

# A Programmer's Guide to ADO.NET in C#

MAHESH CHAND

Apress™

A Programmer's Guide to ADO.NET in C#  
Copyright ©2002 by Mahesh Chand

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-893115-39-9

Printed and bound in the United States of America 12345678910  
Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Technical Reviewer: Ildiko Blackburn, Boost Data Limited  
Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore,  
Karen Watterson, John Zukowski  
Managing Editor: Grace Wong  
Project Manager and Developmental Editor: Tracy Brown Collins  
Copy Editor: Kim Wimpsett  
Production Editor: Kari Brooks  
Composition: Impressions Book and Journal Services, Inc.  
Artist: Cara Brunk, Blue Mud Productions  
Indexer: Valerie Perry  
Cover Designer: Tom Debolski  
Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email [orders@springer-ny.com](mailto:orders@springer-ny.com), or visit <http://www.springer-ny.com>.

Outside the United States, fax +49 6221 345229, email [orders@springer.de](mailto:orders@springer.de), or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 9th Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax: 510-549-5939, email [info@apress.com](mailto:info@apress.com), or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

# Data Components in Visual Studio .NET

IN PREVIOUS CHAPTERS, YOU'VE SEEN the basics of the ADO.NET model and its components. Visual Studio (VS) .NET provides design-time support to work with data components. In this chapter, you'll learn how to use these data components in VS .NET at design-time to create database applications. Using these components is similar to using any Windows control. You just drag the component to a form, set its properties and methods, and you're up and running.

In this chapter I'll start with the Server Explorer, a useful tool for database applications. I'll focus on developing database applications quickly, using data components in VS .NET without writing a lot of code. I'll also show you a step-by-step tutorial to help you develop and run a project. After that, I'll discuss data connection, data adapter, data command, dataset, and data view components in more detail. After finishing this chapter, you'll have a good understanding of data components and how to work with them in VS .NET.

## Creating Your ADO.NET Project

Begin your project by launching VS .NET and choosing New > Project from the Project menu. Choose Visual C# Projects from Project Types and then pick the Windows Application template. If you like, type an appropriate name into the Name field for your first ADO.NET application and click OK (see Figure 4-1).

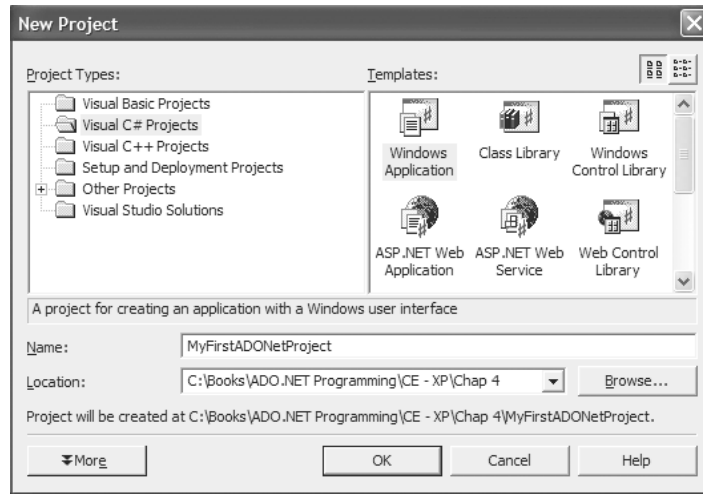
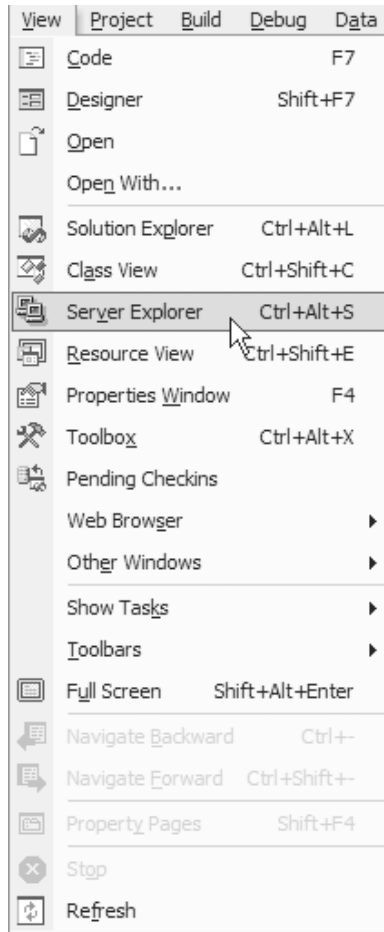


Figure 4-1. Creating a new project

## Using the Server Explorer

The Server Explorer is new to Visual Studio .NET. You can open the Server Explorer by clicking the View > Server Explorer menu item, as shown in Figure 4-2.



*Figure 4-2. Opening the Server Explorer*

The Server Explorer enables you to manage your database servers and connections. If you've ever used ODBC in your applications, then you're probably familiar with the traditional Windows ODBC Administration where you created data source names (DSNs) using ODBC drivers for a data source and then connected your application using this DSN.

Well, now you don't have to worry about it. You can use the Server Explorer to add a new server or a data connection to your list.

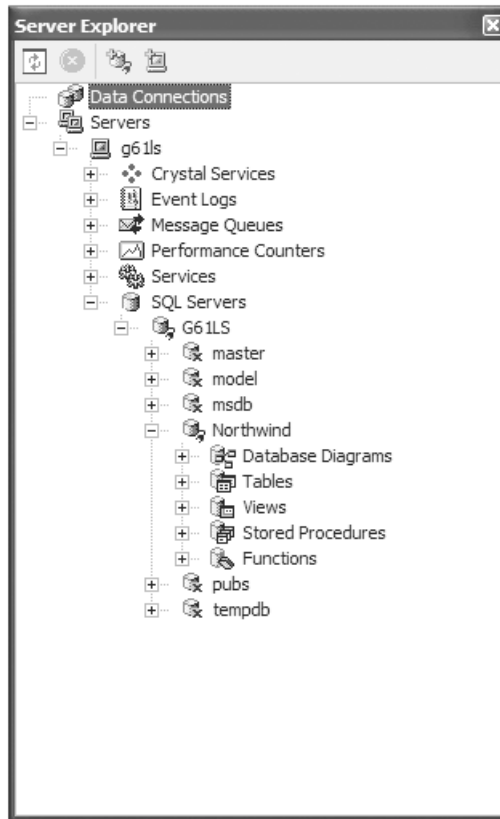


Figure 4-3. Adding a server through the Server Explorer

As you see in Figure 4-3, the Server Explorer has two root nodes: Data Connections and Servers. By right-clicking on these nodes you can add a new data connection or a new server to your list.

Specifically, to add a new server to the Server Explorer, you right-click on the Servers node, select the Add Server menu option, and enter the server name.

### *Adding a New Connection*

Adding a new connection is the next step after adding a server (if you're using a server) to the Server Explorer. You add a new connection to your list by right-clicking on the Data Connections tree item and choosing the Add Connection option. This brings up a Data Link Properties Wizard. The first tab of this wizard, Provider, displays all the data source providers installed on your machine; this is

where you select your database provider. The list could contain any OLE-DB provider, Jet OLE-DB, or other data driver available on your computer. Figure 4-4 shows you a list of providers on my machine.

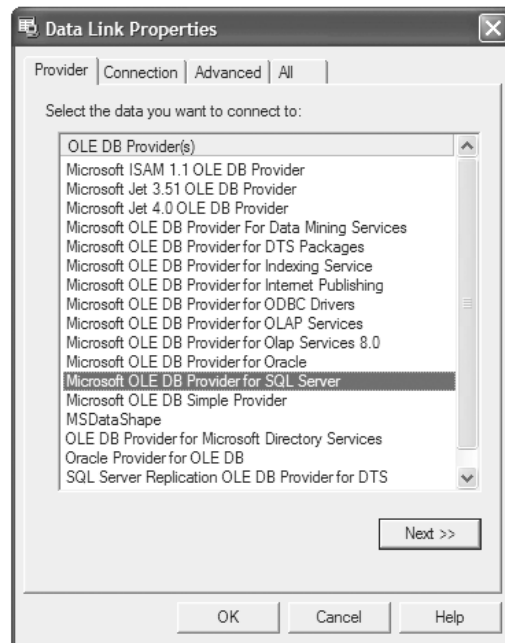


Figure 4-4. Choosing a data provider

The second tab of this wizard, Connection, lets you pick your server and corresponding data source. The drop-down list displays all the available servers. My server is a SQL Server with the default name localhost. After selecting a server, the database drop-down list displays all the available databases on the server. I'll select the Northwind database in this example. By clicking the Test Connection button, you can make sure your database connection is working. If you've provided a wrong user ID or password, the test will throw an error (see Figure 4-5).

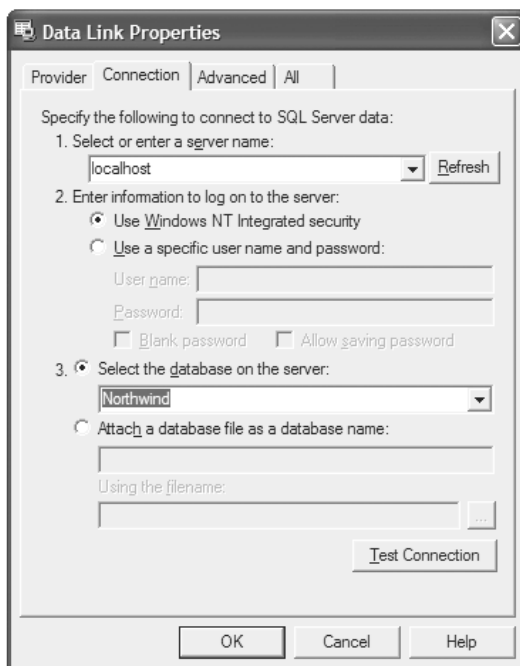
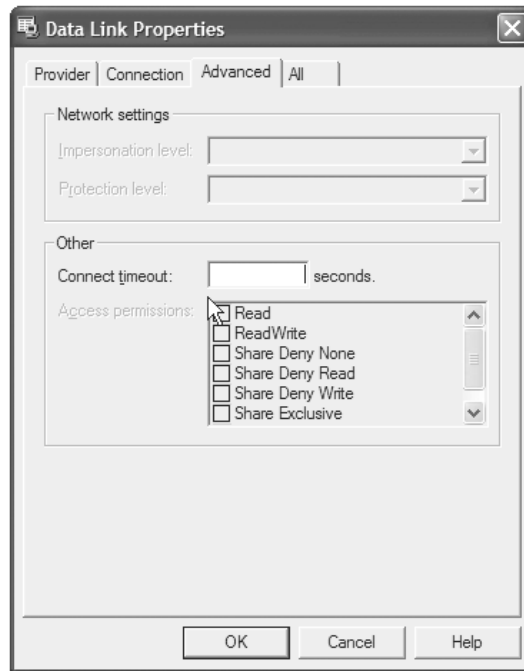


Figure 4-5. Selecting a database from SQL Server

The third tab, Advanced, is for setting connection timeout and access permissions. You can give this connection read, write, or other permissions using the Advanced tab (see Figure 4-6).





*Figure 4-6. Additional options such as permissions and the connection timeout period*

## Managing and Viewing Data

The Server Explorer not only lets you add server and database connections, it also lets you manage and view data. You can add, update, and delete data from a database. The Server Explorer also provides options to create new databases and objects, including tables, views, stored procedures, and so on.

The Server Explorer manages database objects in a tree structure. Each database is a tree node of the server. As you expand the Northwind database node, you can see its children listed as tables, stored procedures, and views (see Figure 4-7).

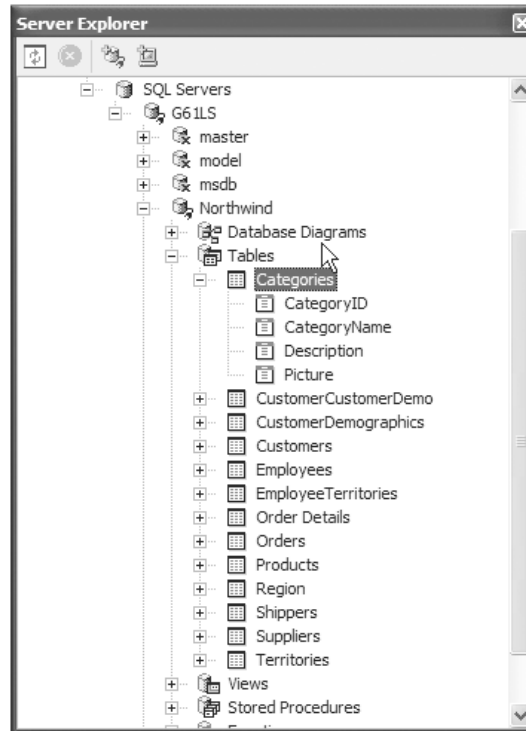


Figure 4-7. The Server Explorer with database tables

If you expand this connection by double-clicking on it, you'll notice it shows tables, views, and stored procedures. You can further expand these to see them in more detail.

Besides showing a list of database objects such as tables, views, stored procedures, and functions, the Server Explorer also lets you view, add, edit, and delete data from a data source. Figure 4-8 shows the Employees table of the Northwind database in the Server Explorer. In Figure 4-8, you see the data in a grid. You can edit this data at any time. For example, to delete a row or a collection of rows, select the rows and hit Delete, or right-click on the selected rows and hit the Delete option. The right-click option of the grid also provides you options to move to the grid's first, next, previous, and last records.

EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate
1	Davolio	Nancy	Sales Representative	Ms.	12/8/1948	5/1/1992
2	Fuller	Andrew	Vice President, Sales	Dr.	2/19/1952	8/14/1992
3	Leverling	Janet	Sales Representative	Ms.	8/30/1963	4/1/1992
4	Peacock	Margaret	Sales Representative	Mrs.	9/19/1937	5/3/1993
5	Buchanan	Steven	Sales Manager	Mr.	3/4/1955	10/17/1993
6	Suyama	Michael	Sales Representative	Mr.	7/2/1963	10/17/1993
7	King	Robert	Sales Representative	Mr.	5/29/1960	1/2/1994
8	Callahan	Laura	Inside Sales Coordinator	Ms.	1/9/1958	3/5/1994
9	Dodsworth	Anne	Sales Representative	Ms.	1/27/1966	11/15/1994
*						

Figure 4-8. The Employee table in the Server Explorer

You can also right-click on a table and choose Retrieve Data from Table to retrieve data of that table, as shown in Figure 4-9.

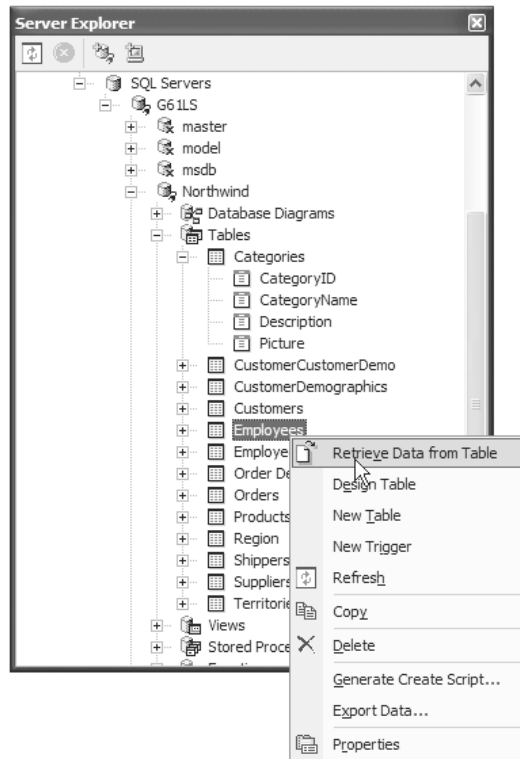


Figure 4-9. Retrieving data from a table in the Server Explorer

## Using Visual Data Components

As mentioned in Chapter 2, “Introduction to Windows Forms,” Microsoft .NET provides many data providers to work with different types of data sources. The class hierarchy model of these data providers remains the same, so programmers won’t have any problem switching between data providers. Some of these data providers are OleDb, Sql, and Odbc. The Odbc data provider was a new addition to the .NET Framework (added after .NET Beta 2). If you don’t have Odbc data providers available in your namespaces, you can install the Odbc data provider by installing Odbc .NET Software Development Kit (SDK) from the Microsoft site (<http://msdn.microsoft.com/data/>).

**NOTE** *This location may change. You can always find the updated URL in the downloads section (<http://www.c-sharpcorner.com/downloads.asp>) of C# Corner.*

If you’re not sure, you can check the toolbox to see if you have an Odbc data provider already installed. The toolbox’s Data tab shows you the available data controls in Visual Studio. These components are DataSet, DataView, SqlConnection, SqlCommand, SqlDataAdapter, OleDbConnection, OleDbCommand, and OleDbDataAdapter (see Figure 4-10).

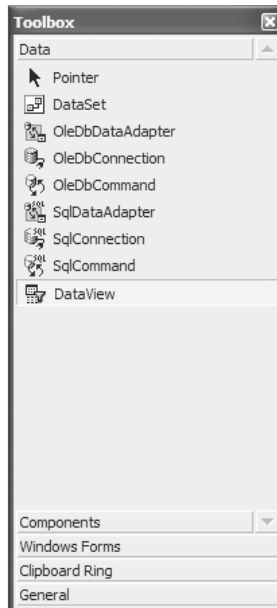


Figure 4-10. Data components

With the OleDb and Sql data components, if you also see ODBC components, then you already have the Odbc data provider installed. Otherwise, you have to install the Odbc data provider. After installing ODBC .NET SDK, you need to go your toolbox to see the ODBC data components. After installing the ODBC .NET SDK, right-click on the toolbox and select Customize Toolbox (see Figure 4-11).

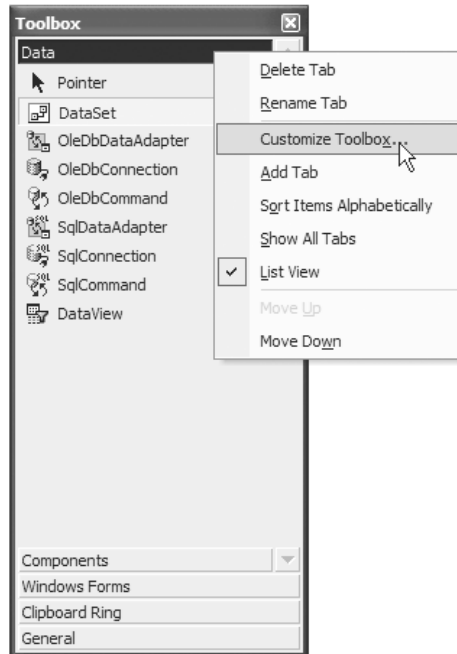


Figure 4-11. The Customize Toolbox option

Now, you'll notice a list of Component Object Model (COM) components and .NET Framework components (see Figure 4-12). Click on the .NET Framework Components tab and select the OdbcCommand, OdbcConnection, OdbcCommandBuilder, and OdbcDataAdapter components. If these components don't show up in the tab, then you need to browse for the component using the Browse button. You can usually find the ODBC components stored as `\Program Files\Microsoft.NET\Odbc.NET\Microsoft.Data.Odbc.dll`.

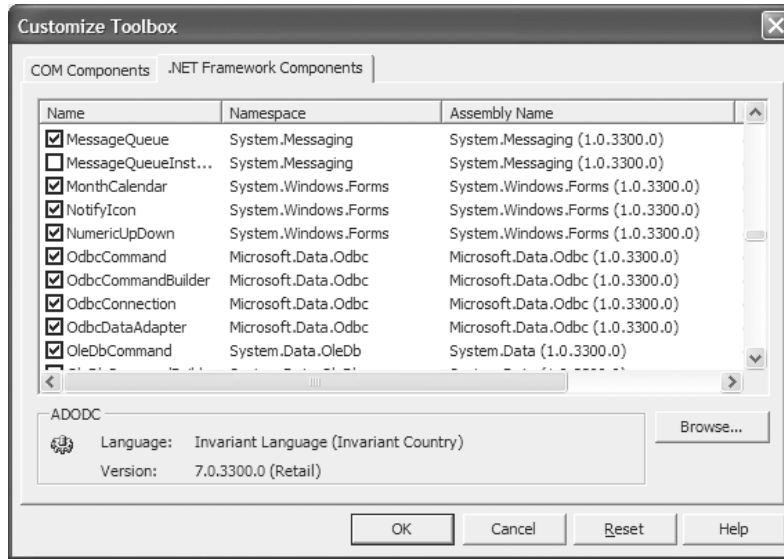


Figure 4-12. ODBC data components

After clicking the OK button, use the Toolbox > Data option to see your ODBC data components (see Figure 4-13).

**NOTE** *If you don't see this file in your Microsoft .Net directory, the ODBC.NET SDK may not have installed on your machine. Try reinstalling it.*

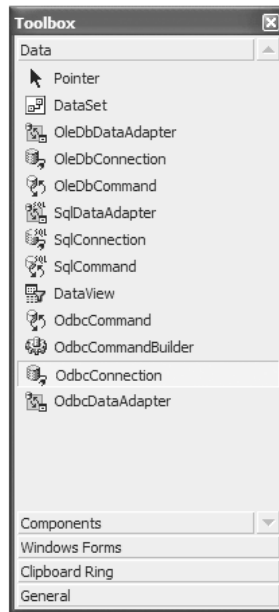


Figure 4-13. Viewing your ODBC data components in the toolbox

As mentioned briefly in Chapter 3, “Overview of ADO.NET,” the .NET Framework Library contains many ADO.NET data providers, including OleDb, Sql, and Odbc. The OleDb data provider wraps up native OLE-DB COM API to work with OLE-DB data sources. To access an OLE-DB data source, you need to install an OLE-DB data provider for that database. Sql data providers work with SQL Server 7 or later databases. Odbc data providers wrap up the ODBC API to work with ODBC data sources (with the help of ODBC Admin and ODBC drivers). Chapter 5 discusses these data providers in more detail. You can even create your own custom data providers. Microsoft and other vendors might add more data providers, which can be added to the library later.

In the .NET Framework, each of these data providers has its own namespaces. For instance, the System.Data.OleDb namespace consists of classes belonging to the OleDb data providers. All of these namespace classes start with OleDb. The System.Data.ODBC and System.Data.SqlClient namespaces consist of classes belonging to the Odbc and Sql data providers, respectively. Similar to OleDb, classes in Odbc start with Odbc, and classes in SqlClient start with Sql.

In Visual C#, some of these classes (or objects) are available from the toolbox; you can add them to a form using drag-drop operation as any other Windows control in the toolbox. These controls are *data components*.

All of these types of components work in pretty much the same way except for the Connection component, whose connection string will vary based on the data source to which you’re connecting.

**NOTE** *In the next section, I'll discuss how you can add these components to your Window Forms applications and set their properties and methods at design-time with the help of the .NET wizards.*

VS .NET also provides a set of data-bound controls. DataGrid, ListBox, and DataList are good examples of some of these data-bound controls. It's fairly easy to work with these controls. You just set a few properties, and they're ready to display your data. For example, setting a DataGrid control's DataSource property displays data from a DataSet object. You'll see these controls in the examples throughout this chapter.

## *Understanding Data Connections*

To connect to a data source, the first thing you need to learn about is a *data connection*.

Each data provider has a connection class, and if you're using VS .NET, you can see these class objects as components in the Toolbox > Data tab. For example, the SqlConnection, OdbcConnection, and OleDbConnection class objects represent a connection for the Sql, Odbc, and OleDb data providers, respectively. See the following:

- SqlConnection creates and manages SQL Server database connections.
- OdbcConnection creates and manages connections to ODBC data sources.
- OleDbConnection creates and manages connections to an OLE-DB data sources.

In VS .NET, you can create a connection component in many ways. You can use the IDE to add a connection object to a project, create it programmatically, or use data adapters that automatically create a connection object for you. In this chapter, we'll be concentrating on adding a connection through VS .NET.

The easiest way to add a connection to a project in VS .NET is to drag a connection component (SqlConnection, OleDbConnection, or OdbcConnection) from the toolbox's Data tab. This action adds a connection object to your project. After that, you can set the connection's properties using the Properties windows. For this demonstration, I'll drop a SqlConnection from the toolbox onto the form. Figure 4-14 shows the Properties window displayed after creating the SqlConnection. Note that the default connection name is the class name with



a unique number appended to it. Because this is the first Connection object, the connection is sqlConnection1.

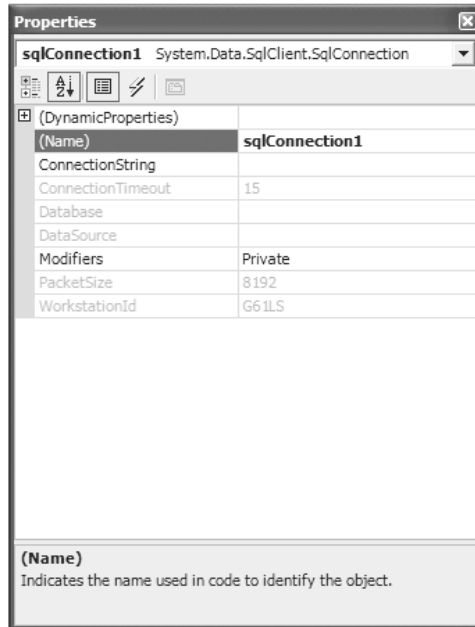


Figure 4-14. The SqlConnection component's properties

As you can see from the Properties window in Figure 4-14, a connection's properties include Database, ConnectionTimeout, DataSource, PacketSize, WorkstationId, Name, and ConnectionString.

