5

Creating an End-to-End Package

Now that you've learned about all the basic tasks and transforms in SSIS, you can jump into some practical applications for SSIS. You'll first start with a normal transformation of data from a series of flat files into SQL Server. Next you'll add some complexity to a transformation by archiving the files automatically. The last example will show you how to make a package that handles basic errors and makes the package more dynamic. As you run through the tutorials, remember to save your package and project often to avoid any loss of work.

Basic Transformation Tutorial

As you can imagine, the primary reason that people use SSIS is to read the data from a source and write it to a destination after it's massaged. This tutorial will walk you through a common scenario where you want to copy data from a flat file source to a SQL Server table without massaging the data. Don't worry; things will get much more complex later in your next package.

Start the tutorial by going online to the Wiley Web site and downloading the sample extract that contains zip code information about cities in Florida. The zip code extract was retrieved from public record data from the 1990 census and has been filtered down to just Florida cities to save on your download time. You'll use this in the next tutorial as well, so it's very important not to skip this tutorial. You can download the sample extract file called ZipCode.txt from this book's Web page at www.wrox.com. Place the file into a directory called C:\SSISDemos.

Open Business Information Development Studio (BIDS) and select File ➪ New ➪ Project. Then select Integration Services Project as your project type. Type **ProSSISChapter5** as the project name, and accept the rest of the defaults (as shown in Figure 5-1).

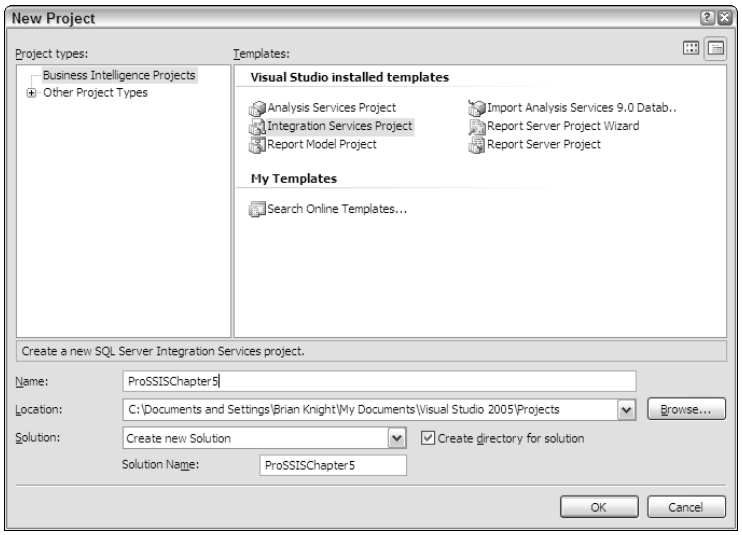

Figure 5-1

The project will be created, and you'll see a default Package.dtsx package file in the Solution Explorer. Right-click on the Package.dtsx file in the Solution Explorer and select Rename. Rename the file ZipLoad.dtsx. When you're asked if you'd like to rename the package as well, click Yes. If the package isn't opened yet, double-click on it to open it in the Package Designer.

Creating Connections

Now that you have the package ready to begin, you need to create a shared connection that can be used across multiple packages. In the Solution Explorer, right-click on Data Sources and select New Data Source. This opens the Data Source Wizard. Select the "Create a data source based on an existing or new connection" radio box and click New, which opens the window to create a new Connection Manager.

> *There are many ways you could have created the connection. For example, you could have created it as you're creating each source and destination. Once you're more experienced with the tool, you'll find what works best for you.*

Your first Connection Manager will be to SQL Server, so select Native OLE DB\SQL Native Client. For the Server Name option, type the name of your SQL Server and enter the authentication mode that is necessary for you to read and write to the database. Lastly, select the AdventureWorks database and click OK. If you don't have the AdventureWorks database, you can pick any other database on the server. You can optionally test the connection. You will then have a data source in the Data Source box that should be selected. Click Next and name the data source AdventureWorks.

You'll use other connections as well, but for those, you'll create connections that will be local to the package only and not shared. With the ZipLoad package open, right-click in the Connection Managers box below and select New Connection from Data Source. You should see the AdventureWorks data source you created earlier. Select that data source and click OK. Once the Connection Manager is created, right-click on it and rename it AdventureWorks if it's not already named that. This is, of course, optional and just keeps us all on the same page.

Next, create a Flat File connection and point it to the ZipCode.txt file in your C:\SSISDemos directory. Right-click in the Connection Manager area of Package Designer, and select New Flat File Connection. Name the connection ZipCode Extract and type any description you like. Point the File Name option to C:\SSISDemos\ZipCode.txt or browse to the correct location by clicking Browse.

You need to set the Format drop-down box to Delimited with <none> set for the Text Qualifier option. The Text Qualifier option allows you to specify that character data is wrapped in quotes or some type of qualifier. This helps you when you have a file that is delimited by commas and you also have commas inside some of the text data that you do not wish to separate by. Setting a Text Qualifier will ignore those commas inside the text data. Lastly, select "Tab {t}" from the Header Row Qualifier drop-down box and check the "Column names in first data row" option. This states that your first row contains the column names for the file. Your final configuration for this page should look like Figure 5-2.

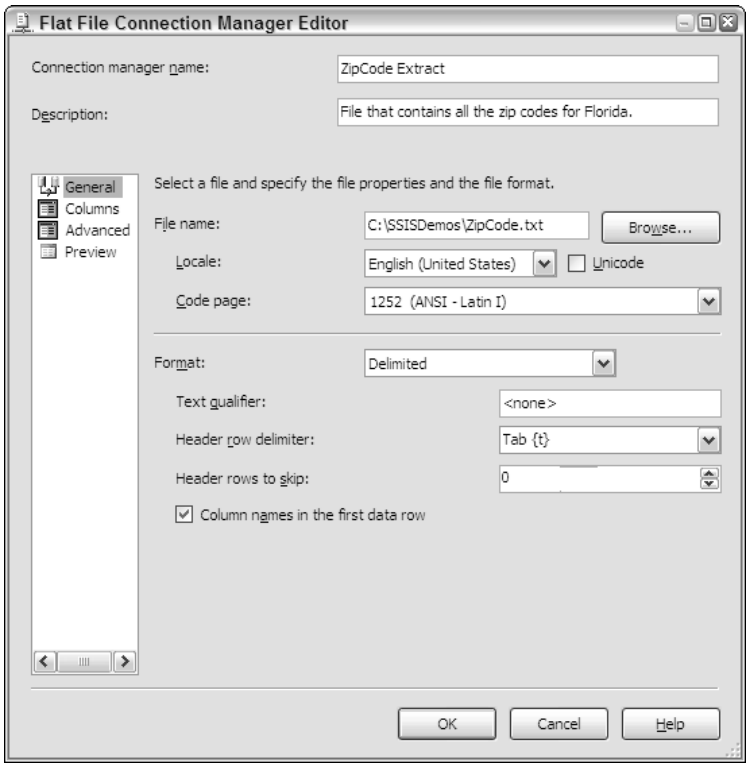

Figure 5-2

You can go to the Columns page to view a preview of the first 101 rows and set the row and column delimiters. The defaults are generally fine for this screen. The Row Delimiter option should be set to {CR}{LF}, which means that a carriage return separates each row. The Column Delimiter option should have carried over from the first page and will again be set to "Tab {t}". In some extracts that you may receive, the header record may be different from the data records and the configurations won't be exactly the same as in the example.

