TRAINING & REFERENCE

murach's ADO.NET 2.0 database programming with VB 2005

(Chapter 3)

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • www.murach.com

Contents

3

How to work with data sources and datasets

In this chapter, you'll learn how to use data sources and datasets to develop database applications. This makes it easier than ever to generate Windows forms that work with the data that's in the data sources. And this is especially useful for developing simple applications or prototyping larger applications.

How to create a data source

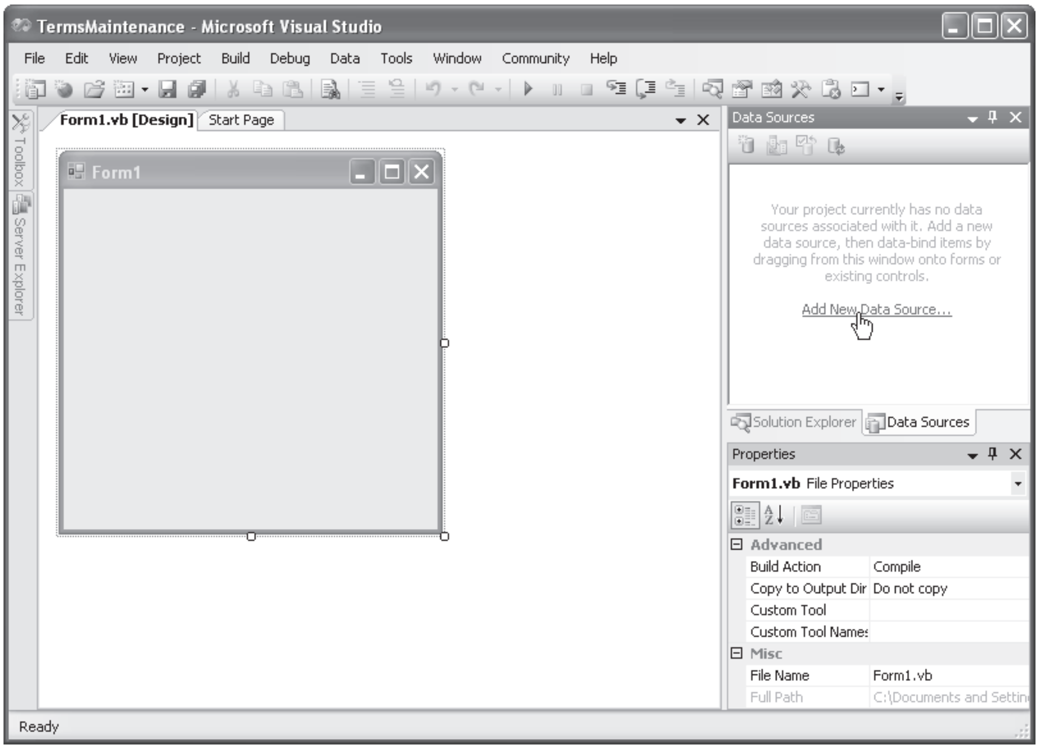
Before you can take advantage of Visual Studio 2005's new features for working with data, you must create a *data source* for the application. As its name implies, a data source specifies the source of the data for an application. Since most applications get their data from a database, the figures that follow show how to create a data source that gets data from a database.

How to use the Data Sources window

The data sources that are available to a project are listed in the Data Sources window as shown in figure 3-1. Here, the second screen shows a data source for the Terms table that's available from the Payables database that's described in figure 1-8 of chapter 1. As you can see, this data source includes three columns from the Terms table named TermsID, Description, and DueDays.

If no data sources are available to a project, the Data Sources window will display an Add New Data Source link as shown in the first screen. Then, you can click on this link to start the Data Source Configuration Wizard described in figures 3-2 through 3-6. This wizard lets you add a new data source to the project. When you're done, you can drag the data source onto a form to create bound controls as described later in this chapter.

An empty Data Sources window

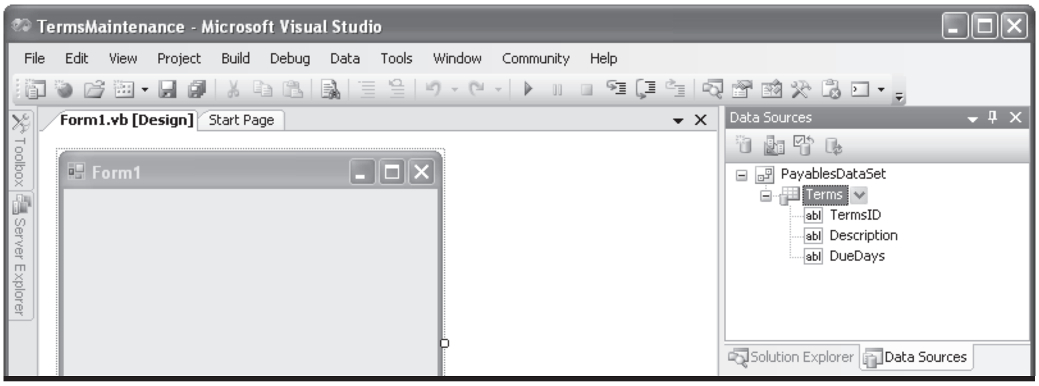


Your project currently has no data sources associated with it. Add a new data source, then data-bind items by dragging from this window onto forms or existing controls.

Form1.vb File Properties

Advanced
Build Action Compile
Copy to Output Dir Do not copy
Custom Tool
Custom Tool Names
Misc
File Name Form1.vb
Full Path C:\Documents and Settin

A Data Sources window after a data source has been added



Description

- A *data source* shows all the tables and columns in the dataset that are available to your application.
- You can display the Data Sources window by clicking on the Data Sources tab that's usually grouped with the Solution Explorer at the right edge of the Visual Studio window or by selecting the Show Data Sources command from the Data menu.
- To create a data source, you can click the Add New Data Source link. Then, you can drag the data source to a form to create controls that are bound to the data source.

Figure 3-1 How to use the Data Sources window

How to start the Data Source Configuration Wizard

If your project doesn't already contain a data source, you can start the Data Source Configuration Wizard by clicking the Add New Data Source link that's shown in the previous figure. However, if your project already contains a data source, this link won't be available. In that case, you can start the Data Source Configuration Wizard by using one of the techniques listed in figure 3-2.

The last technique is to add a SQL Server or Access database file to the project. You may want to do that if the application is for a single user. That way, the database can easily be distributed with the application.

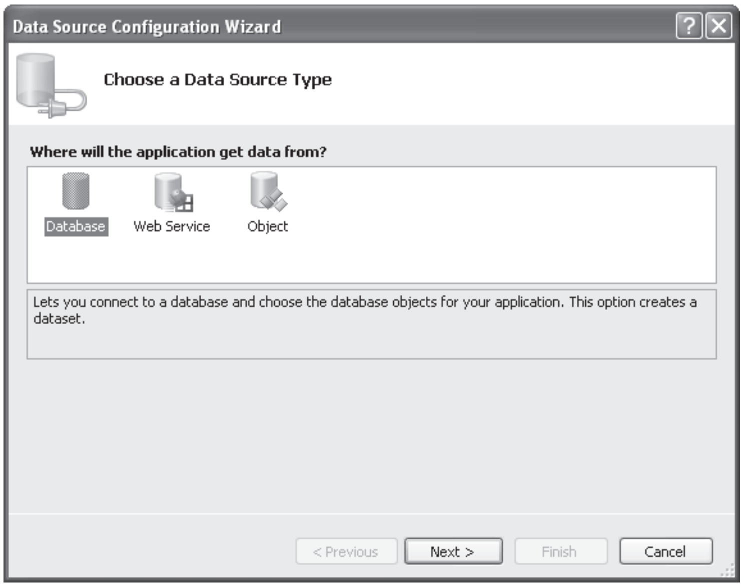
If you add a database file to your project, you should know that by default, that file is copied to the output directory for the project every time the project is built. (The output directory is the directory where the executable file for the application is stored.) Then, when you run the application, the application works with the copy of the database file in the output directory. That means that any changes that you make to the database aren't applied to the database file in the project folder. And each time you rebuild the application, the database in the output directory is overwritten by the unchanged database in the project directory so you're back to the original version of the database.

If you want to change the way this works, you can select the database file in the Solution Explorer and change its "Copy to Output Directory" property from "Copy always" to "Copy if newer." Then, the database file in the output directory won't be overwritten unless the database file in the project directory contains more current data.

How to choose a data source type

Figure 3-2 also shows the first step of the Data Source Configuration Wizard. This step lets you specify the source from which your application will get its data. To work with data from a database as described in this chapter, you select the Database option. However, you can also select the Web Service option to work with data from a web service that's available from the Internet or from an intranet. Or, you can select the Object option to work with data that's stored in a business object. This option lets you take advantage of the objects that are available from the middle layer of an application as described in chapter 9.

The first step of the Data Source Configuration Wizard


Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
Database
Web Service
Object
< Previous
Next >
Finish
Cancel

Lets you connect to a database and choose the database objects for your application. This option creates a dataset.

How to start the Data Source Configuration Wizard

- Click on the Add New Data Source link that's available from the Data Sources window when a project doesn't contain any data sources.
- Click on the Add New Data Source button at the top of the Data Sources window.
- Select the Add New Data Source command from Visual Studio's Data menu.
- Add a SQL Server (.mdf) or Access (.mdb) data file to the project using the Project➔Add➔Existing Item command. Then, the wizard will skip to the step shown in figure 3-6 that lets you choose the database objects you want to include.

How to choose a data source type

- To get your data from a database, select the Database option. This option lets you create applications like the ones described in this chapter.
- To get your data from a web service, select the Web Service option. This option lets you browse the web to select a web service that will supply data to your application.
- To get your data from a business object, select the Object option. This option lets you create applications like the ones described in chapter 9.

Note

- Before you start this procedure, you need to install your database server software on your own PC or on a network server, and you need to attach your database to it. For more information, please refer to appendix A.