**NOTE** *The connection properties depend on the data provider. Some properties may not be available for other data providers. For example, the WorkstationId property is available in Sql data providers but not in OleDb or ODBC data providers.*

### Understanding Connection Strings

The ConnectionString property is the main property of a connection. By clicking the drop-down list of the ConnectionString property, you can see all the available data connections. If you don't have a data connection, you can use its New Connection option (see Figure 4-15), which launches the Data Link Properties Wizard. Refer to the previous "Using the Server Explorer" section.

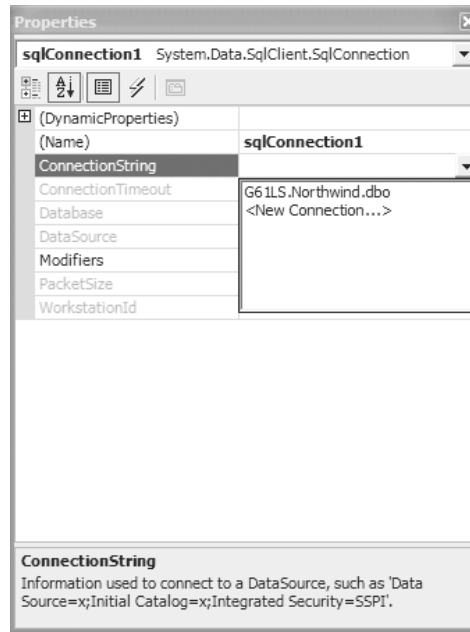


Figure 4-15. ConnectionString property options

After choosing the New Connection option and launching the Data Link Properties Wizard, you choose a server in the Connection tab. On my machine, the SQL Server's name is G61LS, the user ID and password aren't entered because I'm using Windows NT Integrated Security. You need to enter your server name (or select from the drop-down list), and enter your user ID and password if you're not using Windows NT Integrated Security option (see Figure 4-16).

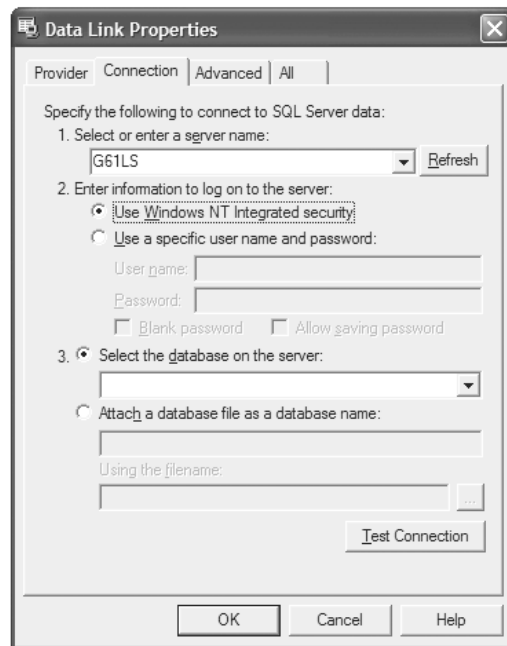


Figure 4-16. Data Link Properties Wizard

The SqlConnection string looks like following:

```
"data source=MCB;initial catalog=Northwind;persist security info=False;" +
"user id=sa;workstation id=MCB;packet size=4096"
```

**NOTE** In Chapter 5, I'll discuss a connection and its properties in more detail and show how to set them programmatically.

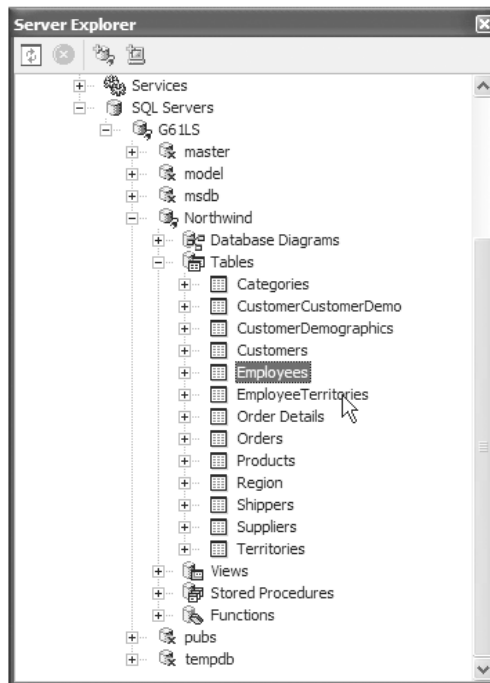
## Working with SQL Data Adapters

A *data adapter* is another important component of a data provider. Similar to the connection, each data provider has a corresponding data adapter class. All data adapters in ADO.NET work in the same way, which means if you know how to work with Sql data adapters, you can use OleDb, ODBC, and other data adapters easily. The `SqlDataAdapter`, `OleDbDataAdapter`, and `OdbcDataAdapter` classes represent data adapter components in Sql, OleDb, and ODBC data

providers, respectively. Besides creating a data adapter programmatically (see Chapter 5 for more details), VS .NET provides you with various ways to create data adapters. Two common ways are by using the Server Explorer and by using the Data Adapter Configuration Wizard.

### *Creating Data Adapters with the Server Explorer*

It's easy to create a data adapter using the Server Explorer. You just drag and drop database objects to a form, and the IDE takes care of everything for you. The IDE writes code that you can use programmatically or bind data controls at design-time. To add a new connection to a project, expand your database in the Server Explorer and drag a table from the Server Explorer to your form (see Figure 4-17).



*Figure 4-17. Creating an adapter using the Server Explorer*

This action creates a connection and a data adapter. You can even drag selected columns or stored procedures on the form. VS .NET takes care of the rest. Right-click on the form and choose View Code to examine the code generated by the wizard; in this example, you'll see one SqlConnection component and one SqlDataAdapter component along with a set of SqlCommand components:

```
private System.Data.SqlClient.SqlConnection sqlConnection1;
private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
```

Once you have a DataAdapter, you can use it to populate datasets and work with its properties. We'll discuss DataSet basics and how to construct them manually in Chapter 5 in more detail. With VS .NET, you can even generate datasets using the visual representation of the DataAdapter. We'll discuss how to populate a DataSet using VS .NET IDE wizards in the "Generating Typed DataSets Using Data Adapter" section of this chapter.

### *Creating Data Adapters with the Data Adapter Configuration Wizard*

The Data Adapter Configuration Wizard is a powerful tool to develop database applications. To see how you can create data adapters using the this wizard, you'll create a new Window Forms–based sample project.

In this first sample project, I'll show you how to create SQL data adapters, read data from a SQL Server data source, and display the data from a data adapter to a DataGrid control. Just follow the following simple steps in the next several sections. After completing these steps, you'll see how easy it is to develop database applications using the Data Adapter Configuration Wizard.

#### ***Step 1: Selecting a Project Template***

First, create a Windows Application template as you did at the beginning of the chapter (see Figure 4-18).

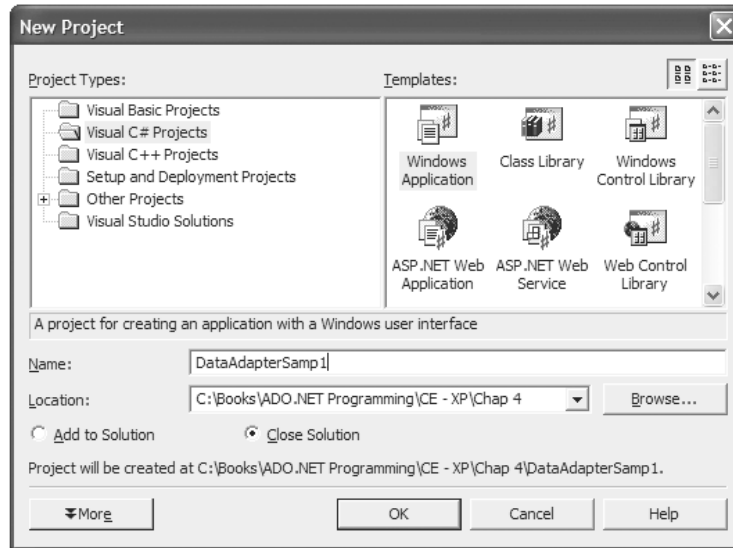


Figure 4-18. Creating a Windows Application project

### **Step 2: Adding a DataGrid Control to the Form**

Now add a DataGrid control to the form by dragging a DataGrid control from the Toolbox > Windows Forms category to the form.

### **Step 3: Adding a Data Adapter Component**

Next, drag a SqlDataAdapter control from the Toolbox > Data category to the form. As you drop the data adapter (Sql, OleDb, or ODBC), the Data Adapter Configuration Wizard pops up.

#### *Welcome Page*

The first page of this wizard is just a welcome screen (see Figure 4-19).



Figure 4-19. The Data Adapter Configuration Wizard welcome screen

### *Choose Your Data Connection Page*

The second page of the wizard lets you create a new connection or pick from a list of available connections on your machine. In this example, I'm using the default Northwind SQL Server database that comes with Visual Studio. As you can see in Figure 4-20, the Northwind connection is available in the list. Don't confuse it with G61LS, which is specific to my machine name. This name will be different for different machines. If you don't have any connection listed, you can use the New Connection button, which launches the Data Link Properties Wizard (discussed in the "Connection Strings" section).



*Figure 4-20. Choosing the Northwind SQL Server database in the Data Adapter Configuration Wizard*

### *Choose a Query Type*

The next page of the wizard is for command set types. A command set could consist of a SQL statement or a new or already existing stored procedure (see Figure 4-21).



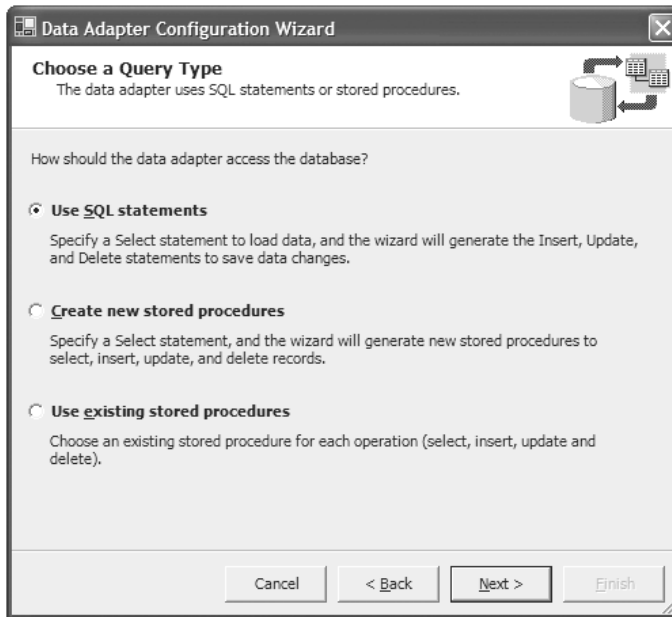


Figure 4-21. Choosing a query type in the Data Adapter Configuration Wizard

#### *Generate the SQL Statement*

The next page of the Data Adapter Configuration Wizard lets you build a SQL statement or a stored procedure (see Figure 4-22).



*Figure 4-22. Creating a Select statement through the Data Adapter Configuration Wizard*

### *Query Builder*

The Query Builder option lets you pick tables from your data source. First, select the Employees table to read in the Employee data. You actually have the option of selecting as many tables as you want, but for now select only one table (see Figure 4-23) and click the Add button.

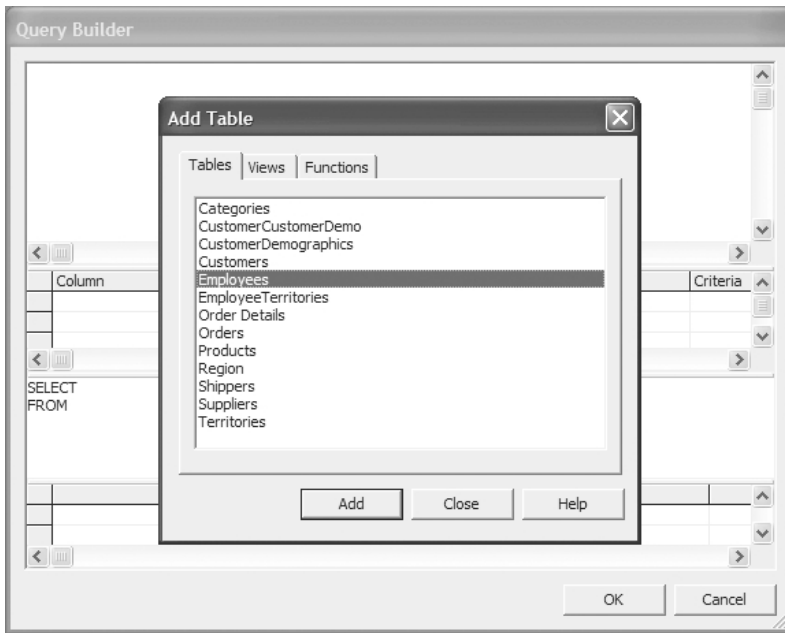


Figure 4-23. The Query Builder

If you've ever used Microsoft Access, you'll find that the Query Builder is similar to it. In Access, you can create queries by dragging tables and their columns to the grid (or checking the columns), and the Query Builder builds a SQL query for your action. In this sample, I'll select EmployeeID, FirstName, and LastName from the Employees table to build our SQL statements (see Figure 4-24).

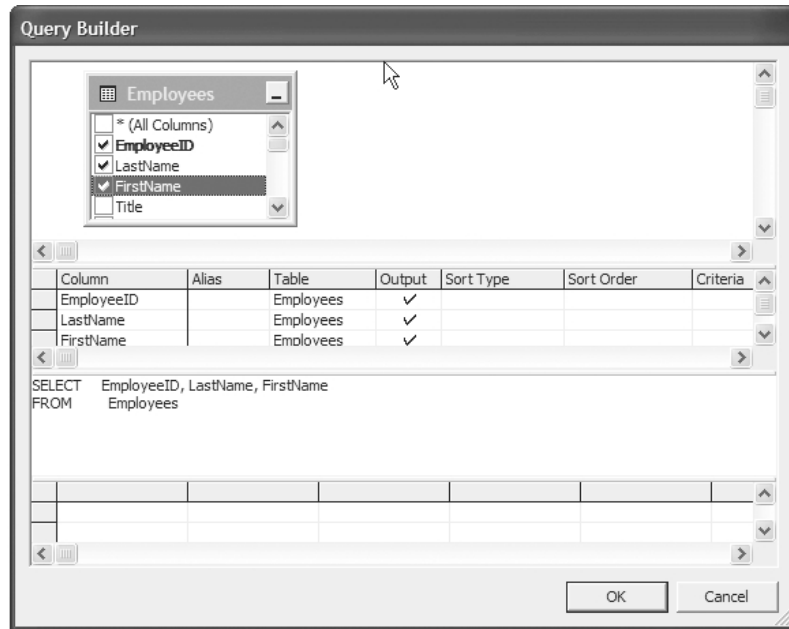


Figure 4-24. Building columns in the query

Now, I'll select three columns from the Employees table. The result looks like Figure 4-25.



Figure 4-25. The Query Builder selection

**NOTE** You can even write your own SQL statement if you don't want to use the Query Builder. For performance reasons, if you only want a few columns, then use column names instead of using SELECT \* statements.

### View Wizard Results

The View Wizard Results page shows you the action being taken by the wizard; in this example, it was successful. The Details section shows that the wizard has generated SQL Select, Insert, Update, and Delete statements and mappings (see Figure 4-26).

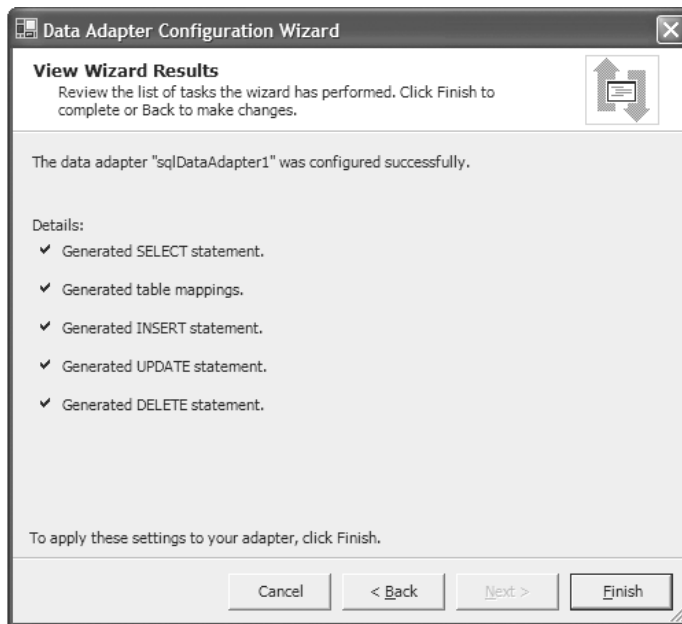


Figure 4-26. The View Wizard Results page

Now you can click the Finish button to complete the process.

Now, if you examine the form in Figure 4-27, you'll see two components: `sqlConnection1` and `sqlDataAdapter1`. The wizard sets the properties of these components for you. Now you can use the data adapter to populate your datasets. Don't forget to resize the DataGrid you added to the project.

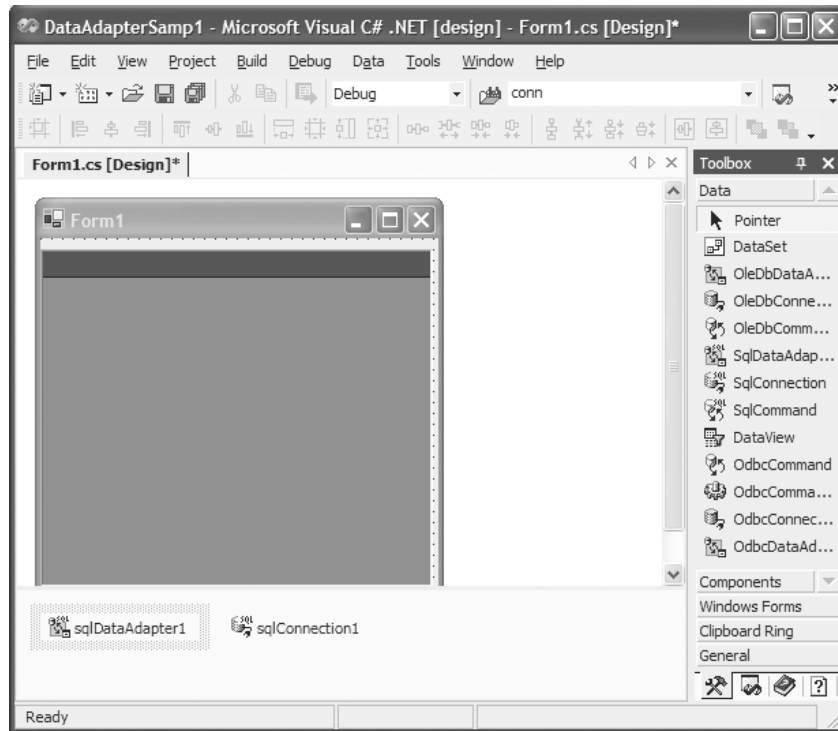


Figure 4-27. SqlConnection and SqlDataAdapter shown in the form designer

#### **Step 4: Setting and Reviewing Data Adapter Properties**

OK, now that you have a DataAdapter on your form, let's take a look at the SqlDataAdapter component properties. You can see its properties by right-clicking on the adapter and selecting the Properties menu item. The Properties window looks like Figure 4-28.

The wizard also shows the available command properties, including InsertCommand, DeleteCommand, SelectCommand, and UpdateCommand (see Figure 4-28).

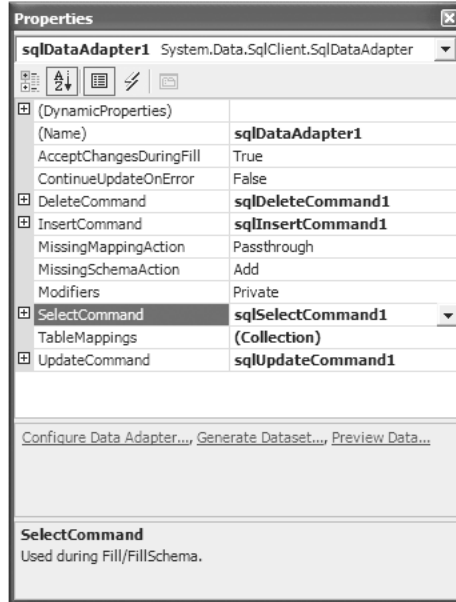


Figure 4-28. The data adapter in the Properties window

You can set `DataAdapter` properties by clicking on these properties. `SqlCommand` and `TableMappings`, for example, are important properties. A data adapter has four `SqlCommand` properties—`SelectCommand`, `DeleteCommand`, `InsertCommand`, and `UpdateCommand`—that all execute SQL commands on the data source. For example, if you look at the `SelectCommand` property in Figure 4-29, you'll see the SQL `Select` statement.

**NOTE** Chapter 5 covers `SelectCommand`, `InsertCommand`, `UpdateCommand`, and `DeleteCommand` in more detail.

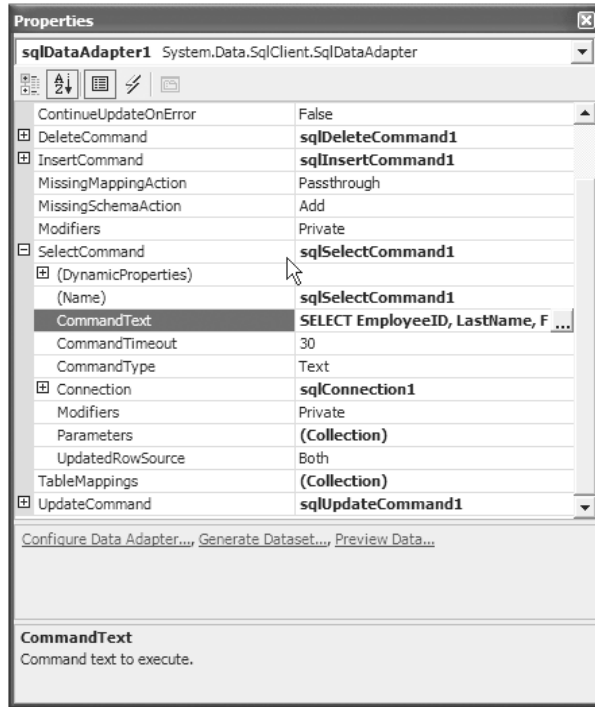


Figure 4-29. Setting the SQL SelectCommand in the data adapter

As you also see in Figure 4-29, you can set CommandText, CommandType, Connection, and so on using the Properties dialog box. If you double-click on CommandText, it pops up the Query Builder where you can rebuild your query (see Figure 4-30).



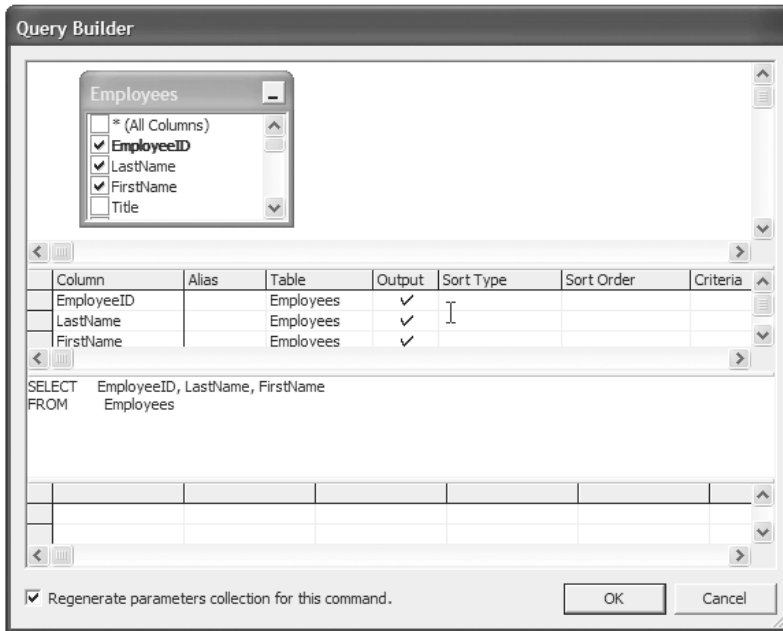


Figure 4-30. Relaunching the Query Builder from the CommandText property

The TableMapping class represents a mapping of DataColumn in the data source to DataColumn in the DataSet. I'll discuss DataTables and table mappings in more detail in Chapter 5. If you click on the TableMappings property (which is a collection of TableMapping objects), it brings up the Table Mappings dialog box.