The Advanced page is where you can specify the data types for each of the three columns. The default for this type of data is a 50-character string, which is excessive in this case. Click Suggest Types to comb through the data and find the best data type fit for the data. This will open the Suggest Column Types dialog box, where you should accept the default options and click OK.

You can now see that the data types in the Advanced page have changed for each column. One column in particular was incorrectly changed. When combing through the first 100 records, the Suggest Column Types dialog box selected a "two-byte signed integer [DT_I2]" for the zip code column. While this would work for the data extract you have, it won't work once you get to some states that have zip codes that begin with a zero. Change this column to a string by selecting string [DT_STR] from the DataType drop-down box, and change the length of the column to 5 by changing the OutputColumnWidth option (shown in Figure 5-3). The last configuration change is to change the TextQualified option to False and click OK.

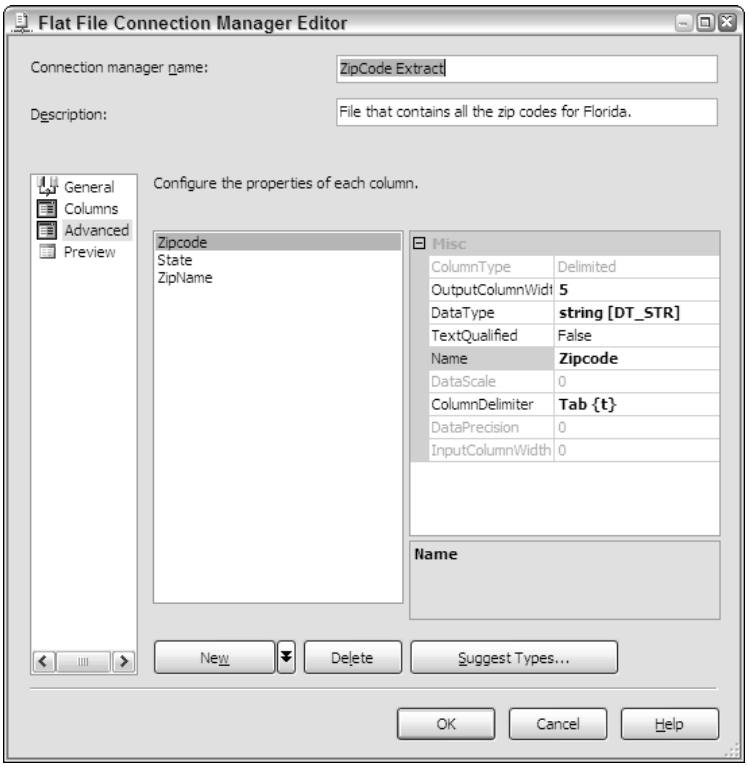

Figure 5-3

Creating the Tasks

With the first few connections created, you can go ahead and create your first task. In this tutorial, you'll have only a single task, which will be the Data Flow task. In the Toolbox, drag the Data Flow task over to the design pane in the Control Flow tab. Next, right-click on the task and select Rename to rename the task "Load ZipCode Info."

Creating the Data Flow

Now comes the more detailed portion of almost all of your packages. Double-click on the task to drill into the data flow. This will automatically take you to the Data Flow tab. You'll see that "Load ZipCode Info" was transposed to the Data Flow Task drop-down box. If you had more than this one Data Flow task, then more would appear as options in the drop-down box.

Drag and drop a Flat File Source onto the data flow design pane, and then rename it "Florida ZipCode File." All the rename instructions in these tutorials are optional, but they will keep you on the same page and make your operational people happier because they'll understand what's failing. Open the "Florida ZipCode File" source and point it to the Connection Manager called ZipCode Extract. Go to the Columns page and take notice of the columns that you'll be outputting to the path. You've now configured the source, and you can click OK.

Next, drag and drop a SQL Server Destination onto the design pane and rename it AdventureWorks. Connect the path (green arrow) from the "Florida ZipCode File" source to the AdventureWorks destination. Double-click on the destination and select AdventureWorks from the Connection Manager drop-down box. For the Use a Table or View option, click the New button next to the drop-down box. This is how you can create a table inside BIDS without having to go back to SQL Server Management Studio. The default DDL for creating the table will use the destination's name (AdventureWorks), and the data types may not be exactly what you'd like, as shown below:

```
CREATE TABLE [AdventureWorks] (
    [Zipcode] VARCHAR(5),
    [State] VARCHAR(2),
    [ZipName] VARCHAR(16)
)
```

Suppose this won't do for your picky DBA, who is concerned about performance. In this case, you should rename the table ZipCode (taking out the brackets) and change each column's data type to a more suitable size and type:

```
CREATE TABLE ZipCode (
    Zipcode CHAR(5),
    State CHAR(2),
    ZipName VARCHAR(16)
)
```

Once you have completed changing the DDL, click OK and the table name will be transposed into the table drop-down box. Finally, go to the Mapping page to ensure that the inputs are mapped to the outputs correctly. SSIS attempts to map the columns based on name, and in this case, since you just created the table with the same column names, it should be a direct match, as shown in Figure 5-4.

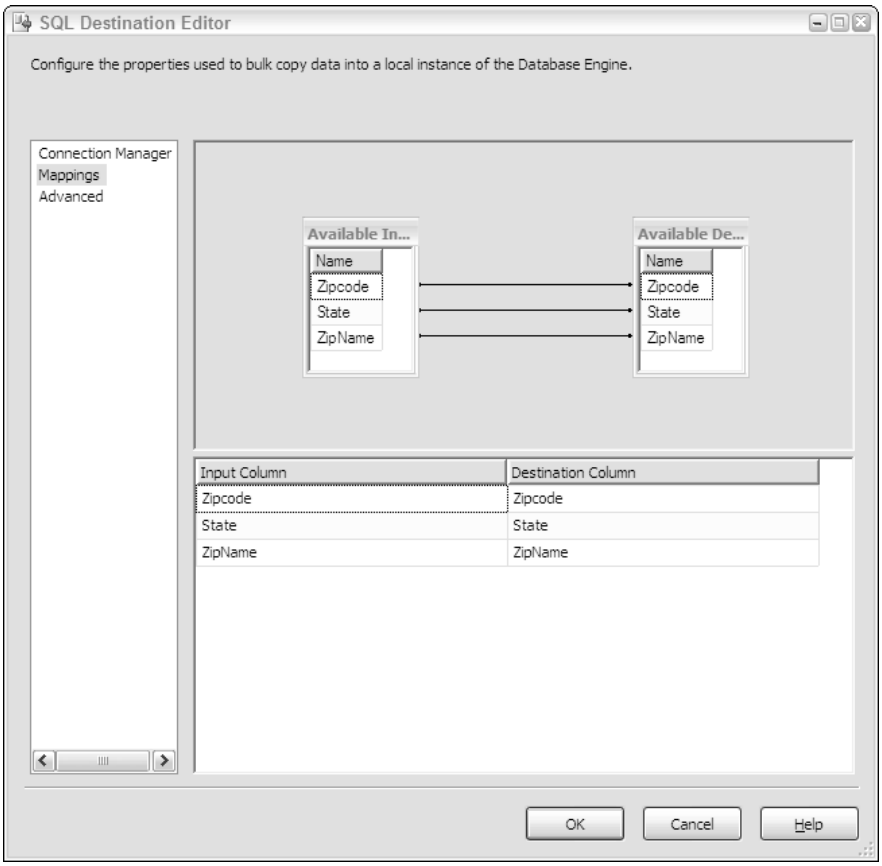

Figure 5-4

Once you've confirmed that the mappings look like Figure 5-4, click OK.