Figure 3-2 How to start the Data Source Configuration Wizard and choose a data source type

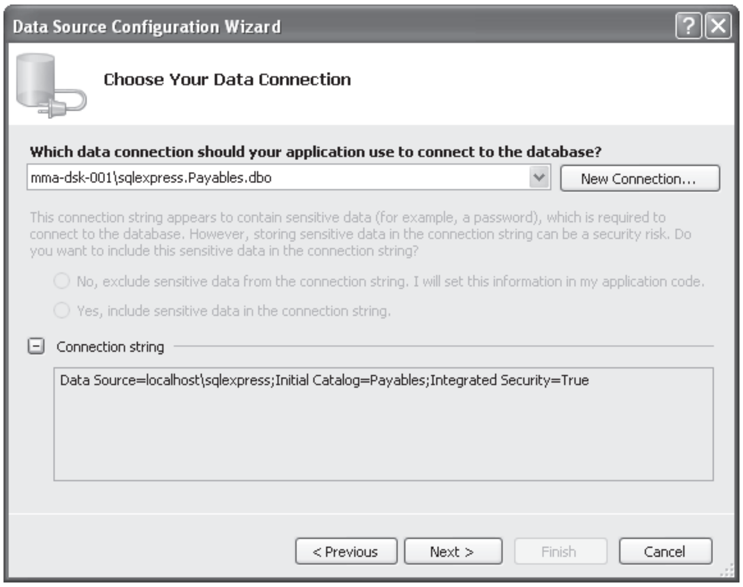
How to choose the connection for a data source

The second step of the Data Source Configuration Wizard, shown in figure 3-3, lets you choose the data connection you want to use to connect to the database. If you've previously defined a data connection, you can choose that connection from the drop-down list. To be sure you use the right connection, you can click the button with the plus sign on it to display the connection string.

If the connection you want to use hasn't already been defined, you can click the New Connection button. Then, you can use the dialog boxes shown in the next figure to create the connection.

Before I go on, you should know that once you create a connection using the Data Source Configuration Wizard, it's available to any other project you create. To see a list of the existing connections, you can open the Server Explorer window (View➔Server Explorer) and then expand the Data Connections node. You can also use the Server Explorer to create data connections without creating a data source. See chapter 19 for more information.

The second step of the Data Source Configuration Wizard



Which data connection should your application use to connect to the database?

mma-dsk-001\sqlexpress.Payables.dbo

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

No, exclude sensitive data from the connection string. I will set this information in my application code.

Yes, include sensitive data in the connection string.

```
Data Source=localhost\sqlexpress;Initial Catalog=Payables;Integrated Security=True
```

Description

- When you click the Next button in the first step of the Data Source Configuration Wizard, the Choose Your Data Connection step shown above is displayed.
- If you've already established a connection to the database you want to use, you can choose that connection. Otherwise, you can click the New Connection button to display the Add Connection dialog box shown in the next figure.
- To see the connection string for an existing connection, click the button with the plus sign on it.

Figure 3-3 How to choose the connection for a data source

How to create a connection to a database

If you click the New Connection button from the second step of the Data Source Configuration Wizard, the Add Connection dialog box shown in figure 3-4 is displayed. This dialog box helps you identify the database that you want to access and provides the information you need to access it. That includes specifying the name of the server that contains the database, entering the information that's required to log on to the server, and specifying the name of the database. How you do that, though, varies depending on whether you're running SQL Server Express on your own PC or whether you're using a database server that's running on a network server.

If you're using SQL Server Express on your own PC and you've downloaded and installed it as described in appendix A, you can use the localhost keyword to specify that the database server is running on the same PC as the application. This keyword should be followed by a backslash and the name of the database server: SqlExpress.

For the logon information, you should select the Use Windows Authentication option. Then, SQL Server Express will use the login name and password that you use to log in to Windows as the name and password for the database server too. As a result, you won't need to provide a separate user name and password in this dialog box.

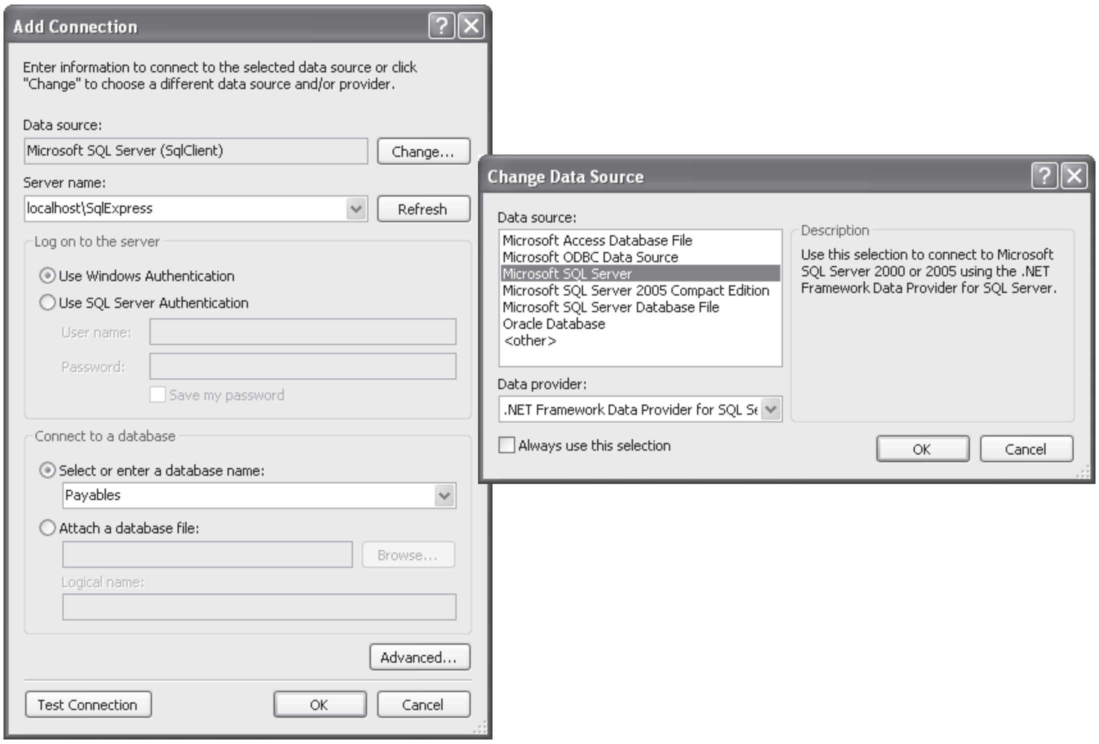
Last, you enter or select the name of the database that you want to connect to. In this figure, for example, the connection is for the Payables database that's used throughout book. When you're done supplying the information for the connection, you can click the Test Connection button to be sure that the connection works.

In contrast, if you need to connect to a database that's running on a database server that's available through a network, you need to get the connection information from the network or database administrator. This information will include the name of the database server, logon information, and the name of the database.

The first time you create a data connection, Visual Studio displays the Change Data Source dialog box shown in this figure before it displays the Add Connection dialog box. The Change Data Source dialog box lets you choose the data source and data provider you want to use for the data connection. By default, the data source is Microsoft SQL Server and the data provider is .NET Framework Data Provider for SQL Server. This works for SQL Server 7, 2000, and 2005 databases including SQL Server Express databases. If that's what you want, you can just click the OK button. Then, Visual Studio will assume that you want to use those values for any data connections you create in the future.

If you ever want to change the data source, you can click the Change button in the Add Connection dialog box to display the Change Data Source dialog box. Then, you can select the data source and data provider you want to use. If you want to access an Oracle database, for example, you can select the Oracle Database item in the Data Source list. Then, you can choose the data provider for Oracle or the data provider for OLE DB from the Data Provider drop-down list.

The Add Connection and Change Data Source dialog boxes



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: localhost\SqlExpress

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:] Payables

Attach a database file:

Logical name:

Change Data Source

Data source:

- Microsoft Access Database File
- Microsoft ODBC Data Source
- [✓ Microsoft SQL Server]
- Microsoft SQL Server 2005 Compact Edition
- Microsoft SQL Server Database File
- Oracle Database
- <other>

Description: Use this selection to connect to Microsoft SQL Server 2000 or 2005 using the .NET Framework Data Provider for SQL Server.

Data provider: .NET Framework Data Provider for SQL Se

Always use this selection

Description

- The first time you create a connection, the Change Data Source dialog box is displayed. You can use this dialog box to choose the data source and data provider you want to use by default for the connections you create. If you ever want to change the data source for a connection, you can click the Change button in the Add Connection dialog box to redisplay the Change Data Source dialog box.
- To create a connection, specify the name of the server that contains the database, enter the information that's required to log on to the server, and specify the name of the database you want to connect to.
- To be sure that the connection is configured properly, you can click the Test Connection button in the Add Connection dialog box.

Express Edition differences

- The Change Data Source dialog box provides only two options: Microsoft Access Database File and Microsoft SQL Server Database File.
- The Add Connection dialog box is simpler, and it includes a Database File Name text box that you use to specify the database. To do that, you click the Browse button to the right of the text box and use the resulting dialog box to point to the data file for the database.

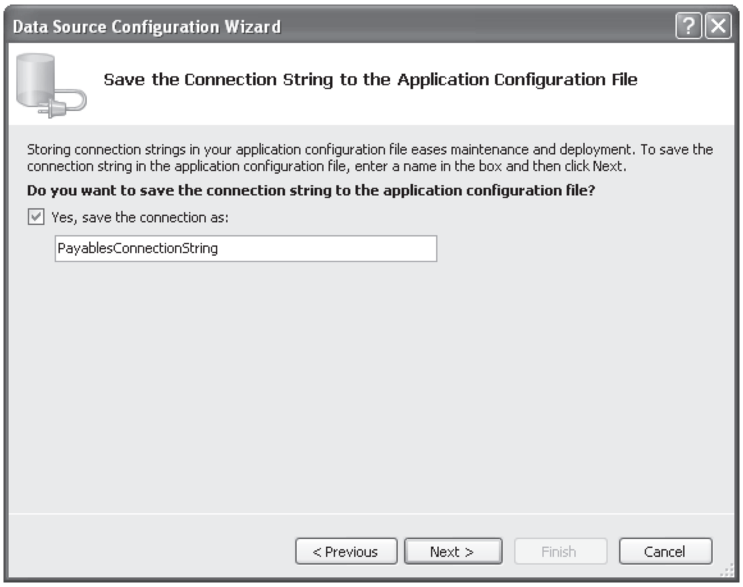
Figure 3-4 How to create a connection to a database

How to save a connection string in the app.config file

After you select or create a data connection, the third step of the Data Source Configuration Wizard is displayed. This step, shown in figure 3-5, asks whether you want to save the connection string in the application configuration file (app.config). In most cases, that's what you'll want to do. Then, any table adapter that uses the connection can refer to the connection string by name. That way, if the connection information changes, you only need to change it in the app.config file. Otherwise, the connection string is stored in each table adapter that uses the connection, and you'll have to change each table adapter if the connection information changes.