As you can see from Figure 4-31, the Table Mapping dialog box has two columns: Source Table and Dataset Table. The Source Table column is a list of actual columns, and the Dataset Table column is a list of the column names used in the dataset. By default, dataset column names are the same as the source table. This is useful when you want to use different names in a program. You can change dataset columns by editing the column itself. Of course, you can't change source columns, but you can reorder them by using the column drop-down list.

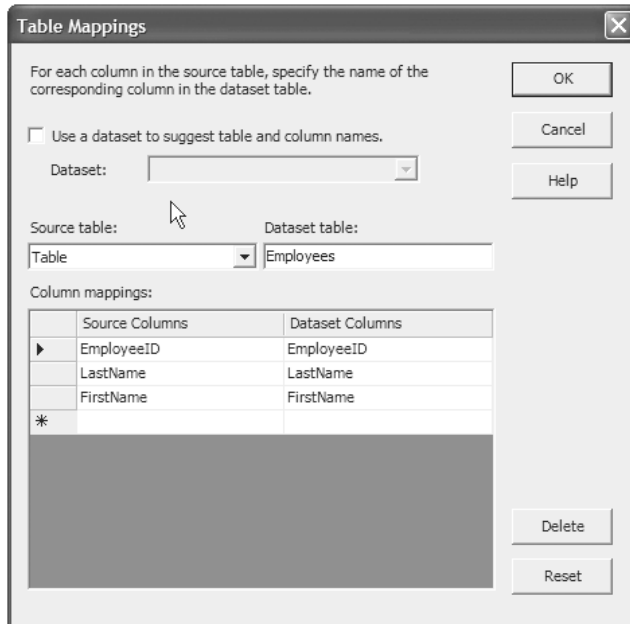


Figure 4-31. Table Mappings dialog box

By using this dialog box, you can even delete columns from your mapping using the Delete button.

#### **Step 4: Reviewing Other Options**

If you look closely at data adapter properties, you'll see three links: Configure Data Adapter, Generate Dataset, and Preview Data (see Figure 4-32).

The Configure Data Adapter option calls the Data Adapter Configuration Wizard, discussed earlier in this chapter. If you want to reset the wizard to change your options, you can use this link.

The Generate Dataset option lets you generate a dataset for this data adapter. I'll discuss how to generate datasets using data adapter properties in the "Working with OleDb Data Adapters" section of this chapter.

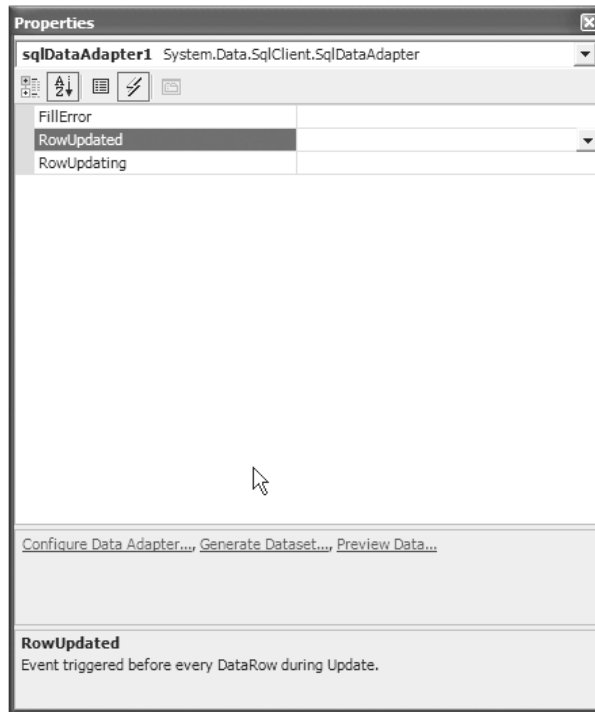


Figure 4-32. Data Adapter option links

The Preview Data option enables you to view the DataSet schema. You can even preview the data in the DataSet by clicking the Fill button. The Data Adapter Preview dialog box looks like Figure 4-33.

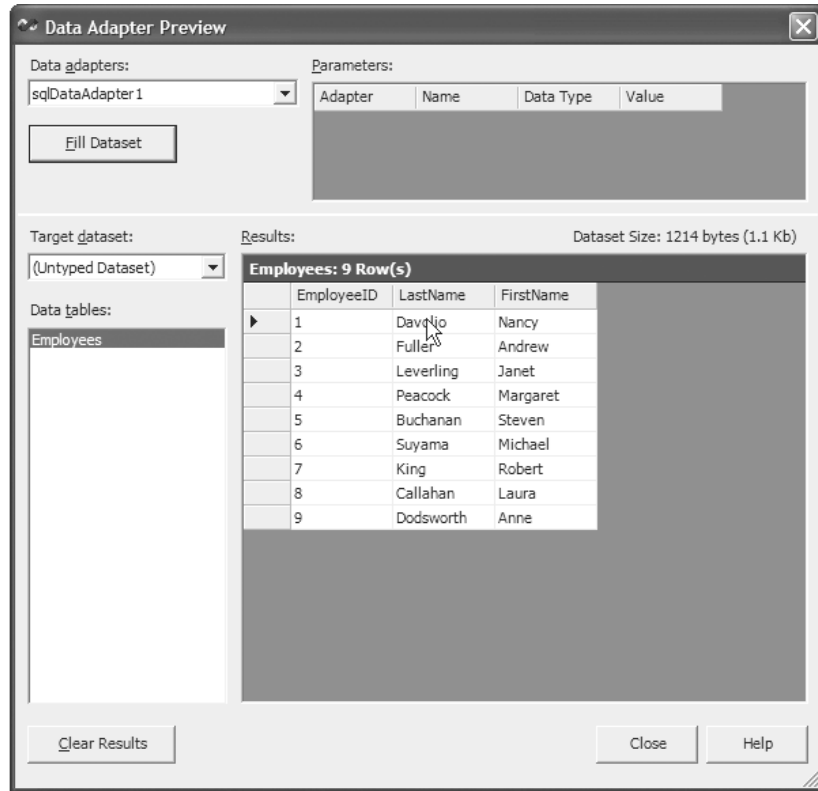


Figure 4-33. Previewing data for the data adapter

The Fill Dataset button in Figure 4-33 fills data into a grid based upon the current state of the SelectCommand in the DataAdapter.

### Step 5: Reviewing the Source Code

Now it's time to examine the code and see what the wizard has done for you automatically. You can see the source code by right-clicking on the form and selecting the View Source option.

**NOTE** *If you don't want to know what the wizard has automatically done for you, you can skip this step.*

All source code generated by the Windows form designer is defined in the `InitializeComponent` method of the file . Right-click on your form and choose `View Code`. Upon examining the source code, you'll see where the wizard has added two components, `sqlConnection1` and `sqlDataAdapter1`, to your source file as well as four `SqlCommand` components. Scroll down to the `Windows Designer Generated Code` option and expand it. This will reveal the contents of the `InitializeComponent` routine (see Listing 4-1).

*Listing 4-1. Added Sql Server provider components*

```
namespace DataAdapterSamp1
{
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.DataGrid dataGrid1;
        private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
        private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
        private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
        private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
        private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
        private System.Data.SqlClient.SqlConnection sqlConnection1;
        // more Source code
        private void InitializeComponent()
        {
            this.dataGrid1 = new System.Windows.Forms.DataGrid();
            this.sqlDataAdapter1 = new System.Data.SqlClient.SqlDataAdapter();
            this.sqlSelectCommand1 = new System.Data.SqlClient.SqlCommand();
            this.sqlInsertCommand1 = new System.Data.SqlClient.SqlCommand();
            this.sqlUpdateCommand1 = new System.Data.SqlClient.SqlCommand();
            this.sqlDeleteCommand1 = new System.Data.SqlClient.SqlCommand();
            this.sqlConnection1 = new System.Data.SqlClient.SqlConnection();
            ...
            // more code
            ...
        }
    }
}
```

Do a search for the `ConnectionString` by hitting `Ctrl+F` to bring up the search dialog box. If you examine the `InitializeComponent()` method, you'll see that the wizard sets `SqlConnection's ConnectionString` property to the following:

```
this.sqlConnection1.ConnectionString = "data source=(local);initial catalog" +
"=Northwind;persist security info=False;user id" +
"=mahesh;workstation id=7LJML01;packet size=4096";
```

It also sets the `CommandText` property of the `SqlCommand` with the corresponding `SELECT`, `INSERT`, `UPDATE`, and `DELETE` SQL statements. The `Connection` property of `SqlCommand` is set to `SqlConnection`:

```
this.sqlSelectCommand1.CommandText = "SELECT LastName, " +
"EmployeeID, FirstName FROM Employees";
this.sqlSelectCommand1.Connection = this.sqlConnection1;
```

If you examine the Listing 4-2, you'll see that `DataAdapter` is connected to a `Connection` through data commands, and the `TableMapping` property is responsible for mapping tables and their columns. Note that the `TableMappings` between `DataSet` columns and `DataSource` columns generated by the wizard have exactly the same column names.

*Listing 4-2. DataAdapter connection through TableMapping*

```
private void InitializeComponent()
{
    //
    // some code here
    //
    this.sqlDataAdapter1.DeleteCommand = this.sqlDeleteCommand1;
    this.sqlDataAdapter1.InsertCommand = this.sqlInsertCommand1;
    this.sqlDataAdapter1.SelectCommand = this.sqlSelectCommand1;
    Please break up code.
    this.sqlDataAdapter1.TableMappings.AddRange
    (new System.Data.Common.DataTableMapping[]
    {new System.Data.Common.DataTableMapping
    ("Table", "Employees", new System.Data.Common.DataColumnMapping[]
    {
        new System.Data.Common.DataColumnMapping("LastName", "LastName"),
        new System.Data.Common.DataColumnMapping("EmployeeID", "EmployeeID"),
        new System.Data.Common.DataColumnMapping("FirstName", "FirstName")}
    }
    );
    // .....
    //.....
}
```

It looks like the wizard did a lot of the work for you!

### ***Step 6: Filling the DataGrid Control with Data***

Until now, you didn't have to write a single line of code. Now, though, you'll add a few lines of code and then you'll be all set to see the data from your data source. First, you'll create a method, `FillDBGrid`, which fills a `DataSet` object. Then you'll read data from a `DataSet` object and populate the `DataGrid` control.

The `Fill` method of `SqlDataAdapter` fills data from a data adapter to the `DataSet`. You call `Fill` method in `FillDBGrid` method. Once you have a `DataSet` containing data, you can do anything with it including creating views for that data. (I discussed multiple views of a `DataSet` object in the previous chapter.) In this example, you set a `DataGrid` control's `DataSource` property to the `DataSet.DefaultViewManager`, which binds the `DataSet` object to the `DataGrid` control (see Listing 4-3).

#### *Listing 4-3. FillDBGrid method*

```
private void FillDBGrid()
{
    DataSet ds = new DataSet();
    sqlDataAdapter1.Fill(ds);

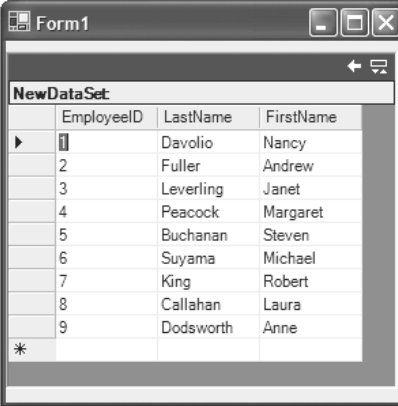
    dataGrid1.DataSource = ds.DefaultViewManager;
}
```

Now you simply call `FillDBGrid` from the `Form1` constructor or the `Form_Load` event or from a button-click handler. In this example I'll call it from the form constructor just after the `InitializeComponent()` call, as you can see in Listing 4-4.

#### *Listing 4-4. Calling the FillDBGrid method from the Form1 constructor*

```
public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();
    FillDBGrid();
    //
    // TODO: Add any constructor code after InitializeComponent call
    //
}
```

Now build and run the project. The result looks like Figure 4-34. Easy, huh?



	EmployeeID	LastName	FirstName
▶	1	Davolio	Nancy
	2	Fuller	Andrew
	3	Leverling	Janet
	4	Peacock	Margaret
	5	Buchanan	Steven
	6	Suyama	Michael
	7	King	Robert
	8	Callahan	Laura
	9	Dodsworth	Anne
*			

Figure 4-34. Output of the Employee data to a DataGrid control

## Working with OleDb Data Adapters

In the previous section, I discussed Sql data adapters. Now, let's take a quick look at OleDb data adapters. Actually, all data adapters (Sql, OleDb, and ODBC) work exactly the same way. I'll take you through a quick step-by-step tutorial on how to use OleDb data adapters. To give you more of a variety, you're going to use OleDb with an Access 2000 database.

As you already know, the first step in working with ADO.NET is to add a new connection using the Server Explorer. For the purposes of consistency, I've used the Northwind Microsoft Access 2000 database for these examples. Feel free, however, to use any data source that has an OLE DB provider available on your machine.

In the Data Link Properties dialog box, choose the Microsoft Jet 4.0 OLD DB Provider (see Figure 4-35).



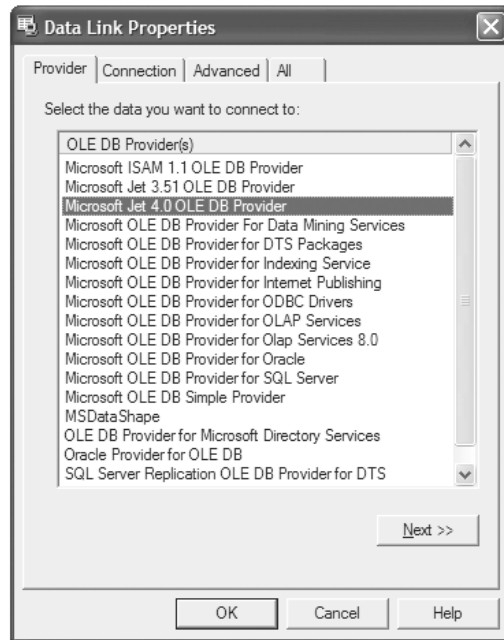


Figure 4-35. Choosing the OLE DB driver for Access

And the database is C:\Northwind.mdb, as you can see in Figure 4-36.

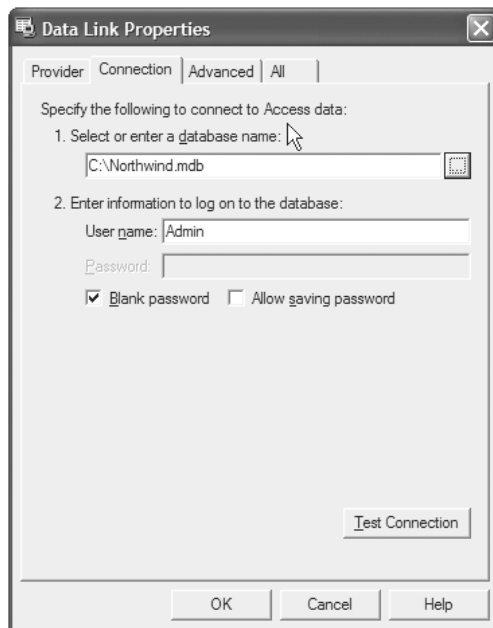


Figure 4-36. Choosing the database in Server Explorer

## Adding an OleDbDataAdapter

Working with either an OleDbDataAdapter or an ODBCDataAdapter is the same as working with the SqlDataAdapter. You can use either the Server Explorer or the Data Adapter Configuration Wizard to create an OleDb data adapter. In this example, I'll use the Data Adapter Configuration Wizard. Drop an OleDbDataAdapter control from Toolbox > Data to your application form. This action will bring up the Data Adapter Configuration Wizard.

On the second page of the wizard, Choose Your Data Connection, you can either create a new connection or pick an existing connection (see Figure 4-37).



Figure 4-37. Configuring an OleDb data adapter for Access

On the next page, select the Use SQL Statement option and click the Next button (see Figure 4-38).



Figure 4-38. Choosing the query type in the Data Adapter Configuration Wizard

This will bring you to the Add Table selection page. As you can see from Figure 4-39, I'm picking the Orders table. Then, click the Add button.

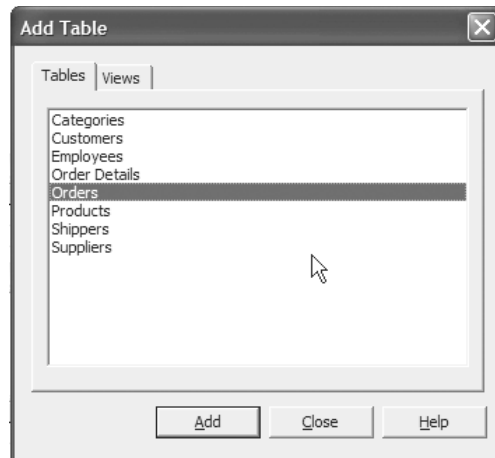


Figure 4-39. Adding a table to the query in the Data Adapter Configuration Wizard

After clicking Add, the Query Builder brings up a table column selector, as shown in Figure 4.40.

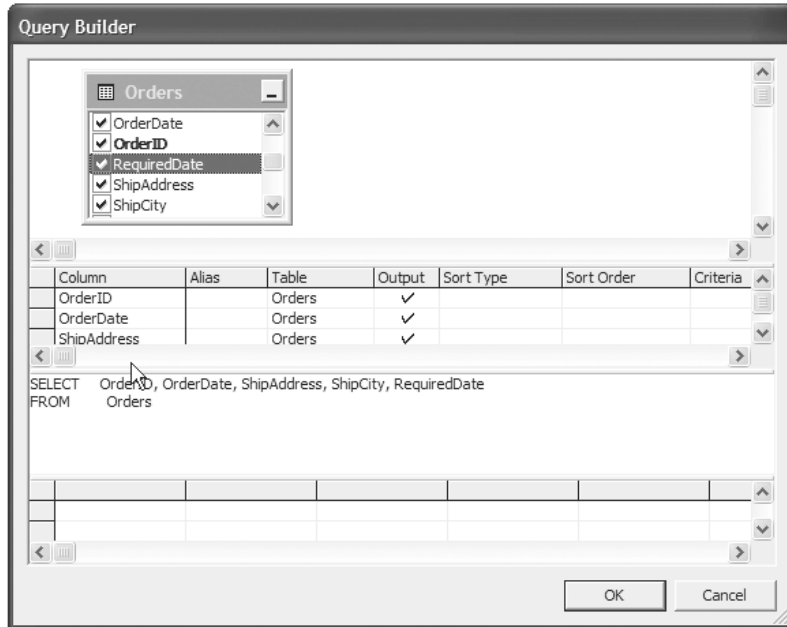


Figure 4-40. Choosing columns for the query in the Data Adapter Configuration Wizard

I chose OrderID, OrderDate, ShipAddress, ShipCity, and RequiredDate for my query by checking the columns in the Orders window. This builds the query shown in the third pane of the Query Builder. Clicking OK displays the final query, as shown in Figure 4-41.

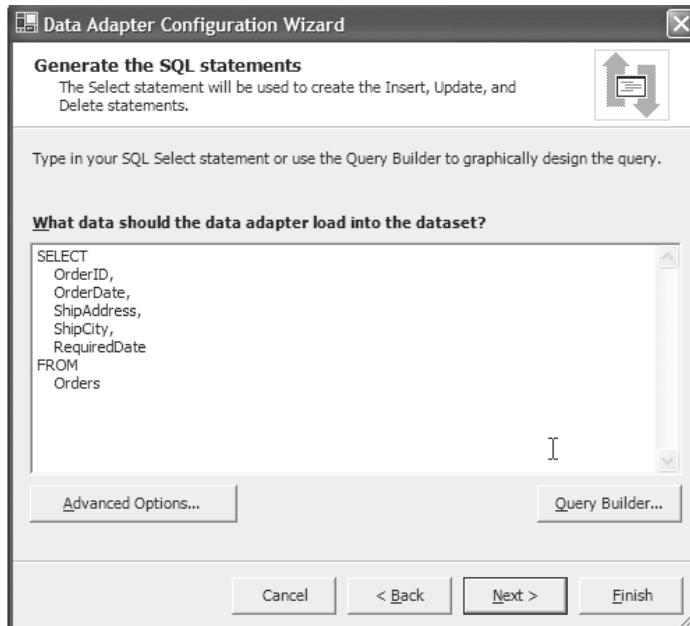


Figure 4-41. Generating the SQL statements in the Data Adapter Configuration Wizard

Clicking on the Advanced Options button brings up the Advanced SQL Generation Options dialog box, as shown in Figure 4-42.

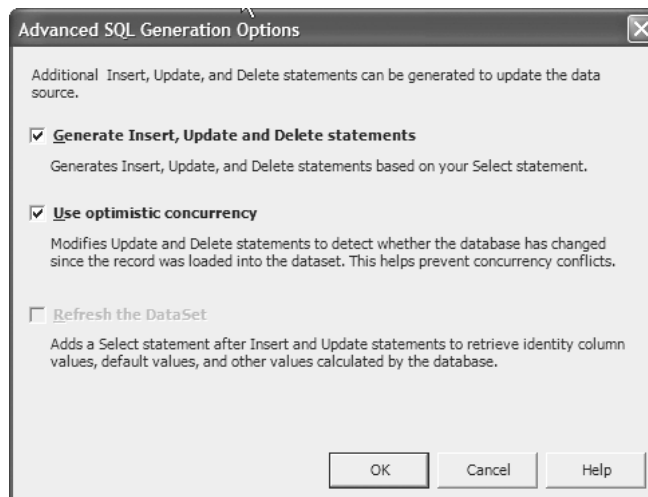


Figure 4-42. Advanced options in the Data Adapter Configuration Wizard

In this dialog box you can opt not to generate INSERT, UPDATE, or DELETE statements by turning off the first option. This is useful if you're planning on only reading the database and don't want all this extraneous code generated.

The second option, Use Optimistic Concurrency, causes the wizard to use optimistic concurrency. Optimistic concurrency checks to see if the row being updated in the database has already been changed by someone else during the update process. The data provider manages this by using a WHERE clause in the UPDATE statement that checks for the original data in the dataset. If it doesn't find the original data, it won't update the data source. A data provider maintains two sets of parameters: one with the original data and one with the current data. The current data parameters work in the UPDATE statement (this is the data you're trying to update the database with), and the original data parameters work in the WHERE clause (these parameters are the check to make sure the database hasn't been updated). If you turn off the Use Optimistic Concurrency option, the WHERE clause only contains the primary key and no original parameter data is generated. You can probably turn this off to speed things up if the application is only for a single user. Below are the differences between the Select statements generated with optimistic concurrency on and off.

This is the code with optimistic concurrency turned off:

```
dateCommand1.CommandText = @"UPDATE Orders SET OrderDate = ?, "+
"RequiredDate = ?, ShipAddress = ?, ShipCity = ? WHERE (OrderID = ?)" +
"AND (OrderDate = ? OR ? IS NULL AND OrderDate IS NULL) AND "+
"(RequiredDate = ? OR ? IS NULL AND RequiredDate IS NULL) AND "+
"(ShipAddress = ? OR ? IS NULL AND ShipAddress IS NULL) AND "+
"(ShipCity = ? OR ? IS NULL AND ShipCity IS NULL)";
```

This is the code with optimistic concurrency on:

```
this.oleDbUpdateCommand1.CommandText = @"UPDATE Orders SET OrderID = ?, "+
"OrderDate = ?, RequiredDate = ?, ShipAddress = ?, ShipCity = ?" +
"WHERE (OrderID = ?) AND (OrderDate = ?) AND (RequiredDate = ?)" +
"AND (ShipAddress = ?) AND (ShipCity = ?)" ;

"SELECT OrderID, OrderDate, RequiredDate, ShipAddress, "+
"ShipCity FROM Orders WHERE (OrderID = ?)";
```

You may also notice the SQL Select statement tacked onto the end of the SQL UPDATE statement. The Refresh the DataSet option adds this statement. Turning this option off will remove the Select statement. You had to uncheck this for the OleDb adapter or else Insert and Update don't work. This isn't true, however, for the SqlServer adapter.

Clicking Next brings up the results screen. As you can see in Figure 4-43, the Data Adapter Configuration Wizard has done quite a bit of work! It's generated all of the commands for the adapter, all of the mappings, and, although not indicated, all of the parameters.