Completing the Package

With the basic framework of the package now constructed, you need to add one more task into the control flow to ensure that you can run this package multiple times. To do this, click on the Control Flow tab and drag an Execute SQL task over to the design pane. Rename the task "Purge ZipCode Table." Double-click on the task and select AdventureWorks from the Connection drop-down box. Finally, type the following query for the SQLStatement option (you can also click the ellipsis button and enter the query):

```
DELETE FROM ZipCode
```

Click OK to complete the task configuration. Connect the task as a parent to the "Load ZipCode Info" task. To do this, click the "Purge ZipCode Table" task and drag the green arrow onto the "Load ZipCode Info" task.

Saving the Package

Your first package is now complete. Go ahead and save the package by clicking the Save icon in the top menu or by selecting File ➪ Save Selected Items. It's important to note here that by clicking Save, you're saving the .DTSX file to the project, but you have not saved it to the server yet. To do that, you'll have to deploy the solution or package. The tutorial will cover that in the last section of this chapter.

Executing the Package

With the package complete, you can attempt to execute it. Do this by selecting the green arrow in the upper menu. You can also right-click on the ZipCode.dtsx package file in the Solution Explorer and select Execute Package. The package will take a few moments to compile and validate, and then it will execute.

You can see the progress under the Progress tab or in the Output window. In the Controller Flow tab, you'll see the two tasks go from yellow to green (hopefully). If both turn green, then the package execution was successful. In the event your package failed, you can look in the Output window to see why. The Output window should be open by default, but in case it's not, you can open it by clicking View ➪ Other Windows ➪ Output.

You can go to the Data Flow tab to see how many records were copied over. You can see the Controller tab in the left image in Figure 5-5 and the Data Flow tab in the right image. Notice the number of records displays in the path as SSIS moves from transform to transform.

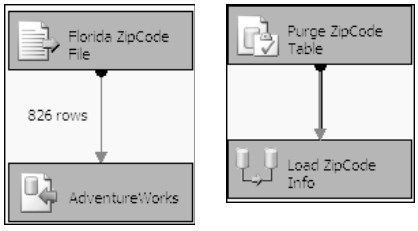

Figure 5-5

By default, when you execute a package, you'll be placed in debug mode. Changes you make in this mode will not be made available until you run the package again. To break out of this mode, click the square stop icon or click Stop Debugging under the Debug menu.

Typical Mainframe ETL with Data Scrubbing

With the basic ETL out of the way, you will now jump into a more complex SSIS package that attempts to scrub data. You can start this scenario by downloading some public data from the Florida Department of Corporations at ftp://dossftp.dos.state.fl.us/public/doc/cor/. Manually download the file 010305c.dat from the FTP site and place the file into a directory called C:\SSISDemos.

In this scenario, you run a credit card company that's interested in marketing to newly formed domestic corporations in Florida. You want to prepare a data extract each day for the marketing department to perform a mail merge and perform a bulk mailing. Yes, your company is an old-fashioned, snail-mail spammer.

Luckily the Department of State for Florida has an interesting extract you can use to empower your marketing department. As with most extracts that come from a legacy system, this extract unfortunately is in a fixed-width format. This means that the file is not delimited by any symbol, and you'll have to manually configure each column's width. In these types of events, your mainframe group or the third party would send you a record layout for the file extract. In this case, you can see the record layout here: http://www.sunbiz.org/corpweb/inquiry/corfile.html. The record layout for this extract would look like this:

```
01  ANNUAL_MICRO_DATA_REC.
    03  ANNUAL_COR_NUMBER                    PIC X(12).
    03  ANNUAL_COR_NAME                      PIC X(48).
    03  ANNUAL_COR_STATUS                    PIC X(01).
    03  ANNUAL_COR_FILING_TYPE               PIC X(15).
    03  ANNUAL_COR_2ND_MAIL_ADD_1            PIC X(42).
    03  ANNUAL_COR_2ND_MAIL_ADD_2            PIC X(42).
    03  ANNUAL_COR_2ND_MAIL_CITY             PIC X(28).
    03  ANNUAL_COR_2ND_MAIL_STATE            PIC X(02).
    03  ANNUAL_COR_2ND_MAIL_ZIP              PIC X(10).
    03  ANNUAL_COR_2ND_MAIL_COUNTRY          PIC X(02).
    03  ANNUAL_COR_FILE_DATE                 PIC X(08).
    03  ANNUAL_COR_FEI_NUMBER                PIC X(14).
    03  ANNUAL_MORE_THAN_SIX_OFF_FLAG        PIC X(01).
    03  ANNUAL_LAST_TRX_DATE                 PIC X(08).
    03  ANNUAL_STATE_COUNTRY                 PIC X(02).
    03  ANNUAL_REPORT_YEAR_1                 PIC X(04).
    03  ANNUAL_HOUSE_FLAG_1                  PIC X(01).
    03  ANNUAL_REPORT_DATE_1                 PIC X(08).
    03  ANNUAL_REPORT_YEAR_2                 PIC X(04).
    03  ANNUAL_HOUSE_FLAG_2                  PIC X(01).
    03  ANNUAL_REPORT_DATE_2                 PIC X(08).
    03  ANNUAL_REPORT_YEAR_3                 PIC X(04).
    03  ANNUAL_HOUSE_FLAG_3                  PIC X(01).
    03  ANNUAL_REPORT_DATE_3                 PIC X(08).
    03  ANNUAL_RA_NAME                       PIC X(42).
    03  ANNUAL_RA_NAME_TYPE                  PIC X(01).
    03  ANNUAL_RA_ADD_1                      PIC X(42).
    03  ANNUAL_RA_CITY                       PIC X(28).
    03  ANNUAL_RA_STATE                      PIC X(02).
    03  ANNUAL_RA_ZIP5                       PIC X(05).
    03  ANNUAL_RA_ZIP4                       PIC X(04).
    03  ANNUAL_PRINCIPALS                    OCCURS 6 TIMES.
        05  ANNUAL_PRINC_TITLE                   PIC X(04).
        05  ANNUAL_PRINC_NAME_TYPE               PIC X(01).
        05  ANNUAL_PRINC_NAME                    PIC X(42).
        05  ANNUAL_PRINC_ADD_1                   PIC X(42).
        05  ANNUAL_PRINC_CITY                    PIC X(28).
        05  ANNUAL_PRINC_STATE                   PIC X(02).
        05  ANNUAL_PRINC_ZIP5                    PIC X(05).
        05  ANNUAL_PRINC_ZIP4                    PIC X(04).
    03  FILLER                               PIC X(04).
```

Each row represents another column, and you can see the length of each column after the word `PIC`. This type of format is very typical from legacy systems and even from newer ones. You won't be using all the columns in this example, so you won't have to define them all. The business goals of this package are as follows:

- Create a package that finds the files in the C:\SSISDemos directory and loads the file into your relational database.
- Archive the file after you load it to prevent it from being loaded twice.
- The package must self-heal. If a column is missing data, the data should be added automatically.
- If the package encounters an error in its attempt to self-heal, output the row to an error queue.

Start a new package in your existing ProSSISChapter5 BIDS project from the first tutorial. Right-click in the Solution Explorer and select Add ➪ New Item. From the New Item dialog box, choose "New SSIS Package." This will create Package1.dtsx, or some numeric variation on that name. Rename the file CorporationLoad.dtsx, and the package should also be renamed. Double-click on the package to open it.

Just like the last package you created, right-click in the Connection Managers area and select a new Connection Manager from the Data Source that points to the AdventureWorks data connection. You now have two packages using the same Shared Connection. Next, create a new Flat File Connection Manager just as you did in the last tutorial. When the configuration screen opens, call the connection "Corporation Extract" in the General page. Type any description you'd like. For this Connection Manager, you're going to configure the file slightly differently. This time, the file will not be delimited by any character or tab. Instead, select Fixed Width from the Format drop-down box. You will also not have any type of column headers this time, so there's no need to check the same "Column names in the first data row" option.