This figure also shows how the connection string is stored in the app.config file. Although this file contains XML data, you should be able to understand it even if you don't know XML. Here, for example, you can see that the connectionStrings element contains an add element that contains three attributes. The first attribute, name, specifies the name of the connection string, in this case, PayablesConnectionString. The second attribute, connectionString, contains the actual connection string. And the third attribute, providerName, identifies the data provider, in this case, SqlClient.

The third step of the Data Source Configuration Wizard


Data Source Configuration Wizard
Save the Connection String to the Application Configuration File

Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.

Do you want to save the connection string to the application configuration file?

☑ Yes, save the connection as:

PayablesConnectionString

< Previous
Next >
Finish
Cancel

The information that's stored in the app.config file

```
<connectionStrings>
    <add name="TermsMaintenance.My.MySettings.PayablesConnectionString"
        connectionString="Data Source=localhost\sqlexpress;
                          Initial Catalog=Payables;
                          Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Description

- By default, the connection string is saved in the application configuration file (app.config). If that's not what you want, you can remove the check mark from the Yes option in the third step of the Data Source Configuration Wizard shown above.
- If you don't save the connection string in the app.config file, the string is specified for the connection of each table adapter you create from the data source. Because of that, we recommend you always save the connection string in the app.config file. Then, only the name of the connection string is stored in the connection for each table adapter.
- You can also enter the name you want to use for the connection string in this dialog box. By default, the connection string is given a name that consists of the database name appended with "ConnectionString".

Figure 3-5 How to save a connection string in the app.config file

How to choose database objects for a data source

Figure 3-6 shows how you can use the last step of the Data Source Configuration Wizard to choose the database objects for a data source. This step lets you choose any tables, views, stored procedures, or functions that are available from the database. In some cases, you can just select the table you need from the list of tables that are available from the database. Then, all of the columns in the table are included in the dataset. In this figure, for example, the Terms table is selected.

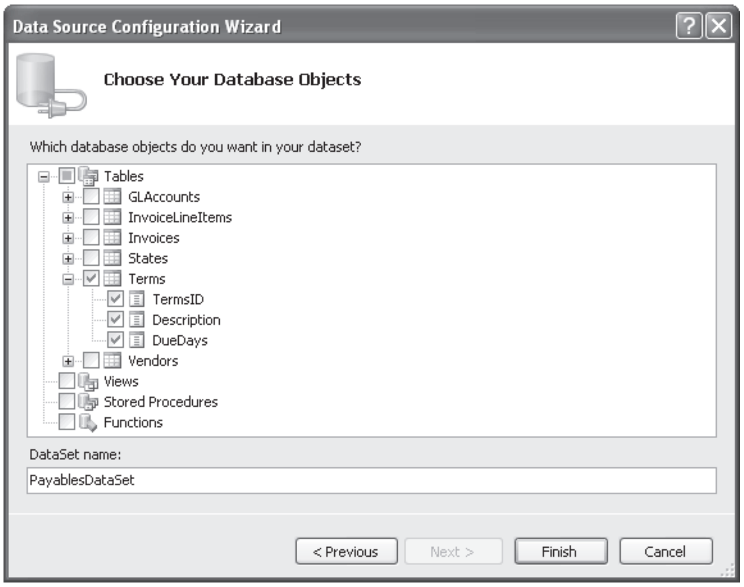
If you want to include selected columns from a table, you can expand the node for the table and select just the columns you want. Later in this chapter, for example, you'll see a Vendor Maintenance application that uses selected columns from the Vendors table. Note that if an application will allow rows to be added to a table, you can omit a column only if it can have a null value or if it's defined with a default value. Otherwise, you have to provide a value for it.

If you include a column with a default value in a dataset, you need to realize that this value isn't assigned to the column in the dataset, even though the dataset enforces the constraints for that column. Instead, the column will be defined with a default value of null, even though null values aren't allowed in columns with default values. As a result, an exception will be thrown whenever a new row is added to the dataset and a value other than null isn't provided for that column.

This means that either the user or the application must provide an acceptable value for the column. One way to do that is to provide a way for the user to enter a value for the column. Another way is to use the Dataset Designer to set the DefaultValue property for this column as described in this figure. You'll learn more about working with the Dataset Designer later in this chapter.

In a larger project, you might want to include several tables in the dataset. Then, the dataset will maintain the relationships between those tables whenever that's appropriate. Or, you might want to use views, stored procedures, or functions to work with the data in the database. If you have experience working with these database objects, you shouldn't have any trouble understanding how this works. Otherwise, you can refer to *Murach's SQL Server 2005 for Developers* for more information.

The last step of the Data Source Configuration Wizard



Data Source Configuration Wizard

Choose Your Database Objects

Which database objects do you want in your dataset?

- [■] Tables
  - [ ] GLAccounts
  - [ ] InvoiceLineItems
  - [ ] Invoices
  - [ ] States
  - [✓] Terms
    - [✓] TermsID
    - [✓] Description
    - [✓] DueDays
  - [ ] Vendors
- [ ] Views
- [ ] Stored Procedures
- [ ] Functions

DataSet name: PayablesDataSet

Description

- In the last step of the Data Source Configuration Wizard, you can choose the database objects that you want to include in the dataset for your project.
- In this step, you can choose from any tables, views, stored procedures, or functions that are available from the database. In addition, you can expand the node for any table, view, stored procedure, or function and choose just the columns you want to include in the data source.
- You can also enter the name you want to use for the dataset in this dialog box. By default, the name is the name of the database appended with “DataSet”.

How to work with columns that have default values

- If a column in a database has a default value, that value isn’t included in the column definition in the dataset. Because of that, you may want to omit columns with default values from the dataset unless they’re needed by the application. Then, when a row is added to the table, the default value is taken from the database.
- If you include a column that’s defined with a default value, you must provide a value for that column whenever a row is added to the dataset. One way to do that is to let the user enter a value. Another way is to display the Dataset Designer as described in figure 3-16, click on the column, and use the Properties window to set the DefaultValue property.

Figure 3-6 How to choose database objects for a data source

The schema file created by the Data Source Configuration Wizard

After you complete the Data Source Configuration Wizard, the new data source is displayed in the Data Sources window you saw in figure 3-1. In addition to this data source, Visual Studio generates a file that contains the *schema* for the dataset class. This file defines the structure of the dataset, including the tables it contains, the columns that are included in each table, the data types of each column, and the constraints that are defined for each table.

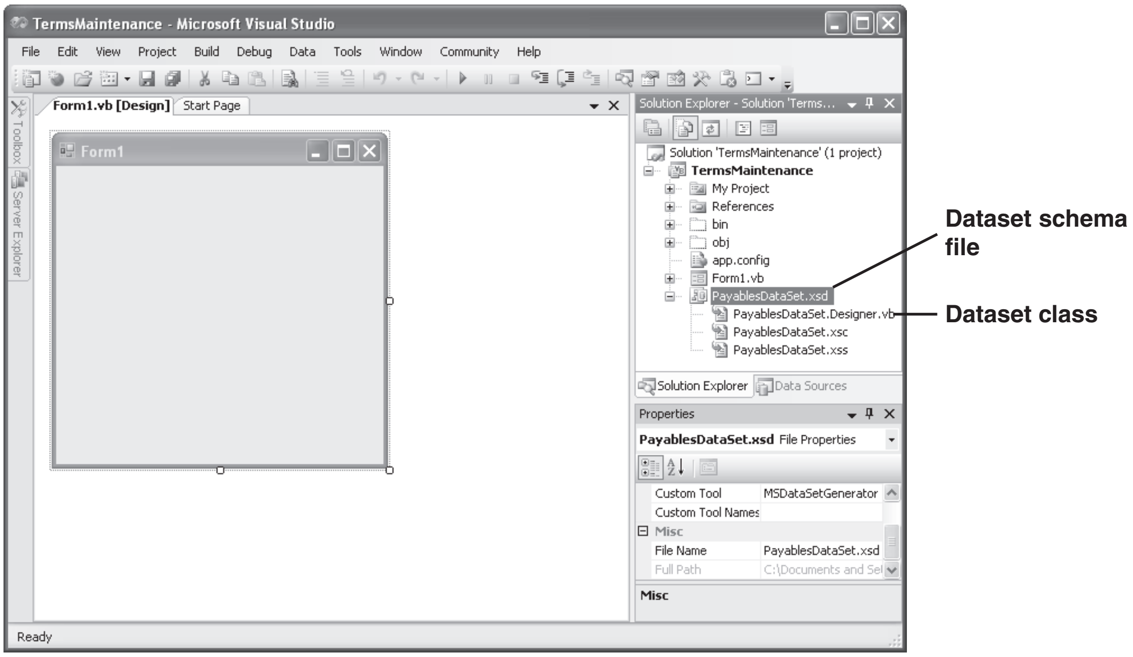
This schema file is listed in the Solution Explorer window and is given the name you specified for the dataset in the last step of the Data Source Configuration Wizard with a file extension of *xsd*. In figure 3-7, for example, you can see the schema file named PayablesDataSet.xsd. As you'll learn later in this chapter, you can view a graphic representation of this schema by double-clicking on this file.

Beneath the schema file, the Solution Explorer displays the file that contains the generated code for the dataset class. In this figure, this code is stored in the PayablesDataSet.Designer.vb file. When you create bound controls from the data source as shown in this chapter, the code in this class is used to define the dataset object that the controls are bound to. Although you may want to view this code to see how it works, you shouldn't change it. If you do, the dataset may not work correctly.

By the way, you should know that a dataset that's created from a dataset class like the one shown here is called a *typed dataset*. The code in the dataset class makes it possible for you to refer to the tables, rows, and columns in the typed dataset using the simplified syntax you'll see in this chapter and the next chapter.

In contrast, you'll learn how to create and work with an *untyped dataset* in chapter 16. As you'll see, you create this type of dataset using code.

A project with a dataset defined by a data source



PayablesDataSet.xsd File Properties

Custom Tool	MSDataSetGenerator
Custom Tool Names	
Misc	
File Name	PayablesDataSet.xsd
Full Path	C:\Documents and Set

Description

- After you create a data source, it's displayed in the Data Sources window. Then, you can use it to create bound controls as shown in this chapter.
- Visual Studio also generates a file that contains the *schema* for the dataset defined by the data source. This file appears in the Solution Explorer and has a file extension of *xsd*. It defines the structure of the dataset, including the tables it contains, the columns in each table, the data types of each column, and the constraints for each table.