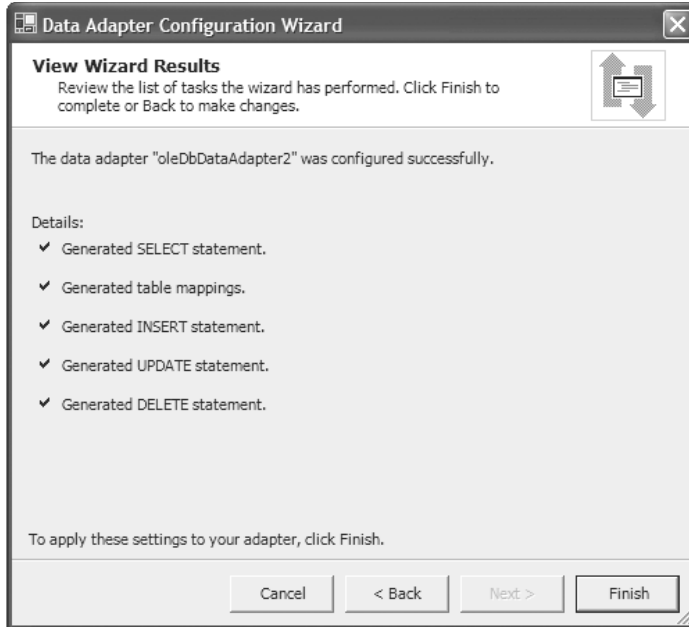


Figure 4-43. View Wizard Results page of the Data Adapter Configuration Wizard

If you examine the form designer, you'll see the wizard added two components to your form: `oleDbConnection1` and `oleDbDataAdapter1`. The source code generated by the wizard is similar to the source generated for the `SqlDataAdapter`. You'll notice differences, though, in the `ConnectionString` and the parameters if you were to go through the same process with a `SqlDataAdapter`. The `OdbcDataAdapter` will also generate similar code.

### *Populating DataSet and Filling the DataGrid*

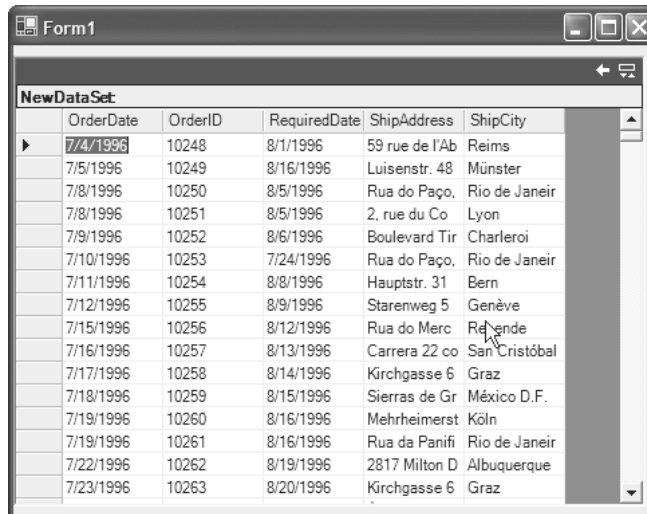
Now, to test whether everything went fine, create a Windows Forms application and add an `OleDbDataAdapter` using the previous steps. Then, add a `DataGrid` control to the form, as well as all the code listed in Listing 4-5 on the `Form_Load` event or a button-click handler.

*Listing 4-5. Adding the code on the Form\_Load event*

```
private void Form1_Load(object sender, System.EventArgs e)
{
    DataSet ds = new DataSet();
    // Populate DataSet by calling Fill method
    OleDbDataAdapter1.Fill(ds);
    // Set DataGrid's DataSource property
    dataGrid1.DataSource = ds.DefaultViewManager;
}
```

If you remember the SqlDataAdapter example, you know that it contained almost the same code. As you can see from Listing 4-5, you create a DataSet object and call OleDbDataAdapter's Fill method to fill data from the data adapter to the dataset. After that you use the DataGrid control's DataSource property and set it as DataSet's DefaultViewManager.

Now build and run the project. Your output should look like Figure 4-44.



OrderDate	OrderID	RequiredDate	ShipAddress	ShipCity
7/4/1996	10248	8/1/1996	59 rue de l'Ab	Reims
7/5/1996	10249	8/16/1996	Luisenstr. 48	Münster
7/8/1996	10250	8/5/1996	Rua do Paço.	Rio de Janeiro
7/8/1996	10251	8/5/1996	2, rue du Co	Lyon
7/9/1996	10252	8/6/1996	Boulevard Tir	Charleroi
7/10/1996	10253	7/24/1996	Rua do Paço.	Rio de Janeiro
7/11/1996	10254	8/8/1996	Hauptstr. 31	Bern
7/12/1996	10255	8/9/1996	Starenweg 5	Genève
7/15/1996	10256	8/12/1996	Rua do Merc	Reims
7/16/1996	10257	8/13/1996	Carrera 22 co	San Cristóbal
7/17/1996	10258	8/14/1996	Kirchgasse 6	Graz
7/18/1996	10259	8/15/1996	Sierras de Gr	México D.F.
7/19/1996	10260	8/16/1996	Mehrheimerst	Köln
7/19/1996	10261	8/16/1996	Rua da Panifi	Rio de Janeiro
7/22/1996	10262	8/19/1996	2817 Milton D	Albuquerque
7/23/1996	10263	8/20/1996	Kirchgasse 6	Graz

*Figure 4-44. Filling a DataGrid with the Orders table*



## Using DataSet and DataView Components

After discussing data adapters and data connections, you got a pretty good idea of how to take advantage of VS .NET design-time support to develop data-bound Windows Form database applications.

The DataSet and DataView components are two powerful and easy-to-use components of the ADO.NET model. In this section, you'll see how to utilize DataSet and DataView components at design-time. In Chapter 5, I'll discuss their properties and methods in more detail and show how to use them programmatically. The DataSet and DataView components fall in the *disconnected* components category, which means you can use these components with or without data providers. I'll discuss connected and disconnected data components in Chapter 5 in more detail. These components work in the same way for all data providers, including Sql, OleDb, and Odbc.

### *Understanding Typed DataSets in Visual Studio .NET*

There are two types of datasets: typed datasets and untyped datasets. As discussed in Chapter 3 (and in more detail in Chapter 5), a typed dataset has an XML schema attached to it. The XML schema defines members for a dataset corresponding to database table columns, and you can access data through these columns. Untyped datasets are ones that are created at run-time and don't have an schema attached to them. I'll now show you how you can generate typed datasets using a VS .NET wizard.

### *Generating Typed DataSets Using Data Adapters*

You can generate typed datasets by using any of the data adapters. You can either generate a dataset by right-clicking on a data adapter and selecting the Generate Dataset menu option or by using the data adapter Properties windows. To generate a dataset from data adapter's Properties window, choose the Generate Dataset hyperlink, which generates a DataSet object, and the wizard writes the code for you (see Figure 4-45).

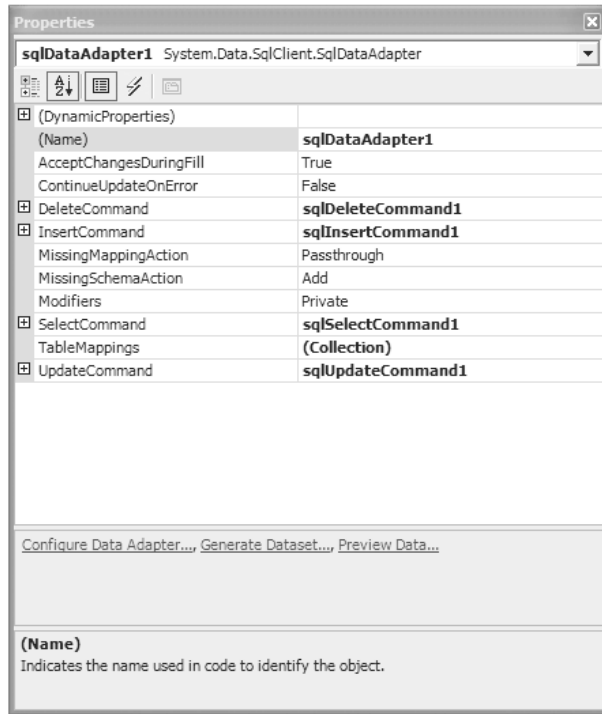


Figure 4-45. Generating a typed dataset from the Properties window

This action pops up a dialog box, which generates a dataset. Type your dataset name and click OK (see Figure 4-46).



*Figure 4-46. Dialog box for generating a dataset*

This action adds a dataset (if you check Add This Dataset to the Designer check box) and pops up the dataset Properties dialog box (see Figure 4-47).

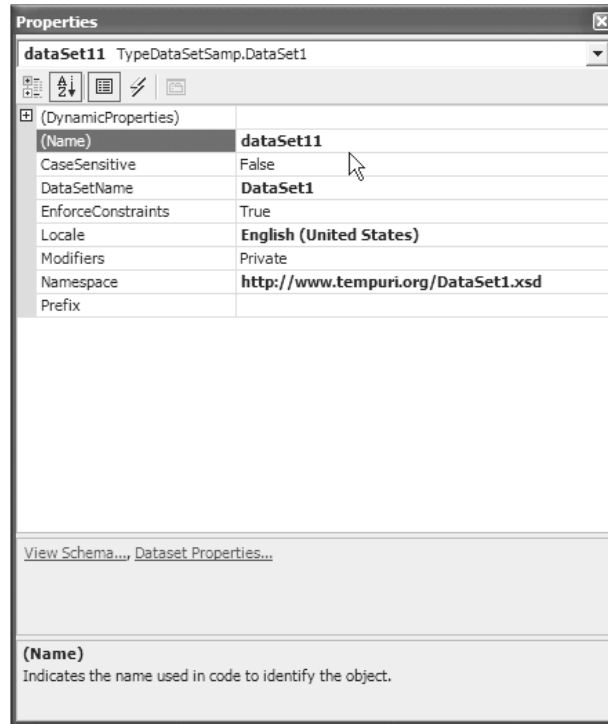


Figure 4-47. A dataset's Properties window showing a typed dataset

Every dataset generated by the IDE creates an XML schema for the dataset. Figure 4-47 provides you with two hyperlinks at the bottom of the dialog: View Schema and DataSet Properties. View Schema lets you view the DataSet schema, and the DataSet Properties hyperlink lets you set the DataSet properties. By following these links you can set the DataSet's column names and other properties (see Figure 4-48).

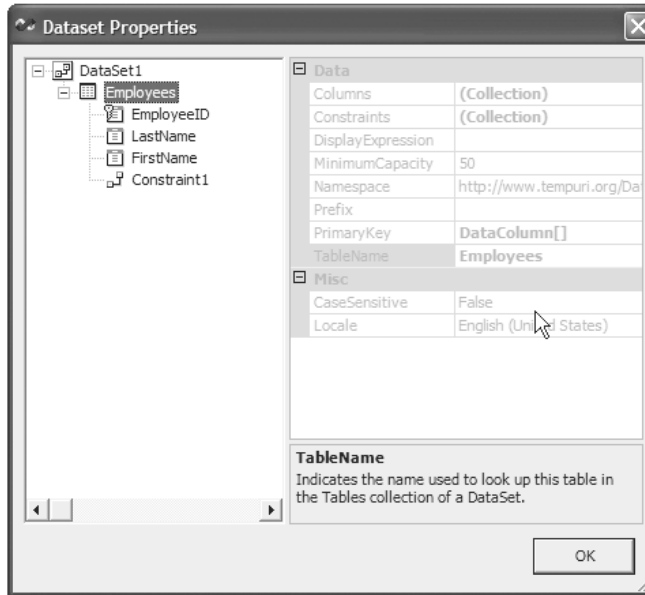


Figure 4-48. Setting DataSet names and additional properties

This action also adds one class inherited from a DataSet and one XML schema (DataSet1.xsd). The Class View of the DataSet is a derived class and looks like Figure 4-49.

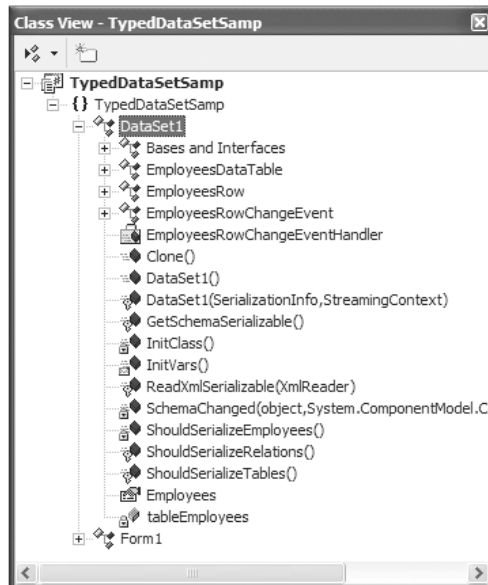


Figure 4-49. A VS .NET-generated typed DataSet class

You can now create an instance of this class instead of creating a DataSet programmatically. This class has a member corresponding to each column of the table to which it's attached:

```
MyDataSet ds = new MyDataSet();
```

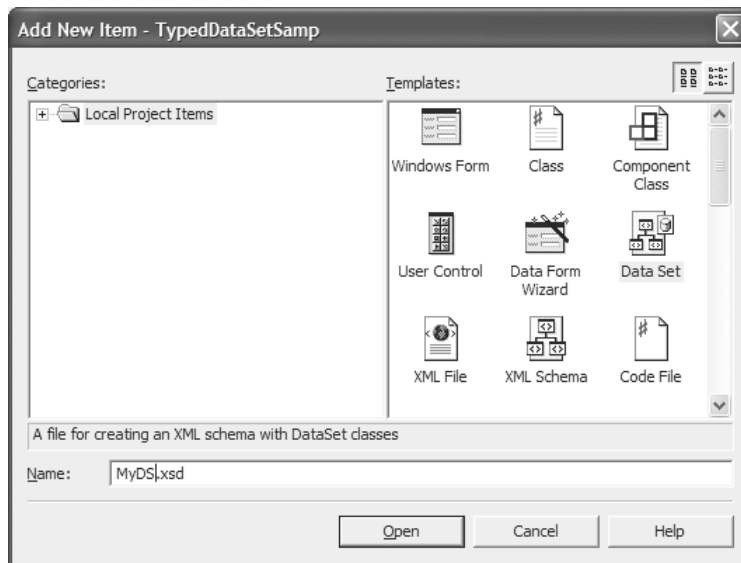
The beauty of typed datasets is that you can access the data in the columns using MyDataSet object members.

Besides creating a DataSet using the Data Adapter Configuration Wizard, there is another good way to do so. I'll discuss this alternate solution in the following section.

### *Adding Typed DataSets*

In the previous discussion, you saw how you can generate DataSet objects from a data adapter. There are other ways to create a typed DataSet object.

You can click on the Project menu and choose Add New Item (or click Ctrl+D). This brings up the Add New Item window where you'll find the Data Set template (see Figure 4-50).



*Figure 4-50. Creating a typed DataSet from the Add New Item window*

After adding the DataSet, the designer creates an XSD (XML schema) file and adds it to your project area. As you can see from Figure 4-51, myDS.xsd is empty.

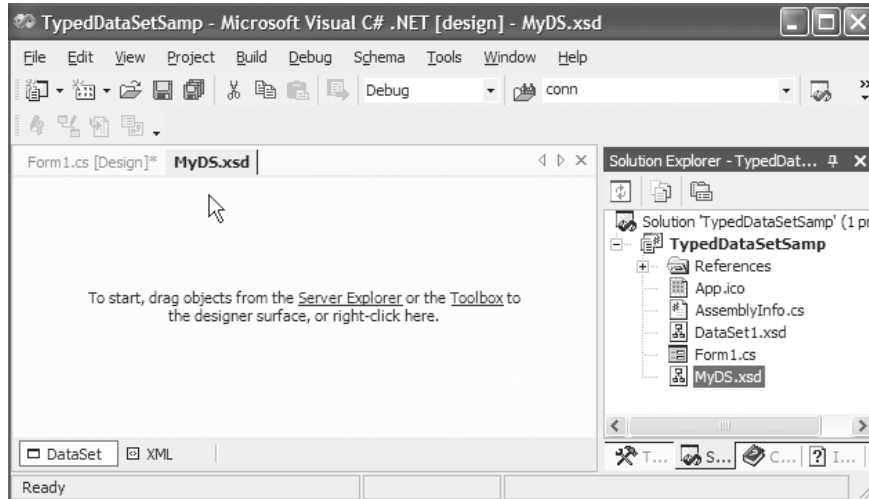


Figure 4-51. myDS.xsd in VS .NET

Next, drop a table (or multiple tables) from the Server Explorer to the form (see Figure 4-52).

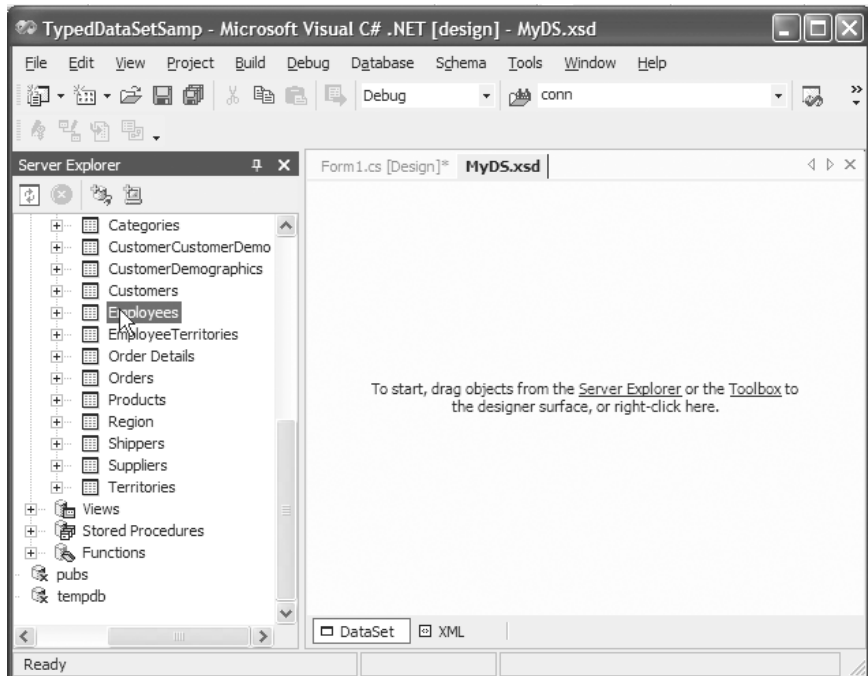


Figure 4-52. Drag and drop tables from the Server Explorer to the form to create a typed DataSet

This action adds one XML schema (MyDS.xsd), which looks like Figure 4-53.

E Employees (Employees)	
E EmployeeID	int
E LastName	string
E FirstName	string
E Title	string
E TitleOfCourtesy	string
E BirthDate	dateTime
E HireDate	dateTime
E Address	string
E City	string
E Region	string
E PostalCode	string
E Country	string
E HomePhone	string
E Extension	string
E Photo	base64Binary
E Notes	string
E ReportsTo	int
E PhotoPath	string
*	

Figure 4-53. Design View of the XML schema of the DataSet



It also automatically adds the typed DataSet class that inherits from DataSet. As you can see in Figure 4-54, the myDS class contains members used to access data from the database.

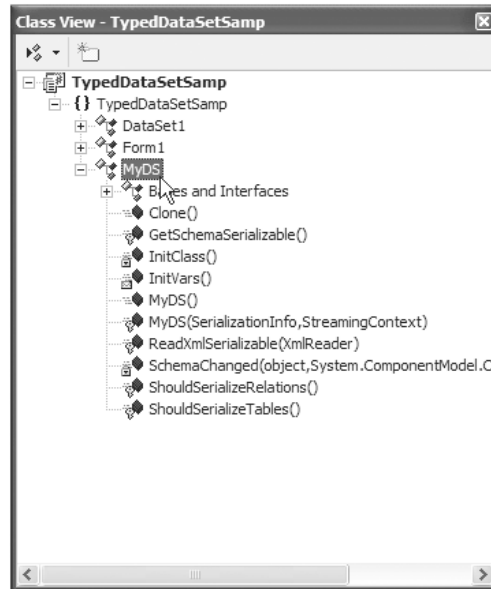


Figure 4-54. Wrapper class generated for the typed DataSet

Once you have this class, you can create an instance of this class and work with its property fields directly:

```
MyDSet ds = new MyDSet();
```

**NOTE** See Chapter 5 for a more extensive example on using datasets.

## Understanding DataView

A DataView represents a view of a DataSet object. You can set filters on the data or sort on data in the DataSet through different DataViews and produce different views of the data. For example, you can create a DataSet with three tables and create three different DataView objects for each table. Once you have a DataView object, you can attach it with any data-bound control, such as a DataGrid or a ComboBox control using data-bound control's DataSource property.

To create a DataView at design-time, drag the DataView from Toolbox > Data onto your form. Then create a DataSet object and set the DataView's Table property to a table in the typed DataSet (see Figure 4-55).

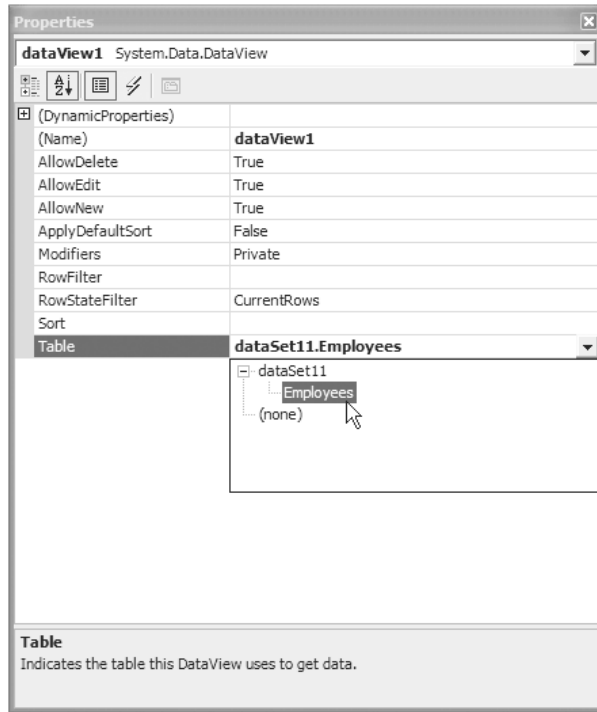


Figure 4-55. DataView Properties window

## Using the Data Form Wizard

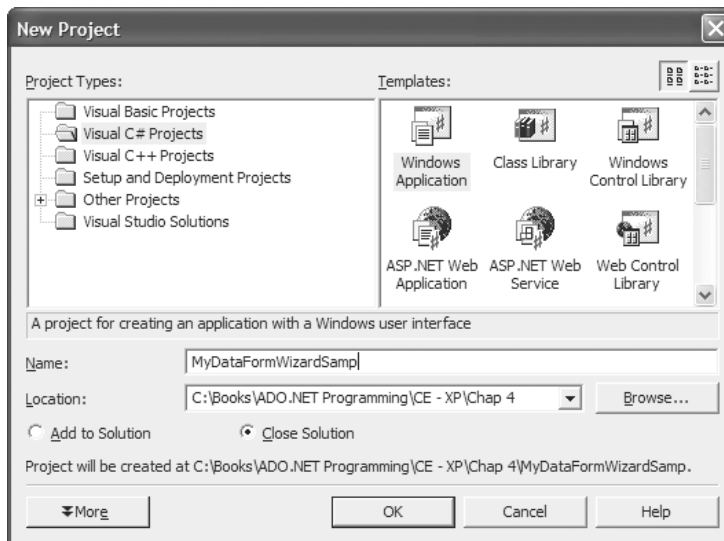
At the end of this chapter, I'd like to discuss Data Form Wizard, one more useful tool to develop database applications. You can use the Data Form Wizard to develop your database application with viewing, updating, and deleting capabilities. This is probably the fastest way to develop database applications in .NET (unless you're an extremely fast typist).

In this section, you'll use a Data Form Wizard to write a fully functioning database application including features such as inserting, updating, and deleting data without writing a single line of code. In this simple example, I've used the familiar Northwind database. I'll use both the Customers and Orders tables to show you a data relationship between table data.

Like many parts of this book, this topic is in the form of tutorial. Just follow the simple steps, and in a few minutes you'll be able to run a wonderful application. In this section, you're going to create a Windows application. After that you'll add a Data Form Wizard to it and call the Data Form Wizard from the main application.

### *Step 1: Selecting a Project Template*

Create a new Windows project by selecting **New Project > Visual C# Projects > Windows Application** and typing your application name (see Figure 4-56).



*Figure 4-56. Creating a Windows Application project*

### *Step 2: Adding a Data Form Wizard Item*

Now add a Data Form Wizard by selecting **Project > Add New Item > Data Form Wizard** from the available templates. You can type the name of your DataForm class in the Name field of the dialog box (see Figure 4-57).

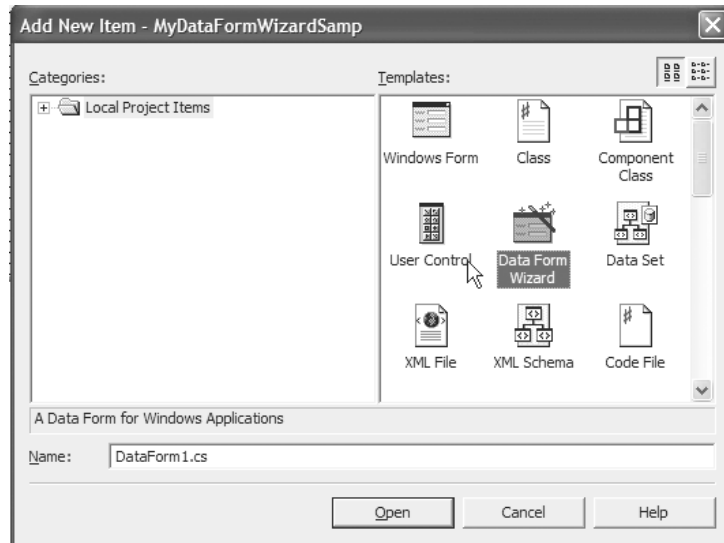


Figure 4-57. Using the Data Form Wizard

Now click Open, which calls the Data Form Wizard.

