CHAPTER

10

SQL Server Integration Services

IN THIS CHAPTER

An Overview of SQL Server Integration Services
Creating Packages
Deploying Packages
Programming with the SQL Server Integration Services APIs

SQL Server Integration Services is an all-new subsystem for SQL Server 2005. With SQL Server 2005 Microsoft has replaced the old Data Transformation Services (DTS) with the all-new SQL Server Integration Services (SSIS). It's important to understand that SSIS isn't a reworked version of DTS. Instead, Microsoft rewrote SSIS from the ground up. Microsoft's goal for SQL Server 2005's Integration Services was to make it an enterprise ETL platform for Windows on a par with any of the stand-alone enterprise-level ETL products. The new SSIS is completely redesigned and built using managed .NET code, giving it a more robust foundation. The new SSIS features a new graphical designer, a greatly enhanced selection of data transfer tasks, better support for programmability and improved run-time. In this chapter you'll learn how to develop data integration packages using SSIS. First, this chapter will start off by giving you an overview of the new SSIS. Next, you'll learn about how to create and deploy packages using the SSIS Designer. Then this chapter will wrap up by showing you how you can create and run SSIS packages programmatically using the Microsoft.SqlServer.Dts namespace.

NOTE

Don't be confused by the DTS moniker in the namespace. SSIS is not built on top of DTS, nor does it use any of the old DTS code. Microsoft simply didn't get around to renaming the APIs to match the name of the new subsystem.

An Overview of SQL Server Integration Services

The new Integration Services architecture is divided into two main sections: the Data Transformation Pipeline (DTP) and the Data Transformation Runtime (DTR). The split is designed to make a clear delineation between data flow and control flow. In the previous versions of DTS, the data flow engine was stronger than the control flow capabilities. This new division essentially makes the control flow portion of SSIS a first-class component on the same level as the data flow component. The new DTP essentially takes the place of the old DTS Data Pump that was used in the SQL Server 7 and 2000. Its primary function is to handle the data flow between the source and target destinations. The DTR is essentially a job execution environment that controls the control flow that's used in an SSIS package. Each of these components exposes its own distinct object model that you can program against. In Figure 10-1 you can see an overview of the new SQL Server Integration Services architecture.

The new Integration Services DTP and DTR are discussed in more detail in the following sections. More information about the new Integration Services tool set is also presented later in this chapter.

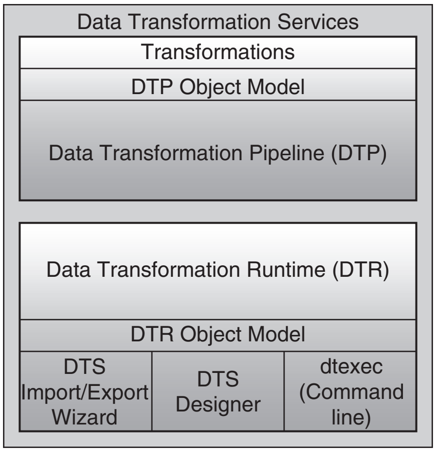


Figure 10-1 *Integration Services architecture*

Data Transformation Pipeline (DTP)

The DTP takes care of the data flow and transformations that take place as rows are moved between the data source and the data target. DTP uses data adapters to connect to the source and destination data sources. As you can see in Figure 10-1, the DTP engine is accessed using the DTP Pipeline object model. This object model is the API that is used by both the built-in transformations supplied by Microsoft and any user-created custom transformations. Transformations move and optionally manipulate row data as they move data from the source columns to the destination columns. You can get a more detailed look at the new DTP architecture in Figure 10-2.

SQL Server 2005 provides a number of source and destination data adapters. Out of the box, SQL Server 2005's Integration Services comes with adapters for SQL Server databases, XML, flat files, and other OLE DB–compliant data sources. While the job of the data adapters is to make connections to the data's source and destination endpoints, the job of the transformations is to move and optionally manipulate the data as it's moved between the source and destination endpoints. Transformation can be as simple as a one-to-one mapping between the source columns and the target columns, or it can be much more complex, performing such tasks as selectively moving columns between the source and target, creating new target columns using one-to-many mappings, or computing derived columns. SQL Server 2005's Integration Services comes with a substantial number of built-in transformations. In addition to these built-in transformations, you can build your own custom transformations by taking advantage of the DTP object model API.

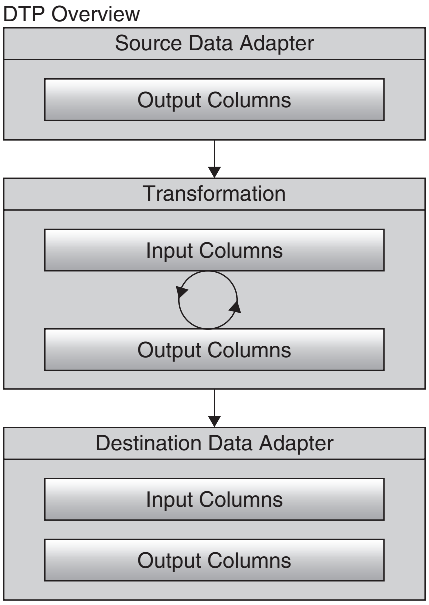


Figure 10-2 *Data Transformation Pipeline components*

Data Transformation Runtime (DTR)

The DTR consists of the DTR engine and the DTR components. DTR components are objects that enable you to govern the execution of SSIS packages. The DTR components are used to build work flows, containers provide structured operations, tasks provide data transfer and transformation functionality, and constraints control the sequence of a work flow in a package. You can see an overview of the new DTR architecture in Figure 10-3.

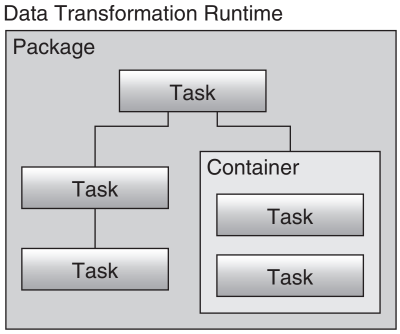


Figure 10-3 *Data Transformation Runtime overview*

The primary DTR components are packages, containers, and tasks. *Tasks* are collections of DTR components; each task is composed of data sources and target destinations as well as data transformations. *Containers* are used to organize and structure related tasks. These containers and tasks are grouped together to form packages. The Integration Services *package* is the physical unit that groups together all of the functions that will be performed in a given transfer operation. Packages are executed by the DTR to perform data transfers. Integration Services packages can be easily rerun or even moved to a different system and executed stand-alone.

The primary purpose of the DTR engine is to control the execution of Integration Services packages. The DTR controls the work flow of the tasks contained in an Integration Services package. In addition, the DTR engine stores package layout; runs packages; and provides debugging, logging, and event handling services. The DTR engine also enables you to manage connections and access Integration Services package variables.

The DTR is accessed using the DTR object framework. The DTR run-time object framework is the API that supports the Integration Services Import/Export Wizard and the Integration Services Designer in addition to the command-line dtexec tool. The Import/Export Wizard and the Designer are used to create packages. Programs that use the DTR object model can automate the creation and execution of Integration Services packages as is shown later in this chapter.

Creating Packages

You can create SSIS packages in three ways: using the SSIS Import and Export Wizard, using the SSID Designer, or programmatically using the DTR object model. In the next section of this chapter you'll see how to create SSIS interactively, first by using the SSIS Import and Export Wizard and then by using the SSIS Designer.

Using the SSIS Import and Export Wizard

The SQL Server 2005 Integration Services SSIS Import and Export Wizard provides a series of dialogs that lead you through the process of selecting the data source, the destination, and the objects that will be transferred. The wizard also allows you to optionally save and execute the SSIS package. Saving the packages generated with the Integration Services Import/Export Wizard and then editing them in the Integration Services Designer is a great way to learn more about Integration Services—especially if you're just getting started with Integration Services or if you're transitioning to the new SQL Server 2005 Integration Services from one of the earlier versions.

You can start the Integration Services Import/Export Wizard by entering **dtswizard** at the command line. The wizard steps you through the process of creating a package. The first action is choosing a data source. In the Data Source drop-down, you select the provider that you want to use. The connection options change depending on the provider that you select. If you select the Microsoft OLE DB Provider for SQL Server, you select the server that you want to connect to and then the database and the type of authentication that you need to use. Clicking Next leads you through the subsequent wizard dialogs. The next dialog allows you to select the data destination, which is essentially identical to the data source dialog except that it defines where the data will be transferred to. After you select the data source and destination, the wizard prompts you to select the data to be transferred and then to optionally save and execute the Integration Services package. As each task in the package executes, the transfer window is dynamically updated, showing the Integration Services package's transfer progress.

Using the SSIS Designer

While the Integration Services Import/Export Wizard is useful for simple ad hoc transfers, ETL (extraction, transformation, and loading) tasks typically require significantly more sophistication and complex processing than the SSIS Import and Export Wizard exposes. By their nature, ETL tasks are far more than just simple data transfers from one destination to another. Instead, they often combine data from multiple sources, manipulate the data, map values to new columns, create columns from calculated values, and provide a variety of data cleanup and verification tasks. That's where the new Integration Services Designer comes into play. The Integration Services Designer is a set of graphical tools that you can use to build, execute, and debug SSIS packages.

Package Overview

In this example the package will be performing an FTP transfer; the results of that FTP transfer will be a flat file; that flat file in turn will be transferred to a SQL Server database. As the flat file is being transferred to the SQL Server database, a lookup operation will occur that matches the incoming vendor product ID numbers to product IDs contained in the AdventureWorks products table. If the lookup succeeds, then the record with the corrected product ID will be written to the destination table. Otherwise, if the lookup fails, the data will be written to a log file.

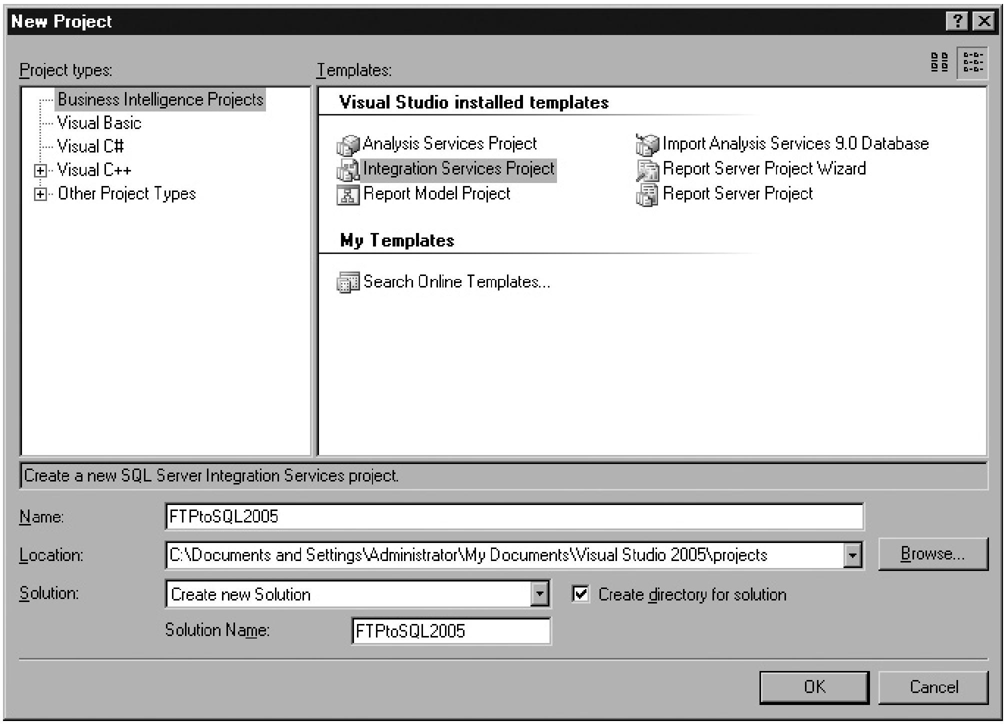
To build the SSIS package, you first start the SSIS Designer using the Business Intelligence Development Studio (BIDS).

NOTE

Don't be confused by the fact that the SSIS Designer is started from the Business Intelligence Development Studio. SSIS is not limited to just Analysis Services projects. The projects developed in the Business Intelligence Development Studio are fully capable of working with relational data.

Open the BIDS and then select the File | New | Project option to open the New Project dialog. To create a new Integration Services project, select Business Intelligence Projects from the Project Types list and then Integration Project from the list of templates, as is shown in Figure 10-4. When the SSIS Designer first starts, you're presented with a blank design surface like the one shown in Figure 10-5.

Defining Tasks At this point, to build an SSIS package, you need to drag and drop tasks from the Control Flow toolbox onto the design surface that represent the actions that you want the package to perform. To construct the sample package, you need to use an FTP task, a SQL task, and a Data Flow task. As you might imagine,



New Project

Project types:
- Business Intelligence Projects
- Visual Basic
- Visual C#
- Visual C++
- Other Project Types

Templates:

Visual Studio installed templates
- Analysis Services Project
- Integration Services Project
- Report Model Project
- Import Analysis Services 9.0 Database
- Report Server Project Wizard
- Report Server Project

My Templates
- Search Online Templates...

Create a new SQL Server Integration Services project.