- Subordinate to the schema file is a file that contains the generated code for the dataset class. Visual Studio uses this class to create a dataset object when you add the data source to a form.

Note

- To see the files that are subordinate to the schema file, click the Show All Files button at the top of the Solution Explorer. Then, expand the node for the schema file.

Figure 3-7 The schema file created by the Data Source Configuration Wizard

How to use a data source

Once you've created a data source, you can bind controls to the data source and then use the bound controls to add, update, and delete the data in the data source. In this chapter, for example, you'll learn how to bind the DataGridView control and TextBox controls to a data source. The DataGridView control is new to .NET 2.0 and has been designed specifically for working with data sources. Although it is similar to the DataGrid control that was available with previous versions of .NET, it also provides some significant enhancements.

How to generate a DataGridView control from a data source

By default, if you drag a table from the Data Sources window onto a form, Visual Studio adds a DataGridView control to the form and *binds* it to the table as shown in figure 3-8. This creates a DataGridView control that lets you browse all the rows in the table as well as add, update, and delete rows in the table. To provide this functionality, Visual Studio adds a toolbar to the top of the form that provides navigation buttons along with Add, Delete, and Save buttons.

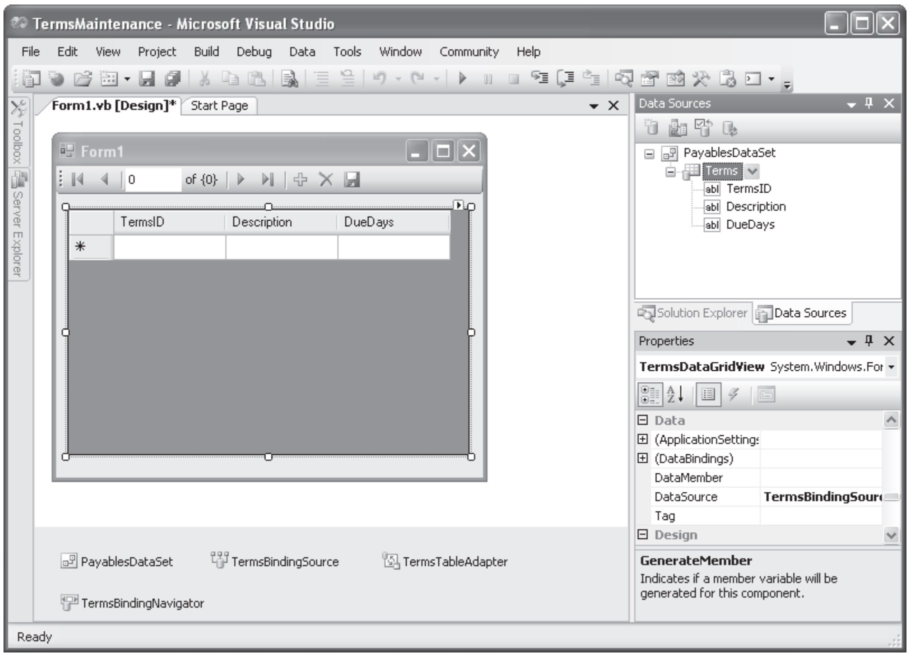
To bind a DataGridView control to a table, Visual Studio uses a technique called *complex data binding*. This just means that the *bound control* is bound to more than one data element. The DataGridView control in this figure, for example, is bound to all the rows and columns in the Terms table.

When you generate a DataGridView control from a data source, Visual Studio also adds four additional objects to the Component Designer tray at the bottom of the Form Designer. First, the DataSet object defines the dataset that contains the Terms table. Second, the TableAdapter object provides commands that can be used to work with the Terms table in the database. Third, the BindingSource object specifies the data source (the Terms table) that the controls are bound to, and it provides functionality for working with the data source. Finally, the BindingNavigator defines the toolbar that contains the controls for working with the data source.

Before I go on, I want to point out that the TableAdapter object is similar to the DataAdapter object you learned about in the previous chapter. However, it has a built-in connection and, as you'll see in chapter 4, it can contain more than one query. Also, a TableAdapter object can only be created by using Visual Studio design tools like the Data Source Configuration Wizard.

I also want to mention that, in general, you shouldn't have any trouble figuring out how to use the binding navigator toolbar. However, you may want to know that if you click the Add button to add a new row and then decide you don't want to do that, you can click the Delete button to delete the new row. However, there's no way to cancel out of an edit operation. Because of that, you may want to add a button to the toolbar that provides this function. You'll learn how to do that in the next chapter.

A form after the Terms table has been dragged onto it



The controls and objects that are created when you drag a data source to a form

Control/object	Description
DataGridView control	Displays the data from the data source in a grid.
BindingNavigator control	Defines the toolbar that can be used to navigate, add, update, and delete rows in the DataGridView control.
BindingSource object	Identifies the data source that the controls on the form are bound to and provides functionality for working with the data source.
DataSet object	Provides access to all of the tables, views, stored procedures, and functions that are available to the project.
TableAdapter object	Provides the commands that are used to read and write data to and from the specified table in the database.

Description

- To *bind* a DataGridView control to a table in a dataset, just drag the table from the Data Sources window onto the form. Then, Visual Studio automatically adds a DataGridView control to the form along with the other controls and objects it needs to work properly. Because the DataGridView control is bound to the table, it can be referred to as a *bound control*.
- To bind a DataGridView control to a data table, Visual Studio uses a technique called *complex data binding*. This means that the control is bound to more than one data element, in this case, all the rows and columns in the table.

Figure 3-8 How to generate a DataGridView control from a data source

A Terms Maintenance application that uses a DataGridView control

At this point, the DataGridView control and binding navigator toolbar provide all the functionality needed for an application that can be used to maintain the data in the Terms table. Figure 3-9 shows how this application appears to the user at runtime. Note that the appearance and operation of the DataGridView control haven't been changed from their defaults. In most cases, however, you'll want to at least make some minor changes in the appearance of this control. You'll learn how to do that in the next chapter when I present some additional skills for working with the DataGridView control.

This figure also presents the code that Visual Studio generates when you create this application, which includes everything needed to make it work. As a result, you can create an application like this one without having to write a single line of code. If you've ever had to manually write an application that provides similar functionality, you can appreciate how much work this saves you.

When this application starts, the first event handler in this figure is executed. This event handler uses the Fill method of the TableAdapter object to load data into the DataSet object. In this example, the data in the Terms table of the Payables database is loaded into the Terms table of the dataset. Then, because the DataGridView control is bound to this table, the data is displayed in this control and the user can use it to modify the data in the table by adding, updating, or deleting rows.

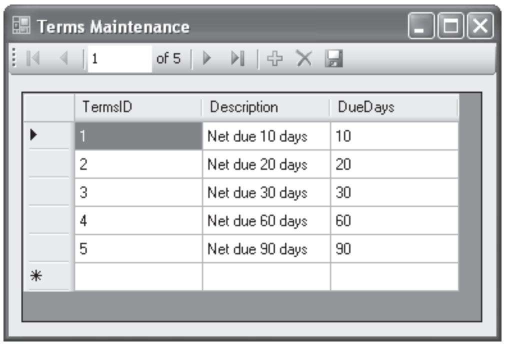
When the user changes the data in the DataGridView control, those changes are saved to the dataset. However, the changes aren't saved to the database until the user clicks the Save button in the toolbar. Then, the second event handler in this figure is executed. This event handler starts by calling the Validate method of the form, which causes the Validating and Validated events of the control that's losing focus to be fired. Although you probably won't use the Validated event, you may use the Validating event to validate a row that's being added or modified. However, I've found that this event doesn't work well with the binding navigator toolbar, so you won't see it used in this book.

Next, the EndEdit method of the BindingSource object applies any pending changes to the dataset. That's necessary because when you add or update a row, the new or modified row isn't saved until you move to another row.

Finally, the Update method of the TableAdapter object saves the Terms table in the DataSet object to the Payables database. When this method is called, it checks each row in the table to determine if it's a new row, a modified row, or a row that should be deleted. Then, it causes the appropriate SQL Insert, Update, and Delete statements to be executed for these rows. As a result, the Update method works efficiently since it only updates the rows that need to be updated.

Now that you understand this code, you should notice that it doesn't provide for any exceptions that may occur during this processing. Because of that, you need to add the appropriate exception handling code for any production applications that you develop so that they won't crash. You'll learn how to do that later in this chapter.

The user interface for the Terms Maintenance application



	TermsID	Description	DueDays
▶	1	Net due 10 days	10
	2	Net due 20 days	20
	3	Net due 30 days	30
	4	Net due 60 days	60
	5	Net due 90 days	90
*			

The code that's generated by Visual Studio

```
Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
    'TODO: This line of code loads data into the 'PayablesDataSet.Terms'
    'table. You can move, or remove it, as needed.
    Me.TermsTableAdapter.Fill(Me.PayablesDataSet.Terms)
End Sub

Private Sub TermsBindingNavigatorSaveItem_Click( _
        ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles TermsBindingNavigatorSaveItem.Click
    Me.Validate()
    Me.TermsBindingSource.EndEdit()
    Me.TermsTableAdapter.Update(Me.PayablesDataSet.Terms)
End Sub
```

The syntax of the Fill method