### Step 3: Walking through the Data Form Wizard

The first page of the wizard is a welcome page telling you what the wizard is about to do (see Figure 4-58).



Figure 4-58. Welcome page of the Data Form Wizard

### Step 4: Choosing the Dataset You Want

On the second page of the wizard, you can choose a dataset name that will later be used to access the data. You can either create a new dataset name or select an existing one. In this example, I'll choose MyDS as the dataset name (see in Figure 4-59).

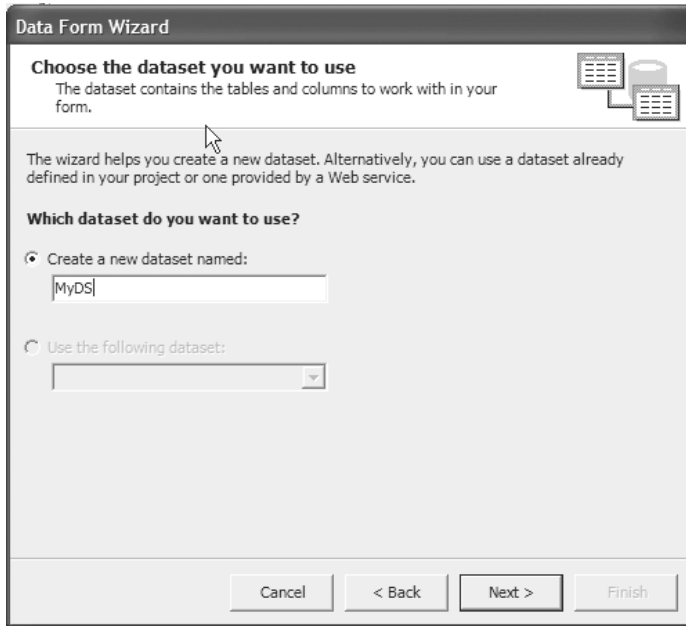


Figure 4-59. Choosing a DataSet in the Data Form Wizard

### Step 5: Choosing a Data Connection

The next page of the wizard asks you to provide a connection. The combo box displays your available connection. If you didn't create a connection, use the New Connection button, which launches the Server Explorer discussed earlier in this chapter. I'll select the usual database, Northwind (see Figure 4-60).

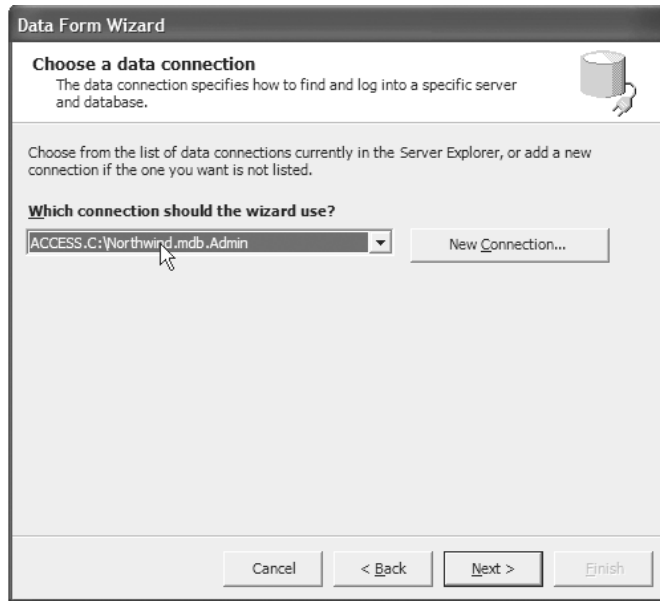


Figure 4-60. Choosing a data connection in the Data Form Wizard

### Step 6: Choosing Tables or Views

The next page of the wizard lets you pick the tables and views you want to connect to the dataset. As you can see in Figure 4-61, I select the Customers and Orders tables in the Available Items list on this page and use the > button to add these tables to the Selected Items list.

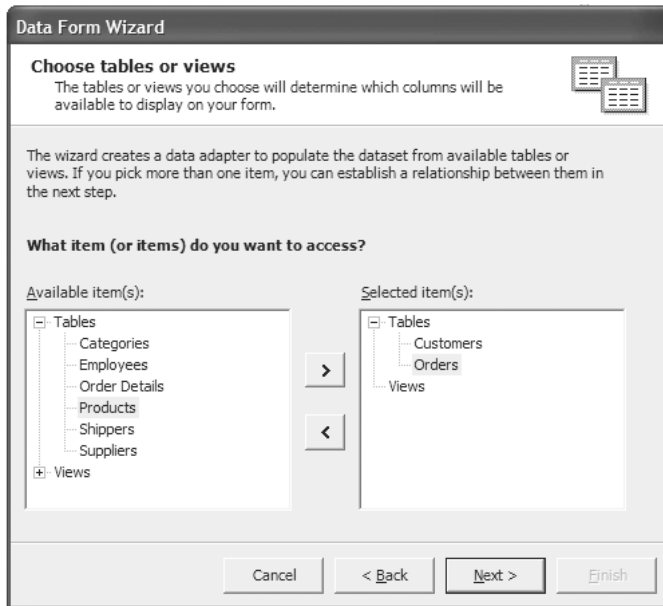


Figure 4-61. Choosing a DataTable or DataView in the Data Form Wizard

Now you're ready to create a relationship between these two tables.

### Step 7: Creating a Relationship between Tables

The next page lets you define a relationship between the Customers and Orders tables. It's useful to provide a relationship between tables when you have a master-detail relationship database. In other words, a customer may have many orders associated with it, so there is a relationship through the CustomerID in the Orders table joined to information about the customer in the Customers table. Now, say you want to see all the orders of a customer based on the CustomerID. If you do this manually, you need to write code to select data from the Orders table to correspond to a CustomerID and then fill data to the form. If you use Data Form Wizard instead, it does everything for you. Neat, huh?

This is the same step you're going to see on the Create a Relationship between Tables page of the wizard. You're going to create a relationship between the Customers and Orders tables based on the CustomerID. I named the relationship between Customers and Orders table CustOrderRelation. You also need to pick the associated primary key and foreign key that links the parent to the child table. Once you've chosen the joining key (CustomerID), you have to click the > button to tell the wizard that you want to add it.

When you run the final program, you'll see how you can filter all orders for a customer based on the CustomerID. As you can see from Figure 4-62, you need to pick one table as parent and another table as a child based on the relationship between them. In this example, the Customers table is the parent table, and the Orders table is the child table.

**Data Form Wizard**

**Create a relationship between tables**  
The wizard will use the relationships to generate code that keeps the tables synchronized as you work with them.

Relationships are based on common keys between tables. Name your new relation, choose the parent and child tables and key fields, and then add it to the relations list using the arrow button.

Name: CustOrderRelation

Parent table: Customers

Child table: Orders

Keys: CustomerID, CustomerID

Relations:

Cancel < Back Next > Finish

*Figure 4-62. Selecting Customers as the parent and Orders as the child table to create the CustOrderRelation relationship*

After adding the relationship to the Relations list, the wizard looks like Figure 4-63.



**Data Form Wizard**

**Create a relationship between tables**  
 The wizard will use the relationships to generate code that keeps the tables synchronized as you work with them.

Relationships are based on common keys between tables. Name your new relation, choose the parent and child tables and key fields, and then add it to the relations list using the arrow button.

Name:

Parent table:  Child table:

Keys:

Relations:  
CustOrderRelation

Figure 4-63. CustOrderRelation listed in the Relations list

### Step 8: Choosing Tables and Columns to Display on the Form

The next page of the wizard lets you select which tables and columns you want to show on the form. For this example, select all the columns from both of the tables (this is the default selection). As you can see in Figure 4-64, the Customers table is the master, and the Orders table is the detail table.

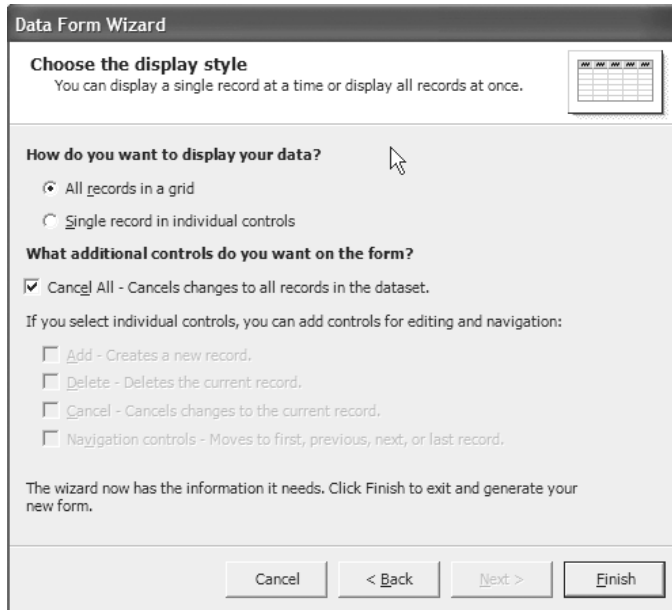
Figure 4-64. Choosing columns to display on the Data Form Wizard

### Step 9: Choosing the Display Style

This page is an important part of creating your form. Actually, the Data Form Wizard adds a Windows form with some controls on it and writes code to fill, update, delete, and navigate data. There are two ways to view the data, and you choose your option on this page. These two options are:

- All Records in a Grid
- Single Record in Individual Controls

Figure 4-65 displays these options.



*Figure 4-65. Choosing between a grid and individual controls on the Data Form Wizard*

The output of All Records in a Grid looks like Figure 4-66. After that you can resize controls on the form.

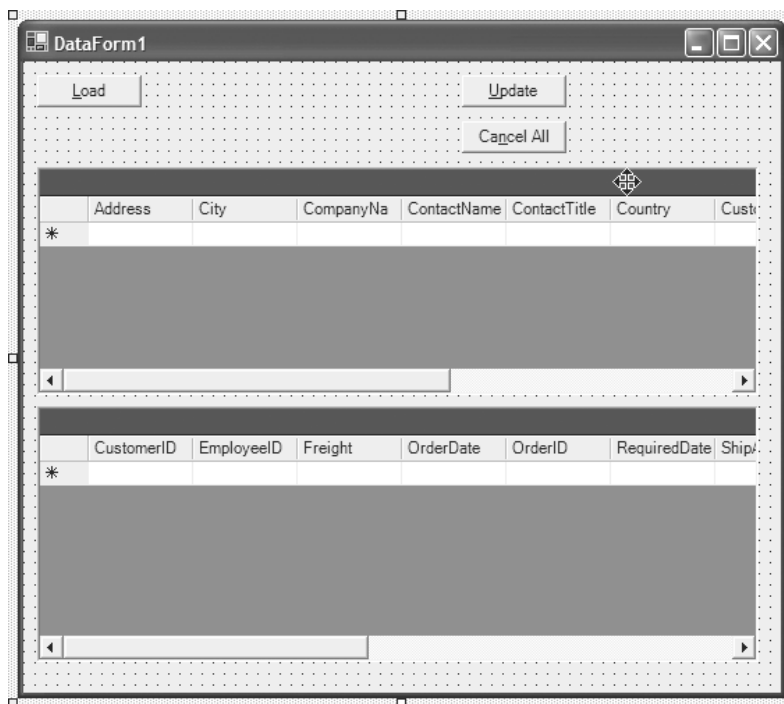


Figure 4-66. Grid DataForm output

The second option, Single Record in Individual Controls, shows data in text boxes and provides you with navigation controls. As you can see from Figure 4-67, the Single Record in Individual Controls option activates Add, Delete, Cancel, and Navigation controls check boxes. You can uncheck the check boxes if you don't want to add that feature in your project.

**Data Form Wizard**

**Choose the display style**  
You can display a single record at a time or display all records at once.

**How do you want to display your data?**

All records in a grid

Single record in individual controls

**What additional controls do you want on the form?**

**Cancel All** - Cancels changes to all records in the dataset.

If you select individual controls, you can add controls for editing and navigation:

**Add** - Creates a new record.

**Delete** - Deletes the current record.

**Cancel** - Cancels changes to the current record.

**Navigation controls** - Moves to first, previous, next, or last record.

The wizard now has the information it needs. Click Finish to exit and generate your new form.

Cancel < Back Next > Finish

*Figure 4-67. The Single Record in Individual Controls option*

The form generated by this option looks like Figure 4-68. As you can see from Figure 4-68, each column of the table has a field on the form.

*Figure 4-68. Data Form Wizard–generated form for the Single Record in Individual Control option*

After your selection of data display style, you click Finish button. The Data Form Wizard adds the Windows form `DataForm1` and the class `DataForm1.cs` corresponding to it.

### *Step 10: Calling the Data Form Wizard Form from the Application*

Now you need to change one more thing. You need to call `DataForm1` when you start your application. By default, your application calls the `Form1` form on start up.

```
static void Main()
{
    Application.Run(new Form1());
}
```

So, you need to replace Form1 with your Data Form Wizard's form name. In this example, Listing 4-6 replaces Form1 with DataForm1 in the Main method.

*Listing 4-6. Calling DataForm1 from the application*

```
static void Main()
{
    Application.Run(new DataForm1());
}
```

**NOTE** *If you've modified the name of your Data Form Wizard-generated form, you need to call that form instead of DataForm1.*

### *Step 11: Viewing the Output*

Now you should see the output shown in Figure 4-69 when you run your application (if you selected the grid view option).

The Load and Update buttons load and update the data, respectively, and Cancel All cancels all the operations. The neat thing is if you move into the top grid, corresponding information changes in the bottom grid. Neat, huh?

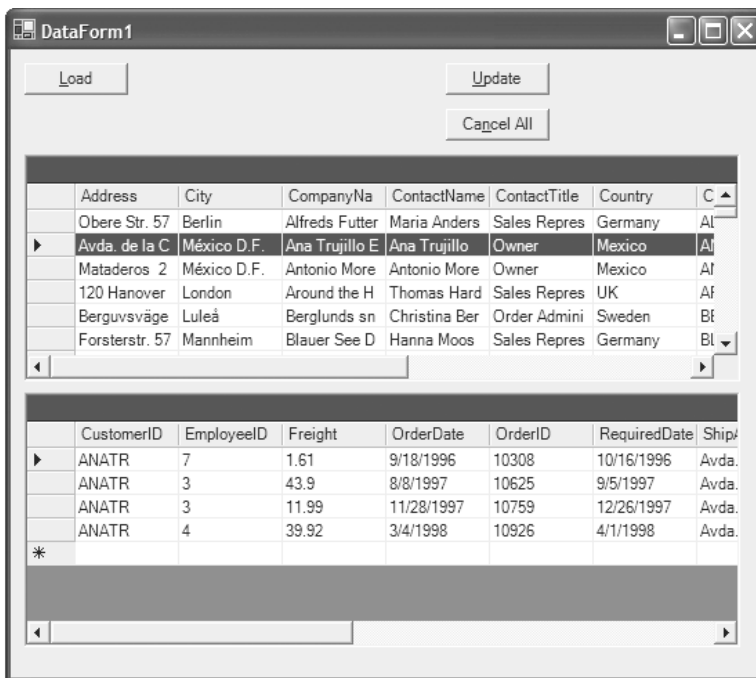


Figure 4-69. Data Form Wizard with all records in a grid option

Figure 4-70 shows the output when you select the Single Record in Individual Control option. By using this view option, you can add, edit, delete, and navigate records easily.



The screenshot shows a window titled "DataForm1" with a standard Windows interface (minimize, maximize, close buttons). The window contains a form with the following fields and values:

- Address: Obere Str. 57
- City: Berlin
- CompanyName: Alfreds Futterkiste
- ContactName: Maria Anders
- ContactTitle: Sales Representati
- Country: Germany
- CustomerID: ALFKI
- Fax: 030-0076545
- Phone: 030-0074321
- PostalCode: 12209
- Region: (empty)

Navigation and action buttons include: Load, Update, Cancel All, Add, Delete, Cancel, and a status bar showing "1 of 93".

	CustomerID	EmployeeID	Freight	OrderDate	OrderID	Re
▶	ALFKI	6	29.46	8/25/1997	10643	9/2
	ALFKI	4	61.02	10/3/1997	10692	10/
	ALFKI	4	23.94	10/13/1997	10702	11/
	ALFKI	1	69.53	1/15/1998	10835	2/1
	ALFKI	1	40.42	3/16/1998	10952	4/2
	ALFKI	3	1.21	4/9/1998	11011	5/7
*						

Figure 4-70. Textbox output with navigational controls

Finally, compile and run your application. Without writing a single line of code, you just created a fully functional database application.

The Load button on the individual control form loads the data, and the Add, Update, and Delete buttons on the form inserts, updates, and deletes records, respectively.

## Data Form Wizard: Looking under the Hood

You just saw how you can develop fully functional database applications in no time with the help of the Data Form Wizard. Now let's see what the wizard does for you in the actual code. (The inherent beauty of VS .NET is that it magically hides all the messy code for you.) The wizard adds two items to your project: MyDS.xsd and DataForm1.cs.

## Understanding MyDS.xsd

MyDS.xsd is an XML schema for the dataset you've added to the project. It's similar to the one discussed in the "Understanding Typed DataSets in Visual Studio .NET" section of this chapter.

## Understanding DataForm1.cs

The second item added by the wizard is the DataForm1 class, a class derived from System.Windows.Forms.Form. The DataForm1 class defines its entire functionality. The InitializeComponent method creates the data connection, the data command, the data adapter, the dataset, and other data components.

The LoadDataSet method loads the data from the data source into the controls by calling FillDataSet (see Listing 4-7).

*Listing 4-7. LoadDataSet method generated by the Data Form Wizard*

```
public void LoadDataSet()
{
    // Create a new dataset to hold the records
    //returned from the call to FillDataSet.
    // A temporary dataset is used because filling
    //the existing dataset would
    // require the databindings to be rebound.
    MyDataFormWizardSamp.MyDS objDataSetTemp;
    objDataSetTemp = new MyDataFormWizardSamp.MyDS();
    try
    {
        // Attempt to fill the temporary dataset.
        this.FillDataSet(objDataSetTemp);
    }
    catch (System.Exception eFillDataSet)
    {
        // Add your error handling code here.
        throw eFillDataSet;
    }
    try
    {
        // Empty the old records from the dataset.
        objMyDS.Clear();
        // Merge the records into the main dataset.
        objMyDS.Merge(objDataSetTemp);
    }
}
```

```

        catch (System.Exception eLoadMerge)
        {
            // Add your error handling code here.
            throw eLoadMerge;
        }
    }
}

```

FillDataSet fills the dataset from the data adapter by calling the Fill method on each data adapter. Note that with the Data Form Wizard, a DataAdapter is created for each table, one DataAdapter for the Customers table and one DataAdapter for the Orders table. Both DataAdapters fill the same DataSet. Listing 4-8 shows the FillDataSet method.

*Listing 4-8. The FillDataSet method generated by the Data Form Wizard*

```

public void FillDataSet(MyDataFormWizardSamp.MyDS dataSet)
{
    // Turn off constraint checking before the dataset is filled.
    // This allows the adapters to fill the dataset without concern
    // for dependencies between the tables.
    dataSet.EnforceConstraints = false;
    try
    {
        // Open the connection.
        this.oleDbConnection1.Open();
        // Attempt to fill the dataset through the OleDbDataAdapter1.
        this.oleDbDataAdapter1.Fill(dataSet);
        this.oleDbDataAdapter2.Fill(dataSet);
    }
    catch (System.Exception fillException)
    {
        // Add your error handling code here.
        throw fillException;
    }
    finally
    {
        // Turn constraint checking back on.
        dataSet.EnforceConstraints = true;
        // Close the connection whether or not the exception was thrown.
        this.oleDbConnection1.Close();
    }
}
}

```

The `UpdateDataSource` method updates the data source from the `DataSet`. The `UpdateDataSet` method calls `UpdateDataSource`, which utilizes the `Update` method of the data adapters. Listing 4-9 shows the `UpdateDataSource` method.

*Listing 4-9. The `UpdateDataSource` and `UpdateDataSet` methods generated by the Data Form Wizard*

```
public void UpdateDataSource(MyDataFormWizardSamp.MyDS ChangedRows)
{
    try
    {
        // The data source only needs to be updated if there
        //are changes pending.
        if ((ChangedRows != null))
        {
            // Open the connection.
            this.oleDbConnection1.Open();
            // Attempt to update the data source.
            oleDbDataAdapter1.Update(ChangedRows);
            oleDbDataAdapter2.Update(ChangedRows);
        }
    }
    catch (System.Exception updateException)
    {
        // Add your error handling code here.
        throw updateException;
    }
    finally
    {
        // Close the connection whether or not the exception
        //was thrown.
        this.oleDbConnection1.Close();
    }
}
```

## Summary

Congratulations! Now you have completed one more step toward understanding ADO.NET and its components. After completing this chapter, you should have a pretty good idea of how to write database applications using VS .NET.

In this chapter, you learned about visual data components in Visual Studio .NET. The Server Explorer is a handy utility added to VS .NET IDE to help you manage your database connections.

Data adapters let you connect to a data source a design-time and can be used to populate DataSet objects. Data adapters also allow you to add, update, and delete data through data command objects. VS .NET also lets you generate typed datasets, which create a DataSet with properties of tables and columns specific to a data source.

DataView is a bindable view of a DataSet. You can sort and filter a DataSet with a DataView and use it to bind to a graphical component in many of the Windows form controls.

Finally, the Data Form Wizard is a useful tool in which you can generate full-fledged database applications with features such as insert, delete, update in no time. In the next chapter, I'll discuss ADO.NET data providers and other ADO.NET components and show how to work with them programmatically. Chapter 5 will also cover data component's methods and properties.