A fixed-width file means that each column is not delimited by any character or tab. Instead, you have to manually specify where each column begins and ends. Most mainframe files you receive will be sent to you in this format. As you'll see in a moment, it's not nearly as easy to configure. You can take the record layout that was discussed earlier and deduce how you should set the starting points of each column. To do this, go to the Columns page in the Flat File Connection Manager Editor (shown in Figure 5-6). Then, start by entering a Row Width of 1172 characters for this file (this number would be given to you by the originator of the file typically).

Next, specify where each column ends by adding a vertical bar at each column end point. Add that bar by left-clicking on the spot where the column ends. The ruler above the data shows you what character you're on before adding a new vertical bar, and the data preview in the Source Data Columns area may help guide you. For this example, you can use the following table as a hint to remind you where you need to place lines. To remove the vertical bar, you can double-click on it, or to move it, just click and drag it to the proper spot.

Position Stops at	Field Description
12	Corporate number
60	Corporation name
61	Corporate status
65	Filing Type

Table continued on following page

Position Stops at	Field Description
118	Mailing address line 1
160	Mailing address line 2
188	City
190	State
200	Zip code
202	Country
210	Filing date
224	FEI number
1172	Records you'll throw out

Notice that for this example, you're throwing out most of the data. It is grouped in one column that you'll remove in the path.

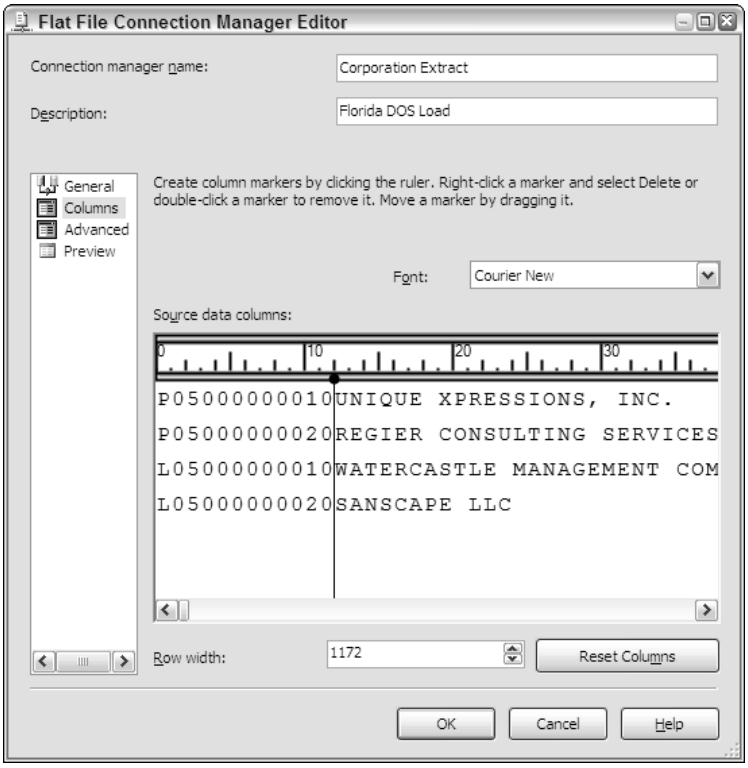

Figure 5-6

After you finish adding all the columns, go to the Advanced tab. Click Suggest Types as you did in the last example. In the Suggest Column Types dialog box, click OK to accept the defaults. Again, this dialog box will accomplish most of the data type discovery for you, but it will make some mistakes that you will have to repair. The first mistake is Column 8 (ZipCode column). You may need to change this DataType option to String [DT_STR] and the OutputColumnWidth to 10 characters. Finally, you'll want to change Column 10 to String [DT_STR]. After you've properly set these two columns, click OK to save the Connection Manager.

Creating the Data Flow

With the mundane work of creating the connections now out of the way, you can go ahead and create the fun transformation. As you did in the last package, you must first create a Data Flow task by dragging it from the Toolbox. Name this task "Load Corporate Data." Double-click on the task to go to the Data Flow tab.

Drag and drop onto the design pane a Flat File source and rename it "Uncleansed Corporate Data." Double-click the source and select Corporate Extract as your Connection Manager that you'll be using. Next, go to the Columns tab and uncheck Column 11 and Column 12 since the marketing department won't need that data. You'll add the destination and transformation in a moment after the scenario is expanded a bit.

Handling Dirty Data

Before you go deeper into this scenario, you should take a time-out to look more closely at this data. As you were creating the connection, a very observant person (I did not notice this until it was too late) may have noticed that some of the important data that you'll need is missing. For example, the city and state are missing from some of the records.

To fix this for the marketing department, you'll use some of the transforms that were discussed in the last chapter to send the good records down one path and the bad records down a different path. You will then attempt to cleanse the bad records and then send those back through the main path. There may be some records you can't cleanse (such as corporations with foreign postal codes) that you'll just have to write to an error log and deal with at a later date.

First, standardize the postal code to a five-digit format. Currently, some have five digits and some have the full 10-digit zip code with a dash (five digits, a dash, and four more digits). Some are nine-digit zip codes without the dash. To standardize the zip code, you use the Derived Column transform. Drag the transform over from the Toolbox and rename it "Standardize Zip Code."

Connect the source to the transformation and double-click on the transform to configure it. Expand the Columns tree in the upper-left corner, find [Column 8], and drag it onto the grid below. This will pre-fill some of the information for you in the derived columns grid area. You now need to create an expression that will take the various zip codes formats in the [Column 8] output column and output only the first five characters. One way of doing this is with the `SUBSTRING` function. If you choose to solve the business problem with that method, the code would look like this:

```
SUBSTRING([Column 8],1,5)
```

This code should be typed into the Expression column in the grid. Next, select that the derived column will replace the existing Column 8 output by selecting that option from the Derived Column drop-down box. You can see what the options should look like in Figure 5-7. Once you've completed the transformation, click OK.

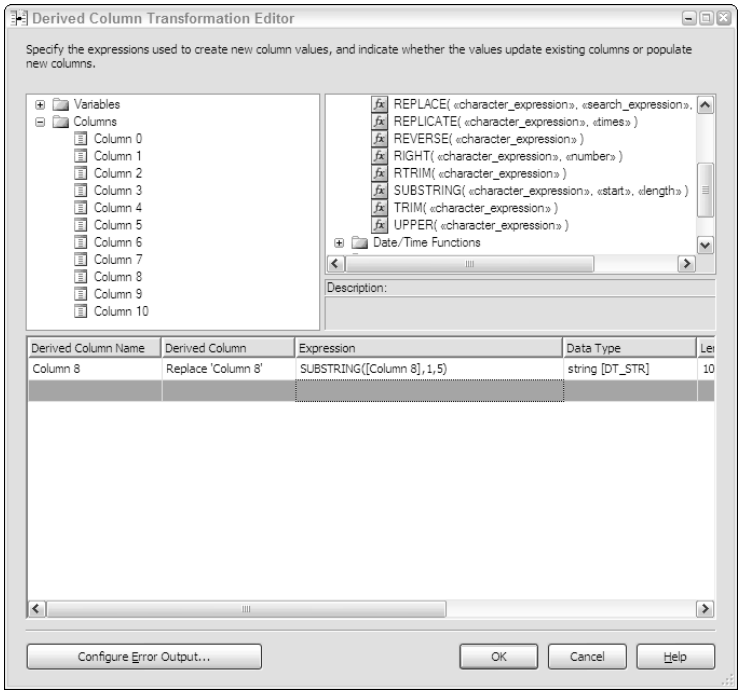

Figure 5-7

The Conditional Split Transformation

Now that you've standardized the data slightly, drag and drop the Conditional Split transformation onto the design pane and connect the green arrow from the source to the Conditional Split. Rename the transform "Find Bad Records." The Conditional Split transformation will enable you to push certain bad records into a data-cleansing process.