```
TableAdapter.Fill(DataSet.TableName)
```

The syntax of the Update method

```
TableAdapter.Update(DataSet.TableName)
```

Description

- Visual Studio automatically generates the code shown above and places it in the source code file when you drag a data source onto a form. If necessary, you can edit this code.
- The generated code uses the Fill and Update methods of the TableAdapter object that's generated for the table to read data from and write data to the database. It also uses the EndEdit method of the BindingSource object to save any changes that have been made to the current row to the dataset.
- The Validate method causes the Validating and Validated events of the control that is losing the focus to be fired. You can use the Validating event to perform any required data validation for the form.
- Users of a DataGridView control can sort the rows by clicking on a column heading and can size columns by dragging the column separators to the left or right.

Figure 3-9 A Terms Maintenance application that uses a DataGridView control

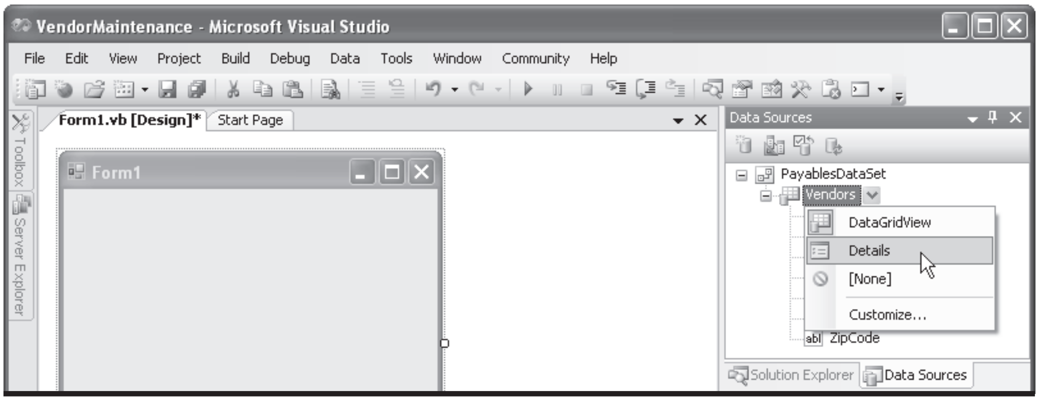
How to change the controls associated with a data source

If the DataGridView control isn't appropriate for your application, you can bind the columns of a data source to individual controls as shown in figure 3-10. Here, the data source consists of several columns from the Vendors table.

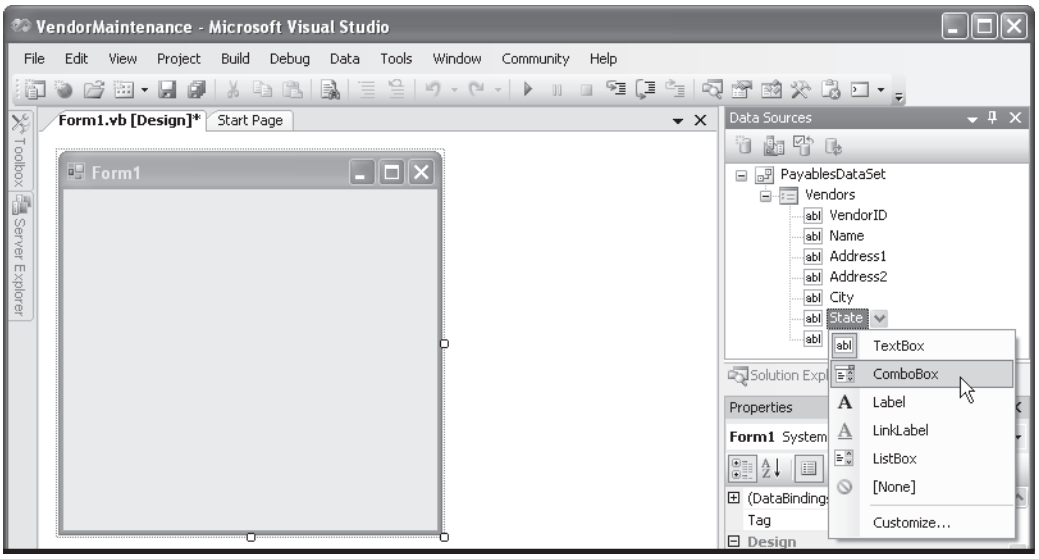
To associate the columns in a table with individual controls, you select the Details option from the drop-down list that's available when you select the table in the Data Sources window. This is illustrated in the first screen in this figure. Then, if you drag that table from the Data Sources window onto a form, Visual Studio generates a label and a bound control for each column in the table.

For most string and numeric columns, Visual Studio generates a TextBox control. That's the case for the Vendors table, as you'll see in the next figure. If you want to change the type of control that's associated with a column, though, you can select the column in the Data Sources window and then use the drop-down list that's displayed to select a different type of control. You can see the list of controls that are available in the second screen in this figure.

How to change the default control for a data table


VendorMaintenance - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Form1.vb [Design]* Start Page
Form1
Data Sources
PayablesDataSet
Vendors
DataGridView
Details
[None]
Customize...
ZipCode
Solution Explorer Data Sources

How to change the default control for a column in a data table



Description

- By default, a data table is associated with a DataGridView control. To change this default so that each column in the table is displayed in a separate control, select the Details option from the drop-down list for the table.
- By default, most string and numeric columns within a data table are associated with the TextBox control. To change this default, select the type of control you want to use from the drop-down list for the column.

Figure 3-10 How to change the controls associated with a data source

How to generate detail controls from a data source

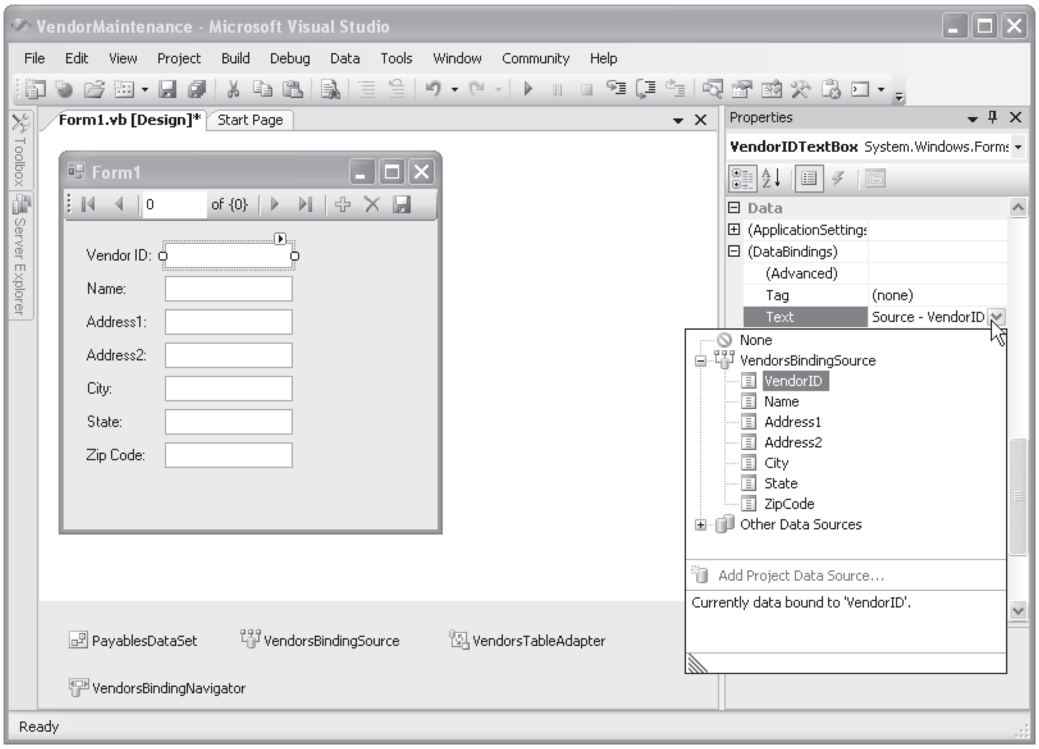
If you change the control type that's associated with a table from DataGridView to Details and then drag that table from the Data Sources window onto a form, Visual Studio will add the appropriate controls to the form as shown in figure 3-11. In addition, it will bind those controls to the appropriate columns in the table, and it will add a Label control for each column to identify it. In this figure, for example, you can see that Visual Studio added a TextBox control and a Label control for each of the seven columns in the Vendors table. In addition, it added DataSet, BindingSource, TableAdapter, and BindingNavigator objects, plus a binding navigator toolbar, just as it does when you generate a DataGridView control.

Notice that when you use text boxes to work with the data in a table, only one row of the table is displayed at a time. Then, Visual Studio uses *simple data binding* to bind each text box to a single column value. To do that, it sets the Text property in the DataBindings collection to the name of the data column that the control is bound to. In this figure, for example, you can see the drop-down list for the Text property of the DataBindings collection. It shows that the Vendor ID text box is bound to the VendorID column of the VendorsBindingSource object.

Once the labels and text boxes are displayed on the form, you can use standard skills for editing the labels and text boxes to get the form to work correctly. For example, if you want to change the text that's displayed in a label, you can select the label and edit its Text property. If you don't want the user to be able to enter data for a particular column, you can change the ReadOnly property of the text box to True. Or, if you don't want to display a column, you can delete the label and text box for that column.

Alternatively, instead of dragging the entire table onto the form, you can drag just the columns you want. In addition, if you want to create a read-only form, you can edit the BindingNavigator toolbar to remove its Add, Delete, and Save buttons. You'll learn how to do that in the next chapter.

A form after the Vendors table has been dragged onto it



Description

- When you drag a table whose columns are associated with individual controls to a form, Visual Studio automatically adds the controls along with labels that identify the columns. It also adds a binding navigator toolbar and the objects for working with the bound data just as it does for a DataGridView control.
- To display the value of a column in a text box, Visual Studio sets the Text property in the DataBindings collection to the name of the data column. This is known as *simple data binding* because the control is bound to a single column value. To change the binding, you can use the drop-down list for the Text property as shown above.

Note

- When you drag individual controls to a form, don't drop them at the top of the form. If you do, the toolbar will overlay the first label and text box and make them difficult to move.

Figure 3-11 How to generate detail controls from a data source

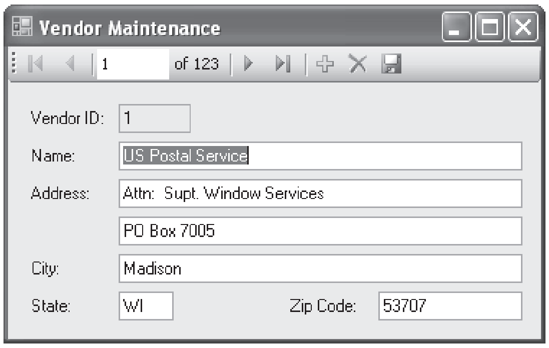
A Vendor Maintenance application that uses TextBox controls

Figure 3-12 shows the user interface for a Vendor Maintenance application that uses the Label and TextBox controls shown in the previous figure. However, I rearranged and made several changes to those controls.

First, I changed the label for the Address1 text box to "Address:" and I removed the label from the Address2 text box. Next, I changed the sizes of the text boxes so that they are appropriate for the data they will be used to display. Finally, I changed the ReadOnly property of the VendorID text box to True so the user can't enter data into this control, and I change the TabStop property of this text box to False so that it isn't included in the tab sequence.

This figure also shows the code for the Vendor Maintenance application. If you compare this code with the code for the Terms Maintenance application in figure 3-9, you'll see that it's almost identical. The only difference is that the code for the Vendor Maintenance application works with the Vendors table, table adapter, and binding source instead of the Terms table, table adapter, and binding source.

The user interface for the Vendor Maintenance application



Vendor ID: 1
Name: US Postal Service
Address: Attn: Supt. Window Services
PO Box 7005
City: Madison
State: WI Zip Code: 53707

The code for the application

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        'TODO: This line of code loads data into the
        'PayablesDataSet.Vendors' table.
        'You can move, or remove it, as needed.
        Me.VendorsTableAdapter.Fill(Me.PayablesDataSet.Vendors)
    End Sub

    Private Sub VendorsBindingNavigatorSaveItem_Click( _
            ByVal sender As System.Object, ByVal e As System.EventArgs) _
            Handles VendorsBindingNavigatorSaveItem.Click
        Me.Validate()
        Me.VendorsBindingSource.EndEdit()
        Me.VendorsTableAdapter.Update(Me.PayablesDataSet.Vendors)
    End Sub

End Class
```