Name: FTPtoSQL2005

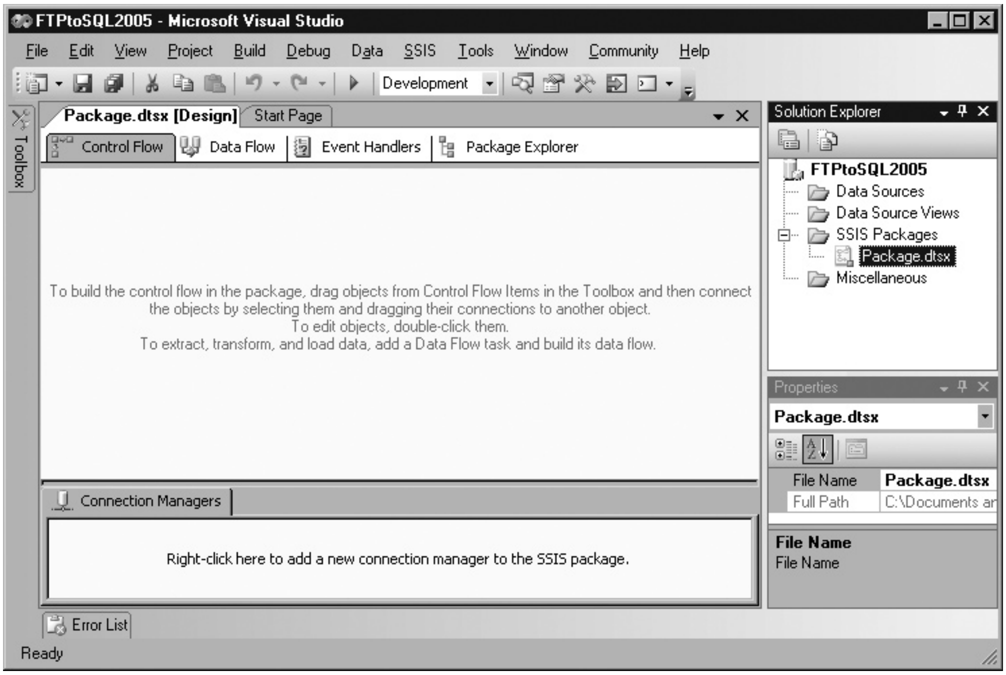
Location: C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\projects

Solution: Create new Solution

☑ Create directory for solution

Solution Name: FTPtoSQL2005

Figure 10-4 *Opening a data transformation project*



To build the control flow in the package, drag objects from Control Flow Items in the Toolbox and then connect the objects by selecting them and dragging their connections to another object.
To edit objects, double-click them.
To extract, transform, and load data, add a Data Flow task and build its data flow.

Right-click here to add a new connection manager to the SSIS package.

File Name	Package.dtsx
Full Path	C:\Documents ar

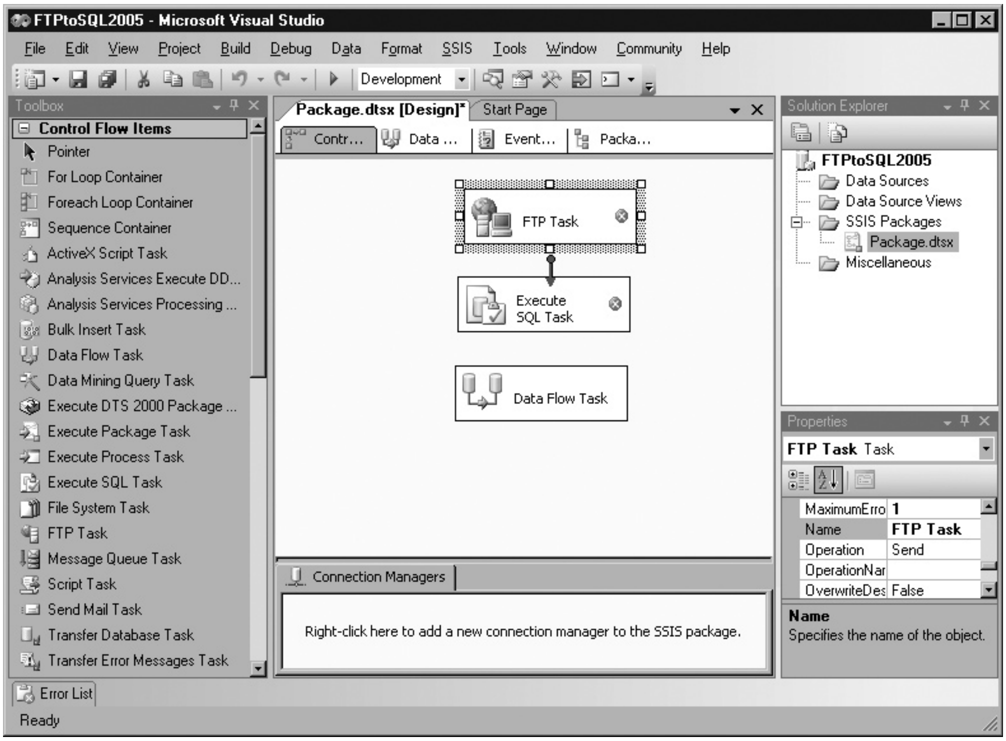
File Name
File Name

Figure 10-5 *The SSIS design surface*

the FTP task will transfer the file from the remote system. The SQL task will be used to create a new task to store the FTP data, and the Data Flow task will transfer the data from the flat file to the SQL Server table and will also perform the lookups. You can see these tasks laid out on the SSIS design surface in Figure 10-6.

You might notice in Figure 10-6 that the tasks are all marked with a red *x*. This indicates that the task has not yet been defined. At this point two things need to happen: the precedence between the tasks needs to be defined, and the tasks each need to be defined. To define the precedence between the tasks is easy.

Defining Precedence The precedence essentially defines which task will be executed first, which second, and so on. To define precedence, click each task. This causes a green arrow indicating precedence to appear at the bottom of the task. First click the FTP task and drag the green arrow to the SQL task. Then click the SQL task and drag the green arrow to the Data Flow task. This forces the FTP task to complete before the SQL task is performed. Likewise, the SQL task must be performed before the Data Flow task. If you do not define precedence, the tasks will be executed in parallel.



Right-click here to add a new connection manager to the SSIS package.

MaximumErro	1
Name	FTP Task
Operation	Send
OperationNar	
OverwriteDes	False

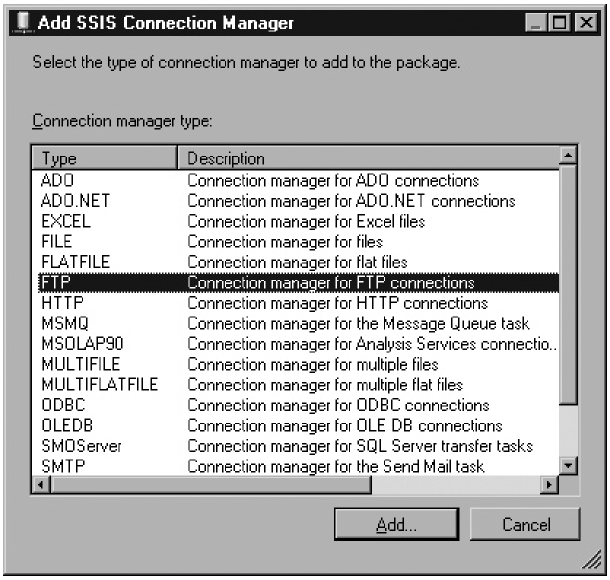
Name
Specifies the name of the object.

Figure 10-6 *The SSIS package tasks*

Defining Connections and Tasks Next the connections that will be used for each task must be defined. In our example, the FTP task will need an FTP connection to the remote host, the SQL task will need an OLE DB connection to the target database, and the data flow task will need a flat file connection for the resulting FTP file and an OLE DB connection to transfer the data to the SQL Server table. To create the FTP connection, right-click in the Connection Manager pane that you can see in the bottom of Figure 10-6 and then select the New Connection option to display the Add SSIS Connection dialog that you can see in Figure 10-7. To define the FTP connection, select FTP from the list of connection types and then click Add to display the FTP Connection Manager that is illustrated in Figure 10-8.

Enter the name of the FTP server in the Server Name prompt and the authentication information that is required to connect to the FTP server in the Credentials group. You can also optionally change the port from the default FTP port of 21 as well as the time-out values. Clicking Test Connection allows you to verify that the values that you've entered are correct. Click OK to save the FTP connection information.

After creating the FTP Connection Manager, you can now finish defining the FTP task. Double-click the FTP Task in the SSIS Designer to display the FTP Task Editor.



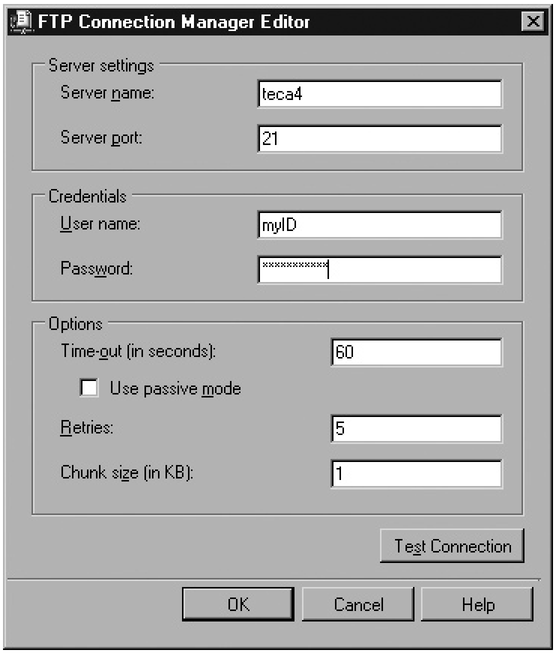
Add SSIS Connection Manager

Select the type of connection manager to add to the package.

Connection manager type:

Type	Description
ADO	Connection manager for ADO connections
ADO.NET	Connection manager for ADO.NET connections
EXCEL	Connection manager for Excel files
FILE	Connection manager for files
FLATFILE	Connection manager for flat files
FTP	Connection manager for FTP connections
HTTP	Connection manager for HTTP connections
MSMQ	Connection manager for the Message Queue task
MSOLAP90	Connection manager for Analysis Services connectio..
MULTIFILE	Connection manager for multiple files
MULTIFLATFILE	Connection manager for multiple flat files
ODBC	Connection manager for ODBC connections
OLEDB	Connection manager for OLE DB connections
SMOServer	Connection manager for SQL Server transfer tasks
SMTP	Connection manager for the Send Mail task

Figure 10-7 *The FTP connection*



FTP Connection Manager Editor

Server settings

Server name: teca4

Server port: 21

Credentials

User name: myID

Password: ************

Options

Time-out (in seconds): 60

☐ Use passive mode

Retries: 5

Chunk size (in KB): 1

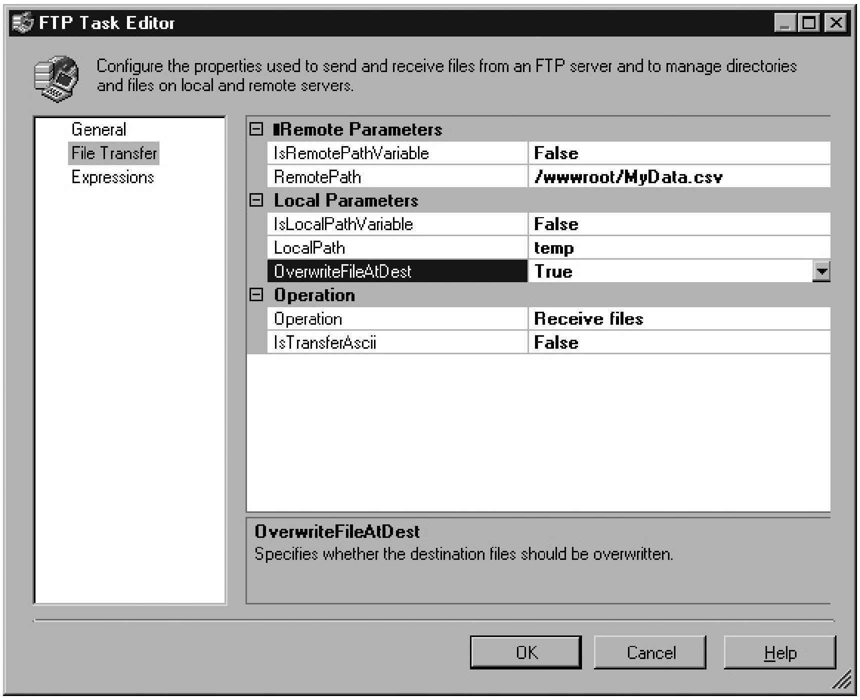
Figure 10-8 *FTP Connection Manager*

On the General screen select the FTP Connection Manager that you just created at the Ftp Connection prompt. Then click the File Transfer item to describe the transfer that will take place. You can see the File Transfer properties in Figure 10-9.

Under the Operation property select Receive Files from the drop-down list to execute an FTP Get operation. Next, under the RemotePath property enter the remote server directory where the file to download will be found. In this example you can see that the file that will be transferred is named /wwwroot/MyData.csv. Next, set the LocalPath property to the directory on the system where you want to receive the file. In Figure 10-9 the value of temp is used, which indicates that the file will be received in the c:\temp directory. Select a value of True for OverwriteFileAtDest if you want to recreate the file each time it is transferred regardless of the presence of an existing file. Click OK to save the settings in the FTP task.

After the FTP connection is defined, you can test it by right-clicking the task in the SSIS designer and then selecting the Execute Task option from the pop-up menu. Running the task will result in an FTP transfer, and the file MyData.csv will be created in the c:\temp directory.

Next, the Execute SQL task must be defined. As with the FTP task, you first need to create a connection for the task to use in order to connect to the desired database to execute a SQL Create Table command. To create an OLE DB connection for SQL



Configure the properties used to send and receive files from an FTP server and to manage directories and files on local and remote servers.

Remote Parameters	
IsRemotePathVariable	False
RemotePath	/wwwroot/MyData.csv
Local Parameters	
IsLocalPathVariable	False
LocalPath	temp
OverwriteFileAtDest	True
Operation	
Operation	Receive files
IsTransferAscii	False

OverwriteFileAtDest
Specifies whether the destination files should be overwritten.

