C++® sample is a Data Transformation Services (DTS) custom task with a user interface. It displays and allows the user to update the value of a global variable. The global variable name is specified as a custom task property.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\CustomTasks\DTSTskGVUpdate

Running the Sample

1. In the Visual C++ development environment, open and run DTSTskGVUpdate.dsw .
2. On the **Build** menu, click **Build DTSTskGVUpdate.dll** to build the project.
3. In Microsoft SQL Server™ Enterprise Manager, right-click **Data Transformation Services**, click **Properties**, and then click **Refresh Cache**, if caching is active.
4. Include the custom task in a DTS package.

For more information about how this example is implemented, see [DTS Example: Including a User Interface in Visual C++](#).

See Also

[Implementing and Testing a DTS Custom Task](#)

[DTS Programming Samples](#)

DTS Programming

DTSTskPropIcon

This Microsoft® Visual C++® sample is a simple Data Transformation Services (DTS) custom task. It displays a message whose text is specified as a custom task property.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\Dts\CustomTasks\DTSTskPropIcon

Running the Sample

1. In the Visual C++ development environment, open and run DTSTskPropIcon.dsw.
2. On the **Build** menu, click **Build DTSTskPropIcon.dll** to build the project.
3. In Microsoft SQL Server™ Enterprise Manager, right-click **Data Transformation Services**, click **Properties**, and then click **Refresh Cache**, if caching is active.
4. Include the custom task in a DTS package.

For more information about how this example is implemented, see [DTS Example: Adding Properties and Icons in Visual C++](#).

See Also

[Implementing and Testing a DTS Custom Task](#)

[DTS Programming Samples](#)

DTS Programming

Packages

The samples supplied with Microsoft® SQL Server™ 2000 demonstrate how to create and execute packages and how to solve typical business problems. You can also use the packages as templates for custom solutions tailored to the business needs of your organization.

For more information, see [DTS Designer Example: Copying Northwind Data](#). This example shows how to create connections, how to create a simple transformation task that copies data, and how to run a package.

These sample Data Transformation Services (DTS) packages are installed along with the DTS sample programs.

File name	Description
DTS - Workflow Example.dts	Create simple and complex precedence constraints.
DTS - Transform Customers.dts	Use Microsoft ActiveX® scripts to transform Northwind data to tempdb .
OLTP to Star Schema - Sample Package.dts	Transform an OLTP database to a Star schema.
DTS - Execute SQL DDL and DML.dts	Create new databases, tables, and indexes.
DTS - Transfer Database and Objects.dts	Copy objects between instances of SQL Server.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\Dts\Misc\packages

See Also

[DTS Programming Samples](#)

DTS Programming

Complex Transformation Sample from SQL Server to Excel

This program copies data from the **pubs** database to a Microsoft® Excel spreadsheet, transforming it into a Microsoft PivotTable®. The program also demonstrates how to create a Data Transformation Services (DTS) object.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtspackages\Dtsexmp2

Running the Sample

Here are the steps for running the Cptexmp application:

1. Open and compile Cptaxdll.vbp.

This registers the CreatePivotTable custom task used by Cptexmp.vbp and tstuiapp.vbp.

2. Open and run Cptexmp.vbp or tstuiapp.vbp.

See Also

[DTS Programming Samples](#)

DTS Programming

DTS Package Sample Supporting Multiple Source and Destination Providers

This sample allows you to create a package from a variety of data sources and destinations. It creates a simple package that you can create from an SQL statement or a Microsoft® ActiveX® script. You can choose a data source and destination from an enumerated list.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtspackages\dtsexmp3

Running the Sample

Open and run dtsqry.vbp.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSActiveScriptTask

This Microsoft® Visual Basic® sample demonstrates how to run a Visual Basic Scripting Edition (VBScript) script as part of a Data Transformation Services (DTS) Task. The script in this task shows a Message Box.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSActiveScriptTask

Running the Sample

1. Open the DTSActiveScriptTask.vbp project.
2. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSApplicationObject

This Microsoft® Visual Basic® sample illustrates some of the information that can be obtained from the Data Transformation Services (DTS) **Application** object.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtspackages\DTSApplicationObject

Running the Sample

1. Open the DTSAppInfo.vbp project.
2. Run the application.

See Also

[DTSAppObject](#)

[DTS Programming Samples](#)

DTS Programming

DTSAppObject

This Microsoft® Visual Basic® sample demonstrates how to use the Data Transformation Services (DTS) **Application** object on the local server.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtssysteminfo\DTSAppObject

Running the Sample

1. Open the DTSApplicationObject.vbp.
2. Run the application.

See Also

[DTSApplicationObject](#)

[DTS Programming Samples](#)

DTS Programming

DTSBulkInsertTask

This Microsoft® Visual Basic® sample demonstrates how to use Data Transformation Services (DTS) to perform a bulk insert operation from a flat text file. Bulk insert operations are possible from a wide range of OLE DB provider file types.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSBulkInsertTask

Running the Sample

1. Open the BulkInsertTask.vbp project.
2. Run the application.

This sample assumes that you have created the **pubs2..authors** table as a copy of the **pubs..authors** table.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSCopyDatabase

This Microsoft® Visual Basic® sample demonstrates how to use the Data Transformation Services (DTS) **TransferObjectsTask** object to copy a database. This sample copies all objects except dependencies, logins, and users from one named database on the local server to another named database on the local server. If you wish to include dependencies, logins, or users in the transfer, set the **IncludeDependencies**, **IncludeLogins**, and **IncludeUsers** properties to TRUE.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSCopyDatabase

Running the Sample

This sample assumes that you have created the pubs2 database with no data in it.