# Index

## Symbols and Numbers

`///` (three slashes), using with comments in source code, 67  
`<!-- >` tag, description of, 357, 362  
- (unary) operators, using with for loop statements, 42  
& (ampersand), role XML-document references, 370  
, (comma), using with UPDATE statement in SQL, 676  
&& (conditional and) operator, using with if . . . else statement, 40–41  
" (double quote) entity in XML, representing, 370  
; (semicolon)  
    advisory about, 46  
    role in XML-document references, 370  
' (single quote) entity in XML, representing, 370  
# (hash symbol), role in XML-document character references, 370  
++ (unary) operators, using with for loop statements, 42  
<-- and --> pairs, role in XML comments, 369  
< and > characters in XML documents, advisory about, 369  
< (less than)  
    entity in XML, 370  
    operator in SQL, 673  
<> (not equal to) operator in SQL, 673  
= (equal to) operator in SQL, 673  
> (greater than)  
    entity in XML, 370  
    operator in SQL, 673  
@ (at) sign, appearance in SQL Server Insert commands, 339  
|| (conditional or) operators, using with if . . . else statements, 40–41  
0-3 isolation levels, details of, 663  
1-4 values for locking types, descriptions of, 664  
1NF (first normal form), explanation of, 654–656  
2NF (second normal form), explanation of, 656–658  
3NF (third normal form), explanation of, 658–659  
4NF (fourth normal form), explanation of, 659  
5NF (fifth normal form), explanation of, 659–660

## A

AcceptChanges method  
    of ADO.NET DataRow class, 252  
    of ADO.NET DataSet class, 276  
    of ADO.NET DataTable class, 259  
Access connection string for ODBC, displaying, 291  
Access tables, exporting to text files, 633–637  
accessibility modifiers  
    example of, 32  
    functionality of, 33–35  
Action property of  
    XmlNodeChangedEventArgs,  
        description of, 566  
AddCat1 stored procedure, code for, 588  
Added member of ADO.NET DataRowState enumeration, 257  
additive operators, examples of, 38  
AddNew method of ADO.NET DataView class, description of, 278  
AddNews property of ADO.NET DataView class, description of, 277  
AddRange method of Form class, code for, 89  
AddRow\_Click method, code for, 270  
ADO (ActiveX Data Objects)  
    versus ADO.NET, 128–129  
    recordset cursor types and values in, 661  
ADO recordsets, using in ADO.NET, 595–598  
ADODB namespace  
    accessing databases with, 597–598  
    adding references to, 603  
    including in projects, 596–597  
    viewing, 596  
ADOMD (ActiveX Data Objects Multi-Dimensional Library)  
    adding references to, 603  
    functionality of, 601  
    testing, 603  
ADOMD namespaces, adding to projects, 602  
ADO.NET (ActiveX Data Objects .NET)  
    accessing OLAP server data with, 600–611  
    adding database support to Web services with, 516–517  
    adding parameters to stored procedures with, 585–587  
    versus ADO, 128–129  
    advantages of, 126–128  
    and COM interoperability, 594–595  
    CommandBuilder utility in, 135–136  
    concurrency control in, 343–347  
    creating Command objects for, 140–141

- creating DataAdapter objects for, 140–141
  - creating OleDb Command objects for, 301–303
  - creating stored procedures with, 573–580, 576–577
  - DataViewManager class, 278–279
  - deployment of, 127
  - editing, deleting, and executing stored procedures with, 575–578
  - executing and reading results of stored procedures in, 307–308
  - executing stored procedures with programmatically, 583–589
  - explanation of, 123–124
  - filling data to DataSets or DataReader objects in, 141
  - getting database schemas from, 607–610
  - introduction to, 123
  - and ODBC, 125–126
  - and OLE-DB, 125–126
  - overview of namespaces and classes in, 129–132
  - performance and scalability of, 128
  - pessimistic concurrency in, 346–347
  - purpose of, 125–126
  - and relational databases, 660–661
  - returning data from stored procedures with, 578
  - returning values from stored procedures with, 586
  - role of Command components in, 230–231
  - role of Command object in, 133–135
  - role of Command objects in, 300–301
  - role of Connection components in, 230–231
  - role of Connection object in, 133–134
  - role of DataAdapter components in, 230–231
  - role of DataSet class in, 273
  - role of DataSet components in, 231–232
  - role of DataSets in, 124, 274–277
  - role of DataTable components in, 232
  - role of DataView class in, 273
  - role of DataViews in, 277–278
  - role of managed class in, 127
  - role of managed code in, 127
  - role of parameters in, 337–339
  - role of XML in, 124
  - saving SELECT statements as stored procedures with, 578–579
  - selecting records with, 610–611
  - support for XML, 127
  - System.Data namespace class, 237–238
  - transactions in, 342–343
  - typed and untyped DataSets in, 276–277
  - using ADO recordsets in, 595–598
  - using ADOX with, 598–600
  - using DataReaders with, 313–314
  - using output parameters and stored procedures with, 588
  - using views with, 588–594
  - viewing stored procedures with, 574
  - visual data components of, 127
- ADO.NET applications
- adding namespace references to, 139–140
  - choosing .NET data providers for, 138–139
  - closing connections to, 142
  - constructing SQL Server Connection objects for, 148
  - creating in VS .NET, 153–154
  - displaying data in, 141
  - establishing connections in, 140
  - writing, 138–145
  - writing with Visual Studio .NET IDE, 145–151
- ADO.NET architecture, examining, 230–232
- ADO.NET class hierarchy, exploring, 232–237
- ADO.NET components, understanding, 132–138
- ADO.NET data provider namespace references, adding to projects, 283
- ADO.NET data providers
- choosing, 282–283
  - connecting to databases, 283–285
  - explanation of, 229
  - introduction to, 279–283
  - opening and closing connections for, 285–289
- ADO.NET disconnected classes
- DataRelation, 257–258
  - DataRows, 251–256
  - DataRowState, 256–257
  - DataTable, DataColumn, and DataRow, 241–243
  - DataTables, 258–273
  - explanation of, 229
  - introduction to, 237
  - role of DataColumns in, 241–243
  - System.Data namespaces, 237
  - System.Data.Common namespaces, 239–240
- ADO.NET events
- calling Fill and Update methods of DataAdapter for, 557–558
  - introduction to, 545–546
  - role of FillError property of data adapters in, 553
  - testing data adapter events used with, 554–559
  - using connection events with, 547–553
  - using DataAdapter events with, 553–559
  - using DataSet events with, 559–560
  - using DataTable events with, 560–565
  - using DataView and DataViewManager events with, 568–570
  - using XmlDataDocument events with, 565–568
- ADO.NET objects, transferring XML names to, 393
- AdoNetApp1.cs application, code for, 143–144



- ADOX (ActiveX Data Objects Extensions for Data Definition Language and Security)
  - using from managed code, 600
  - using with ADO.NET, 598–600
- aliases, using with SQL, 679–680
- AllowCustomPaging property of ASP.NET DataGrid control, description of, 465
- AllowDBNull property of ADO.NET DataColumn disconnected class, description of, 244
- AllowDelete property of ADO.NET DataView class, description of, 277
- AllowEdit property of ADO.NET DataView class, description of, 277
- AllowPaging property
  - of ASP.NET DataGrid control, 465
  - example of, 495
  - setting, 495
- AllowSorting property
  - of ASP.NET DataGrid control, 465
  - setting, 495
- amp XML built-in entities and references, description of, 370
- ampersand (&), role XML-document references, 370
- apos XML built-in entities and references, description of, 370
- AppendChild XML method, functionality of, 398–399
- Application class of System.Windows.Forms, description of, 101–102
- array elements, iterating with foreach loop statement, 43–44
- array types, explanation of, 25–29
- arrays
  - imitating with indexer class members, 63
  - sorting, searching, and copying, 26–29
- ASP.NET
  - adding even handlers to button-click events with, 448–449
  - adding server-side controls with, 460–462
  - advantages of, 437
  - binding Web Forms controls at design-time with, 472–478
  - creating data views and connecting to datasets with, 475–476
  - creating guest books with, 478–490
  - creating tables at design-time with, 504–507
  - creating tables programmatically with, 507–509
  - creating Web applications with, 438–442
  - data binding in, 462
  - installing, 436
  - introduction to, 435
  - platform requirements of, 436
  - setting control properties for, 444–448
  - setting page properties for, 441–442
  - table controls for, 504
  - using data-bound controls with, 462–464
  - using DataGrid and DataList controls with, 464–469
  - viewing data in DataGrid controls with, 452–455
  - viewing page options for, 441
- ASP.NET applications
  - adding, editing, and deleting data in, 496–503
  - adding new records to, 499–500
  - developing with VS .NET, 450–455
  - enabling automatic paging in, 494–495
  - enabling paging at design-time in, 491–494
  - executing SQL queries in, 497–499
  - filling data to ListBox controls with, 451–452
  - MyGuestBook example, 479–490
  - updating data in, 501
  - using ExecuteNonQuery method with, 498
  - using ExecuteSQL method with, 499
  - using SQL SELECT statements with, 498
- ASP.NET server-side controls
  - explanation of, 455
  - and .NET Framework library, 457–459
  - types of, 455–457
- assemblies
  - definition of, 594
  - understanding, 8–10
- assignment operators, examples of, 38
- asynchronous Web services, executing, 539–542
- at (@) sign, appearance in SQL Server Insert commands, 339
- atomic columns, role in 1NF, 654
- Attr node in XML, description of, 374
- Attribute member of XmlNodeType enumeration, description of, 384
- AttributeCount property of XmlReader class, description of, 387
- attributes
  - adding to XML nodes, 402
  - explanation of, 31
  - returning for XML nodes, 385–386
  - role in XML documents, 371
  - in XML, 361–362
- AutoGenerateColumns property of ASP.NET DataGrid control, description of, 465
- AutoIncrement property of ADO.NET DataColumn disconnected class, description of, 244
- AutoIncrementSeed property of ADO.NET DataColumn disconnected class, description of, 244
- AutoIncrementStep property of ADO.NET DataColumn disconnected class, description of, 244
- automatic memory management feature of C#, explanation of, 4
- AVG function, using in SQL, 673

## B

<b> tag, description of, 357

BackColor property of ASP.NET DataGrid control, description of, 465

BackColor property of ASP.NET DataGrid control, description of, 465

BaseURI property of XmlReader class, description of, 387

BCNF (Boyce-Codd normal form), explanation of, 659

BCNI (Boyce-Codd normal form), explanation of, 654

BeginEdit method of ADO.NET DataRow class, description of, 252

BeginInit method  
 of ADO.NET DataSet class, 276  
 of ADO.NET DataView class, 278

Begin(IsolationLevel) method  
 of Transaction class, description of, 342  
 in Sql data providers, 347

BeginTransaction method of ADO.NET Connection class, description of, 285

binary compatibility with base classes, C# support for, 4–5

binary operators, definition of, 37

BinarySearch property of array class, description of, 27

BindData method, example of, 260–264

<body> tag, description of, 357

Boiler.cs class example, implementing events and event handlers with, 58–60

bookstore elements, adding to XML schemas, 423–424

books.xml  
 deleting all items in, 400–401  
 loading from strings, 394–395  
 output of, 360–361

bool C# type alias, details of, 18

BorderColor property of ASP.NET DataGrid control, description of, 465

BorderStyle property of ASP.NET DataGrid control, description of, 465

BorderWidth property of ASP.NET DataGrid control, description of, 465

boxing, definition of, 30

<br> tag, description of, 357

break statement, functionality of, 44

Broken member of ADO.NET ConnectionType enumerations, description of, 285

Broken property of ConnectionState enumeration, description of, 549

Button class of System.Windows.Forms, description of, 101

button-click handlers, creating ASP.NET tables with programmatically, 508–509

button event handlers, writing code for, 269

byte C# type alias, details of, 18

## C

C#  
 automatic memory management feature of, 4  
 background of, 1  
 case-sensitivity of, 1  
 characteristics and features of, 2–5  
 compiling from command line, 144–145  
 and DLLs, 4  
 evolution of, 2  
 exception types in, 65  
 garbage collection in, 4  
 lack of support for inheritance, 3  
 language and cross-platform interoperability in, 5  
 as modern language, 3  
 as object-oriented language, 3  
 and open source, 1–2  
 operators in, 38  
 performing type conversions in, 29–31  
 role of manifests in, 5  
 role of namespaces in, 4, 7  
 role of object types in, 3–4  
 scalability of, 4–5  
 simplicity and flexibility of, 3  
 standard input and output streams in, 10–11  
 support for binary compatibility with base classes, 4–5  
 support for function overloading, 52–54  
 typesafety of, 3–4  
 versioning control in, 4–5

C# code, compiling from command line, 6

C# components, understanding, 7–10

C# Corner Web site, 3

C# editors, availability of, 5

C# types  
 explanation of, 4  
 introduction to, 17–18

CancelEdit method of ADO.NET DataRow class, description of, 252

candidate key, role in BCNF, 659

Caption property of ADO.NET DataColumn disconnected class, description of, 244

CarRec struct type example, 19–20

Catalog object of ADOX, functionality of, 599

CDATA member of XmlNodeType enumeration, description of, 384

CDATA sections, role in XML documents, 369

CellPadding property of ASP.NET DataGrid control, description of, 465

cells, adding to rows of ASP.NET tables at design-time, 506–507

CellSpacing property of ASP.NET DataGrid control, description of, 465

ChangeDatabase method of ADO.NET Connection class, description of, 285

- Chaos isolation level for transactions,
  - descriptions, 343
- char C# type alias, details of, 18
- character and entity references, role in XML documents, 370
- CheckBox ASP.NET data-bound control,
  - description of, 464
- CheckBox class of System.Windows.Forms,
  - description of, 101
- CheckBoxList ASP.NET data-bound control,
  - description of, 464
- checked operators, explanation of, 38–39
- child nodes, moving to, 417–418
- ChildNodes property of XmlNode class,
  - functionality of, 394
- ChildRelation property of ADO.NET
  - DataTable class, description of, 259
- class constructors, calling, 50–51
- class events, using delegate reference types
  - with, 24
- class keyword, definition of, 7
- class members, elements of, 46
- class method example, 52
- class objects, using indexer class members
  - with, 63
- class property member example, 56–58
- class reference types, explanation of, 21–22
- Class View window, displaying Windows
  - Forms classes in, 92–100
- classes
  - adding to VS .NET IDE Windows Forms application, 92–96
  - in ADO.NET, 129–132
  - functionality of, 46
  - implementing multiple interfaces with, 22–23
  - for ODBC .NET data providers, 618
- Clear method
  - of ADO.NET DataSet class, 276
  - of ADO.NET DataTable class, 259
- Clear property of array class, description of, 27
- CLI (Common Language Infrastructure) and C#, 1–2
- clients, creating for Web services, 525–539
- Clone method
  - of ADO.NET DataSet class, 276
  - of ADO.NET DataTable class, 259
- Clone property of array class,
  - description of, 27
- Close method
  - of ADO.NET Connection class, 285
  - of ADO.NET DataReaders, 315
  - using with XML documents, 392
- Close property of XmlReader class, description of, 388
- Closed member of ADO.NET ConnectionType
  - enumerations, description of, 285
- Closed property of ConnectionState
  - enumeration, description of, 549
- CLR (Common Language Runtime)
  - handling stack overflow with, 38–39
  - incorporation into Mono Project, 2
- CLR types, converting to XSD types, 393
- code segments, using goto statement with, 43–44
- ColorDialog class, functionality of, 101, 120–121
- ColumnChanged event of DataTable event,
  - description of, 560, 562
- ColumnChanging event of DataTable event,
  - description of, 560, 562
- ColumnMapping property of ADO.NET
  - DataColumn disconnected class, description of, 244
- ColumnName property of ADO.NET
  - DataColumn disconnected class, description of, 244
- columns, displaying with Data Form Wizard, 215–216
- Columns object of ADOX, functionality of, 599
- Columns property
  - of ADO.NET DataTable class, 259
  - of ASP.NET DataGrid control, 465
- COM (Component Object Model) and ADO.NET, 594–595
- COM libraries, adding references to, 595–596
- ComboBox class of System.Windows.Forms,
  - description of, 101
- comma (,), using with UPDATE statement in SQL, 676
- Command components, role in ADO.NET, 230–231
- command line, compiling C# from, 144–145
- command-line Windows forms applications,
  - writing, 70–78
- Command object, role in ADO.NET, 134–135
- Command objects
  - calling stored procedures with, 306–309
  - creating for ADO.NET applications, 140–141
  - functionality of, 300–301
  - role in ADO.NET, 133–134
- Command property of
  - OleDbRowUpdatedEventArgs, description of, 554
- CommandBuilder objects
  - creating, 335–336
  - functionality of, 334–335
- CommandBuilder utility, using with ADO.NET, 135–136
- CommandText property of
  - OleDbCommand, description of, 301
- CommandType item
  - functionality of, 305
  - of OleDbCommand, 301

- Comment member of XmlNodeType enumeration, description of, 384
- Comment node in XML, description of, 375
- comments
  - role in XML, 369
  - using `///` (three slashes) with, 67
- Commit method of Transaction class, description of, 342
- CommitTransaction, functionality of, 340
- common dialogs, creating with
  - Windows.Forms namespace, 118–121
- CommonAppDataPath method of
  - Windows.Forms.Application class, description of, 102
- CommonDialog class of
  - System.Windows.Forms, description of, 101
- Compare objects, sample output of, 15
- CompareValidator ASP.NET server-side control, description of, 457
- complexType items, adding to XML schemas, 424–425
- concurrency control
  - in ADO.NET, 343–347
  - definition of, 341
  - Web site for, 665
- conditional and (&&) operators, using with if . . . else statements, 40–41
- Conditional built-in attribute, description of, 31
- conditional operators, examples of, 38
- conditional or (||) operators, using with if . . . else statements, 40–41
- Configure Data Adapter option, using with SQL data adapters, 184
- Connecting member of ADO.NET
  - ConnectionType enumerations, description of, 285
- Connecting property of ConnectionState enumeration, description of, 549
- Connection class, using with ADO.NET data providers, 283–285
- Connection components, role in ADO.NET, 230–231
- connection events
  - adding programmatically at design-time, 549
  - testing, 549
  - writing code for execution of, 551–552
- connection events, using with ADO.NET events, 547–553
- Connection Lifetime connection pooling setting, description of, 299
- Connection objects
  - creating for ADO.NET applications, 140
  - creating with different constructors, 286
  - role in ADO.NET, 133–134
- connection pooling, understanding, 297–300
- Connection property of OleDbCommand, description of, 301
- Connection Reset connection pooling setting, description of, 299
- connection strings
  - for ODBC with various databases, 291
  - for Ole Db with various databases, 290
  - role in VS .NET, 167–169
- connections
  - adding with Server Explorer, 156–159
  - creating with different strings, 298–299
  - establishing for ADO.NET data providers, 283–285
  - establishing in ADO.NET applications, 140
  - opening and closing for ADO.NET data providers, 285–289
- connections and disconnected data, management by ADO.NET versus ADO, 128
- ConnectionState enumeration properties, list of, 548–549
- ConnectionString class, searching, 187
- ConnectionString property of ADO.NET
  - Connection objects, description of, 284
- ConnectionTimeout property of ADO.NET
  - Connection objects, description of, 284
- console-based ADO.NET applications, creating, 142–145
- Console class and members
  - accessing, 7
  - displaying, 11
- constant class member
  - functionality of, 49
  - and inheritance, 46
- constants, definition of, 36
- Constraint class in ADO.NET System.Data namespaces, description of, 237
- ConstraintCollection class in ADO.NET System.Data namespaces, description of, 237
- constraints, definition of, 242
- Constraints property of ADO.NET DataTable class, description of, 259
- constructors, overloading, 50
- ContextMenu class of
  - System.Windows.Forms, description of, 101
- Continue property of FillEventArgs, description of, 553
- continue statement, functionality of, 45
- Control class of Windows.Forms namespace, functionality of, 102
- Control classes of System.Windows.Forms, description of, 101
- control flow, explanation of, 40
- controls
  - adding to VS .NET IDE Windows Forms application, 79–82
  - adding to Windows Forms, 73–74
- Copy method
  - of ADO.NET DataSet class, 276
  - of ADO.NET DataTable class, 259

- Copy property of array class, description of, 27
  - CopyTo property of array class, description of
    - of array class, 27
  - Count property of ADO.NET DataView class, description of, 277
  - CREATE TABLE statement, using with SQL, 676–677
  - CREATE VIEW statement, using with SQL, 681
  - CreateCommand method of ADO.NET
    - Connection class, description of, 285
  - CreateCustomersTable, calling with form's constructor, 260
  - CreateCustomersTable method, code for, 268–269
  - CreateCustomerTable method, example of, 260–265
  - CreateInstance property of array class, description of, 27
  - CreateNavigator method of XmlNode class, functionality of, 394
  - CreateOrdersTable method, example of, 260–264
  - CROSS JOINS, using with SQL, 680
  - .cs extension, adding to Windows Forms applications, 70
  - Ctrl+F keyboard shortcut, searching
    - ConnectionString class with, 187
  - Ctrl+F5 keyboard shortcut, running VS .NET IDE Windows Forms application with, 91
  - cubes
    - getting dimensions of, 605–607
    - getting from FoodMart 2000 database, 604–605
    - role in OLAP, 600
  - CurrentPageIndex property of ASP.NET
    - DataGrid control, description of, 465
  - Cursor classes of System.Windows.Forms, description of, 101
  - cursors, role in relational databases, 660–661
  - CustEmpView, displaying, 592–593
  - custom ASP.NET server-side controls, explanation of, 457
  - customer/order relationship
    - example of, 260–264
    - explanation of, 257–258
  - CustomValidator ASP.NET server-side control, description of, 457
  - CustOrderRelation relationship, creating with Data Form Wizard, 214–215
  - CustOrdersDetail stored procedure, displaying output of, 582–583
- D**
- data
    - displaying in ADO.NET applications, 141
    - managing and viewing with Server Explorer, 159–161
    - reading and storing, 313–314
    - retrieving from views programmatically, 593–594
  - Data Adapter Configuration Wizard
    - binding ASP.NET Web Forms controls at design-time with, 472–478
    - binding DataGrid controls with, 476–477
    - filling datasets with, 477–478
  - Data Adapter Configuration Wizard, creating
    - SQL adapters with, 171
  - data adapters, generating DataSet objects from, 474–475
  - data-bound controls, using with ASP.NET, 462–464
  - data columns, understanding, 243–244
  - data components, using with VS .NET, 162–166
  - data connection pages, choosing for SQL data adapters, 173
  - data connections
    - choosing with Data Form Wizard, 211–212
    - role in VS .NET, 166–169
  - Data Form Wizard
    - adding items to, 209–210
    - calling from applications, 220–221
    - choosing data connections with, 211–212
    - choosing DataSet objects with, 211
    - choosing display style for, 216–220
    - choosing project templates for, 209
    - choosing tables and columns to display with, 215–216
    - choosing tables with, 212
    - choosing views with, 212
    - creating relationships between tables with, 213–215
    - examining functionality of, 223–226
    - generating FillDataSet method with, 225
    - generating LoadDataSet method with, 224–225
    - generating UpdateDataSet method with, 226
    - generating UpdateDataSource method with, 226
    - Grid DataForm sample output in, 218
    - role of DataForm1.cs class in, 224–226
    - role of MyDS.xsd in, 224
    - viewing output from, 221–223
    - walking through, 210
  - data providers
    - choosing for ADO.NET applications, 138–139
    - choosing with Server Explorer, 157
    - Connection class for, 134
  - data sources, connecting through ODBC
    - DSN, 294–296
  - data synchronization, definition of, 410
  - DataAdapter class
    - description of, 240
    - methods of, 323
  - DataAdapter components
    - adding for SQL data adapters, 172–173
    - role in ADO.NET, 230–231

- DataAdapter constructors, overloaded forms of, 320
- DataAdapter events
  - adding from Properties Window of Query Builder, 554–555
  - using with ADO.NET events, 553–559
- DataAdapter objects
  - connecting through TableMapping property, 188
  - constructing, 320–321
  - constructing for SqlDataAdapter objects, 148
  - creating for ADO.NET applications, 140–141
  - example of, 323–326
  - functionality of, 319–320
  - generating typed DataSets with, 199–204
  - inserting, updating, and deleting data with, 327
  - performing table and column mapping with, 332–334
  - properties of, 322
  - relationship to Command objects in ADO.NET, 134–135
  - relationship to DataSet and DataView objects, 275
  - role in ADO.NET, 136
  - using FillError event handlers with, 556
- DataAdapter properties, setting and reviewing for SQL data adapters, 180–184
- DataBase property of ADO.NET Connection objects, description of, 284
- database schemas
  - explanation of, 653
  - getting from ADO.NET, 607–610
- database table columns versus fields, 242
- database tables. *See* tables
- DataColumn class
  - in ADO.NET System.Data namespaces, 237
  - creating for ADO.NET applications, 245–247
  - properties of, 244
  - relationship to DataRow and DataTable classes, 241
- DataColumn constructors, creating columns with, 246–247
- DataColumn properties, setting, 247–248
- DataColumnChangeEventHandler ADO.NET event, functionality of, 546
- DataColumnCollection class in ADO.NET System.Data namespaces, description of, 237
- DataColumnMapping class in ADO.NET System.Data.Common namespaces, description of, 240
- DataColumnMapping example, 333–334
- DataColumnMappingCollection class in ADO.NET System.Data.Common namespaces, description of, 240
- DataColumns, adding to DataTables, 248–251
- DataForm1.cs class, role in Data Form Wizard, 224–226
- DataGrid ASP.NET data-bound control, description of, 463
- DataGrid control properties, setting at design-time, 466–467
- DataGrid controls
  - adding to forms for SQL data adapters, 172
  - creating with Data Adapter Configuration Wizard, 476–477
  - displaying HTML view of, 494–495
  - displaying Orders table data in, 325
  - filling in VS .NET, 197–198
  - filling with data, 189–190
  - paging in, 490–493
  - setting images as next and previous page text for, 495
  - using AutoFormat option with, 467–468
  - using Borders property page with, 471–472
  - using Columns property page with, 469
  - using Format property page with, 470–471
  - using Paging property page with, 469–470
  - using Property Builder with, 468
  - using with ASP.NET, 464–469
  - viewing ASP.NET data in, 452–455
- DataKeyField property of ASP.NET DataGrid control, description of, 465
- DataList ASP.NET data-bound control, description of, 463
- DataList controls, using with ASP.NET, 464–466
- DataReaders
  - filling data to, 141
  - functionality of, 313–314
  - initializing and closing, 314–315
  - properties and methods of, 315
  - reading with, 315–317
  - using with SQL Server databases, 316–317
- DataRelation class in ADO.NET System.Data namespaces, description of, 237
- DataRelation constructor, functionality of, 257–258
- DataRelationCollection class in ADO.NET System.Data namespaces, description of, 237
- DataRow class
  - in ADO.NET System.Data namespaces, 237
  - explanation of, 251–256
  - relationship to DataColumn and DataTable classes, 241
- DataRow objects
  - adding rows to DataTables with, 253–255
  - adding to DataTables, 328
  - functionality of, 241
- DataRowChangeEventHandler ADO.NET event, functionality of, 546
- DataRowCollection class in ADO.NET System.Data namespaces, description of, 237