Figure 10-9 *FTP file transfer task properties*

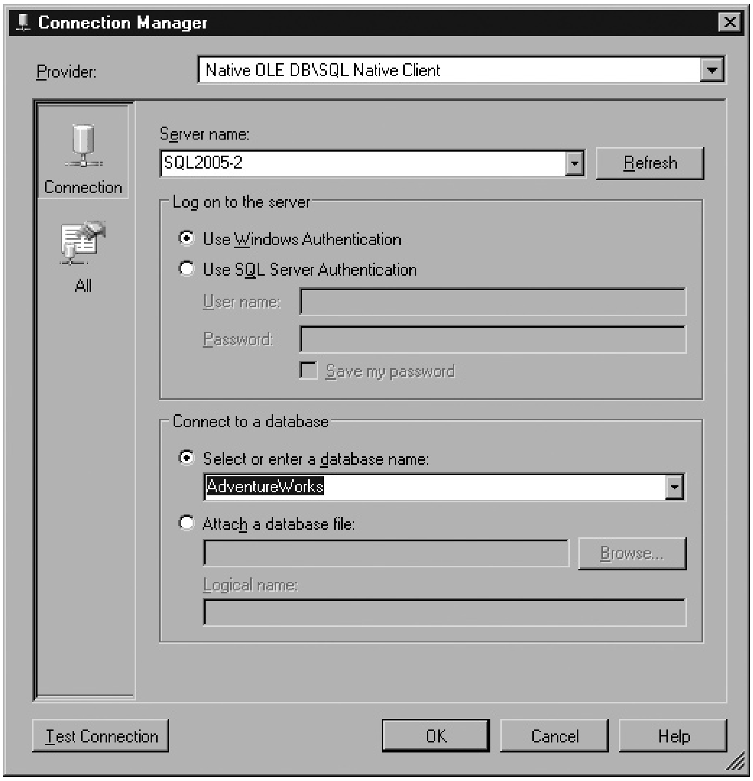
Server, right-click in the Connection Manager and select New OLE DB Connection from the list. Then click New to create a new OLE DB connection. This will display the Connection Manager dialog shown in Figure 10-10.

NOTE

You could have selected an ADO.NET Connection type as well. However, most SSIS transformations can use only the OLE DB Connection type. Therefore, selecting the OLE DB connection enables the package to reuse the same connection for a variety of operations.

The Provider drop-down box should show .Native OLE DB\SQL Native Client. Fill in the server name, the required authentication information for the server, and the target database. Here you can see that this connection will use the server SQL2005-2. It will connect using Windows authentication, and AdventureWorks will be the default database. Click OK and then OK again to create a new OLE DB connection.

After defining the OLE DB connection, double-click the SQL task to assign values to the SQL task properties. Here you need to fill in the ConnectionType, Connection,



Provider: Native OLE DB\SQL Native Client

Server name: SQL2005-2

Log on to the server

● Use Windows Authentication

○ Use SQL Server Authentication

User name:

Password:

□ Save my password

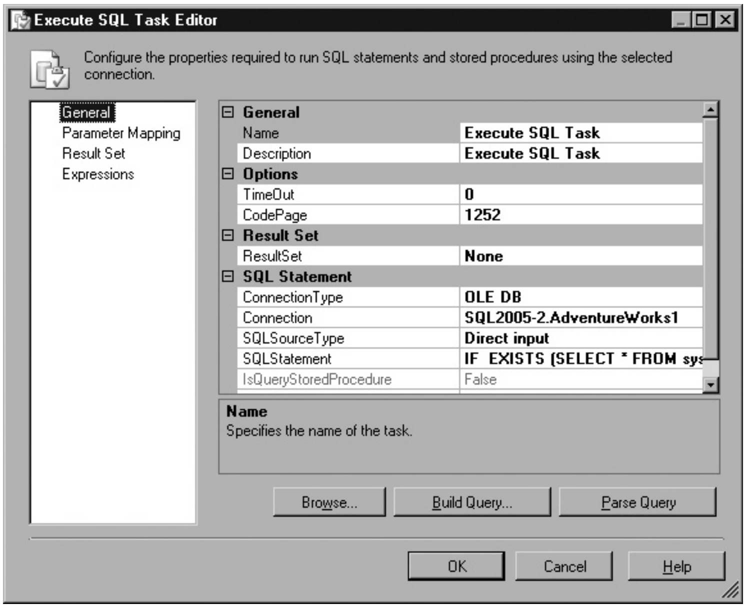
Connect to a database

● Select or enter a database name: AdventureWorks

○ Attach a database file:

Logical name:

Figure 10-10 *OLE DB Connection Manager*



Execute SQL Task Editor

Configure the properties required to run SQL statements and stored procedures using the selected connection.

General	
Name	Execute SQL Task
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	SQL2005-2.AdventureWorks1
SQLSourceType	Direct input
SQLStatement	IF EXISTS (SELECT * FROM sys
IsQueryStoredProcedure	False

Name
Specifies the name of the task.

Figure 10-11 *SQL task properties*

and SQLStatement properties. You can see the completed Execute SQL Task properties shown in Figure 10-11.

As you can see in Figure 10-11, the ConnectionType property has been set to OLE DB, and the Connection property has been assigned the name of the OLE DB connection that was created earlier, in this case, SQL2005-2.AdventureWorks1. Next, the SQLStatement property must be assigned a SQL command. This example will use a SQL statement that first drops and then creates the destination table. You can see the complete SQL statement in the following listing:

```
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[Purchasing].[ProductShipments]') AND type in (N'U'))
  DROP TABLE [Purchasing].[ProductShipments]
GO

CREATE TABLE [Purchasing].[ProductShipments](
  [ShipProductID] [varchar](15) NOT NULL,
  [AdwProductID] [int] NOT NULL,
  [Name] [varchar](50) NOT NULL,
  [ProductNumber] [varchar](25) NOT NULL,
  [ShipDate] [datetime] NULL,
  [Units] [int] NOT NULL
) ON [PRIMARY]
GO
```

The SQL code in this listing will create a table called Purchasing.ProductShipments in the AdventureWorks database. The columns here pass on the values that are provided by the FTP transfer. Take note of the data types of these columns, as they will need to match the data types used by the Data Flow task later on.

After filling out the ConnectionType, Connection, and SQLStatement properties, click OK to save the Execute SQL task.

As you saw earlier with the FTP task, you can test the Execute SQL task by right-clicking the task in the SSIS designer and selecting the Execute Task option from the pop-up menu. This will run the SQL statement; in this case, the Purchasing .ProductShipments table will be created in the AdventureWorks database.

Defining the Data Flow Next, the Data Flow task needs to be defined. Double-click the Data Flow task to switch the SSIS Designer to the Data Flow tab. This will cause the toolbox to change from the Control Flow toolbox to the Data Flow toolbox. While the Control Flow toolbox shows the different tasks that are available, the Data Flow toolbox shows the available data sources, transformations, and destinations. To define the data flow for this package, first drag the Flat File Source onto the design surface from the Data Flow Source portion of the toolbox. Next, go to the Data Flow Transformations section of the toolbox and drag the Lookup transformation onto the designer. Then go to the Data Flow Destination section of the toolbox and drag the SQL Server Destination onto the SSIS data flow design surface. The design surface should appear like the one shown in Figure 10-12.

You assign precedence and values to each of the data flow elements in the same way that you did to the control flow tasks. To assign precedence to the data flow elements, first click the Flat File Source item and drag the green arrow to the Lookup transformation. Next, click the Lookup transformation and drag the green arrow to the SQL Server Destination item. This will cause the data flow to start with the flat file source, perform a lookup, and then move on to the SQL Server destination. To define each of the data flow elements, double-click the element that you want to work with to open the editor and then assign the appropriate value.

To define the Flat File Source, double-click the Flat File Source transformation to display the Flat File Source Editor. There, click New to create a new Flat File Manager. This will display the Flat File Connection Manager shown in Figure 10-13.

In the Flat File Connection Manager name the connection by filling in the Connection Manager Name property. This example uses the value of FTP File Output. Then tell the Flat File Manager about the file that you will be using as input by filling in the File Name property with the name of the file that will be read. In Figure 10-13 you can see that the Flat File Manager will be reading the file c:\temp\Mydata.csv. If you know that the incoming data will have header values in the first row, as many csv files do, then check the Column Names In The First Data Row check box. Click OK to save the settings.

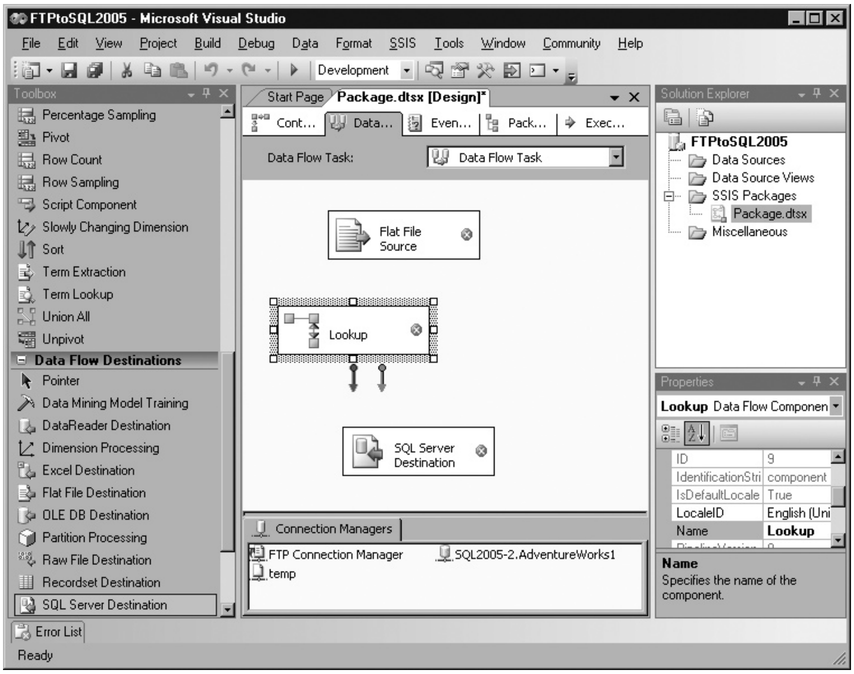
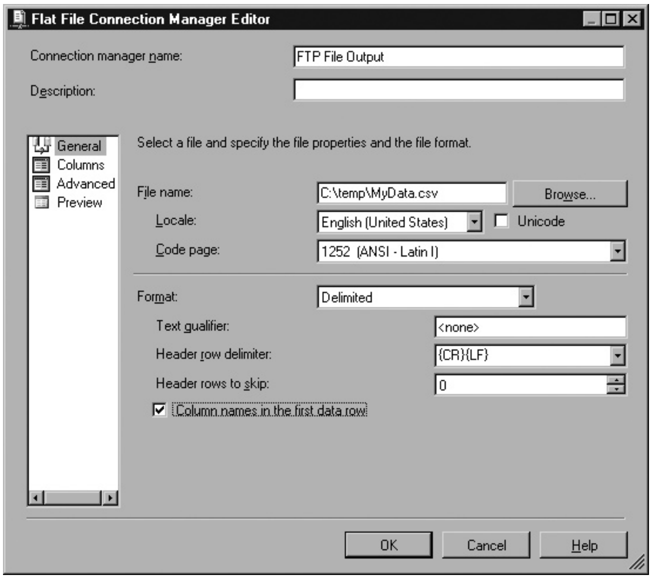


Figure 10-12 *The SSIS data flow design surface*



Connection manager name: FTP File Output

Description:

Select a file and specify the file properties and the file format.

File name: C:\temp\MyData.csv

Locale: English (United States) □ Unicode

Code page: 1252 (ANSI - Latin I)

Format: Delimited

Text qualifier: <none>

Header row delimiter: {CR}{LF}

Header rows to skip: 0

☑ Column names in the first data row

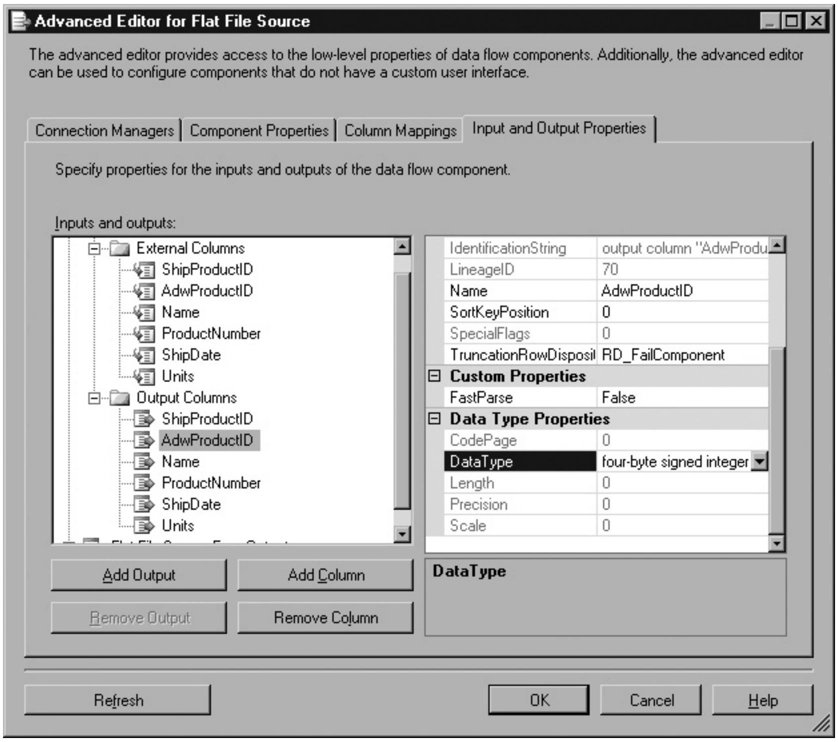
Figure 10-13 *Flat File Connection Manager*

NOTE

Testing the earlier FTP task will produce a file that you can use to connect to and preview with the Flat File Connection Manager.