1. Open the DTSCopyDatabase.vbp project.
2. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSExecProcess

This Microsoft® Visual Basic® sample demonstrates how to run a Microsoft Win32® application from a Data Transformation Services (DTS) task. The sample creates a DTS package and a custom task that runs the Windows Calculator for 15 seconds.

Depending on the **FailPackageOnTimeout** and **FailOnError** settings in the sample, an error message is returned if Calc.exe does not exit in 15 seconds.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSExecProcess

Running the Sample

1. Open the DTSExecProcess.vbp project.
2. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSExecSQLTask

This Microsoft® Visual Basic® sample demonstrates how to execute an SQL statement while running a Data Transformation Services (DTS) package.

This sample:

- Creates a package.
- Opens a connection to the **pubs2** database.
- Creates a custom task, which executes an SQL statement to INSERT a record into the **Sales** table.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSExecSQLTask

Running the Sample

1. Create the **pubs2** database that is a copy of the **pubs** database on the local server.
2. Open the DTSExecuteSQLTask.vbp project.
3. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSExecutePackage

This Microsoft® Visual Basic® sample demonstrates how to execute programmatically a Data Transformation Services (DTS) package that has been saved in .dts format to a structured storage file. The package in the sample performs a copy operation from the **pubs** database to the **pubs2** database.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSExecutePackage

Running the Sample

1. Create the **pubs2** database that is a copy of the **pubs** database on the local server.
2. Open the DTSExecutePackage.vbp project.
3. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSFTPTask

This Microsoft® Visual Basic® sample demonstrates how to use Data Transformation Services (DTS) to copy files that are not Microsoft® SQL Server™ files from a source to a destination. This sample copies Authors.txt in the local directory to the \Test subdirectory.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstasks\DTSFTPTask

Running the Sample

1. Open the DTSFTPTask.vbp project.
2. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSPackageInfo

This Microsoft® Visual Basic® sample illustrates some of the information that can be obtained from the Data Transformation Services (DTS)

Application.GetPackageInfos method. It provides detailed information about the DTS packages on Microsoft® SQL Server™ 2000.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtssysteminfo\DTSPackageInfo

Running the Sample

This application uses Windows Authentication.

1. Open the DTSSQLServerPackages.vbp.
2. Run the application.
3. Click **DTS Package Info**.
4. Select the instance of SQL Server you wish to query.

See Also

[DTS Programming Samples](#)

DTS Programming

DTSTransferObjectsTask

This Microsoft® Visual Basic® sample demonstrates how to use Data Transformation Services (DTS) to transfer various types of database objects from the **pubs** database to the **pubs2** database.

The objects transferred in this sample are:

- **Authors** table.
- **Employee** table.
- **Titleview** view.
- **Byroyalty** stored procedure.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtstas\DTSTransferObjectsTask

Running the Sample

1. Create the **pubs2** database that is a copy of the **pubs** database on the local server.
2. Open the DTSTransferObjectsTask.vbp project.
3. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

FoodMart2000

These Microsoft® Visual Basic® samples demonstrate how to use the Visual Basic file output from the Data Transformation Services (DTS) Import/Export Wizard to convert the FoodMart 2000.mdb database to a Microsoft SQL Server™ 2000 database.

There are three samples in this group.

- Foodmart2000a: A .bas file that converts the Foodmart 2000.mdb file to SQL Server 2000.
- Foodmart2000b: A project that converts the Foodmart 2000.mdb file to SQL Server 2000, but shows how to use the DTS package events to indicate process status on UI.
- Foodmart2000c: A .bas file that converts the Foodmart 2000.mdb file to SQL Server 2000, but also incorporates an error handler.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Misc\FoodMart2000

Running the Sample

These samples assume that you are using Windows Authentication and that the execution account has the necessary privileges.

The initial steps for using these samples are:

1. Install SQL Server 2000 Analysis Services samples.
2. Use SQL Server Enterprise Manager to create a new database called **Foodmart2000**.

3. In FoodMart2000a and FoodMart2000c, modify the path for the DataSourcePath variable name.
4. Execute the FoodMart2000a.vdp, FoodMart2000b.vdp, and FoodMart2000c.vdp projects.

See Also

[DTS Programming Samples](#)

DTS Programming

Pub2Pubs

This Microsoft® Visual Basic® sample copies the **Authors** table from the **pubs** database to the **pubs2** database while performing several operations on various fields. The sample is a single package with two connections, and two Steps/Tasks with multiple custom subtasks.

The comments in the source code of this sample provide detailed explanations of code that is generated by the Data Transformation Services (DTS) Import/Export Wizard.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Misc\Pub2Pubs

Running the Sample

1. Create the **pubs2** database that is a copy of the **pubs** database on the local server.
2. Open the PubsToPub2.vbp project.
3. Run the application.

See Also

[DTS Programming Samples](#)

DTS Programming

Simple DTS Package Sample Using Visual Basic

This Microsoft® Visual Basic® sample demonstrates how to build and execute a Data Transformation Services (DTS) package.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtspackages\Dtsvbpkg

Running the Sample

1. Create two ODBC Data Source Names (DSN), "DTS Source" and "DTS Destination", that refer to the **pubs** database.
2. Open and run dtsvbpkg.vbp

See Also

[DTS Programming Samples](#)

DTS Programming

Simple Transformation Sample Between Two SQL Server Tables

This Microsoft® Visual Basic® sample copies data from a source table to a destination table, after a Transact-SQL script is used to create a sample table in the **pubs** database.

Default Location

C:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\Dts\Dtspackages\Dtsexmp1

Running the Sample

Here are the steps for running the Dtsexmp1 application:

1. Create an ODBC DSN for Microsoft SQL Server™ and make **pubs** the default database.
2. Run the Creattbl.sql script to create the **authorname** table.
3. Open and run Dtsexmp1.vbp.

See Also

[DTS Programming Samples](#)