Figure 3-12 A Vendor Maintenance application that uses TextBox controls

How to handle data errors

When you develop an application that uses a data source, you'll want to provide code that handles any data errors that might occur. In general, those errors fall into three categories: data provider errors, ADO.NET errors, and errors that the DataGridView control detects. You'll learn how to provide for these errors in the topics that follow.

How to handle data provider errors

When you access a database, there is always the possibility that an unrecoverable error might occur. For example, the database server might be shut down when you try to access it, or the network connection to the database server might be broken. Either way, your applications should usually anticipate such problems by catching any database exceptions that might occur.

Figure 3-13 shows the exceptions thrown by the .NET data providers when an unrecoverable error occurs. You can refer to these errors as *data provider errors*. As you can see, each data provider has its own exception class. So, if you're using the SQL Server data provider, you should catch exceptions of the SqlException class. If you're using the Oracle data provider, you should catch exceptions of the OracleException class. And so on.

The code example in this figure shows how you can catch a SqlException that might occur when attempting to fill a dataset using a table adapter. Here, the shaded lines show the code that has been added to the generated code. This code will display an error message when a SqlException occurs, and it uses the Number and Message properties of the SqlException class to display details about the exception. It also uses the GetType method to indicate the type of exception that occurred.

Although it's uncommon, more than one server error can occur as the result of a single database operation. In that case, an error object is created for each error. These objects are stored in a collection that you can access through the Errors property of the exception object. Each error object contains a Number and Message property just like the exception object. However, because the Number and Message properties of the exception object are set to the Number and Message properties of the first error in the Errors collection, you don't usually need to work with the individual error objects.

.NET data provider exception classes

Name	Description
SqlException	Thrown if a server error occurs when accessing a SQL Server database.
OracleException	Thrown if a server error occurs when accessing an Oracle database.
OdbcException	Thrown if a server error occurs when accessing an ODBC database.
OleDbException	Thrown if a server error occurs when accessing an OLE DB database.

Common members of the .NET data provider exception classes

Property	Description
`Number`	An error number that identifies the type of error.
`Message`	A message that describes the error.
`Source`	The name of the provider that generated the error.
`Errors`	A collection of error objects that contain information about the errors that occurred during a database operation.
Method	**Description**
`GetType()`	Gets the type of the current exception.

Code that catches a SQL exception

```
Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
    Try
        Me.VendorsTableAdapter.Fill(Me.PayablesDataSet.Vendors)
    Catch ex As SqlException
        MessageBox.Show("SQL Server error # " & ex.Number _
            & ": " & ex.Message, ex.GetType.ToString)
    End Try
End Sub
```

Description

- Whenever the data provider (SQL Server, Oracle, ODBC, or OLE DB) encounters a situation it can’t handle, a data provider exception is thrown. You can handle these types of exceptions by catching them and displaying appropriate error messages.
- The Number and Message properties pinpoint the specific server error that caused the data provider exception to be thrown.
- The SqlException class is stored in the System.Data.SqlClient namespace.

Figure 3-13 How to handle data provider errors

How to handle ADO.NET errors

When you work with bound controls, *ADO.NET errors* can occur when the data in those controls is saved to the dataset (not the database), or when an Insert, Update, or Delete statement can't be executed against the database. Figure 3-14 presents some of the most common of those errors.

Here, ConstraintException and NoNullAllowedException are subclasses of the DataException class, so you can catch either of these errors by catching DataException errors. In contrast, DBConcurrencyException isn't a subclass of the DataException class, so you must catch DBConcurrencyException errors separately. All of the ADO.NET exception classes are members of the System.Data namespace.

The error-handling code in this figure catches errors caused by the EndEdit method of a binding source and the Update method of a table adapter. The first exception, DBConcurrencyException, occurs if the number of rows that are affected by an insert, update, or delete operation is zero, which typically indicates that concurrency errors have occurred. Then, a message box is used to display an error message, and the Fill method of the table adapter is used to retrieve the current data from the database and load it into the Vendors data table. That will help prevent further concurrency errors from occurring.

Although you might think that a concurrency error would be generated by the database rather than ADO.NET, that's not the case. To understand why, you need to remember that the Update and Delete statements that are generated for a table adapter contain code that checks that a row hasn't changed since it was retrieved. But if the row has changed, the row with the specified criteria won't be found and the SQL statement won't be executed. When the table adapter discovers that the row wasn't updated or deleted, however, it realizes there was a concurrency error and throws an exception.

Like other exception classes provided by the .NET Framework, each ADO.NET exception class has a Message property and a GetType method that you can use to display information about the error. You can see how this property and method are used in the second Catch block in this figure, which catches any other ADO.NET exceptions that may occur. This Catch block displays a dialog box that uses the Message property and the GetType method of the DataException object to describe the error. Then, it uses the CancelEdit method of the binding source to cancel the current edit operation.

Incidentally, to test your handling of concurrency exceptions, you can start two instances of Visual Studio and run the same application from both of them. Then, you can access and update the same row from both instances.

Common ADO.NET exception classes

Class	Description
DBConcurrencyException	The exception that's thrown by the data adapter if the number of rows affected by an insert, update, or delete operation is zero. This exception is typically caused by a concurrency violation.
DataException	The general exception that's thrown when an ADO.NET error occurs.
ConstraintException	The exception that's thrown if an operation violates a constraint. This is a subclass of the DataException class.
NoNullAllowedException	The exception that's thrown when an add or update operation attempts to save a null value in a column that doesn't allow nulls. This is a subclass of the DataException class.

Common members of the ADO.NET exception classes

Property	Description
`Message`	A message that describes the exception.
Method	**Description**
`GetType()`	Gets the type of the current exception.

Code that handles ADO.NET errors

```
Try
    Me.VendorsBindingSource.EndEdit()
    Me.VendorsTableAdapter.Update(Me.PayablesDataSet.Vendors)
Catch ex As DBConcurrencyException
    MessageBox.Show("A concurrency error occurred. " _
        & "The row was not updated.", "Concurrency Exception")
    Me.VendorsTableAdapter.Fill(Me.PayablesDataSet.Vendors)
Catch ex As DataException
    MessageBox.Show(ex.Message, ex.GetType.ToString)
    VendorsBindingSource.CancelEdit()
Catch ex As SqlException
    MessageBox.Show("SQL Server error # " & ex.Number _
        & ": " & ex.Message, ex.GetType.ToString)
End Try
```

Description

- An ADO.NET exception is an exception that occurs on any ADO.NET object. All of these exceptions are members of the System.Data namespace.
- In most cases, you'll catch specific types of exceptions if you want to perform special processing when those exceptions occur. Then, you can use the DataException class to catch other ADO.NET exceptions that are represented by its subclasses.

Figure 3-14 How to handle ADO.NET errors

How to handle data errors for a DataGridView control

Because the DataGridView control was designed to work with data sources, it can detect some types of data entry errors before they're saved to the dataset. If, for example, a user doesn't enter a value for a column that's required by the data source, or if a user tries to add a new row with a primary key that already exists, the DataGridView control will raise the DataError event. Then, you can code an event handler for this event as shown in figure 3-15.

The second parameter that's received by this event handler has properties you can use to display information about the error. The one you'll use most often is the Exception property, which provides access to the exception object that was thrown as a result of the error. Like any other exception object, this object has a Message property that provides a description of the error. You can also use the RowIndex and ColumnIndex properties of the second parameter to identify the row and column that caused the data error.

An event of the DataGridView control

Event	Description
`DataError`	Raised when the DataGridView control detects a data error such as a value that isn't in the correct format or a null value where a null value isn't valid.