Each of the output data types should be changed to match the data types that are used in the SQL Server destination table. To change the output data types, right-click the Flat File Source and select Show Advanced Editor from the pop-up menu to display the Advanced Editor for Flat File Source. Then click the Output Properties tab and expand the Flat File Source | Output Columns node to display a dialog like the one shown in Figure 10-14.

For each column, click the Data Type property and change the type from DT_STR (the default) to the type that will match the columns in the target table. For instance, in Figure 10-14 you can see that the Data Type property of the AdwProductID column has been changed to a four-byte integer, which will match the required output column.



Advanced Editor for Flat File Source

The advanced editor provides access to the low-level properties of data flow components. Additionally, the advanced editor can be used to configure components that do not have a custom user interface.

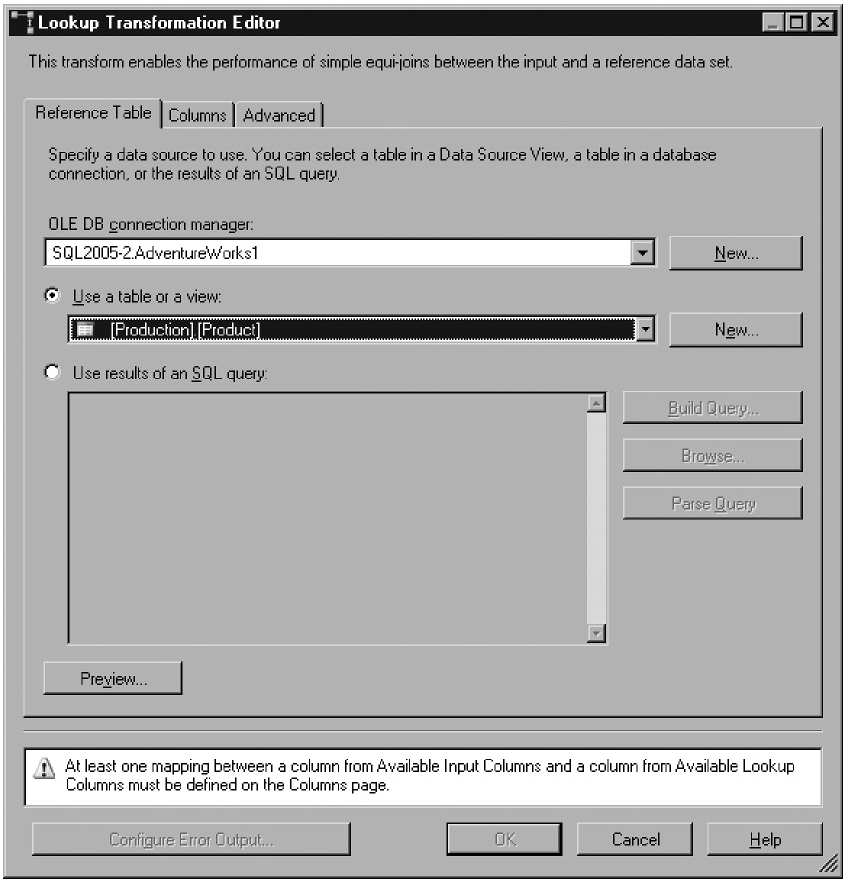
Specify properties for the inputs and outputs of the data flow component.

IdentificationString	output column "AdwProdu
LineageID	70
Name	AdwProductID
SortKeyPosition	0
SpecialFlags	0
TruncationRowDisposit	RD_FailComponent
Custom Properties	
FastParse	False
Data Type Properties	
CodePage	0
DataType	four-byte signed integer
Length	0
Precision	0
Scale	0

Figure 10-14 *Modify the Flat File output column data types*

In this case, it will also match the ProductID column of the Product table, which will later be used to perform a data lookup. After all of the data type changes have been made, click the OK button to save the Flat File property changes.

The next step is to set up the database lookup that will be used to verify that the vendor-supplied product numbers are correct. In this example, the AdventureWorks product number is supplied in the AdWProductID field that's found in the FTP output file. If the value for the AdWproductID matches a value from the Production .ProductID column, then the data will be written to the SQL Server destination table. Otherwise, the data will be written to an error file. To define the Lookup, double-click the Lookup transformation on the data flow design surface. This will display the Lookup Transformation Editor shown in Figure 10-15.



Lookup Transformation Editor

This transform enables the performance of simple equi-joins between the input and a reference data set.

Specify a data source to use. You can select a table in a Data Source View, a table in a database connection, or the results of an SQL query.

OLE DB connection manager: SQL2005-2.AdventureWorks1

● Use a table or a view: [Production].[Product]

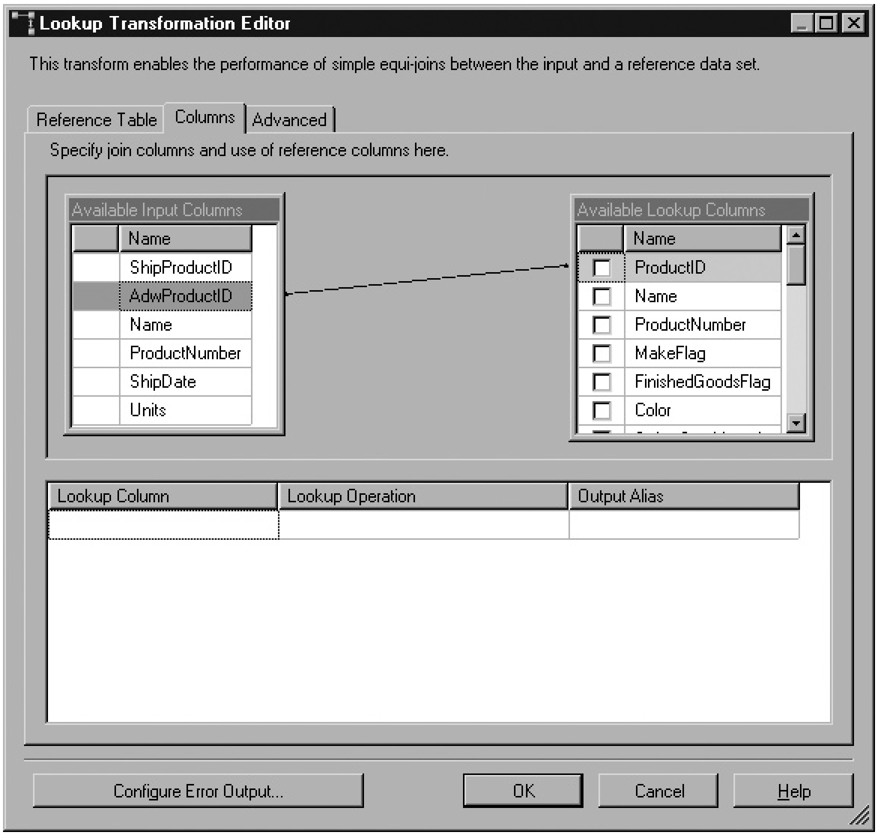
○ Use results of an SQL query:

At least one mapping between a column from Available Input Columns and a column from Available Lookup Columns must be defined on the Columns page.

Figure 10-15 *Lookup Transformation Editor: Select Connection Manager*

Under the Reference Table tab use the drop-down box to select the OLE DB Connection Manager that was created earlier. Here you can see that the OLE DB Connection Manager is set to SQL2005-2.AdventureWorks1. After assigning the connection, the next step is to specify the table or query that will be used for the lookup operation. This example uses the table Production.Product from the AdventureWorks database. After selecting the table, the next step is to specify the columns that will be used in the lookup. To select the columns, click the Columns tab as is shown in Figure 10-16.

On the Columns tab first select the column from the list of Available Input columns that you will use to perform the lookup. Then drag that column over to the matching column in the Available Lookup Columns list. In Figure 10-16 you can see that the value from the incoming AdWProductID column will be used to look up values in the ProductID column from the Production.Product table that was selected earlier. If you only want to perform a lookup operation, you can stop here. Click OK to save the Lookup Transformation settings.



This transform enables the performance of simple equi-joins between the input and a reference data set.

Specify join columns and use of reference columns here.

Available Input Columns

	Name
	ShipProductID
	AdwProductID
	Name
	ProductNumber
	ShipDate
	Units

Available Lookup Columns

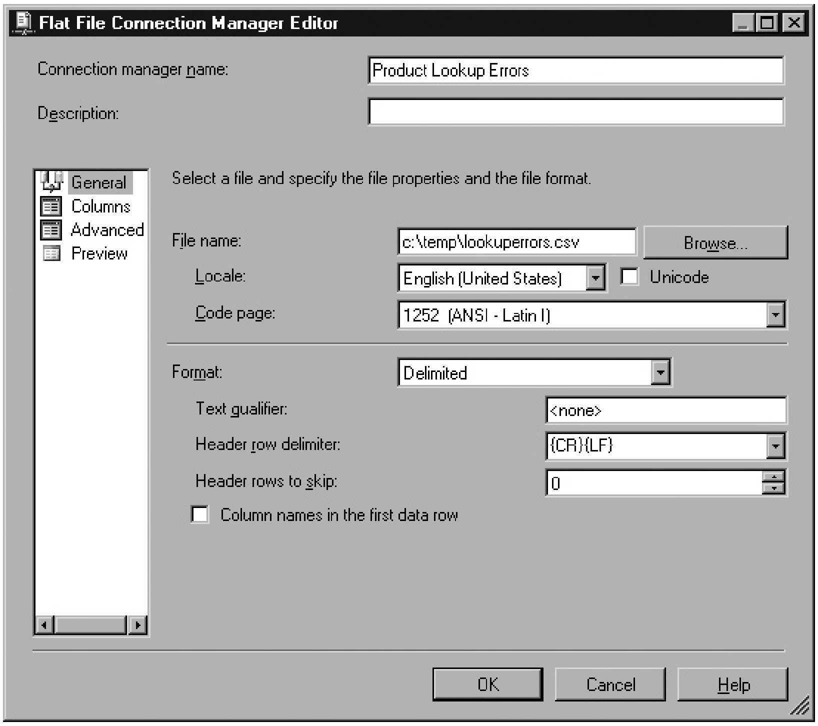
	Name
☐	ProductID
☐	Name
☐	ProductNumber
☐	MakeFlag
☐	FinishedGoodsFlag
☐	Color

Lookup Column	Lookup Operation	Output Alias

Figure 10-16 *Lookup Transformation Editor: Match Columns*

However, if you also want to build an error log to report lookup failures, then you should add a new Flat File Connection Manager that will allow you to output the Lookup transformation error output. As you saw earlier, to add a Flat File Connection, right-click in the Connection Manager pane in the SSIS Designer and then select New Flat File Connection from the pop-up menu to display the Flat File Connection Manager as shown in Figure 10-17.

Give the Flat File connection a name. In this example it is named Product Lookup Errors. Next, use the File Name prompt to specify the folder and file that will be used to write the output. In Figure 10-17 you can see that the lookup errors will be written to the file lookuperrors.csv in the c:\temp directory. The remainder of the prompts control the formation of the output data. In Figure 10-17 all of the defaults have been accepted, which will result in the creation of a comma-separated value (csv) file. Click OK after specifying the properties for the Product Lookup Errors Flat File connection to save the values.



Connection manager name: Product Lookup Errors

Description:

Select a file and specify the file properties and the file format.

File name: c:\temp\lookuperrors.csv

Locale: English (United States) □ Unicode

Code page: 1252 (ANSI - Latin I)

Format: Delimited

Text qualifier: <none>

Header row delimiter: {CR}{LF}

Header rows to skip: 0

□ Column names in the first data row

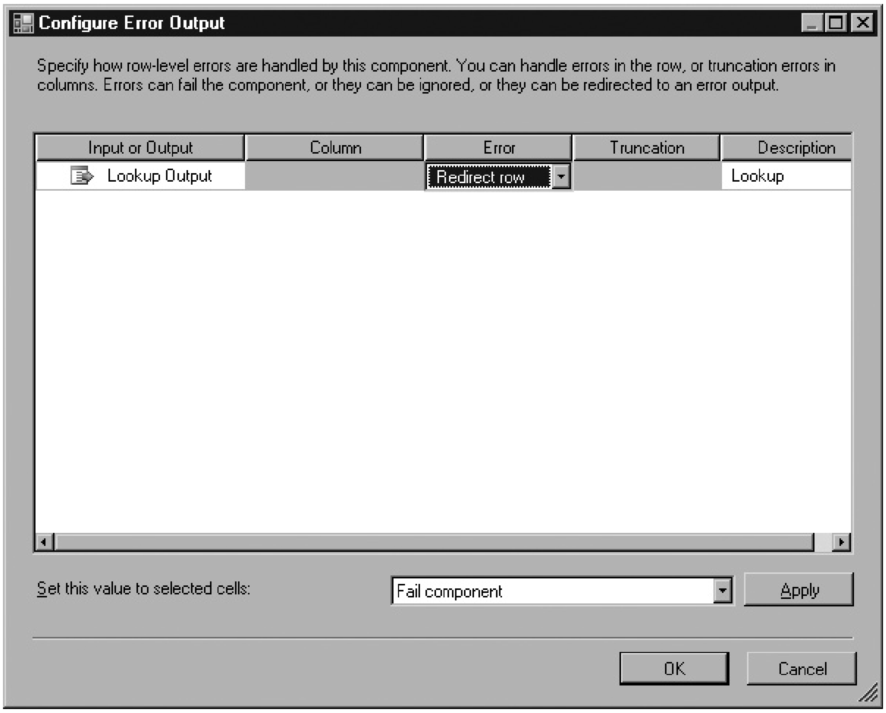
Figure 10-17 *Flat File Connection Manager for lookup error output*

After creating the new Flat File Connection Manager, drag a new Flat File Destination from the data flow toolbox onto the design surface. Double-click the new Flat File Destination to open the editor and then select the Product Lookup Errors connection for the Connection property. Finally, drag the red line from the Lookup Transformation to the Flat File Destination to connect the flat file to the Error Lookup Transformation's error output.