To cleanse the data that has no city or state, you'll write a condition that says that any row that is missing a city or state will be moved to a cleansing path in the data flow. Double-click on the Conditional Split transform after you have connected it from the Derived Column transform to edit the transformation. Create a condition called "Missing State or City" by typing its name in the Output Name column. You will

now need to write an expression that looks for empty records. One method of doing this is to use the `LTRIM` function. The two vertical bars (||) in the following code are the same as an `OR` login in your code. The following code will check for a blank Column 6 or Column 7:

```
LTRIM([Column 6]) == "" || LTRIM([Column 7]) == ""
```

The last thing you'll need to do is give a name to the default output if the coded condition is not met. Call that output "Good Data," as shown in Figure 5-8. The default output is the name of the output that will contain the data that did not meet your conditions.

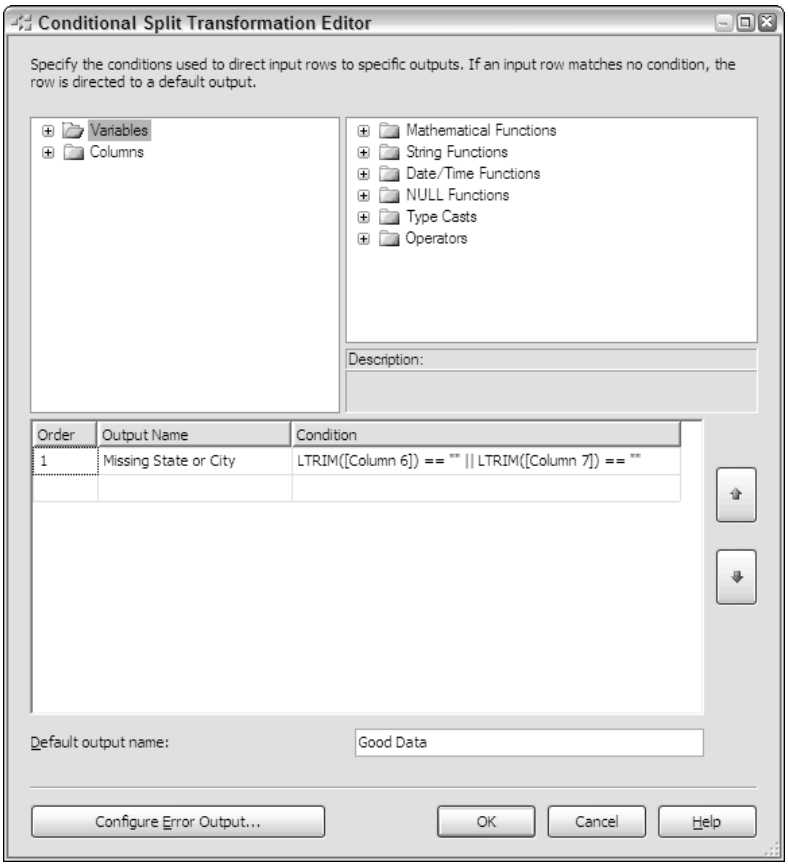

Figure 5-8

If you have multiple cases, always place conditions that you feel will capture most of the records at the top of the list.

The Lookup Transformation

Next, drag and drop the Lookup transformation onto the design pane. When you connect to it from the Conditional Split transformation, you'll see the Input Output Selection dialog box (shown in Figure 5-9). Select "Missing State or Zip" and click OK. This will send any bad records to the lookup transformation from the Conditional Split. Rename the Lookup transformation "Fix Bad Records."

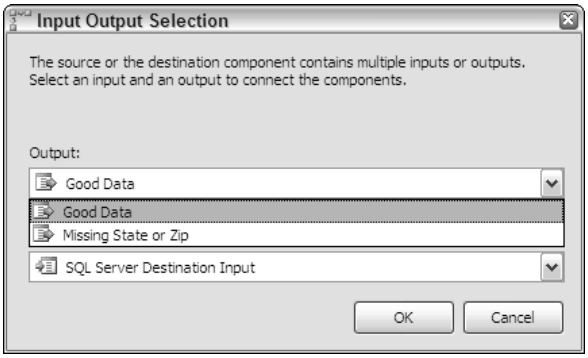

Figure 5-9

The Lookup transformation allows you to map a city and state to the rows that are missing that information by looking the record up against the ZipCode table you loaded earlier. Open up the transformation editor for the Lookup transform, and in the Reference Table page, select AdventureWorks as the Connection Manager that contains your lookup table. Select ZipCode from the Use Table or View drop-down box.

Next, go to the Columns page and drag Column 8 from the left Available Input Columns to the right ZipCode column in the Available Lookup Columns table. This will create an arrow between the two tables as shown in Figure 5-10. Then, check the State and City columns that you wish to output. This will transfer their information to the bottom grid. Select that you wish for these columns to replace the existing Column 6 and Column 7. The ZipName column will replace Column 6 and the State column will replace Column 7 as shown in the grid in Figure 5-10. Click OK to exit the transform editor. There are many more options here, but you should stick with the basics for the time being.

The Union All Transformation

Now that your dirty data is cleansed, go ahead and send the sanitized data back into the main data path by using a Union All transformation. Drag and drop the Union All transform onto the design pane and connect the "Fix Bad Records" Lookup transform and the "Find Bad Records" Conditional Split transform onto the Union All transform. There is nothing more to configure with the Union All transformation.