- DataRowState enumeration, functionality of, 256–257
- DataRowView class in ADO.NET System.Data namespaces, description of, 238
- DataSet class
  - in ADO.NET System.Data namespaces, 238
  - reading XML documents with, 405–406
  - relationship to DataTable and DataView classes, 274
  - role in ADO.NET, 273
  - writing XML documents with, 406–409
- DataSet components
  - role in ADO.NET, 231–232
  - using with VS .NET, 199–208
- dataset data, saving to XML documents, 414
- DataSet events, using with ADO.NET events, 559–560
- DataSet format, displaying XML data in, 411–413
- DataSet objects
  - adding DataTables to, 265
  - choosing with Data Form Wizard, 211
  - connecting to data views in ASP.NET, 475–476
  - constructing and filling for ADO.NET applications, 149
  - in DataView objects in, 138
  - filling data to, 141, 323, 477–478
  - functionality of, 137, 274–276
  - generating from data adapters, 474–475
  - generating from existing XML schemas, 430–432
  - loading XML data with, 411
  - populating in VS .NET, 197–198
  - relationship to DataAdapter and DataView objects, 275
  - role in ADO.NET, 124
  - using Server Explorer with, 427–429
  - using with DataAdapter objects in ADO.NET, 136
- DataSet property
  - of ADO.NET DataSetView class, 279
  - of ADO.NET DataTable class, 259
- DataSet1 class, generating with Data Adapter Configuration Wizard, 474–475
- DataSetName property of ADO.NET DataSet class, description of, 275
- DataSource property
  - of ADO.NET Connection objects, 284
  - of ASP.NET DataGrid control, 465
- DataTable class
  - in ADO.NET System.Data namespaces, 238
  - properties of, 259
  - relationship to DataColumn and DataRow classes, 241
  - relationship to DataSet and DataView classes, 274
- DataTable components, role in ADO.NET, 232
- DataTable events, using with ADO.NET events, 560–565
- DataTable objects, functionality of, 241
- DataTable property of FillEventArgs, description of, 553
- DataTableCollection class in ADO.NET System.Data namespaces, description of, 238
- DataTableMapping class in ADO.NET System.Data.Common namespace, description of, 240
- DataTableMapping objects, using with DataAdapters, 332–333
- DataTableMappingCollection class in ADO.NET System.Data.Common namespaces, description of, 240
- DataTable objects
  - adding DataColumn to, 248–251
  - adding DataRow to, 328
  - adding rows to, 253–255, 270
- DataType property of ADO.NET DataColumn disconnected class, description of, 244
- DataView and DataViewManager events, using with ADO.NET events, 568–570
- DataView class
  - in ADO.NET System.Data namespaces, 238
  - relationship to DataSet and DataTable classes, 274
  - role in ADO.NET, 273
- DataView components, using with VS .NET, 199–208
- DataView objects
  - adding, updating, and deleting rows of, 569–570
  - DataSet objects in, 138
  - relationship to DataAdapter and DataSet objects, 275
  - role in ADO.NET, 277–278
  - role in VS .NET, 207–208
- DataViewManager class
  - in ADO.NET System.Data namespaces, 238
  - role in ADO.NET, 278–279
- DataViewManager property of ADO.NET DataView class, description of, 277
- DataViewSettings property of ADO.NET DataSetView class, description of, 279
- DbDataAdapter class in ADO.NET System.Data.Common namespaces, description of, 240
- DBDataPermission class in ADO.NET System.Data.Common namespaces, description of, 240
- DBTools
  - importing Access Northwind database with, 623–625
  - setting internal data connections of, 626
- DbType property of OleDbParameter class, description of, 338

- deadlocks, explanation of, 665
  - decimal C# type alias, details of, 18
  - DecodeName method, using with XML documents, 393
  - DefaultValue property of ADO.NET
    - DataColumn disconnected class, description of, 244
  - DefaultView property of ADO.NET DataTable class, description of, 259
  - DefaultViewManager property of ADO.NET DataSet class, description of, 275
  - delegate reference types
    - explanation of, 24
    - using with event class members, 58
  - Delete method
    - of ADO.NET DataRow class, 252–253, 255–256
    - of ADO.NET DataView class, 278
  - DELETE statement, using with SQL, 676
  - DeleteCommand property
    - of OleDbDataAdapter Command, 322
    - of OleDbDataAdapters, 322
  - Deleted member of ADO.NET DataRowState enumeration, 257
  - DeleteRow\_Click method, code for, 271
  - delimiters, choosing with Export Text Wizard, 636
  - Depth property
    - of ADO.NET DataReaders, 315
    - of XmlReader class, 387
  - description, role in Web services, 511
  - design-time versus run-time development,
    - role in creating Windows forms, 69–70
  - Design View, displaying orders for Web services in, 525–526, 535
  - destructor class member
    - advisory about, 51
    - functionality of, 51
    - and inheritance, 46
  - Detached member of ADO.NET
    - DataRowState enumeration, 257
  - dialog classes of System.Windows.Forms,
    - description of, 101
  - dialogs, creating with Windows.Forms namespace, 118–121
  - Direction property of OleDbParameter class,
    - description of, 338
  - dirty reads, role in relational database isolation levels, 662
  - \*.disco files, role in Web services, 511, 529
  - disconnected components, definition of, 199
  - discovery files, using with Web services, 529
  - discovery, role in Web services, 511
  - Dispose method, functionality of, 85, 287
  - DllImport built-in attribute, description of, 31
  - DLLs (dynamic link libraries) and C#, relationship between, 4
  - DOCTYPE declaration, role in XML documents, 368
  - document classes in System.Xml namespace,
    - explanation of, 376
  - Document member of XmlNodeType enumeration,
    - description of, 384
  - Document node in XML,
    - description of, 374
  - Document Outline viewer, synchronizing
    - Web-page controls with, 446–448
  - DocumentElement XML method,
    - functionality of, 399
  - DocumentFragment member of
    - XmlNodeType enumeration, description of, 384
  - documenting source code, 67
  - documents, loading with XmlDocument class, 394–395
  - DocumentType member of XmlNodeType enumeration,
    - description of, 384
  - DocumentType node in XML,
    - description of, 374
  - DOM API (application programming interface),
    - explanation of, 378
  - DOM (Document Object Model), overview of, 372–375
  - DOM implementation
    - role of Load methods in, 394–395
    - role of Save methods in, 395
    - role of XmlDocument class in, 394–395
    - role of XmlDocumentFragment class in, 395–396
    - role of XmlElement class in, 397–402
    - role of XmlNode class in, 394
  - double C# type alias, details of, 18
  - double quote (") entity in XML,
    - representing, 370
  - do . . . while loop statement,
    - functionality of, 43
  - drag-and-drop design-time feature in VS .NET,
    - advantages of, 459
  - DROP TABLE statement
    - executing with OleDbCommand, 650–651
    - using with SQL, 677
  - DropDownList ASP.NET data-bound control,
    - description of, 464
  - DSNs (Data Source Names)
    - connecting to data sources through, 294–296
    - creating, 630
    - defining when accessing text files, 638
  - DTD (Document Type Definition),
    - explanation of, 364–366
  - dynamic cursors,
    - explanation of, 660
- ## E
- ECMA (European Computer Manufacturers Association), 1–2
  - EditItemIndex property of ASP.NET DataGrid control,
    - description of, 465
  - EditItemStyle property of ASP.NET DataGrid control,
    - description of, 465
  - element-attributes in XSLT,
    - definition of, 403
  - Element member of XmlNodeType enumeration,
    - description of, 384
  - Element node in XML,
    - description of, 374



- elements
    - in HTML, 356
    - in XML, 365–366, 369
  - Employees table, saving as Excel spreadsheet, 642
  - empty elements in XML documents, explanation of, 370–371
  - EndEdit method of ADO.NET DataRow class, description of, 252
  - EndElement member of XmlNodeType enumeration, description of, 384
  - EndEntity member of XmlNodeType enumeration, description of, 384
  - EndInit method of ADO.NET DataSet class, description of, 276
  - entity and character references, role in XML documents, 370
  - Entity member of XmlNodeType enumeration, description of, 384
  - Entity node in XML, description of, 375
  - EntityReference member of XmlNodeType enumeration, description of, 384
  - enum data types, explanation of, 20–21
  - EOF property of XmlReader class, description of, 387
  - equal to (=) operator in SQL, 673
  - equality operators, examples of, 38
  - Equals method of Object class, explanation of, 12, 14–15
  - Error class, functionality of, 350–352
  - ErrorCode property of InfoMessageEvents, description of, 548
  - errors, catching with SqlException class, 351–352
  - Errors property
    - of FillErrorEventArgs, 553
    - of OleDbRowUpdatedEventArgs, 554
  - Errors property of InfoMessageEvents, description of, 548
  - event class member
    - functionality of, 58–62
    - and inheritance, 46
  - event handlers
    - adding code to VS .NET IDE Windows Forms application, 89–91
    - adding to button-click events with ASP.NET, 448–449
    - adding to button controls in Windows Forms, 74–75
    - adding to menu items with Windows.Forms namespace, 116–117
    - defining, 58
    - implementing with Boiler.cs class, 58–59
    - writing for toolbar buttons with Windows.Forms namespace, 113–115
  - event handling, example of, 61–62
  - event reference types, explanation of, 24–25
  - events
    - adding to Windows Forms, 74–75
    - implementing with Boiler.cs class, 58–59
    - understanding, 61
  - Excel connection string for ODBC, displaying, 291
  - Excel databases
    - accessing, 641–643
    - connecting to, 292
  - exception handling, functionality of, 65–67
  - Exception type in C#, description of, 65
  - Execute method, using with SQL statements in ASP.NET applications, 498
  - ExecuteNonQuery method
    - of Command object, 311
    - using with ASP.NET applications, 498
  - ExecuteScalar method of Command object, description of, 312–313
  - ExecuteSQL method, using with ASP.NET applications, 499
  - Executing member of ADO.NET ConnectionType enumerations, description of, 285
  - Executing property of ConnectionState enumeration, description of, 549
  - Exit, ExitThread methods of
    - Windows.Forms.Application class, description of, 102
  - explicit type conversions, explanation of, 30
  - Export Text Wizard, exporting Access tables to text files with, 635
  - Expression property of ADO.NET DataColumn disconnected class, description of, 244
  - expressions, definition of, 36–39
- ## F
- Fetching property
    - of ADO.NET ConnectionType enumerations, 285
    - of ConnectionState enumeration, 549
  - field class member and inheritance, relationship between, 46
  - FieldCount property of ADO.NET DataReaders, description of, 315
  - fields class member, functionality of, 48–49
  - fields versus database table columns, 242
  - FileDialog class of System.Windows.Forms, description of, 101
  - Fill Data button, effect in ASP.NET, 454
  - Fill method
    - of DataAdapter, 557–558
    - of OleDbDataAdapters, 323
  - FillDataGrid method, example of, 496–497
  - FillDataSet method, generating with Data Form Wizard, 225
  - FillDBGrid method, calling from Form1 constructor, 189
  - FillError event handler code, using DataAdapter with, 556
  - FillError property of data adapters, role in ADO.NET events, 553
  - FillErrorHandler ADO.NET event, functionality of, 546

- FillSchema method of OleDbDataAdapters, description of, 323
  - Finalize method of Object class, use of, 16–17
  - Find method of ADO.NET DataView class, description of, 278
  - FindRows method of ADO.NET DataView class, description of, 278
  - firehose cursors, definition of, 313
  - first child nodes, moving to, 417–418
  - first.exe file, creation of, 6
  - FirstWebApplication project, creating with ASP.NET, 439
  - float C# type alias, details of, 18
  - Font dialog box, displaying, 119–120
  - Font property of ASP.NET DataGrid control, description of, 465
  - <font> tag, description of, 357–358
  - FontDialog class of System.Windows.Forms, description of, 101
  - FoodMart 2000 database, getting all available cubes from, 604–605
  - FooterStyle property of ASP.NET DataGrid control, description of, 465
  - for loop statement, functionality of, 42
  - foreach loop statement, functionality of, 43
  - ForeColor property of ASP.NET DataGrid control, description of, 465
  - foreign keys, role in relational databases, 653
  - ForeignKeyConstraint class in ADO.NET System.Data namespaces, description of, 237
  - Form class of Windows.Forms namespace, functionality of, 102–103
  - Form classes of System.Windows.Forms, description of, 101
  - form controls, adding events with Windows Forms, 74–75
  - Form Designer, examining code in, 84–89
  - form properties, adding to Windows Forms, 72–78
  - Form\_Load event, adding code on, 198
  - forms, definition of, 69
  - forward read-only cursors, definition of, 313, 660
  - function overloading, C# support for, 52
- ## G
- garbage collection in C#, explanation of, 4
  - Generate Dataset option, using with SQL data adapters, 184
  - GetAttribute method
    - using with XML nodes, 385–386
    - of XmlElement class, 398
  - GetAttribute property of XmlReader class, description of, 388
  - GetChanges method of ADO.NET DataSet class, description of, 276
  - GetChildRows method of ADO.NET DataRow class, description of, 252
  - GetFillParameters method of OleDbDataAdapters, description of, 323
  - GetHashCode method of Object class, description of, 12, 16–17
  - GetLength property of array class, description of, 27
  - GetOrderFromDatabase method
    - adding to sample Web service, 519–520
    - testing in sample Web service, 523
  - GetParentRows method of ADO.NET DataRow class, description of, 252
  - GetType method of Object class, description of, 12–15
  - GetType operator versus typeof operator, 39
  - GetValue property of array class, description of, 27
  - GetXml method of ADO.NET DataSet class, description of, 276
  - GetXmlSchema method of ADO.NET DataSet class, description of, 276
  - Getxxx method of ADO.NET DataReader, description of, 315
  - Global.asax files in Web services, explanations of, 514
  - goto statement, functionality of, 43–44
  - greater than (>)
    - entity in XML, 370
    - operator in SQL, 673
  - Grid DataForm sample output in Data Form Wizard, 218
  - GridLines property of ASP.NET DataGrid control, description of, 465
  - GROUP BY clause, using with SQL, 674–675
  - gt XML built-in entities and references, description of, 370
  - guest books, creating in ASP.NET, 478–490
  - GuestBook.mdb, table schema of Guest table in, 479
  - GUI components, role in Web Forms, 438
  - GUI (Graphical User Interface), building for DataTable operations, 266–267
- ## H
- <h1 . . . h6> tag, description of, 357
  - HasAttribute method of XmlElement class, description of, 398
  - HasAttributes property of XmlReader class, description of, 387
  - hash symbol (#), role in XML-document character references, 370
  - hashtables, definition of, 16
  - HasValue property of XmlReader class, description of, 387
  - HAVING clause, using with SQL, 675
  - HeaderStyle property of ASP.NET DataGrid control, description of, 465
  - Height property of ASP.NET DataGrid control, 466
  - “Hello, C# World!” program, writing, 6–7
  - Hello class, creating namespace wrapper for, 8
  - HelloWorldNamespace member, calling from MyOtherNamespace, 10

<hr> tag, description of, 357  
 HTML ASP.NET server-side controls, list of, 455  
 HTML files, simple example of, 356  
 HTML (HyperText Markup Language)  
   explanation of, 356–358  
   versus XML, 359, 361  
 HTML tags  
   examples of, 358  
   explanation of, 356  
   list of, 357  
   navigating with Document Outline viewer,  
   448  
 <html> tag, description of, 357

## I

<i> tag, description of, 357  
 ID property of ASP.NET DataGrid control,  
   description of, 466  
 IDataParameter interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 238  
 IDataParameterCollection interfaces of  
   ADO.NET System.Data name-  
   spaces, description of, 238  
 IDataReader interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 238  
 IDataRecord interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 238  
 IDbCommand interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 239  
 IDbConnection interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 239  
 IDbDataAdapter interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 239  
 IDbDataAdapters, implementing with data  
   provider-specific classes, 320  
 IDbDataParameter interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 239  
 IDbTransaction, implementing with data  
   provider-specific classes, 342  
 IDbTransaction interfaces of ADO.NET  
   System.Data namespaces,  
   description of, 239  
 if . . . else statement, functionality of, 40–41  
 images, adding to toolbar buttons with  
   Windows.Forms namespace,  
   107–113  
 implicit type conversions, explanation of,  
   29–30  
 indexer class member  
   functionality of, 63  
   and inheritance, 46  
 IndexOutOfRangeException type in C#,  
   description of, 65–66  
 InfoMessage ADO.NET event  
   testing, 551, 553

  using with Connection object and  
   ADO.NET events, 547–548  
   writing event handlers for, 550  
 InfoMessageEventHandlers, functionality of,  
   352–353  
 inheritance, functionality of, 64–65  
 InitializeComponent routine, code for, 187  
 INNER JOINS, using with SQL, 680  
 input and output streams in C#, explanation  
   of, 10–11  
 Input member of ParameterDirection enu-  
   meration, description of, 586  
 input parameters, accepting with stored pro-  
   cedures, 581–582  
 InputOutput member of ParameterDirection  
   enumeration, description of, 586  
 INSERT statement  
   adding records to tables with, 311–313  
   using with Oracle databases and OleDb  
   data providers, 646–647  
   using with SQL, 678–679  
 InsertAfter XML method, functionality of, 401  
 InsertCommand property  
   of OleDbDataAdapter Command, 322  
   of OleDbDataAdapters, 322  
 InsertOrder method of OrderRetrievalService  
   project, testing, 533–536  
 InsertOrderFromNode Web method, calling  
   for sample Web service, 538  
 instance constructor class member  
   functionality of, 49–51  
   and inheritance, 46  
 instance fields, definition of, 35  
 int C# type alias, details of, 18  
 interface reference types, explanation of,  
   22–23  
 internal accessibility modifier,  
   description of, 33  
 internal accessibility type for class  
   members, 47  
 interoperability marshaling, definition of,  
   595  
 is operator, explanation of, 39  
 IsClosed property of ADO.NET DataReaders,  
   description of, 315  
 IsDefault property of XmlReader class,  
   description of, 387  
 IsEmptyTag property of XmlReader class,  
   description of, 387  
 IsFixedLength property of array class,  
   description of, 26  
 IsNullable property of OleDbParameter class,  
   description of, 338  
 isolation levels  
   and data consistency, 663  
   role in relational databases, 662–663  
 IsolationLevel, role in database connections,  
   343  
 IsReadOnly property of array class,  
   description of, 26  
 IsStartElement property of XmlReader class,  
   description of, 388

- ITableMapping interfaces of ADO.NET
  - System.Data namespaces, description of, 239
- ITableMappingCollection interfaces of ADO.NET System.Data namespaces, description of, 239
- Item property
  - of ADO.NET DataReaders, 315
  - of ADO.NET DataRow disconnected class, 251
  - of ADO.NET DataView class, 277
  - role in getting XML node information, 381–382
  - of XmlReader class, 387
- ItemArray property of ADO.NET DataRow disconnected class, description of, 251
- J**
- jagged arrays, example of, 26
- JIT (Just-In-Time) compiler, incorporation into Mono Project, 2
- JOIN queries, using with SQL, 679–680
- K**
- keyset cursors, explanation of, 660–661
- L**
- Label class of System.Windows.Forms, description of, 101
- last in wins concurrency control, definition of, 341
- Length property of array class, description of, 26
- less than (<)
  - entity in XML, 370
  - operator in SQL, 673
- LineNumber property of SqlError class, description of, 352
- ListBox ASP.NET data-bound control, description of, 463
- ListBox class of System.Windows.Forms, description of, 101
- ListBox controls, filling data to, 451–452
- ListChanged event, components of, 568, 570
- ListChangedType member of ListChangedEventArgs, description of, 568
- ListView class of System.Windows.Forms, description of, 101
- Load methods, role in DOM implementation, 394–395
- LoadDataSet method, generating with Data Form Wizard, 224–225
- local variables, explanation of, 32
- LocalName property of XmlReader class, description of, 387
- locking modes, role in relational databases, 664–665
- locking, role in relational databases, 661–665
- logical operators, examples of, 38
- long C# type alias, details of, 18
- LookupNamespace property of XmlReader class, description of, 388
- loops
  - exiting with break statement, 44
  - exiting with continue statement, 45
- lt XML built-in entities and references, description of, 370
- M**
- MainMenu class of System.Windows.Forms, description of, 101
- managed classes, role in ADO.NET, 127
- managed code
  - definition of, 3
  - role in ADO.NET, 127
  - using ADOX from, 600
- managed code, definition of, 594
- manifests, role in C#, 5
- markup, definition of, 355
- master/details relationship, explanation of, 257–258
- Max Pool Size connection pooling setting, description of, 299
- MaxLength property of ADO.NET DataColumn disconnected class, description of, 244
- MemberwiseClone method of Object class, use of, 16–17
- Menu control classes of System.Windows.Forms, description of, 101
- menu items, adding to Windows applications with Windows.Forms namespace, 115–118
- MenuItem class of System.Windows.Forms, description of, 101
- Merge method of ADO.NET DataSet class, description of, 276
- MergeFailed dataset event, explanation of, 559–560
- MergefailedEventHandler ADO.NET event, functionality of, 546
- Message property
  - of InfoMessageEvents, 548
  - of OleDbError class, 351
  - of SqlError class, 352
- MessageBox control classes of System.Windows.Forms, description of, 101
- metadata, role in relational databases, 654
- method class member
  - functionality of, 52–56
  - and inheritance, 46
- method overloading example, 52–54
- methods
  - adding to VS .NET IDE Windows Forms application, 97–100
  - of Windows.Forms.Application class, 102
- Microsoft .NET and XML, functionality of, 375–380

Microsoft.Data.Odbc namespace  
 displaying, 616  
 locating, 130

Microsoft.Data.Odbc.dll assembly, adding  
 references to, 615–616

Min Pool Size connection pooling setting,  
 description of, 299

modern language, C# as, 3

Modified member of ADO.NET DataRowState  
 enumeration, 257

modifiers, definition of, 32

Mono Project, explanation of, 2

Move methods of XPathNavigator class,  
 descriptions of, 417

MoveToAttribute property of XmlReader  
 class, description of, 388

MoveToContent method, navigating nodes  
 in XML documents with,  
 384–385

MoveToContent property of XmlReader class,  
 description of, 388

MoveToElement property of XmlReader class,  
 description of, 388

MoveToFirstAttribute property of XmlReader  
 class, description of, 388

MoveToMethod, navigating nodes in XML  
 documents with, 384–385

MoveToNextAttribute property of XmlReader  
 class, description of, 388

msadomd.dll library  
 adding references to, 602  
 locating, 601

MSXML parser, description of, 363

multi-item data-bound controls, using with  
 ASP.NET, 462–463

multiple arrays, example of, 26

multiplicative operators, examples of, 38

myClass, indexers signature of, 63

MyDS.xsd XML schema, role in Data Form  
 Wizard, 224

MyForm.cs example, creating with Windows  
 Forms, 71–72

MyGuestBook ASP.NET application  
 adding forms to, 483–487  
 compiling and running, 487–490  
 controls on, 480  
 creating, 479–482  
 source code for adding guest data to data-  
 base, 481–482  
 submission page for, 481

MyOtherNamespace namespace members,  
 calling, 9

mySP stored procedure  
 displaying output of, 580–581  
 executing using SQL data providers, 584

MySQL database  
 connecting through Odbc data providers,  
 296–297  
 downloading, 621  
 finding ODBC drivers for, 622–623  
 starting as service, 621–622  
 using, 632–633

MySQL Server, importing Northwind data-  
 base into, 625–630

MyTable data, viewing in DataGrid control,  
 650

myTestCls derived from  
 System.Windows.Forms.Form  
 example, 96

## N

Name property of XmlReader class, descrip-  
 tion of, 387

Name property, reading XML node informa-  
 tion with, 381–384

namespace references, adding to ADO.NET  
 applications, 139–140

namespaces  
 in ADO.NET, 129–132  
 role in C#, 4, 7  
 understanding, 8–10

NamespaceURI property of XmlReader class,  
 description of, 387

NameTable property of XmlReader class,  
 description of, 387

NativeError property of OleDbError class,  
 description of, 351

nested transactions, definition of, 340

.NET base class library, terminology advisory,  
 123

.NET data providers, choosing for ADO.NET  
 applications, 138–139

.NET Framework and Web services, relation-  
 ship between, 512

.NET Framework library and ASP.NET server-  
 side controls, relationship  
 between, 457–459