Next, to tell the Lookup Transformation to direct error rows to the flat file, double-click the transformation to open the Lookup Transformation Editor. Then click the Columns tab followed by the Configure Error Output button to display the Configure Error Output dialog shown in Figure 10-18.

On the Configure Error Output dialog use the drop-down beneath Error and select the Redirect Row option. This will redirect any error output to the flat file that was defined using the Flat File Connection Manager. Then click OK.

The final step to complete the configuration of the SSIS package's data flow is the definition of the SQL Server destination. On the data flow design surface double-click the SQL Server Destination object to open up the SQL Destination Editor that's shown in Figure 10-19.



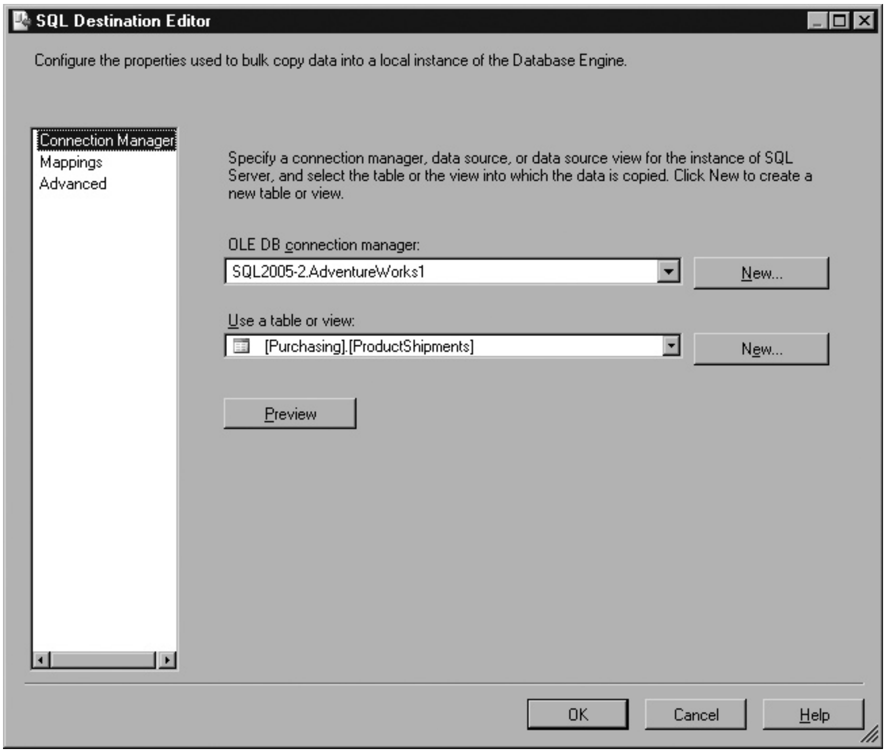
Configure Error Output

Specify how row-level errors are handled by this component. You can handle errors in the row, or truncation errors in columns. Errors can fail the component, or they can be ignored, or they can be redirected to an error output.

Input or Output	Column	Error	Truncation	Description
Lookup Output		Redirect row		Lookup

Set this value to selected cells: Fail component

Figure 10-18 *Configure Error Output*



SQL Destination Editor

Configure the properties used to bulk copy data into a local instance of the Database Engine.

Specify a connection manager, data source, or data source view for the instance of SQL Server, and select the table or the view into which the data is copied. Click New to create a new table or view.

OLE DB connection manager: SQL2005-2.AdventureWorks1

Use a table or view: [Purchasing].[ProductShipments]

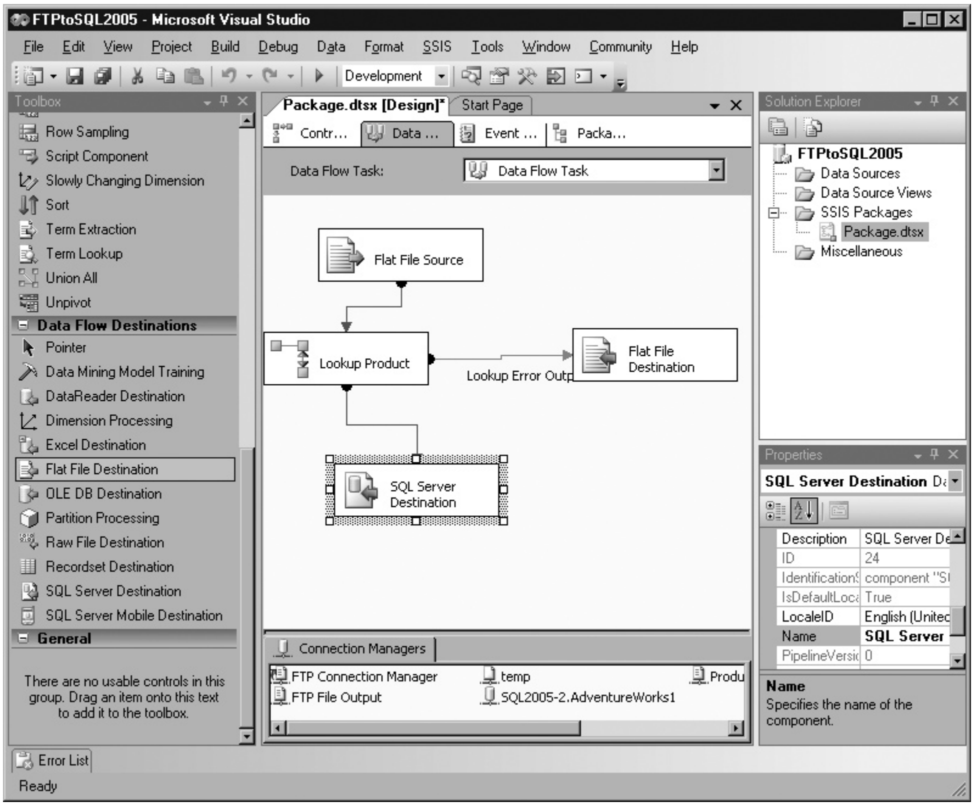
Figure 10-19 *SQL Destination Editor*

The first step in defining the OLE DB Destination is to select the appropriate OLE DB Connection Manager. Click the OLE DB Connection Manager drop-down and then select the SQL2005-2.AdventureWorks1 OLE DB Connection Manager. Next, in the Use A Table Or View drop-down select the table that will store the output. In Figure 10-19 you can see that the Purchasing.ProductShipments table has been selected.

NOTE

Testing the Execute SQL task that was previously created will produce a Purchasing .ProductsShipments table that you can use to define the OLE DB Connection Manager.

At this point the configuration of the SSIS package has been completed. You can optionally change the column mappings or view the contents of the ProductsShipments file. Clicking OK will save the properties configurations of the OLE DB Destination and close the SQL Destination Editor. The completed data flow is shown in Figure 10-20.



Description	SQL Server De
ID	24
Identification	component "S
IsDefaultLoca	True
LocaleID	English (Unitec
Name	SQL Server
PipelineVersi	0

Figure 10-20 *The completed data flow*

The elements on the data flow designer reflect the flow of events that will happen when the SSIS package executes the Data Flow task. Here you can see that the Flat File source (which points to the c:\temp\MyData.csv file) will be read as input. For each row read a lookup to the AdventureWorks Production.Products table will be performed. If a match for the incoming ProductID is found in the Production.Products table, then the data will be written to the Purchasing.ProductShipments table in the AdventureWorks database. If the incoming ProductID has an error and doesn't match any rows in the Products table, then the data will be written to the Flat File Destination (which points to the c:\temp\lookuperrors.csv file).

NOTE

If you want to view the data that's being sent between any of the sources, transformations, or destinations on the data flow designer, you can click either the green or red connection line to display the Edit Data Path dialog. From the Edit Data Path dialog select Data Viewers and Add. When data flows over the selected data path, it will be displayed in the data viewer.

Running the Package

After the SSIS package has been designed, you can run it from the SSIS Designer by clicking the green arrow on the toolbar, pressing F5, or selecting the Debug | Start Debugging option from the menu.

In order to execute the package, a file must be available for import that can be found on the remote FTP server. The following listing shows the contents of the sample import file that is capable of testing the SSIS package:

```
ShipProductID,AdwProductID,Name,ProductNumber,ShipDate,Units
10-504,504,Cup-Shaped Race,RA-2345,,38055
10-505,505,Cone-Shaped Race,RA-7490,,38055
10-506,506,Reflector,RF-9198,,38055
10-507,507,LL Mountain Rim,RM-M464,,38055
10-508,508,ML Mountain Rim,RM-M692,,38055
10-509,509,HL Mountain Rim,RM-M823,,38055
10-510,510,LL Road Rim,RM-R436,,38055
10-511,511,ML Road Rim,RM-R600,,38055
10-512,512,HL Road Rim,RM-R800,,38055
10-514,594,LL Mountain Seat Assembly,SA-M198,,38055
10-515,595,ML Mountain Seat Assembly,SA-M237,,38055
10-516,596,HL Mountain Seat Assembly,SA-M687,,38055
10-517,597,LL Road Seat Assembly,SA-R127,,38055
10-518,518,ML Road Seat Assembly,SA-R430,,38055
```

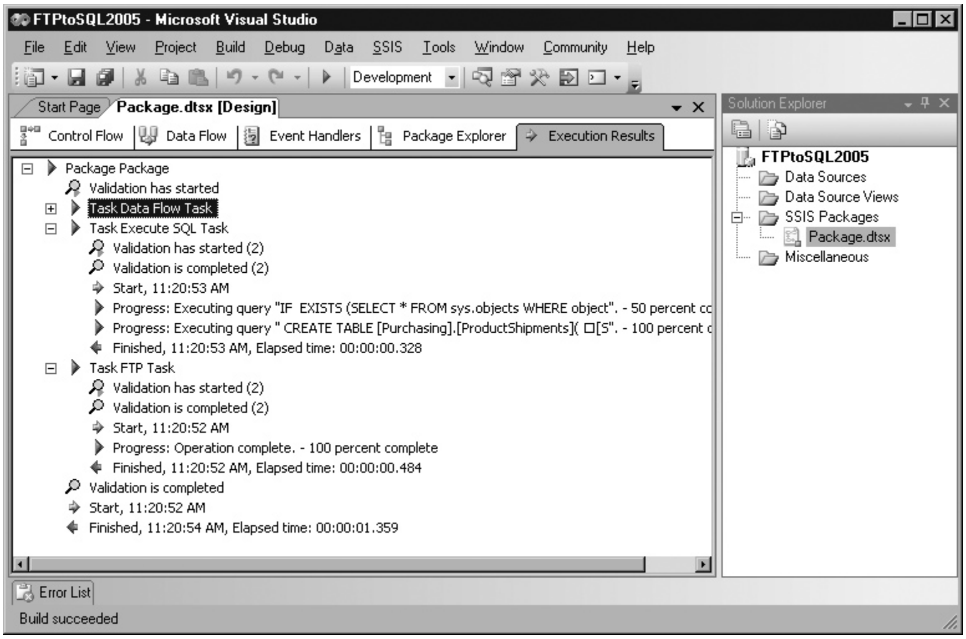
Four rows in this test file will produce error output. These are the rows with the values 594, 595, 596, and 597, as there are no matching values for these in the Production.Products table.

Running the SSIS package from the designer will show you the package's run-time status under the Execution Results tab as is shown in Figure 10-21, where you can see the status of each of the different tasks that compose the FTPtoSQL2005 package. The FTP task was completed first, at 11:20:52, followed by the Execute SQL task, which was completed at 11:20:53, and the Data Flow task was completed last.

Using Breakpoints

The SSIS Designer provides a fully interactive development environment, and in addition to being able to create SSIS packages it also offers the ability to debug them. You can use the SSIS Designer to set breakpoints at the package level, the container level, or the individual tasks level of an SSIS package.

Integration Services provides ten break conditions that you can enable on all tasks and containers. In addition, some tasks and containers include additional task-specific



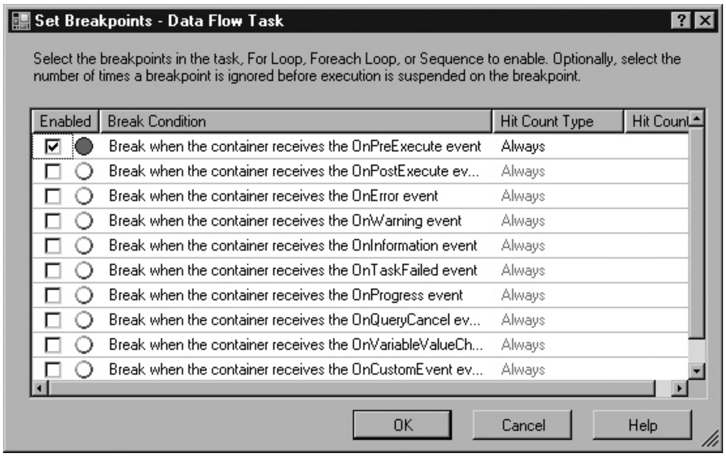
```
⊟ Package Package
    Validation has started
  ⊞ Task Data Flow Task
  ⊟ Task Execute SQL Task
      Validation has started (2)
      Validation is completed (2)
      Start, 11:20:53 AM
      Progress: Executing query "IF  EXISTS (SELECT * FROM sys.objects WHERE object". - 50 percent cc
      Progress: Executing query " CREATE TABLE [Purchasing].[ProductShipments]( □[S". - 100 percent c
      Finished, 11:20:53 AM, Elapsed time: 00:00:00.328
  ⊟ Task FTP Task
      Validation has started (2)
      Validation is completed (2)
      Start, 11:20:52 AM
      Progress: Operation complete. - 100 percent complete
      Finished, 11:20:52 AM, Elapsed time: 00:00:00.484
    Validation is completed
    Start, 11:20:52 AM
    Finished, 11:20:54 AM, Elapsed time: 00:00:01.359
```

Build succeeded

Figure 10-21 *Package execution results*

breakpoint conditions. When a breakpoint is encountered, the package halts execution and you can examine the contents of variables and other elements in the package.