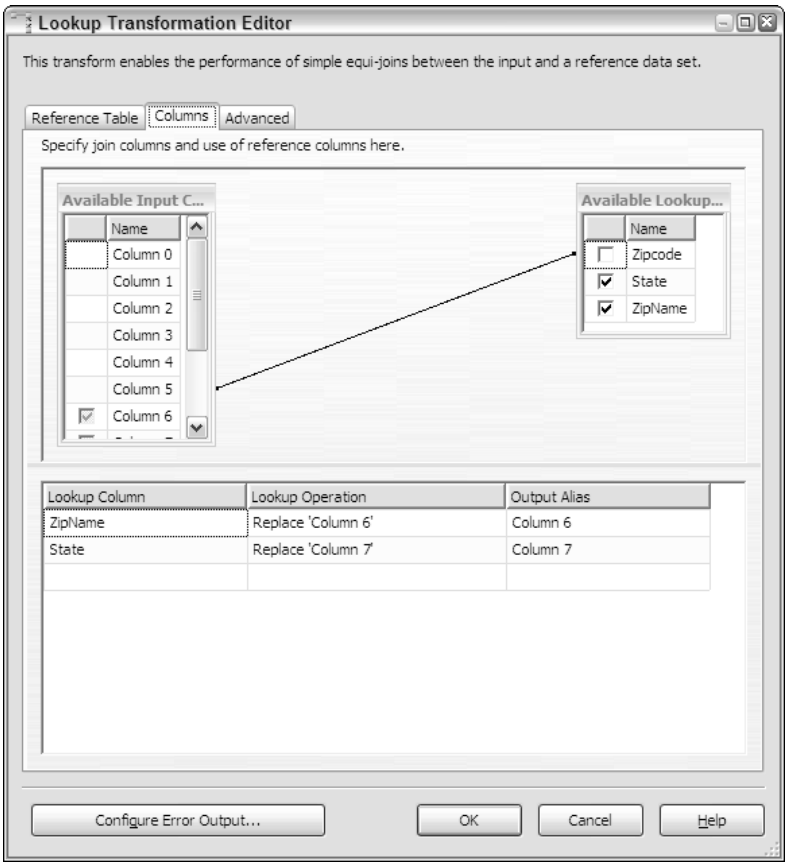

Figure 5-10

Finalizing

The last step in the data flow is to send the data to an OLE DB destination. Drag the OLE DB destination to the design pane and rename it "Mail Merge Table." Connect the Union All transform to the destination. Double-click on the destination and select AdventureWorks from the Connection Manager drop-down box. For the Use a Table or View option, select the New button next to the drop-down box. The default DDL for creating the table will use the destination's name (AdventureWorks), and the data types may not be exactly what you'd like, as shown here:

```
CREATE TABLE [Mail Merge Table] (
    [Column 0] VARCHAR(12),
    [Column 1] VARCHAR(48),
    [Column 2] VARCHAR(1),
    [Column 3] VARCHAR(4),
    [Column 4] VARCHAR(53),
    [Column 5] VARCHAR(42),
    [Column 6] VARCHAR(28),
    [Column 7] VARCHAR(2),
    [Column 8] VARCHAR(10),
    [Column 9] VARCHAR(2),
    [Column 10] VARCHAR(10)
)
```

Go ahead and change the schema to something a bit more useful. Change the table name and each column to a more logical name like this:

```
CREATE TABLE MarketingCorporation(
    CorporateNumber varchar(12),
    CorporationName varchar(48),
    FilingStatus char(1),
    FilingType char(4),
    AddressLine1 varchar(53),
    AddressLine2 varchar(42),
    City varchar(28),
    State char(2),
    ZipCode varchar(10),
    Country char(2),
    FilingDate varchar(10) NULL
)
```

You may have to map the columns this time because the column names are different. Go to the mappings page and map each column to its new name.

Handling More Bad Data

The unpolished package is essentially complete, but it has one fatal flaw that you're about to discover. Go ahead and execute the package. Depending on the year you're performing this demo, it may fail (it was written in 2005, and each year in the file is overwritten with a new year). Don't panic if your package fails. In 2005, the 010305c.dat file contained some superfluous data from individuals outside the country. You can see this by adding a grid data viewer between the "Find Bad Records" and "Fix Bad Records" transforms.

If you do this, you can see, for example, that in the 010305c.dat file, four records were sent to be cleansed by the Lookup transformation. Of those, only two had the potential to be cleansed. The other two records were for companies outside the country and could not be located in the Lookup transform that contained only Florida zip codes. You may remember that the business requirement was to only send marketing a list of domestic addresses for their mail merge product. They didn't care about the international addresses because you didn't have a business presence in those countries. The grid data viewer pointed out the problem in the last two records, though, and it looks something like Figure 5-11:

Column 6	Column 7	Column 8	Column 9
JACKSONVILLE, FLORIDA		32207	
BOYNTON BEACH		33436	US
CARACAS, 1062, VENEZUE...			
BELIZE CITY			BZ

Figure 5-11

The error you'd see in the output window when you executed the package would look something like this:

```
Error: 0xC020901E at Load Corporate Data, Fix Bad Records [87]: Row yielded no
match
during lookup.

Error: 0xC0209029 at Load Corporate Data, Fix Bad Records [87]: The "component "Fix
Bad
Records" (87)" failed because error code 0xC020901E occurred, and the error row
disposition on "output "Lookup Output" (89)" specifies failure on error. An error
occurred on the specified object of the specified component.

Error: 0xC0047022 at Load Corporate Data, DTS.Pipeline: The ProcessInput method on
component "Fix Bad Records" (87) failed with error code 0xC0209029. The identified
component returned an error from the ProcessInput method. The error is specific to
the
component, but the error is fatal and will cause the Data Flow task to stop
running.

Error: 0xC0047021 at Load Corporate Data, DTS.Pipeline: Thread "WorkThread0" has
exited
with error code 0xC0209029.

Error: 0xC0047039 at Load Corporate Data, DTS.Pipeline: Thread "WorkThread1"
received a
shutdown signal and is terminating. The user requested a shutdown, or an error in
another
thread is causing the pipeline to shutdown.

Error: 0xC0047021 at Load Corporate Data, DTS.Pipeline: Thread "WorkThread1" has
exited
with error code 0xC0047039.
```

You need to not fail the package on these types of problems, so send the bad records to an error queue that can be reviewed and cleaned out manually. To do this properly, you'll audit each record that fails and create an ErrorQueue table on the SQL Server. Drag over the Audit transformation from your Toolbox. Rename the Audit transformation "Add Auditing Info" and connect the failure path (the red arrow coming out of the "Fix Bad Records" transform after you click it) to the Audit transform.

You'll immediately see the Configure Error Output dialog box (shown in Figure 5-12). This screen allows you to tell SSIS how to react if certain errors occur in the transformation. The rows under the Truncation column distinguish what would happen if a row is too large to be added to the transform. The Error column tells SSIS how to react in the event of a transformation error. You can see to the right in the Description column what type of error it's expecting to handle for that particular transformation. For example, for the Lookup transformation the expected error this will trap is a lookup failure, meaning the

Lookup transformation was unable to find a match for the inputted record. For this example, select each of the error types to redirect the row as shown in Figure 5-10. The default is to fail the component, which would fail the entire package, or you can choose to just ignore the error completely by selecting Ignore from the same drop-down box.

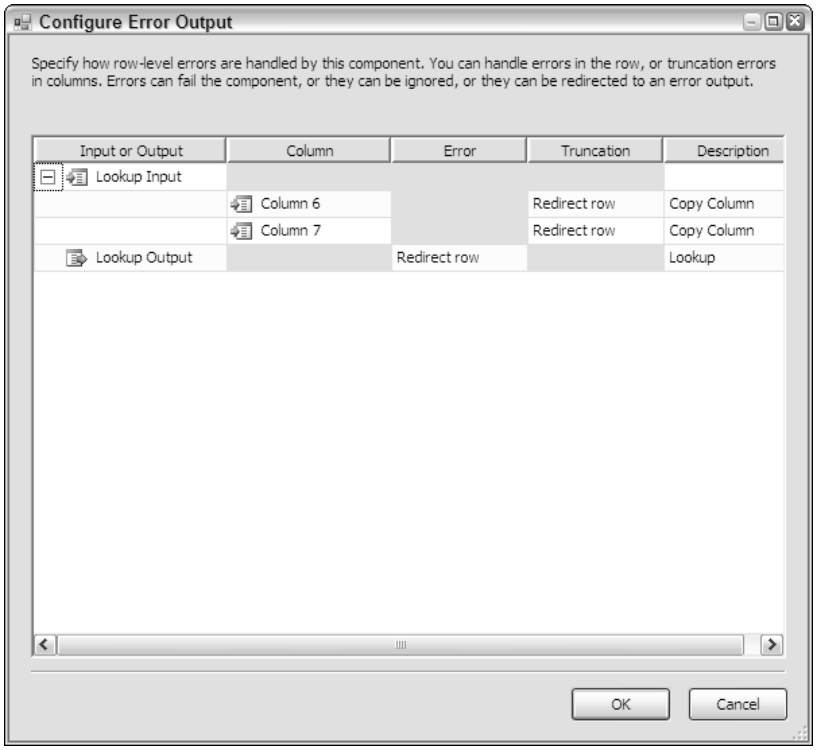

Figure 5-12

Once you've configured the Error Output handling as shown in Figure 5-12, click OK.