Three properties of the DataGridViewDataErrorEventArgs class

Property	Description
`Exception`	The exception that was thrown as a result of the error. You can use the Message property of this object to get additional information about the exception.
`RowIndex`	The index for the row where the error occurred.
`ColumnIndex`	The index for the column where the error occurred.

Code that handles a data error for a DataGridView control

```
Private Sub TermsDataGridView_DataError(ByVal sender As System.Object, _
        ByVal e As System.Windows.Forms.DataGridViewDataErrorEventArgs) _
        Handles TermsDataGridView.DataError
    Dim row As Integer = e.RowIndex + 1
    Dim errorMessage As String = "A data error occurred." & vbCrLf _
        & "Row: " & row & vbCrLf _
        & "Error: " & e.Exception.Message
    MessageBox.Show(errorMessage, "Data Error")
End Sub
```

Description

- You can code an event handler for the DataError event of the DataGridView control to handle any data errors that occur when working with the DataGridView control.
- You can use the Exception, RowIndex, and ColumnIndex properties of the second parameter of the event handler to display a meaningful error message.

Figure 3-15 How to handle data errors for a DataGridView control

How to use the Dataset Designer

The *Dataset Designer* lets you work with a dataset schema using a graphic interface. In the topics that follow, you'll learn three basic skills for working with the Dataset Designer. Then, in chapter 5, you'll learn some additional skills for using this designer.

How to view the schema for a dataset

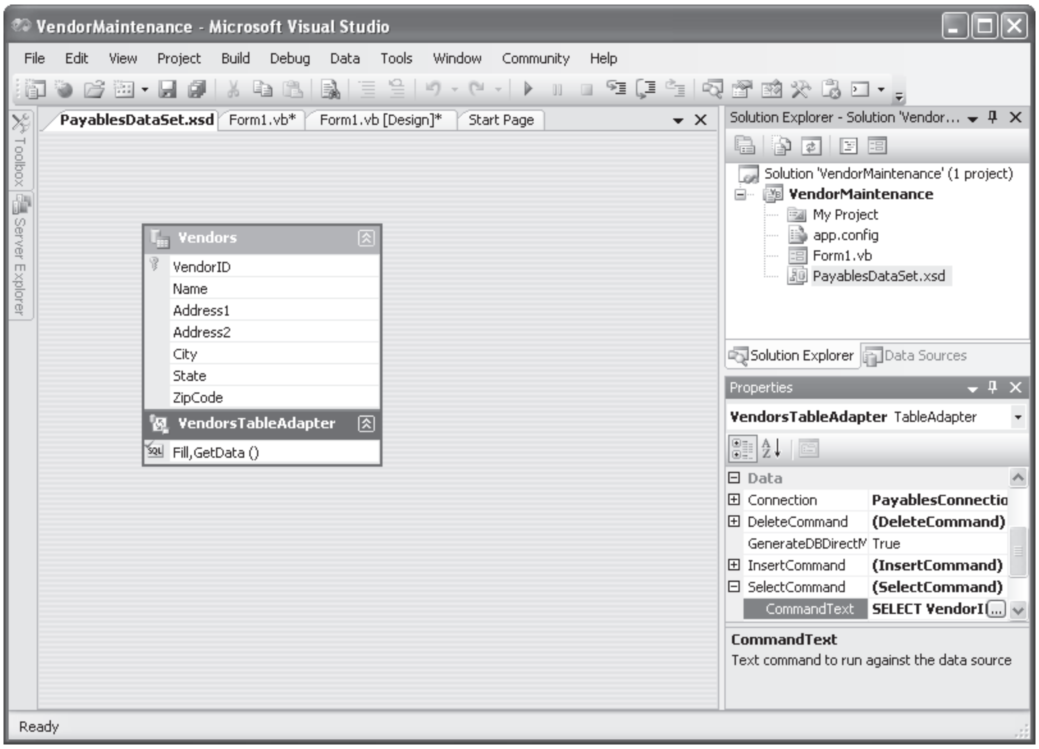
To learn more about a dataset, you can display its schema in the Dataset Designer. In figure 3-16, for example, you can see the schema for the Payables dataset used by the Vendor Maintenance application. For this simple application, this dataset contains just the Vendors table since this is the only table used by the application. The key icon in this table indicates that the VendorID column is the primary key for the table.

For each table in a dataset, the dataset schema also includes a table adapter that lists the queries that can be used with the table. Each table adapter includes at least a *main query* named Fill that determines the columns that are used when you drag the table from the Data Sources window. This query is also used to generate the Insert, Update, and Delete statements for the table. In addition, the table adapter includes any other queries you've defined for the table. You'll learn more about defining additional queries in the next two chapters.

If you click on a table adapter in the Dataset Designer, you'll see that its properties in the Properties window include the ADO.NET objects that the table adapter defines. That includes a Connection object, as well as SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand objects. If you expand any of these command objects, you can look at the CommandText property that defines the SQL statement it executes. In this figure, for example, you can see the beginning of the Select statement for the SelectCommand object that's used by the Fill query of the table adapter for the Vendors table. If you click on the ellipsis button for this property, you can work with the query using the Query Builder that's described in chapter 5.

Note that the Dataset Designer also makes it easy to set the properties for a column in a table that's in the dataset. To do that, just select a column and use the Properties window. For instance, you can use this technique to set the DefaultValue property for a column in the dataset, which is something that you often have to do.

The schema displayed in the Dataset Designer


VendorMaintenance - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
PayablesDataSet.xsd | Form1.vb* | Form1.vb [Design]* | Start Page
Toolbox
Server Explorer
Vendors
VendorID
Name
Address1
Address2
City
State
ZipCode
VendorsTableAdapter
Fill,GetData ()
Solution Explorer - Solution 'Vendor...
Solution 'VendorMaintenance' (1 project)
VendorMaintenance
My Project
app.config
Form1.vb
PayablesDataSet.xsd
Solution Explorer
Data Sources
Properties
VendorsTableAdapter TableAdapter
Data
Connection PayablesConnectio
DeleteCommand (DeleteCommand)
GenerateDBDirectM True
InsertCommand (InsertCommand)
SelectCommand (SelectCommand)
CommandText SELECT VendorI
CommandText
Text command to run against the data source
Ready

Description

- To view the schema for the dataset of a data source, double-click on the schema file for the dataset (.xsd) in the Solution Explorer, or select the schema file and click the View Designer button at the top of the Solution Explorer. The schema is displayed in the *Dataset Designer*.
- To view the properties for a table adapter in the Properties window, select the table adapter in the Dataset Designer. These properties include the Connection object that's used to connect to the database, and the SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand objects that are used to work with the data in the database.
- For each table adapter, the query named Fill is the *main query*. This query determines the columns that are used when you drag a table from the Data Sources window onto a form. The Insert, Update, and Delete statements for the table are also based on this query.
- To view the properties for a query, select the query in the Dataset Designer.
- To work with the SQL statement in a CommandText property, you can click on the ellipsis button that appears when that property is selected. This displays the statement in the Query Builder, which you'll learn about in chapter 5.
- To view and set the properties for a column in a table, select the column. This is an easy way to set the DefaultValue property for a column.

Figure 3-16 How to view the schema for a dataset

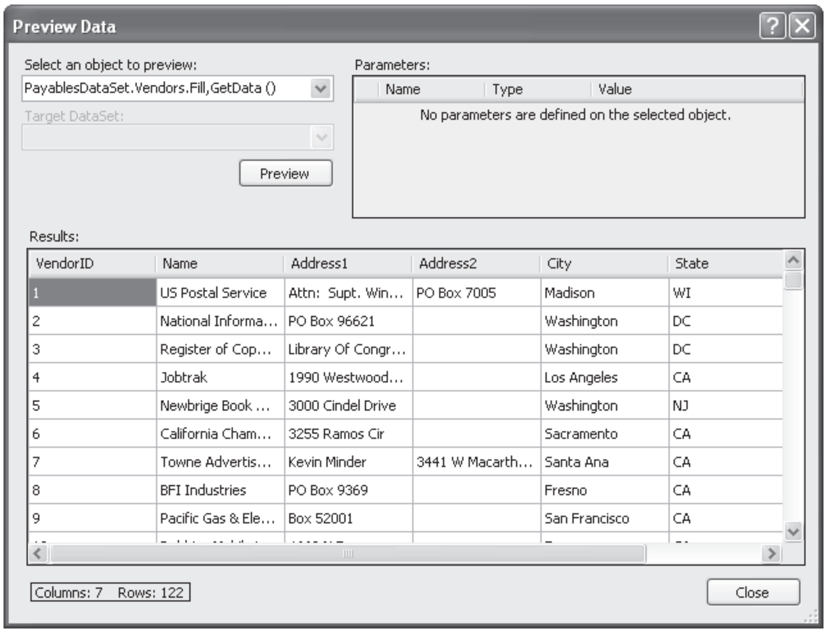
How to preview the data for a query

After you create a query, you can use the Dataset Designer to preview the data it retrieves. To do that, you use the Preview Data dialog box as shown in figure 3-17. Here, the data returned by the Fill query for the Vendors table adapter is being previewed.

To preview the data for a query, you just click the Preview button. When you do, the data will be displayed in the Results grid, and the number of columns and rows returned by the query will be displayed just below the grid. In this example, the query retrieved 7 columns and 122 rows.

In the next chapter, you'll learn how to create queries that use parameters. For those queries, you must enter a value for each parameter in the Value column of the Parameters grid before you can preview its data. For example, suppose you want to retrieve the data for a vendor with a specific vendor ID. Then, you have to enter that vendor ID in the Parameters grid to retrieve the data for that vendor.

The Preview Data dialog box


Preview Data
Select an object to preview:
PayablesDataSet.Vendors.Fill,GetData ()
Target DataSet:
Preview
Parameters:
Close

Name	Type	Value

No parameters are defined on the selected object.