To set a task breakpoint in an SSIS package, right-click the task and select the Edit Breakpoints option from the pop-up menu to display the Set Breakpoints dialog that you can see in Figure 10-22.



Select the breakpoints in the task, For Loop, Foreach Loop, or Sequence to enable. Optionally, select the number of times a breakpoint is ignored before execution is suspended on the breakpoint.

Enabled	Break Condition	Hit Count Type	Hit Count
☑ ●	Break when the container receives the OnPreExecute event	Always	
☐ ○	Break when the container receives the OnPostExecute ev...	Always	
☐ ○	Break when the container receives the OnError event	Always	
☐ ○	Break when the container receives the OnWarning event	Always	
☐ ○	Break when the container receives the OnInformation event	Always	
☐ ○	Break when the container receives the OnTaskFailed event	Always	
☐ ○	Break when the container receives the OnProgress event	Always	
☐ ○	Break when the container receives the OnQueryCancel ev...	Always	
☐ ○	Break when the container receives the OnVariableValueCh...	Always	
☐ ○	Break when the container receives the OnCustomEvent ev...	Always	

Figure 10-22 *Set Breakpoints*

You can set breakpoints on one or more of the conditions by placing a check mark in the Enabled box. In addition, you can use the Hit Count to control if and how frequently encountering the breakpoint will result in the halting of the packages. The default value is Always, meaning the package will always stop when the condition is encountered. However, by specifying a hit count, you can control how many times the condition must be encountered before the package execution is paused.

Using Checkpoints

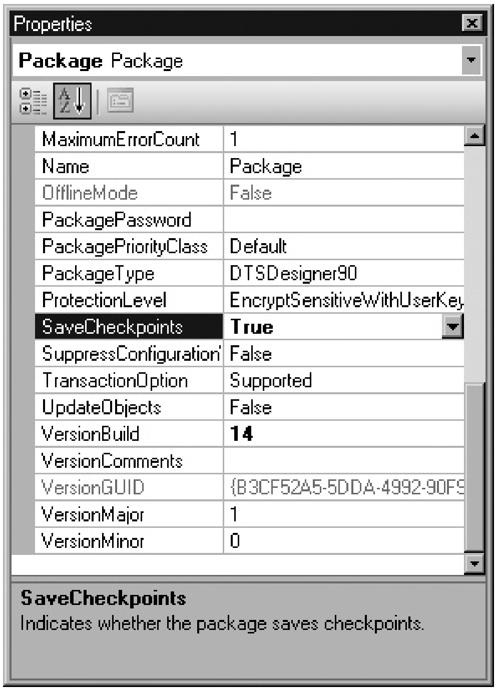
Checkpoints enable a failed SSIS package to be restarted at the spot where the execution was ended. Using checkpoints can significantly improve the recoverability of packages that contain complex operations. In addition, they can provide considerable time savings for the recovery of packages that contain long-running tasks because the package doesn't need to reprocess all of the tasks prior to the checkpoint. When checkpoints are enabled, information about the package's execution is written to a checkpoint file. SSIS will use the data in this file to determine which control flow tasks in the package have been executed. If a package that is using checkpoints fails, the SSIS DTR engine can use the checkpoint file to restart the package at the point of failure.

Checkpoints apply to the package's control flow, not to the data flow. Control-flow containers are the basic unit of checkpoint restartability. When the execution of a package that uses checkpoints is restarted, the package execution begins with the failed control flow task. If that control flow task uses any data flow within that task, then the data flow will be rerun in its entirety—the task will not pick up from the last row transferred. Even with this minor limitation, checkpoints offer a great improvement in package recoverability.

Checkpoints are enabled by setting the package's SaveCheckpoints property to True in the SSIS package properties. You can see the SaveCheckpoints property in Figure 10-23.

Once checkpoints are enabled, you also need to tell the SSIS package where to write the checkpoint data. To do this, you must supply a filename to the CheckpointFileName property. In addition, the way SSIS treats running packages where there is an existing checkpoint file is controlled by the CheckpointUsage property. The CheckpointUsage property supports the following values:

CheckpointUsage Value	Description
Never	The checkpoint file is not used, and the package always starts from the beginning of the control flow.
Always	The checkpoint file is always used, and the package restarts from the point of the previous execution failure. The package's execution will fail if the checkpoint file is not present.
IfExists	The package restarts from the point of the previous execution failure if the checkpoint file exists. If there is no checkpoint file, execution starts at the beginning of the control flow.



MaximumErrorCount	1
Name	Package
OfflineMode	False
PackagePassword	
PackagePriorityClass	Default
PackageType	DTSDesigner90
ProtectionLevel	EncryptSensitiveWithUserKey
SaveCheckpoints	**True**
SuppressConfiguration'	False
TransactionOption	Supported
UpdateObjects	False
VersionBuild	**14**
VersionComments	
VersionGUID	{B3CF52A5-5DDA-4992-90F9
VersionMajor	1
VersionMinor	0

SaveCheckpoints
Indicates whether the package saves checkpoints.

Figure 10-23 *Enabling checkpoints*

Using Transactions

SSIS also supports database transactions. When using transactions, the database changes performed by a package can be committed as a unit if a package runs successfully, or the changes can be rolled back as a unit if the package execution fails. Transactions can be enabled for all SSIS container types, including packages, containers, loops, and sequence containers. You enable transaction support using the container's TransactionOption property, which you set using the SSIS Designer or programmatically. The TransactionOption property supports the following values:

TransactionOption Values	Description
Not Supported	The container does not start a transaction and will not join an existing transaction that was initiated by a parent container.
Supported	The container does not start a transaction but will join an existing transaction that was started by a parent container.
Required	The container starts a transaction. If an existing transaction has already been started by the parent container, the container will join it.

Package Security

SSIS packages can contain sensitive authentication information, and saving those packages opens up the possibly of a security exposure. To protect against this possibility, SSIS supports the encryption of sensitive information. SSIS uses the Triple Data Encryption Standard (3DES) cipher algorithm with a key length of 192 bits, and packages are encrypted either when they are created or when they are exported. SSIS package encryption is controlled using the package's ProtectionLevel property, which supports the following values:

ProtectionLevel Value	Description
DontSaveSensitive	Sensitive data is not saved in the package. When the package is opened, the sensitive data will not be present and the user will need to provide the sensitive data.
EncryptSensitiveWithUserKey	Sensitive data is saved as a part of the package and is encrypted with a key that's based on the user who created or exported the package. Only that user will be able to run the package. If another user opens the package, the sensitive data will not be available.
EncryptSensitiveWithPassword	Sensitive data is saved as a part of the package and is encrypted with a user-supplied password. When the package is opened the user must provide a password to access the sensitive data. If the password is not provided, the package will be opened without the sensitive data.
EncryptAllWithPassword	The entire contents of the package will be encrypted with a user-supplied password. When the package is opened, the user must provide the package's password. If the password is not provided, the package will not be able to be opened.
EncryptAllWithUserKey	The entire contents of the package will be encrypted with a key that's based on the user key for the user who created or exported the package. Only the user who created the package will be able to open it.
ServerStorage	The package is not encrypted. Instead, the package's contents are secured according to the database's object access security. If the ServerStorage value is used, the package must be saved to the sysdtspackages90 table in the msdb database. It cannot be saved to the file system.

Deploying Packages

SSIS supports the deployment of packages through the use of the package configurations and the ability to easily deploy packages using the package deployment utility. In the next section you'll see how to create a configuration for an SSIS package as well as how to use the package deployment utility.

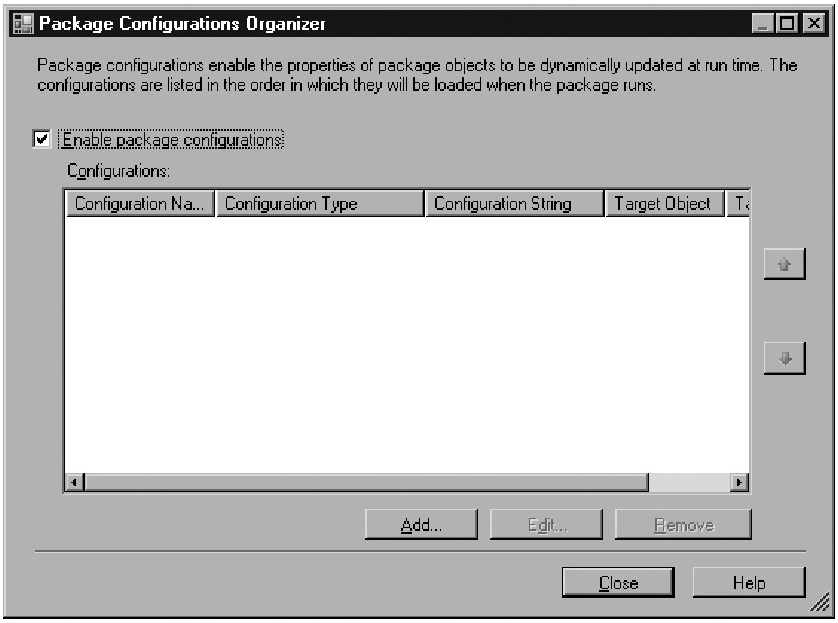
Creating Configurations

Configuration information enables an SSIS package to automatically load external information at run time. You can use configurations to pass in variable values and connection information to an SSIS package at run time. For variables, the Value property is assigned with the value that is passed in when the package is run. Likewise, for connection information the Connection Manager's properties such as ConnectionString, ServerName, and InitialCatalog can be assigned to dynamically change the server system that will be used by an SSIS package.

SSIS configurations are created by using the Package Configuration Organizer, which is started from BIDS. To create a package configuration for an Integration Services project, select the SSIS | Package Configurations option in BIDS. This will start the Package Configurations Organizer tool that is shown in Figure 10-24.

You can create multiple configurations for a single package. The configurations are applied to the package in the order that they are displayed in the Package Organizer. You can use the directional arrows shown on the right side of Figure 10-24 to move a configuration up or down in the list.

You can also create a single configuration that can be applied to multiple packages. For example, if you want to deploy a package to several systems where the only difference in the package properties is the server name, you could create a configuration that uses an environment variable to supply the server name.



Package Configurations Organizer

Package configurations enable the properties of package objects to be dynamically updated at run time. The configurations are listed in the order in which they will be loaded when the package runs.

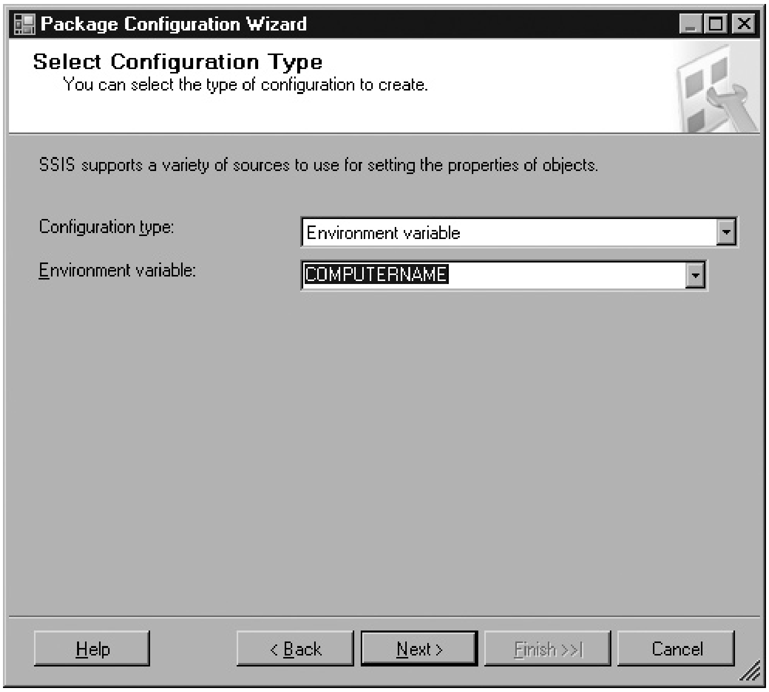
☑ Enable package configurations

Figure 10-24 *The Package Configuration Organizer*

To create a configuration, first check the Enable Package Configurations check box and then click Add to start the SSIS Configuration Wizard. The Configuration Wizard steps you through creating a package configuration. Click past the Wizard welcome screen to display the Configuration Type dialog shown in Figure 10-25.

The Configuration Type drop-down enables you to select the data source that will be used by the configuration. SSIS package configuration supports the following configuration types:

- XML configuration file
- Environment variable
- Registry entry
- Parent package variable
- SQL Server


Package Configuration Wizard

Select Configuration Type

You can select the type of configuration to create.

SSIS supports a variety of sources to use for setting the properties of objects.