When you redirect bad rows, as you did here, two additional columns are added to the data flow: the Error Code and Error Description columns. This can tell someone who's expected to clean the queue what column caused the problem and the description of the error.

With the errors now being handled, double-click the Audit transform to configure that transformation. Go ahead and add two additional columns to the output. Select Task Name and Package Name from the drop-down boxes in the Audit Type column. This will transpose a default Output Column Name. Take out the spaces in each output column name, as shown in Figure 5-13, to make it easier to query later. You'll want to output this auditing information because you may have multiple packages and tasks loading data into the corporation table, and you'll want to track which package actually originated the error.

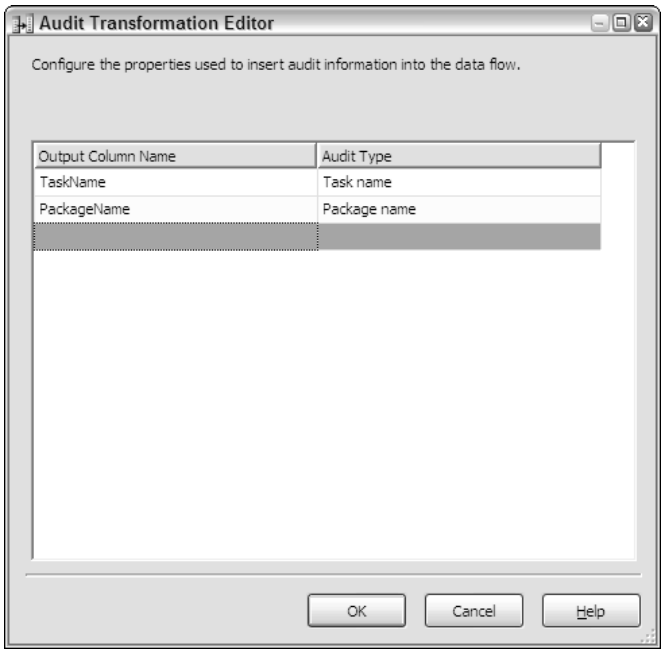

Figure 5-13

The last thing you need to do to polish up the package is to send the bad rows to the SQL Server ErrorQueue table. Drag another OLE DB destination over to the design pane and connect the Audit transformation to it. Rename the destination "Error Queue." Double-click on the destination and select AdventureWorks as the Connection Manager, and click New to add the ErrorQueue table. Name the table "ErrorQueue" and follow a similar schema to the one below:

```
CREATE TABLE ErrorQueue(
    CorporateNumber varchar(12),
    CorporationName varchar(48),
    FilingStatus char(1),
    FilingType char(4),
    AddressLine1 varchar(53),
    AddressLine2 varchar(42),
    City varchar(28),
    State char(2),
    ZipCode varchar(10),
    Country char(2),
    FilingDate varchar(10) NULL,
    ErrorCode INT,
    ErrorColumn INT,
    TaskName NVARCHAR(19),
    PackageName NVARCHAR(30)
)
```

In error queue tables like the one just illustrated, be very generous when defining the schema. In other words, you don't want to create another transformation error trying to write into the error queue table. Instead, you may want to define everything as a varchar column and give more space than is actually needed.

You may have to map the columns this time due to the column names being different. Go to the mappings page and map each column to its new name.

You are now ready to re-execute the package. This time, in my data file, four records needed to be fixed, and two of those were sent to the error queue. The final package would look something like the one shown in Figure 5-14 when executed. Again, your data may vary widely based on what corporation file you downloaded from the Department of State.

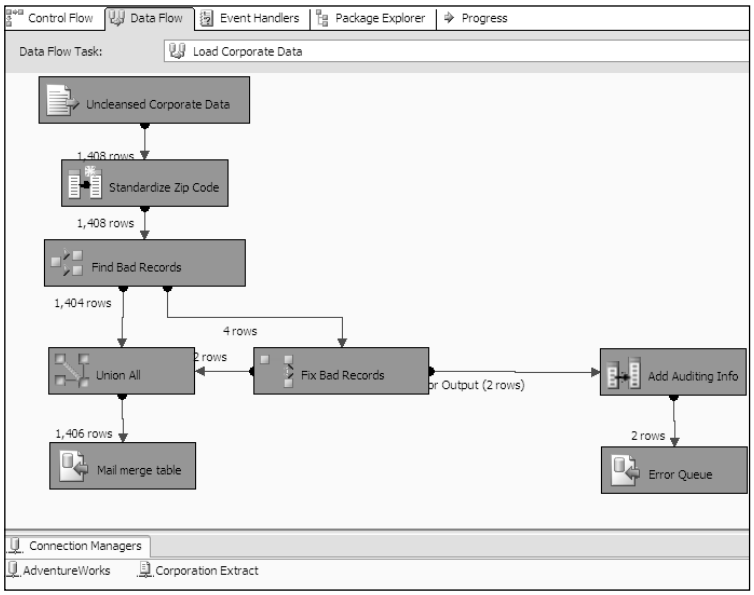

Figure 5-14

Looping and the Dynamic Task

You've come a long way in this chapter to creating a self-healing package, but it's not terribly reusable yet. Your next task in the business requirements is to configure the package so that it reads a directory for any .DAT file and performs the previous tasks to that collection of files. To simulate this example, go ahead and download a few files again from ftp://dossftp.dos.state.fl.us/public/doc/cor/. They can be whatever .DAT file you'd like, but make sure you download at least two more.

Looping

Your first task is to loop through any set of .DAT files in the C:\SSISDemos directory and load them into your database. To meet this business requirement, you'll need to use the Foreach Loop container. Go to the Control Flow tab in the same package that you've been working in, and drag the container onto the design pane. Then, drag the "Load Corporate Data" Data Flow task onto the container. Rename the container "Loop Through Files."

Double-click on the container to configure it. Go to the Collection page and select Foreach File Enumerator from the Enumerator drop-down box. Next, specify that the folder will be C:\SSISDemos and that the files will have the *.DAT extension, as shown in Figure 5-15.

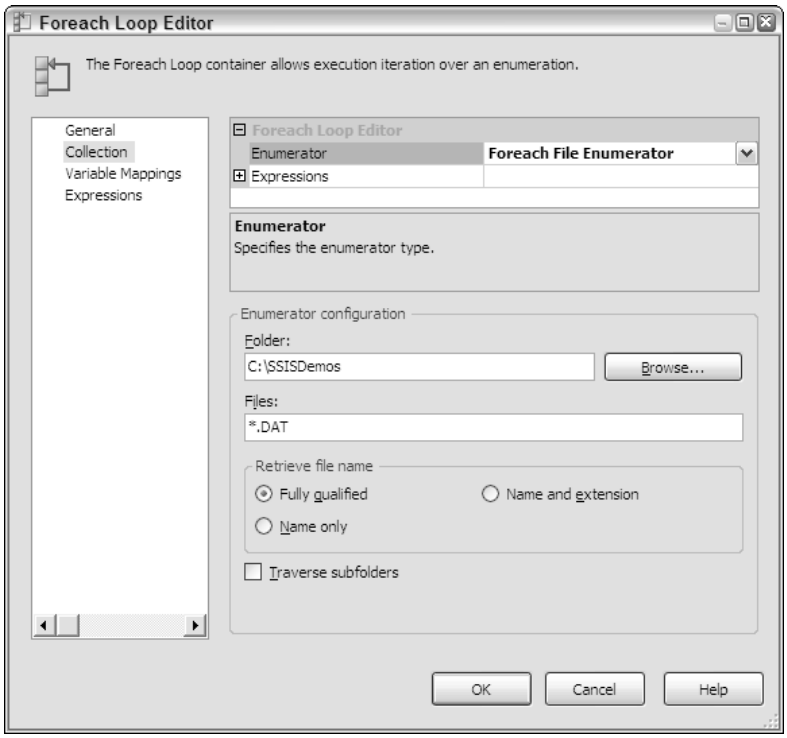

Figure 5-15

You need to now map the variables to the results of the Foreach File Enumeration. Go to the Variable Mappings page inside the Foreach Loop Editor and select <New Variable...> from the Variable column drop-down box. This will open the Add Variable dialog box. For the container, you'll remain at the package level. You could assign the scope of the variable to the container, but you should keep things simple for this example. Name the variable "ExtractFileName" in the Name option and click OK, leaving the rest of the options at their default settings.