Results:

VendorID	Name	Address1	Address2	City	State
1	US Postal Service	Attn: Supt. Win…	PO Box 7005	Madison	WI
2	National Informa…	PO Box 96621		Washington	DC
3	Register of Cop…	Library Of Congr…		Washington	DC
4	Jobtrak	1990 Westwood…		Los Angeles	CA
5	Newbrige Book …	3000 Cindel Drive		Washington	NJ
6	California Cham…	3255 Ramos Cir		Sacramento	CA
7	Towne Advertis…	Kevin Minder	3441 W Macarth…	Santa Ana	CA
8	BFI Industries	PO Box 9369		Fresno	CA
9	Pacific Gas & Ele…	Box 52001		San Francisco	CA

Columns: 7 Rows: 122

Description

- To display the Preview Data dialog box for a query, right-click on the query in the Dataset Designer and select the Preview Data command, or select the query and then use the Data➔Preview Data command.
- To preview the data, click the Preview button. When you do, the data will be displayed in the Results grid, and the number of columns and rows returned by the query will be displayed just below the Results grid.
- If a query requires parameters, you must enter a value for each parameter in the Value column of the Parameters grid. See chapter 4 for more information on query parameters.

Figure 3-17 How to preview the data for a query

How to interpret the generated SQL statements

The Fill method of a table adapter uses the SQL Select statement that's stored in the SelectCommand object for the Fill query of the table adapter to retrieve data from a database. Similarly, the Update method of a table adapter uses the SQL Insert, Update, and Delete statements that are stored in the InsertCommand, UpdateCommand, and DeleteCommand objects of the table adapter to add, update, and delete data from the database.

To help you understand what these statements do, figure 3-18 presents the Select statement for the Vendor Maintenance form and the Insert and Update statements that were generated from this statement. Although these statements may look complicated, the information presented here will give you a good idea of how they work.

To start, notice that the Insert statement is followed by a Select statement that retrieves the row that was just added to the database. That may be necessary in cases where the database generates some of the data for the new row. When a vendor row is added to the database, for example, the database generates the value of the VendorID column. Then, the Select statement in this figure uses the SCOPE_IDENTITY function that you learned about in chapter 1 to retrieve the row with this ID. For now, just realize that if the database doesn't generate or calculate any of the column values, this Select statement, as well as the one after the Update statement, aren't needed.

Also notice that the Update statement uses optimistic concurrency. (Although the Delete statement isn't shown here, it uses optimistic concurrency as well.) Because of that, code is added to the Where clause of this statement to check whether any of the column values have changed since they were retrieved from the database. This code compares the current value of each column in the database against the original value of the column, which is stored in the dataset. If one or more columns can contain a null value, it also checks if both the original value and the current value of those columns are null. That's the case for the Address2 column in the Vendors table. This is necessary because one null value isn't considered equal to another null value. Then, if none of the values have changed, the operation is performed. Otherwise, it's not.

Finally, notice that most of the statements in this figure use one or more parameters. For example, parameters are used in the Values clause of the Insert statement and the Set clause of the Update statement to refer to the current values of the columns in the dataset. Parameters are also used in the Where clause of the Update statement to refer to the original values of the columns in the dataset. The wizard inserts these parameters when it creates the command objects for a table adapter. Then, before each statement is executed, Visual Studio substitutes the appropriate value for each variable.

This should give you more perspective on how the dataset is refreshed and how optimistic concurrency is provided when you use ADO.NET. Because of the disconnected data architecture, these features can't be provided by the database management system or by ADO.NET. Instead, they are provided by the SQL statements that are generated by the Data Source Configuration Wizard.

SQL that retrieves vendor rows

```
SELECT     VendorID, Name, Address1, Address2, City, State, ZipCode
FROM       Vendors
```

SQL that inserts a vendor row and refreshes the dataset

```
INSERT INTO Vendors
             (Name, Address1, Address2, City, State, ZipCode)
VALUES       (@Name,@Address1,@Address2,@City,@State,@ZipCode);

SELECT     VendorID, Name, Address1, Address2, City, State, ZipCode
FROM       Vendors
WHERE      (VendorID = SCOPE_IDENTITY())
```

SQL that updates a vendor row and refreshes the dataset

```
UPDATE    Vendors
SET       Name = @Name, Address1 = @Address1, Address2 = @Address2,
          City = @City, State = @State, ZipCode = @ZipCode
WHERE     ((VendorID = @Original_VendorID) AND
          (Name = @Original_Name) AND (Address1 = @Original_Address1) AND
          (@IsNull_Address2 = 1) AND (Address2 IS NULL) AND
          (City = @Original_City) AND (State = @Original_State) AND
          (ZipCode = @Original_ZipCode)
          OR
          (VendorID = @Original_VendorID) AND
          (Name = @Original_Name) AND (Address1 = @Original_Address1) AND
          (Address2 = @Original_Address2) AND (City = @Original_City) AND
          (State = @Original_State) AND (ZipCode = @Original_ZipCode));

SELECT     VendorID, Name, Address1, Address2, City, State, ZipCode
FROM       Vendors
WHERE      (VendorID = @VendorID)
```

Description

- By default, the Data Source Configuration Wizard adds code to the Where clause of the Update and Delete statements that checks that the data hasn't changed since it was retrieved. (Although the Delete statement isn't shown here, its Where clause is identical to the Where clause of the Update statement.)
- By default, the Data Source Configuration Wizard adds a Select statement after the Insert and Update statements to refresh the new or modified row in the dataset.
- If a column can contain a null value, code is added to the Where clause of the Update and Delete statements that checks if both the original column value and the current value of the column in the database are null. That's necessary because two null values aren't considered equal.
- The SQL statements use parameters to identify the new values for an insert or update operation. Parameters are also used for the original column values, which are used to check that a row hasn't changed for an update or delete operation. And one is used in the Where clause of the Select statement after the Update statement to refer to the current row. The values for these parameters are stored in and retrieved from the dataset.

Figure 3-18 How to interpret the generated SQL statements

Perspective

Now that you've completed this chapter, you should be able to use a data source to create simple applications that let you view and maintain the data in one table of a database. That should give you some idea of how quickly and easily you can create applications when you use the data source feature. And in the next two chapters, you'll learn how you can use data sources and datasets to build more complex applications.

Terms

data source	complex data binding
schema	simple data binding
typed dataset	data provider error
untyped dataset	ADO.NET error
binding a control	Dataset Designer
bound control	main query

Before you do any of the exercises…

Before you do any of the exercises in this book, you need to download the directories and files for this book from our web site and install them on your PC. When you do, a directory named ADO.NET 2.0 VB will be created on your C drive. This directory will contain the subdirectories and files you need to do the exercises. For example, you can build the applications for this chapter in the C:\ADO.NET 2.0 VB\Chapter 03 directory. You also need to install SQL Server Express and attach the Payables database that you've downloaded as explained in appendix A.

Exercise 3-1 Build a DataGridView application

In this exercise, you'll build the application shown in figure 3-9. That will show you how to build a simple application with data sources, a dataset, and a GridView control.

Build the form and test it with valid data

1. Start a new application named TermsMaintenance in your chapter 3 directory, and use the techniques in figures 3-1 through 3-8 to create the data source and drag it onto the form. Then, adjust the size of the form and the DataGridView control as needed, but don't change anything else.
2. Test the application with valid data in three phases. First, sort the rows by clicking on a column header, and size one of the columns by dragging its column separator. Second, change the data in one column of a row, and move

to another row to see that the data is changed in the dataset. Third, add a new row with valid data in all columns, and move to another row to see that the row has been added. At this point, the changes have been made to the dataset only, not the database. Now, click the Save button to save the changes to the database.

Test the form with invalid data and provide exception handling

3. Test the application with invalid data by deleting the data in the Description column of a row and moving to another row. This should cause a NoNullAllowedException that's automatically handled by the DataGridView control so the application doesn't crash.
4. Add an exception handler for the DataError event of a DataGridView control as shown in figure 3-15. To start the code for that handler, click on the control, click on the Events button in the Properties window, and double-click on the DataError event. Then, write the code for the event, and redo the testing of step 3 to see how your code works.
5. When you're through experimenting, end the application and close the project.

Exercise 3-2 Build an application with text boxes

In this exercise, you'll build the application shown in figure 3-12. That will show you how to use data sources with controls like text boxes.

Build the form and test it with valid data

1. Start a new application named VendorMaintenance in your chapter 3 directory, and create a data source for the fields in the Vendor table that are used by the form in figure 3-12. Then, use the techniques in figures 3-10 and 3-11 to drag the data source fields onto the form as text boxes. At this point, the form should look like the one in figure 3-11.
2. Test the application with valid data in three phases. First, use the toolbar to navigate through the rows. Second, change the data in one column of a row, move to another row, and return to the first row to see that the data has been changed in the dataset. Third, add a new row with valid data in all columns, move to another row, and return to the added row to see that the row has been added to the dataset. Now, click the Save button to save the dataset to the database.

Test the form with invalid data and provide exception handling

3. Add a new row to the dataset, but don't enter anything into the City field. Then, click on the Save button. This should cause a NoNullAllowedException, since City is a required field.
4. Add exception handling code for an ADO.NET DataException as shown in figure 3-14 to catch this type of error. Then, run the application and redo the testing of step 3 to see how this error is handled now.

5. Delete the data in the Name column of a row, which means that the column contains an empty string. Next, move to another row, and return to the first row to see that the row has been accepted into the dataset. Then, click on the Save button and discover that this doesn't throw an exception because an empty string isn't the same as a null value. This indicates that data validation is required because an empty string isn't an acceptable value in the database. In the next chapter, you'll learn one way to provide data validation.
6. Adjust the controls on the form and any related properties so the form looks like the one in figure 3-12. This should take just a minute or two.

Use the Dataset Designer

7. Use one of the techniques in figure 3-16 to view the schema for the dataset in the Dataset Designer.
8. Click on the table adapter in the Dataset Designer and review its properties in the Properties window. Then, look at the Select statement that's used for getting the data into the dataset. To do that, click on the plus sign in front of SelectCommand, and click on the ellipsis button for CommandText. This opens up the Query Builder, which you'll learn about in chapter 5, and there you can see the Select statement that's used for getting the data into the dataset. Now, close the Query Builder.
9. Right-click on the query in the Dataset Designer, and preview the data that will be retrieved by that query as shown in figure 3-17.
10. When you're through experimenting, end the application and close the project.