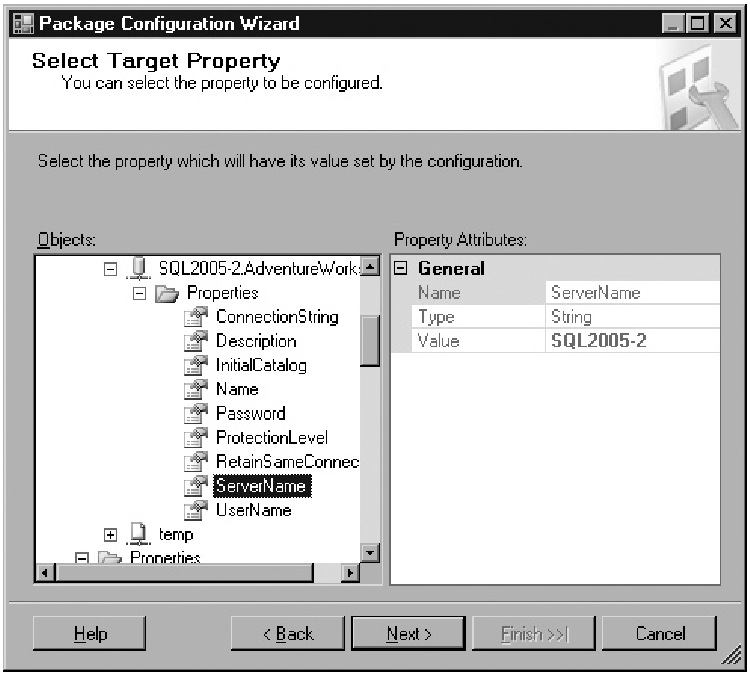
Configuration type: Environment variable

Environment variable: COMPUTERNAME

Figure 10-25 *Configuration Type*

In Figure 10-25 you can see that the type of Environment variable has been selected, along with the COMPUTERNAME variable. XML file configurations and SQL Server configurations support the selection of multiple properties in a single configuration object. The other configuration types permit only one configurable property per configuration. Click Next to display the dialog shown in Figure 10-26, which shows where you select the package's properties or variables that will have their values set by the configuration when the package is run.

In Figure 10-26 you can see that the properties for the OLE DB Connection Manager named SQL2005-2.AventureWorks1 have been expanded and that the ServerName property has been selected. This will enable the COMPUTERNAME environment variable to be substituted for the OLE DB Connection Manager's ServerName in the connection string when this package attempts to use the OLE DB connection. Clicking Next displays the configuration summary screen, which allows you to view and confirm your selections. If you need to make changes, you can use the Back button to page back through the Configuration Wizard and make any needed changes. Otherwise, clicking Next adds the configuration to the package and will display the Package Configuration Organizer with your configuration as you can see in Figure 10-27.



Package Configuration Wizard

Select Target Property

You can select the property to be configured.

Select the property which will have its value set by the configuration.

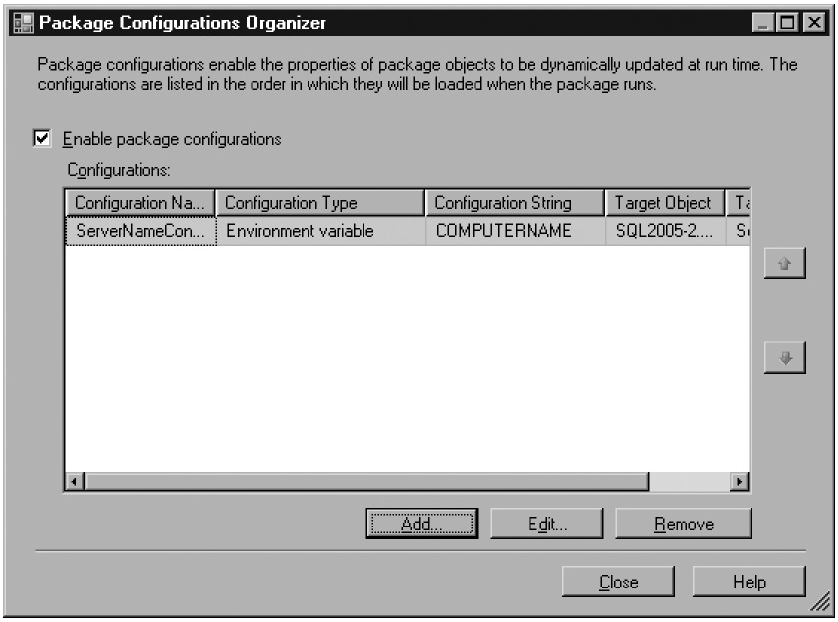
Objects:

- SQL2005-2.AdventureWork
  - Properties
    - ConnectionString
    - Description
    - InitialCatalog
    - Name
    - Password
    - ProtectionLevel
    - RetainSameConnec
    - ServerName
    - UserName
- temp
- Properties

Property Attributes:

General	
Name	ServerName
Type	String
Value	SQL2005-2

Figure 10-26 *Select Target Property*


Package Configurations Organizer
Package configurations enable the properties of package objects to be dynamically updated at run time. The configurations are listed in the order in which they will be loaded when the package runs.
☑ Enable package configurations
Configurations:
Add...
Edit...
Remove
Close
Help

Configuration Na...	Configuration Type	Configuration String	Target Object	Ta
ServerNameCon...	Environment variable	COMPUTERNAME	SQL2005-2....	S

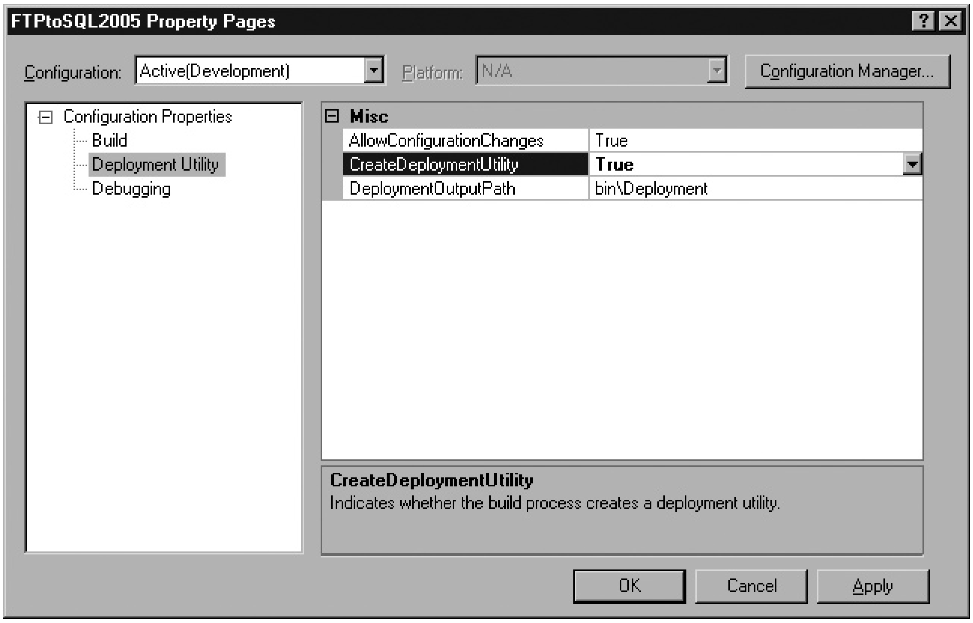
Figure 10-27 *The completed package configuration*

Later if you need to modify a configuration, click Edit to rerun the Configuration Wizard and select different objects and different properties.

Using the Package Deployment Utility

SSIS contains a handy feature called the Package Deployment Utility that allows you to assemble your SSIS packages, package configurations, and supporting files into a deployment folder and build an executable setup file to install your packages. To create the Package Deployment Utility, right-click the project properties in the BIDS Solution Explorer pane and then select the Properties option to display the Property Pages dialog box as shown in Figure 10-28.

Set the CreateDeploymentUtility option to True on the project property page. Then build your project by selecting the Build Solution option on the BIDS menu. Building the project creates the DTSDeploymentManifest.xml file and copies the project along with the DTSInstall.exe utility to the bin/Deployment folder or to the location specified in the DeploymentOutputPath property. The DTSDeploymentManifest.xml file lists the packages and the package configurations in the project. The DTSInstall.exe program runs the Package Installer Wizard.



Configuration: Active(Development)

Platform: N/A

AllowConfigurationChanges True

CreateDeploymentUtility True

DeploymentOutputPath bin\Deployment

CreateDeploymentUtility
Indicates whether the build process creates a deployment utility.

Figure 10-28 *Package Deployment Utility*

Programming with the SQL Server Integration Services APIs

In addition to providing a graphical development environment, SSIS also provides an object model API for both the DTR and the DTP that enables you to programmatically create and execute SSIS packages. Programming the data flow engine enables you to automate the creation and configuration of the SSIS tasks, transformations, and data flow tasks, and to create custom components. The run-time engine is exposed both as a native COM object model and as a fully managed object model. The SSIS data flow engine is written in native code, but it can be controlled programmatically using a managed .NET object model. In this next section you'll see an example of how you can use the SQL Server Integration Services API in a console application to create and execute a new SSIS package.

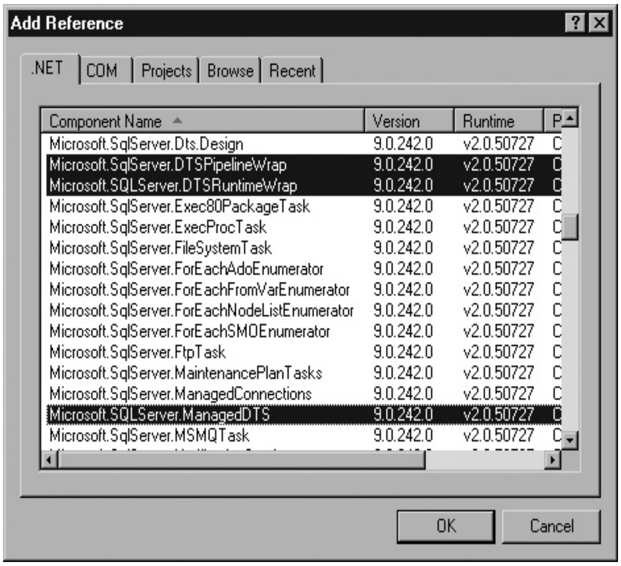
The SQL Server Integration Services API is located in a number of different assemblies: Microsoft.SqlServer.ManagedDTS.dll, SqlServer.DTSPipelineWrap .dll, and SqlServer.DTSRuntimeWrap.dll. To use these assemblies in your program, you need to add references for each of them in your project. Then you can use the Integration Services classes to create both SSIS DTP and DTR objects in your application. To add references to your project, select the Project | Add Reference menu option to display the Add Reference dialog. Scroll through the list until you see the Microsoft.SqlServer.DTSPipelineWrap, Microsoft.SqlServer.DTSRuntimeWrap, and Microsoft.SqlServer.ManagedDTS assemblies listed in the Component Name list. Select these assemblies as is illustrated in Figure 10-29. Click OK to add the references to your project.

Next, add import directives for the SSIS assembly namespaces to the Declarations section of your project. Using import directives enables you to use the classes in the imported namespaces without having to fully qualify the names. The following code listing shows how to create import directives for the SSIS namespace:

```
Imports Microsoft.SqlServer.Dts.Runtime
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports SSISRuntime = Microsoft.SqlServer.Dts.Runtime.Wrapper
```

NOTE

To avoid compile-time errors due to common object names, it's best to use an alternative import name when importing both Microsoft.SqlServer.DtsPipeline.Wrapper and Microsoft.SqlServer .DtsRuntime.Wrapper. That's why the third Imports statement uses the name SSISRuntime.



Component Name	Version	Runtime	P
Microsoft.SqlServer.Dts.Design	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.DTSPipelineWrap	9.0.242.0	v2.0.50727	C
Microsoft.SQLServer.DTSRuntimeWrap	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.Exec80PackageTask	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.ExecProcTask	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.FileSystemTask	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.ForEachAdoEnumerator	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.ForEachFromVarEnumerator	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.ForEachNodeListEnumerator	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.ForEachSMOEnumerator	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.FtpTask	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.MaintenancePlanTasks	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.ManagedConnections	9.0.242.0	v2.0.50727	C
Microsoft.SQLServer.ManagedDTS	9.0.242.0	v2.0.50727	C
Microsoft.SqlServer.MSMQTask	9.0.242.0	v2.0.50727	C

Figure 10-29 *Adding references to SSIS assemblies*

After adding the SSIS references to your project and import directives for the appropriate namespaces, you're ready to begin using the SSIS APIs in your application. The following code sample shows how you can create a package using the SSIS APIs:

```
Module CreateSSISPackage
    Sub Main()
        ' Create the Package
        Console.WriteLine("Creating the MySSIS Package")
        Dim myPackage As New Package()
        myPackage.PackageType = DTSPackageType.DTSDesigner90
        myPackage.Name = "MySSISPackage"
        myPackage.Description = "Created using the SSIS API"
        myPackage.CreatorComputerName = System.Environment.MachineName
        myPackage.CreatorName = "Otey"

        'Add the OLE DB and Flat File Connection Managers
        Console.WriteLine("Creating the MyOLEDBConnection")
        Dim cnOLEDB As ConnectionManager = _
          MyPackage.Connections.Add("OLEDB")
        cnOLEDB.Name = "MyOLEDBConnection"
        cnOLEDB.ConnectionString = _
          "Provider=SQLNCLI;Integrated Security=SSPI;" _
          & "Initial Catalog=AdventureWorks;Data Source=SQL2005-2;"

        Console.WriteLine("Creating the MyFlatFileConnection")
        Dim cnFile As ConnectionManager = _
          myPackage.Connections.Add("FLATFILE")
        cnFile.Properties("Name").SetValue(cnFile, "MyFlatFileConnection")
        cnFile.Properties("ConnectionString").SetValue _
          (cnFile, "c:\temp\MySSISFileExport.csv")
        cnFile.Properties("Format").SetValue(cnFile, "Delimited")
        cnFile.Properties("ColumnNamesInFirstDataRow") _
          .SetValue(cnFile, False)
        cnFile.Properties("DataRowsToSkip").SetValue(cnFile, 0)
        cnFile.Properties("RowDelimiter").SetValue(cnFile, vbCrLf)
        cnFile.Properties("TextQualifier").SetValue(cnFile, """")
```

Near the top of this listing you can see where a new SSIS package object named myPackage is created. Next, the package's properties are assigned values. The most important of these are the PackageType and Name properties, where the values of DTSpackageType.DTSdesigner90 and MySSISPackage are used.