You will then see the User::ExtractFileName variable in the Variable column and the number 0 in the Index option. Since the Foreach File Enumerator option has only one column, you'll only see an index of 0 for this column. If you used a different enumerator option, you would have the ability to enter a number for each column that was returned from the enumerator. Click OK to leave the Foreach Loop editor.

Making the Package Dynamic

Now with the loop created, you need to set the file name in the Corporation Extract Connection Manager to be equal to the file name that the enumerator retrieves dynamically. To meet this business requirement, right-click on the Corporation Extract Connection Manager and select Properties (note that you're clicking on Properties, not on Edit as you've done in the past). In the Properties pane for this Connection Manager, click the ellipsis button next to the Expressions option.

By clicking the ellipsis button, you open the Property Expressions Editor. Select ConnectionString from the Property drop-down box, as shown in Figure 5-16. You can either type in **@[User::ExtractFileName]** in the Expression column or click the ellipsis button and then drag and drop the variable into the expression window. By typing **@[User::ExtractFileName]**, you are setting the file name in the Connection Manager to be equal to the ExtractFileName variable that you set in the Foreach Loop earlier. Click OK to exit the Property Expression Editor. You'll now see in the Property window that there is a single expression by clicking the plus sign.

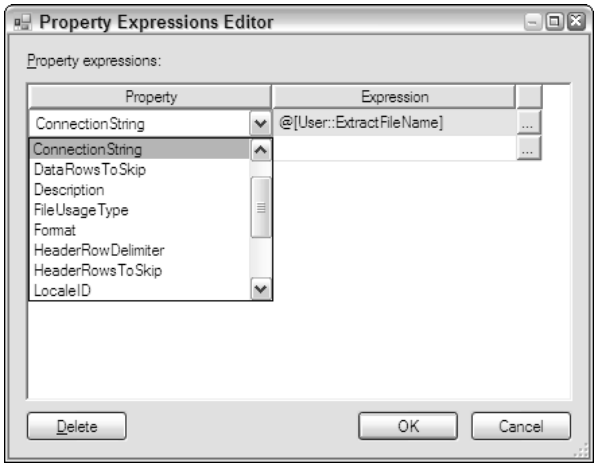

Figure 5-16

As it stands right now, each time the loop finds a .DAT file in the C:\SSISDemos directory, it will set the ExtractFileName variable to that path and file name. Then, the Connection Manager will use that variable as its file name and run the Data Flow task one time for each file it finds. You now have a reusable package that can be run against any file in the format you designated earlier.

The only missing technical solution to complete is the archiving of the files after you load them. Before you begin solving that problem, manually create an archive directory under C:\SSISDemos called C:\SSISDemos\Archive. Right-click in the Connection Manager window and select Create New File Connection. Select Existing Folder for the Usage Type, and point the file to the C:\SSISDemos\Archive directory. Click OK and rename the newly created Connection Manager "Archive File."

Next, drag a File System task into the "Loop Through Files" container and connect it to the "Load Corporate Data" Data Flow task with an On Success constraint (the green arrow should be attached to the File System task). Rename that task "Archive File."

Double-click on the "Archive File" File System task to open the editor (shown in Figure 5-17). Set the Operation drop-down box to Move File. Next, specify that the Destination Connection not be a variable and that it be set to the Archive File Connection Manager that you just created. The SourceConnection drop-down box should be set to the "Corporation Extract" Connection Manager that you created a long time ago. Essentially, what you're configuring is that the file that was pulled earlier from the loop will be moved to whatever directory and file name is in the Archive File Connection Manager.

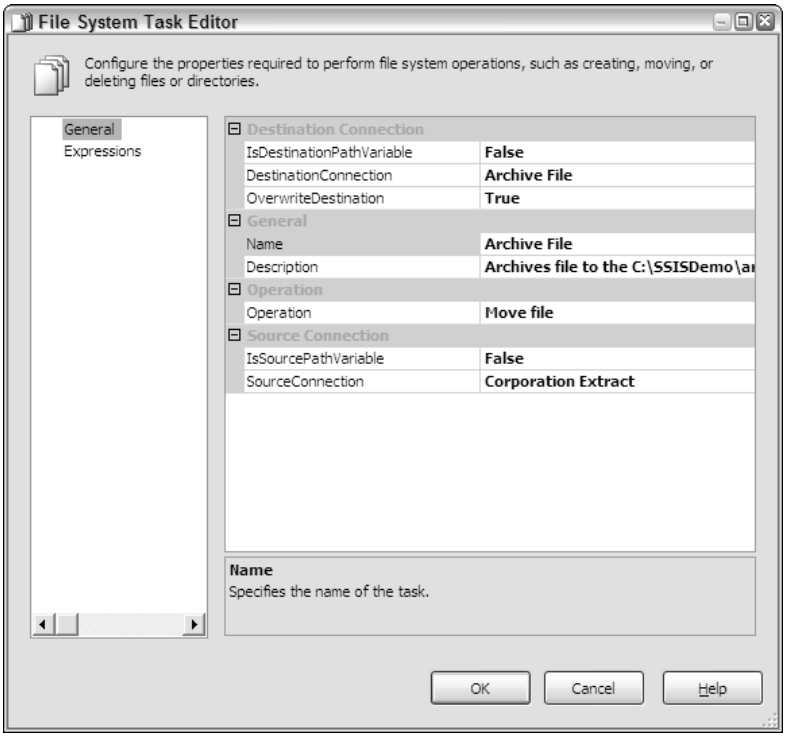

Figure 5-17

Your complete package should now be ready to execute. Go ahead and save the package first before you execute it. If you successfully implemented the solution, your control flow should look something like Figure 5-18 when executed. When you execute the package, you'll see the control flow items flash green once for each .DAT file in the directory.

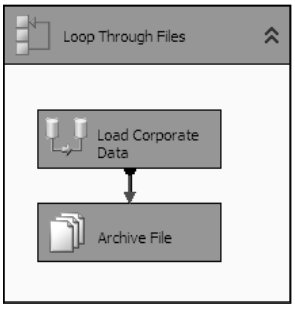

Figure 5-18

Summary

This chapter focused on driving home the basic SSIS transforms, tasks, and containers. You performed a basic ETL procedure and then expanded the ETL to self-heal when bad data arrived from your data supplier. You then set the package to loop through a directory, find each .DAT file, and load it into the database. The finale was archiving the file automatically after it was loaded. With this type of package now complete, you could throw any .DAT file that matched the format you configured and it will load with reasonable certainty. In the upcoming chapter, you'll dive into some more advanced tasks and transforms that use the Advanced Editor.