After creating the package object, the next step is to create Connection Managers for the package. In this example, the package will be performing a simple export from SQL Server to the file system, which requires two Connection Managers: an OLE DB

Connection Manager to connect to SQL Server, and a Flat File Connection Manager to write the export file. First, the OLE DB Connection Manager is created and named MyOLEDBConnection. The Connection's Add method is then used to add the new ConnectionManager object to the package's Connections collection. Then the OLE DB ConnectionManager's ConnectionString property is assigned a connection string that will connect to the AdventureWorks database on the server named SQL2005-2 using integrated security. After that, a similar process creates a Flat File ConnectionManager object named MyFlatFileConnection and adds it to the package's Connections collection. Then the ConnectionString property of the MyFlatFileConnection is assigned the value of c:\temp\MySSISFileExport.csv. The following property assignments set up a delimited file type for the export operation.

After the creation of the package and Connection Manager objects, the next step to the creation of a Data Flow task is shown in the following code listing:

```
'Add a Data Flow Task
Console.WriteLine("Adding a Data Flow Task")
Dim taskDF As TaskHost = _
  TryCast(myPackage.Executables.Add("DTS.Pipeline"), TaskHost)
taskDF.Name = "DataFlow"

Dim DTP As MainPipe
DTP = TryCast(taskDF.InnerObject, MainPipe)

' Add the OLE DB Source
Console.WriteLine("Adding an OLEDB Source")
Dim DFSource As IDTSComponentMetaData90
DFSource = DTP.ComponentMetaDataCollection.New()
DFSource.ComponentClassID = "DTSAdapter.OLEDBSource"
DFSource.Name = "OLEDBSource"

' Connect, populate the Input collections and disconnect
Dim SourceInst As CManagedComponentWrapper = _
  DFSource.Instantiate()
SourceInst.ProvideComponentProperties()
DFSource.RuntimeConnectionCollection(0).ConnectionManagerID _
    = myPackage.Connections("MyOLEDBConnection").ID
DFSource.RuntimeConnectionCollection(0).ConnectionManager _
    = DtsConvert.ToConnectionManager90 _
     (myPackage.Connections("MyOLEDBConnection"))
SourceInst.SetComponentProperty("OpenRowset", "[Sales].[Customer]")
SourceInst.SetComponentProperty("AccessMode", 0)
SourceInst.AcquireConnections(Nothing)
SourceInst.ReinitializeMetaData()
SourceInst.ReleaseConnections()
```

Near the top of this listing you can see where a new Data Flow task named taskDF is created. Here the Data Flow task must be set up to read from the OLE DB Connection Manager that will be the source of the data and then write to the flat file connection that acts as the data destination. After creation of the Data Flow task, an OLE DB data adapter for the source data is created that's named DFSource. The DFSource object's ComponentClassID is set to DTSADpater.OLEDBSource, defining it as an OLE DB connection, and the Name of the data flow source is set to OLEDBSource. Next, an instance of the DFSource object is created in order to populate the input connections. This enables the downstream data flow components to see the input metadata. Here the OLE DB connection is set to the Sales.Customer table from the AdventureWorks database. After the input metadata has been collected, the connection is released.

The next step is to define the data flow destination as is shown in the following listing:

```
' Add the Flat File Destination
Console.WriteLine("Adding a Flat File Destination")
Dim DFDestination As IDTSComponentMetaData90
DFDestination = DTP.ComponentMetaDataCollection.New()
DFDestination.ComponentClassID = _
  "DTSAdapter.FlatFileDestination"
DFDestination.Name = "FlatFileDestination"

' Create an instance of the component
Dim DestInst As CManagedComponentWrapper = _
  DFDestination.Instantiate()
DestInst.ProvideComponentProperties()
DFDestination.RuntimeConnectionCollection(0).ConnectionManagerID _
    = myPackage.Connections("MyFlatFileConnection").ID
DFDestination.RuntimeConnectionCollection(0).ConnectionManager _
    = DtsConvert.ToConnectionManager90 _
       (myPackage.Connections("MyFlatFileConnection"))

' Map a connection between the source and destination
DTP.PathCollection.New().AttachPathAndPropagateNotifications _
  (DFSource.OutputCollection(0), DFDestination.InputCollection(0))

' Add columns to the FlatFileConnectionManager
Dim MyFlatFilecn As _
  SSISRuntime.IDTSConnectionManagerFlatFile90 = Nothing
For Each cm As ConnectionManager In myPackage.Connections
    If cm.Name = "MyFlatFileConnection" Then
        MyFlatFilecn = TryCast(cm.InnerObject, _
          SSISRuntime.IDTSConnectionManagerFlatFile90)
        DtsConvert.ToConnectionManager90(cm)
    End If
Next
```

```
 ' Get the columns from the source
 Dim InColumns As IDTSVirtualInputColumnCollection90 _
     = DFDestination.InputCollection(0).GetVirtualInput() _
 .VirtualInputColumnCollection()

 Dim col As SSISRuntime.IDTSConnectionManagerFlatFileColumn90
 Dim name As SSISRuntime.IDTSName90
 For cols As Integer = 0 To InColumns.Count - 1 Step 1
     col = MyFlatFilecn.Columns.Add()
     ' Set the last column delimiter to CRLF
     If cols = InColumns.Count - 1 Then
         col.ColumnDelimiter = vbCrLf
     Else
         col.ColumnDelimiter = ","
     End If
     col.ColumnType = "Delimited"
     col.DataType = InColumns(cols).DataType
     col.DataPrecision = InColumns(cols).Precision
     col.DataScale = InColumns(cols).Scale
     name = TryCast(col, SSISRuntime.IDTSName90)
     name.Name = InColumns(cols).Name
 Next

DestInst.AcquireConnections(Nothing)
 DestInst.ReinitializeMetaData()

 Dim wrapper As CManagedComponentWrapper = _
   DFDestination.Instantiate()
 Dim vInput As IDTSVirtualInput90 = _
   DFDestination.InputCollection(0).GetVirtualInput()
 For Each vColumn As IDTSVirtualInputColumn90 In _
   vInput.VirtualInputColumnCollection
     wrapper.SetUsageType(DFDestination _
       .InputCollection(0).ID, vInput, vColumn.LineageID, _
         DTSUsageType.UT_READONLY)
 Next

 ' Match the input and output columns
 Dim exCol As IDTSExternalMetadataColumn90
 For Each InCol As IDTSInputColumn90 In _
   DFDestination.InputCollection(0).InputColumnCollection
     exCol = DFDestination.InputCollection(0) _
       .ExternalMetadataColumnCollection(InCol.Name)
     wrapper.MapInputColumn(DFDestination _
       .InputCollection(0).ID, InCol.ID, exCol.ID)
 Next
 DestInst.ReleaseConnections()
```

Here the data flow destination named DFDestination is created. Its ComponentClassID is set to DTSAdapter.FlatFileDestination, defining it as an OLE flat file in the file system, and the Name of the data flow source is set to FlatFileDestination. Next, an instance of the DFDestination object name DestInst is created in order to map the input columns to the columns in the destination output file.

You create the precedence between the data flow source and destination using the DTP object's AttachPathAndPropagateNotifications method. The next section of code adds the column to the FlatFile ConnectionManager. It does this by reading the metadata that was previously retrieved from the OLE DB Source connection. The DFDestination object's GetVirtualInput method populates the input collection, and then a For-Each loop is used to set the attributes for each of the output columns.

Once the collection of the columns has been created, the next step is to map the input columns from the OLE DB Source to the flat file output columns. Here a one-to-one mapping is used, and a simple For-Each loop reads through the input metadata, associating each column to the corresponding output column. After the mappings have been set up, the connection is released using the ReleaseConnections method.

This completes the code needed to create the SSIS package. In the next section of code you can see how to validate, save, and execute the package:

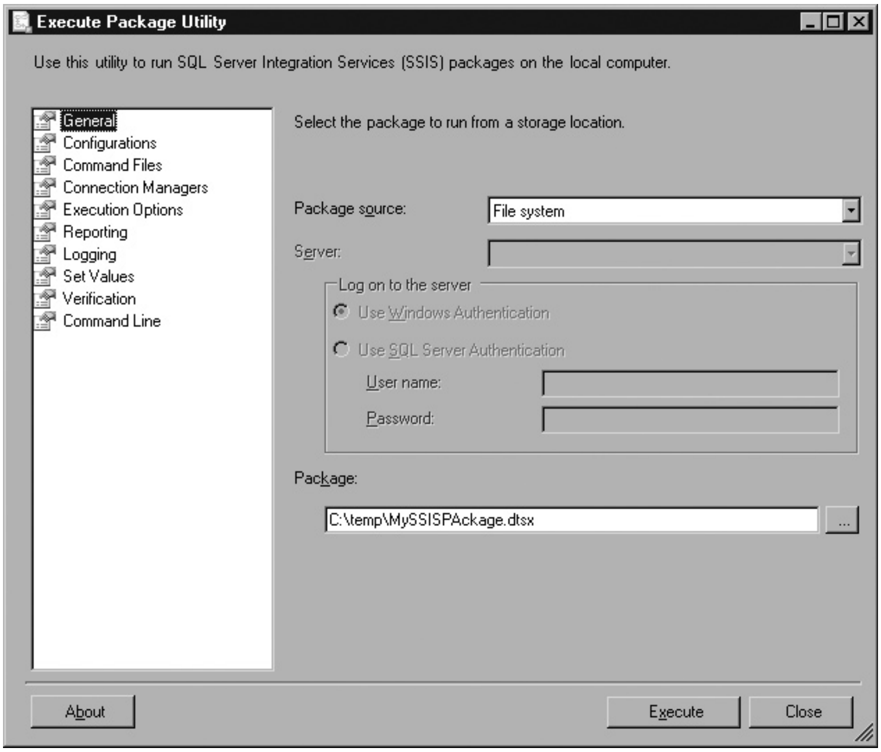
```
        ' Validate the package
        Console.WriteLine("Validating the MySSISPackage")
        Dim pkgStatus As DTSExecResult = myPackage.Validate _
          (Nothing, Nothing, Nothing, Nothing)
        System.Console.WriteLine("Validation result: " & _
          pkgStatus.ToString())

        ' Save the package
        Console.WriteLine("Saving the MySSISPackage")
        Dim SSISExe As New Application()
        SSISExe.SaveToXml("c:\temp\MySSISPAckage.dtsx", myPackage, Nothing)

        ' Execute the Package
        If pkgStatus = DTSExecResult.Success Then
            Console.WriteLine("Executing the MySSISPackage")
            Dim pkgResult As DTSExecResult = myPackage.Execute()
            Console.WriteLine("MySSISPackage results: " _
              & pkgResult.ToString)
        Else
            Console.WriteLine("Package validation failed")
        End If
        Console.ReadKey()
    End Sub

End Module
```

Calling the SSIS package object's Validate method causes the SSIS engine to parse the package, ensuring that all of the settings are valid. Here the results of the Validate method are assigned to the status variable.

Next, regardless of the status, the package is saved to the file system by creating an instance of the SSIS Application object and then using that object's SaveToXML method. The first argument of the SaveToXML method specifies the filename to save the package under, and the second argument passes in an instance the package object.

Finally, the contents of the pkgStatus object are checked to ensure that the package was valid. If it is, then the package's Execute method is called to run the SSIS package and perform the data export. The execution results are returned in the pkgResult variable.

After the SSIS package has been successfully executed, a file containing the exported data named MySSISFileExport.csv along with an SSIS package named MySSISPAckage.dtsx will be found in the c:\temp directory. Double-clicking the MySSISPackage.dtsx package in the file system will launch the Execute Package Utility that you can see in Figure 10-30. The Execute Package Utility allows you to browse the package's properties, optionally changing properties and variables, as well as to execute the package.


Execute Package Utility

Use this utility to run SQL Server Integration Services (SSIS) packages on the local computer.

Select the package to run from a storage location.

Package source: File system

Server:

Log on to the server

● Use Windows Authentication

○ Use SQL Server Authentication

User name:

Password:

Package:

C:\temp\MySSISPAckage.dtsx

Figure 10-30 *The newly created SSIS package*

The previous example illustrated creating and running an SSIS package. However, as you can see in the following listing, if you just want to execute an existing SSIS package, the code is much simpler. The following code listing shows how to execute an SSIS package from a console application that has a reference added for the Microsoft.SqlServer.Dts.Runtime assembly:

```
Imports Microsoft.SqlServer.Dts.Runtime

Module Module1

    Sub Main()
        Dim sPath As String
        Dim oPkg As New Package
        Dim oApp As New Application
        Dim oResults As DTSExecResult

        sPath = "C:\temp\MySSISPackage.dtsx"
        pkg = oApp.LoadPackage(sPath, Nothing)
        oResults = oPkg.Execute()
        Console.WriteLine(oResults.ToString())
        Console.ReadKey()

    End Sub

End Module
```

At the top of this listing you can see an import directive for the Microsoft .SqlServer.Dts.Runtime assembly. Within the Sub Main procedure you can see where the DTS Application object's LoadPackage method is used to load the MySSISPackage.dtsx package file to the c:\temp directory in the file system. In this example the MySSISPackage.dtsx package was created using the code from the previous listings. After loading the package, the Execute method is used to run the package. The results are then displayed on the console.

Summary

SQL Server Integration Services is an all-new subsystem in SQL Server 2005 that completely replaces the older Data Transformation Services subsystem that was present in the older versions of SQL Server. In this chapter you learned about SSIS's simple SSIS Import and Export Wizard for performing basic data transfer operation as well as how to create more complex, multistep packages using the SSIS Designer. You saw how to use package checkpoints for recoverability and transactions to ensure data integrity, as well as how to create configurations for flexible package deployments. In addition, you also saw how to use the SSIS APIs to programmatically create SSIS packages from a .NET application.