.NET Framework, using XslTransformation  
 class with, 403–404

NewIndex member of ListChangedEventArgs,  
 description of, 568

NewParent property of  
 XmlNodeChangedEventArgs,  
 description of, 566

NewRow method of ADO.NET DataTable  
 class, description of, 259

NextResult method of ADO.NET  
 DataReaders, description of, 315

Node property of  
 XmlNodeChangedEventArgs,  
 description of, 566

NodeChanged event handler, code for, 567

NodeInserted event handler, code for, 567

NodeRemoved event handler, code for, 568

nodes. *See* XML nodes

NodeType property of XmlReader class,  
 description of, 387

NodeType property, role in getting XML node  
 information, 382

non-static methods, definition of, 52

None member of XmlNodeType enumera-  
 tion, description of, 384

nonrepeatable reads, role in relational data-  
 base isolation levels, 662

- normalization, role in relational databases, 654
  - Northwind database
    - choosing in Server Explorer for use with OleDb data adapters, 191
    - Customers database table schema in, 241–242
    - displaying in Server Explorer, 288
    - displaying Orders table schema of, 670
    - importing into MySQL Server, 625–630, 625–630
    - importing with DBTools, 623–625
    - listing stored procedures for, 580
    - using ODBC .NET data providers to read data from, 620–621
    - viewing Employee table with Server Explorer, 160–161
    - viewing table schema and data in, 243
  - not equal to (<>) operator in SQL, 673
  - Notation member of XmlNodeType enumeration, description of, 384
  - NullReferenceException type in C#, description of, 65–66
  - Number property of SqlError class, description of, 352
- O**
- Object class
    - explanation of, 12–17
    - Finalize method of, 16–17
    - getting type of, 12–13
    - MemberwiseClone method of, 16–17
  - object types, role in C#, 3–4
  - objects
    - comparing with Equals method in Object class, 14–15
    - destroying with destructor class member, 51
  - ODBC Command objects, creating, 303–305
  - ODBC data components, viewing in toolbox, 164–165
  - ODBC data provider for ADO.NET
    - Command class for, 135
    - connecting to Excel databases with, 292
    - connecting to other providers with, 290–292
    - connecting to SQL Server databases with, 291
    - connecting to text files with, 293–294
    - Connection class for, 134
    - for DataAdapter class, 136
  - Odbc data providers
    - connecting to MySQL databases through, 296–297
    - downloading, 162
  - ODBC Data Source Administrator, examining drivers from, 290
  - ODBC drivers
    - accessing text files with, 640
    - finding for MySQL database, 622–623
  - ODBC DSNs (Data Source Names)
    - connecting to data sources through, 294–296
    - creating, 630–632
  - ODBC .NET data providers
    - accessing data from data sources with, 619–621
    - accessing Excel databases with, 641–643
    - accessing text files with, 637–641
    - customizing toolbox for, 613–614
    - finding components for, 614–615, 617
    - installing, 613–617
    - introduction to, 613
    - using, 617–621
    - using with Oracle 9i databases, 647–651
    - using with Oracle databases, 644–651
    - using with Sybase databases, 651–652
    - verifying presence of, 617
  - ODBC (Open Data Base Connectivity) and ADO.NET, 125–126
  - OdbcCommand, executing DROP TABLE SQL statement with, 650–651
  - OdbcConnection class object, role in VS .NET, 166
  - Odbc.Net folder, browsing, 615
  - OLAP (Online Analytical Processing) Server data, accessing with ADO.NET, 600–611
  - OldIndex member of ListChangedEventArgs, description of, 568
  - OldParent property of XmlNodeChangedEventArgs, description of, 566
  - OLE-DB (Object Linking and Embedding Data Base) and ADO.NET, 125–126
  - OLE DB Services settings, details of, 298
  - OleDb Command objects, creating, 301–303
  - OleDb data adapters, working with, 190–191
  - OleDb data providers
    - accessing Oracle 8i databases with, 645–647
    - Command class for, 135
    - connecting to a SQL Server with, 288
    - connecting to other providers with, 290–292
    - Connection class for, 134
    - for DataAdapter class, 136
    - using with VS .NET, 165
  - OleDb data provider classes, list of, 281–282
  - OleDb data provider event handlers and event argument classes for ADO.NET events, diagram of, 546–547
  - OleDbCommand constructor, functionality of, 141
  - OleDbCommand objects reading data from databases with, description of, 302–303
  - OleDbConnection class object, role in VS .NET, 166

- OleDbConnections, opening and closing, 286–287
  - OleDbDataAdapter class, properties of, 322
  - OleDbDataAdapter Command properties, list of, 322
  - OleDbDataAdapters
    - creating for VS .NET applications, 192–197
    - creating for Web services with Server Explorer, 517–519
    - displaying Orders table data in DataGrids with, 325
    - executing SELECT statements with, 321
  - OleDbError class properties, list of, 351
  - OleDbError collection, utilizing, 350
  - OleDbInfoMessageEventHandler OleDb data provider event handler for ADO.NET events, functionality of, 547
  - OleDbRowUpdatedEventArgs members, list of, 554
  - OleDbRowUpdatedEventHandler OleDb data provider event handler for ADO.NET events, functionality of, 547
  - OleDbRowUpdatingEventHandler OleDb data provider event handler for ADO.NET events, functionality of, 547
  - OleDbType property of OleDbParameter class, description of, 338
  - OnListChanged event handler, code for, 568–569
  - Open method of ADO.NET Connection class, description of, 285
  - Open property of ConnectionState enumeration, description of, 549
  - OpenFileDialog class, functionality of, 119
  - OpenfileDialog class of System.Windows.Forms, description of, 101
  - operands versus operators, 37
  - operator class member and inheritance, relationship between, 46
  - operators
    - definition of, 36–39
    - versus operands, 37
    - types of, 37
  - optimistic concurrency
    - in ADO.NET, 343–347
    - definition of, 341
    - role in OleDbDataAdapters and VS .NET, 196
  - optimistic locking, explanation of, 664–665
  - Oracle 8i databases, accessing with OleDb data providers, 645–647
  - Oracle 9i databases, using ODBC .NET data providers with, 647–651
  - Oracle connection string for ODBC, displaying, 291
  - Oracle databases, using ODBC .NET data providers with, 644–651
  - Order DataSet object, displaying for sample Web service, 524
  - OrderRetrievalService project
    - displaying proxy reference file for, 540
    - testing, 533
  - OrderRetrievalService reference, viewing for sample Web service, 528
  - OrderRetrievalService.asmx.cs sample Web service, displaying, 515–516
  - out parameters, using with methods, 54–56
  - OUTER JOINS, using with SQL, 680
  - Output member of ParameterDirection enumeration, description of, 586
  - output parameters, using with stored procedures, 588–589
- ## P
- <p> tag, description of, 357
  - PacketSize property of ADO.NET Connection objects,
    - description of, 284
  - Page property of ASP.NET DataGrid control, description of, 466
  - Page\_Load event, displaying for ViewGuestBook.aspx, 485–486
  - PageCount property of ASP.NET DataGrid control, description of, 466
  - PageIndexChanged event handler, adding to ASP.NET DataGrid controls, 491–492
  - PagerStyle modes, setting for DataGrid controls, 495
  - PageSize property
    - of ASP.NET DataGrid control, 466
    - setting, 495
  - paging, enabling at design-time, 491–494
  - ParameterDirection enumeration, members of, 585–586
  - ParameterDirection.Output, example of, 588
  - ParameterName property of OleDbParameter class, description of, 338
  - parameters
    - creating, 338
    - role in ADO.NET, 337–339
  - parent/child relationship, explanation of, 257–258
  - ParentRelation property of ADO.NET DataTable class, description of, 259
  - pessimistic concurrency
    - definition of, 341
    - example in ADO.NET, 346
  - pessimistic locking, explanation of, 664
  - phantoms, role in relational database isolation levels, 662
  - PictureBox class of System.Windows.Forms, description of, 101
  - PIs (processing instructions), role in XML documents, 371
  - PJ/NF, explanation of, 659–660

- Precision property of OleDbParameter class, description of, 338
  - Prefix property of XmlReader class, description of, 387
  - Preview Data option, using with SQL data adapters, 185
  - primary keys
    - role in relational databases, 653
    - role in setting DataColumn properties, 248
  - primary operators, examples of, 38
  - PrimaryKey property of ADO.NET DataTable class, description of, 259
  - printing classes of System.Windows.Forms, description of, 101
  - private accessibility modifier, description of, 33
  - Procedure property of SqlError class, description of, 352
  - ProcessingInstruction member of XmlNodeType enumeration, description of, 384
  - ProcessingInstruction node in XML, description of, 374
  - program logic, explanation of, 40
  - ProgressBar class of System.Windows.Forms, description of, 101
  - project templates
    - choosing for Data Form Wizard, 209
    - selecting for SQL data adapters, 171–172
    - selecting for Windows Forms application written in VS .NET IDE, 78–79
  - prologs, role in XML documents, 367–368
  - properties
    - adding to VS .NET IDE Windows Forms application, 97–100
    - setting for Windows Forms, 72–78
    - setting in VS .NET IDE Windows Forms application, 82–84
  - Property Builder, using with DataGrid controls, 468
  - property class member
    - functionality of, 56–58
    - and inheritance, 46
  - protected accessibility modifier, description of, 33
  - protected accessibility type for class members, 47
  - protected internal accessibility modifier, description of, 33
  - protected internal accessibility type for class members, 47
  - Provider property of ADO.NET Connection objects, description of, 284
  - proxy files
    - using with Web services, 529
    - viewing contents of, 540
  - public accessibility modifier, description of, 33
  - public accessibility type for class members, accessibility types and scopes for, 47
- ## Q
- queries, reading batches with DataReaders, 317–319
  - Query Builder
    - relaunching from CommandText property, 183
    - using with SQL data adapters, 176–178
  - Query Editor, launching, 668–669
  - query types, choosing for SQL data adapters, 174
  - quot XML built-in entities and references, description of, 370
- ## R
- radio buttons, adding to Windows applications with Windows.Forms namespace, 116
  - RangeValidator ASP.NET server-side control, description of, 457
  - Rank property of array class, description of, 26
  - Read Committed isolation level, definition of, 663
  - Read method
    - of ADO.NET DataReaders, 315
    - of System.Console class, 10
  - read-only variables, explanation of, 35–36
  - Read property of XmlReader class, description of, 388
  - Read Uncommitted isolation level, definition of, 663
  - ReadAttributeValue property of XmlReader class, description of, 388
  - ReadCommitted isolation level for transactions, descriptions, 343
  - reader and writer classes, role in System.Xml namespace, 376
  - ReadInnerXml property of XmlReader class, description of, 388
  - ReadLine method of System.Console class, description of, 10
  - ReadOnly property of ADO.NET DataColumn disconnected class, description of, 244
  - ReadState property of XmlReader class, description of, 387
  - ReadUncommitted isolation level for transactions, descriptions, 343
  - ReadXml method, functionality of, 405
  - ReadXmlSchema method
    - of ADO.NET DataSet class, 276
    - functionality of, 406
  - ReadXXXX property of XmlReader class, description of, 388
  - records
    - adding to tables with INSERT SQL statement, 311–313



- selecting, 610–611
  - RecordsAffected property
    - of ADO.NET DataReaders, 315
    - of OleDbRowUpdatedEventArgs, 554
  - recordsets
    - versus DataSets, 128–129, 274
    - filling DataAdapters with, 327
  - rectangular arrays, example of, 26
  - ref parameters, using with methods, 54–56
  - reference types, explanation of, 4, 21–25
  - Reference.cs file, contents of, 540–541
  - ReferenceEquals method of Object class,
    - description of, 12, 14–15
  - references
    - adding to ADO.MD, 603
    - adding to ADOX libraries, 599
    - adding to COM libraries, 595–596
    - adding to Microsoft.Data.Odbc.dll assembly, 615–616
    - adding to msadomd.dll library, 602
  - RegularExpressionValidator ASP.NET server-side control, description of, 457
  - RejectChanges method
    - of ADO.NET DataRow class, 252–253, 256
    - of ADO.NET DataSet class, 276
    - of ADO.NET DataTable class, 259
  - relational databases
    - and ADO.NET, 660–661
    - details of, 653–654
    - dirty reads in, 662
    - effect of deadlocks on, 665
    - nonrepeatable reads in, 662
    - and normalization, 654–660
    - phantoms in, 662
    - resources for, 666
    - role of cursors in, 660–661
    - role of isolation levels in, 662–663
    - role of locking in, 661–665
    - role of locking modes in, 664–665
    - role of sets in, 660–661
  - relational operators, examples of, 38
  - Relations property of ADO.NET DataSet class,
    - description of, 275
  - relationships between tables, creating with
    - Data Form Wizard, 213–215
  - ReleaseObjectPool method
    - of ADO.NET Connection class, 285
    - calling, 300
  - RemoveAll methods of XmlElement class,
    - description of, 398–400
  - RemoveAllAttributes methods of XmlElement class,
    - description of, 398
  - RemoveAttribute methods of XmlElement class,
    - description of, 398
  - RemoveAttributeAt methods of XmlElement class,
    - description of, 398
  - RemoveAttributeNode methods of
    - XmlElement class,
      - description of, 398
  - Repeatable Read isolation level, definition of, 663
  - RepeatableRead isolation level for transactions,
    - descriptions, 343
  - Repeater ASP.NET data-bound control,
    - description of, 464
  - ReplaceChild XML method, functionality of, 401
  - RequiredFieldValidator ASP.NET server-side control,
    - description of, 457
  - Reset method
    - of ADO.NET DataSet class, 276
    - of ADO.NET DataTable class, 259
  - return statement, functionality of, 45
  - ReturnValue member of ParameterDirection enumeration,
    - description of, 586
  - Reverse property of array class,
    - description of, 27–29
  - Rollback method of Transaction class,
    - description of, 342
  - Rollback Transaction, functionality of, 340
  - Rollback(SavePoint) transaction method in
    - Sql data providers, 347
  - root nodes
    - getting for XML documents, 399
    - moving to, 417–418
    - role in XML, 359
  - Row property of
    - OleDbRowUpdatedEventArgs,
      - description of, 554
  - RowChanged event of DataTable event,
    - description of, 560, 563–564
  - RowChanging event of DataTable event,
    - description of, 561, 563–564
  - RowDeleted event of DataTable event,
    - description of, 561, 564–565
  - RowDeleting event of DataTable event,
    - description of, 561, 564–565
  - RowFilter property of ADO.NET DataView class,
    - description of, 277
  - rows
    - adding to ASP.NET tables at design-time,
      - 504–505, 507
    - adding to DataTables, 270
    - deleting from DataTables, 271
    - using SELECT DISTINCT SQL statement with, 611
  - Rows property of ADO.NET DataTable class,
    - description of, 259
  - RowState property of ADO.NET DataRow disconnected class,
    - description of, 251
  - RowUpdated event handler, code for, 556–557
  - RowUpdating event handler, code for, 557
  - Run method of Windows.Forms.Application class,
    - description of, 102
- ## S
- Sales by Year stored procedures in Northwind database,
    - code for, 306–307
  - Save methods, role in DOM implementations, 395
  - SaveFileDialog class, functionality of, 119

- SaveFileDialog class of
  - System.Windows.Forms, description of, 101
- savepoints
  - definition of, 340
  - using in SQL Server, 347–349
- Save(SavePointName) transaction method in
  - Sql data providers, 347
- sbyte C# type alias, details of, 18
- scalability in C#, explanation of, 4–5
- Scale property of OleDbParameter class,
  - description of, 338
- schemas, role in XML, 364–366
- SearchButton\_Click method, code for, 272
- SELECT COUNT statement, using with SQL, 674
- SELECT DISTINCT SQL statement, selecting records with, 611, 672
- Select method
  - of ADO.NET DataTable class, 259
  - using with XPathNavigator class, 419–420
- SELECT statements
  - examples of, 670–675
  - executing with SqlDataAdapter, 321
  - saving as stored procedures with ADO.NET, 578–579
- SelectCommand property
  - of OleDbDataAdapter Command, 322
  - of OleDbDataAdapters, 322
  - setting in SQL data adapters, 182
- SelectedIndex property of ASP.NET DataGrid control, description of, 466
- SelectedItem property of ASP.NET DataGrid control, description of, 466
- SELECT . . . FORM XML clause, using with SQL, 682
- semicolon (;)
  - advisory about, 46
  - role in XML-document references, 370
- Serializable built-in attribute, description of, 31
- Serializable isolation level, explanation of, 343, 663
- serialization, explanation of, 377
- server controls, role in Web Forms, 438
- server cubes
  - getting dimensions of, 605–607
  - getting from FoodMart 2000 database, 604–605
  - role in OLAP, 600
- Server Explorer
  - adding connections with, 156–159
  - creating SQL data adapters with, 170–171
  - displaying Northwind database in, 288
  - displaying views available in, 592
  - introduction to, 154–156
  - launching, 427
  - launching to create stored procedures, 573–574
  - managing and viewing data with, 159–161
  - using DataSet objects with, 430–432
- servers, adding to Server Explorer, 156
- ServerVersion property of ADO.NET
  - Connection objects, description of, 284
- Service1.asmx file in Web services
  - explanation of, 514
  - running, 521
- SetAttribute method of XmlElement class,
  - description of, 398
- SetAttributeNode method of XmlElement class,
  - description of, 398
- sets, role in relational databases, 660–661
- SetValue property of array class,
  - description of, 27
- SGML (Standard Generalized Markup Language), explanation of, 355–356
- shift operators, examples of, 38
- short C# type alias, details of, 18
- ShowFooter property of ASP.NET DataGrid control,
  - description of, 466
- ShowHeader property of ASP.NET DataGrid control,
  - description of, 466
- SignificantWhitespace member of XmlNodeType enumeration,
  - description of, 384
- simple types, explanation of, 17–19
- single-dimensional array example, 25–26
- single quote (') entity in XML, representing, 370
- Size property of OleDbParameter class,
  - description of, 338
- sizeof operator, explanation of, 39
- Skip method, using with XML nodes, 386
- Skip property of XmlReader class,
  - description of, 388
- SOAP (Simple Object Access Protocol), passing XML nodes with, 537–539
- Solution Explorer, using with Web services, 513
- Sort property
  - of ADO.NET DataView class, 277
  - of array class, 27–29
- source code
  - documenting, 67
  - reviewing for SQL data adapters, 186–189
- Source property
  - of OleDbError class, 351
  - of SqlError class, 352
- Source property of InfoMessageEvents,
  - description of, 548
- SourceColumn property of OleDbParameter class,
  - description of, 338
- SourceVersion property of OleDbParameter class,
  - description of, 338
- Splitter class of System.Windows.Forms,
  - description of, 101
- Sql Command objects, creating, 303–305
- SQL data adapters
  - adding DataAdapter components to, 172–173
  - adding DataGrid controls to forms for, 172
  - choosing data connection pages for, 173

- choosing query types for, 174
- creating with Data Adapter Configuration Wizard, 171
- creating with Server Explorer, 170–171
- generating SQL statements for, 175
- introduction to, 169–170
- previewing data for, 186
- reviewing source code for, 186–189
- selecting project templates for, 171–172
- setting and reviewing DataAdapter properties for, 180–184
- setting SelectCommand in, 182
- using Query Builder with, 176–178
- Sql data provider for ADO.NET Command class for, 135
- connecting to SQL Server databases with, 288–289
- Connection class for, 134
- for DataAdapter class, 136
- transaction methods in, 347
- using savepoints with, 347–349
- SQL data providers, executing mySP stored procedure with, 584–585
- SQL queries
  - executing in Web Forms, 497–499
  - executing with ADO.NET Command object, 134–135
- SQL Server
  - connecting through ODBC, 291–292
  - connecting through SqlClient data provider, 292
  - connecting with OleDb data provider, 288
  - using savepoints with, 347–349
- SQL Server Connection objects, constructing for ADO.NET applications, 148
- SQL Server databases
  - accessing with SqlCommand, 304–305
  - getting tables from, 608–609
  - using DataReaders with, 316–317
- SQL Server exception, advisory about, 150–151
- SQL statements
  - generating for SQL data adapters, 175
  - testing, 667
- SQL (Structured Query Language)
  - aliases, 679–680
  - conditional statements used in, 673
  - CREATE TABLE statement with, 676–677
  - CREATE VIEW statement, 681
  - CROSS JOINS, 680
  - DELETE statement with, 676
  - DROP TABLE statement with, 677
  - GROUP BY clause, 674–675
  - HAVING clause with, 675
  - INNER JOINS, 680
  - INSERT statement, 678–679
  - JOIN queries with, 679–680
  - OUTER JOINS, 680
  - resources for, 682–683
  - SELECT COUNT statement, 674
  - SELECT statements, 670–675
  - SELECT . . . FORM XML clause, 682
  - SUM and AVG functions, 673
  - TRUNCATE TABLE statement with, 677–678
  - UPDATE statement with, 675–676
  - using views with, 680–682
- SQL View option, choosing, 668
- SqlClient data provider, connecting to SQL Server through, 292
- SqlCommand
  - accessing SQL Server databases with, 304–305
  - calling stored procedures with, 306
- SqlCommandBuilder, using, 335–337
- SqlConnection class object
  - role in VS .NET, 166, 169, 171
  - viewing in form designer, 180
- SqlDataAdapters
  - connecting to ADO.NET databases with, 148
  - displaying Customers tables data in DataGrids with, 326
  - executing SELECT statements with, 321
  - viewing in form designer, 180
- SqlError class, properties of, 352
- SqlException class, catching errors with, 351–352
- SqlParameter class, using with ADO.NET stored procedures, 585
- SqlState property of OleDbError class, description of, 351
- start and end tags, role in XML documents, 368–369
- State property
  - of ADO.NET Connection objects, 284
  - of SqlError class, 352
- StateChange ADO.NET event
  - testing, 551–552
  - writing event handlers for, 550
- StateChange event, using with Connection object and ADO.NET events, 547–548
- StateChangeEventHandler ADO.NET event, functionality of, 546
- StatementType property of OleDbRowUpdatedEventArgs, description of, 554
- static constructor class member
  - functionality of, 49–51
  - and inheritance, 46
- static cursors, explanation of, 660
- static methods, definition of, 52
- static variables, explanation of, 35–36
- Status property of OleDbRowUpdatedEventArgs, description of, 554
- stored procedures
  - accepting input parameters with, 581–582
  - adding parameters to, 585–587
  - adding subtotal listings to, 309
  - calling with Command objects, 306–309
  - creating with ADO.NET, 573–580, 576–577

- stored procedures (*continued*)
    - editing, deleting, and executing with ADO.NET, 575–578
    - executing with ADO.NET Command object, 134–135
    - executing and reading results in ADO.NET, 307–308
    - executing from VS.NET, 580–583
    - executing programmatically with ADO.NET, 583–589
    - returning data with, 578
    - returning values from, 586
    - using output parameters with, 588–589
    - viewing, 574
  - StoredProcedure member of CommandType enumeration, description of, 305
  - string conversion, performing with ToString method of Object class, 16
  - StringWriter class, definition of, 351
  - <strong> tag, description of, 357
  - struct types, explanation of, 19–20
  - stylesheets, expressing with XSL, 402–404
  - SUM function, using in SQL, 673
  - switch statement
    - functionality of, 41–42
    - versus goto statement, 44
  - switches, exiting with break statement, 44
  - Sybase databases, using ODBC .NET data providers with, 651–652
  - synchronous Web services, explanation of, 539
  - System.Console class, functionality of, 10–11
  - System.Data assembly in ADO.NET, contents in IL DASM utility, 129–131
  - System.Data namespace
    - event handler and event argument classes defined in, 546
    - hierarchy in ADO.NET, 232–233
    - interfaces in ADO.NET, 238–239
  - System.Data.Common namespace in ADO.NET
    - diagram of, 234
    - functionality of, 131, 237, 239–240
  - System.Data.dll assembly adding namespace references to, 139
  - System.Data.OleDb namespace
    - functionality of, 131
    - hierarchy in ADO.NET, 235–236
  - SystemData.SqlClient namespace in ADO.NET, functionality of, 131–132
  - System.Data.SqlTypes namespace hierarchy in ADO.NET, diagram of, 234–235
  - SystemException type in C#, description of, 65
  - System.String class, getting type of, 12–13
  - SystemWeb namespace, role in .NET Framework library, 457–458
  - System.Web.Services namespace, role in .NET Framework library, 459
  - System.Web.UI namespace, role in .NET Framework library, 458
  - System.Web.UI.HtmlControls namespace, role in .NET Framework library, 458
  - SystemWeb.UI.WebControls namespace, role in .NET Framework library, 459
  - System.Windows.Forms common classes, list of, 101
  - System.Windows.Forms namespace
    - adding menu items to applications with, 115–118
    - adding toolbars to applications with, 103–115
    - common Control class in, 103
    - creating dialogs with, 118–121
    - functionality of, 100
  - System.Xml namespace
    - adding reference for System.Xml.dll assembly to, 379–380
    - role in Microsoft .NET and XML, 375–377
  - System.Xml.dll assembly, adding reference to System.Xml namespace, 379–380
  - System.Xml.Schema namespace, role in Microsoft .NET and XML, 377
  - System.Xml.Serialization namespace, role in Microsoft .NET and XML, 377
  - System.Xml.XPath namespace, role in Microsoft .NET and XML, 377–378
  - System.Xml.Xsl namespace, role in Microsoft .NET and XML, 378
- ## T
- tab-delimited text files, using ODBC to read data from, 293
  - Table control, introduction to, 503–509
  - table information, reading with TableDirect CommandType, 309–311
  - Table Mappings dialog box, displaying for SQL data adapters, 183–184
  - Table object of ADOX, functionality of, 599
  - Table property
    - of ADO.NET DataRow disconnected class, 251
    - of ADO.NET DataView class, 277
  - table schemas, getting, 609–610
  - <table> tag, description of, 357
  - TableDirect CommandType, reading table information with, 309–311
  - TableDirect member
    - of CommandType enumeration, 305
  - TableMapping property
    - of OleDbRowUpdatedEventArgs, 554
  - TableMapping property, making DataAdapter connections through, 188
  - TableMappings property
    - of DataAdapter class, 332
    - of OleDbDataAdapters, 322
  - TableName property of ADO.NET DataTable class, description of, 259
  - tables
    - adding records with INSERT SQL statement, 311–313
    - adding to View Designer, 590

- choosing with Data Form Wizard, 212
  - creating and adding data to, 648–649
  - creating at design-time with ASP.NET, 504–507
  - creating with DataTables and DataColumns, 249–250
  - displaying with Data Form Wizard, 215–216
  - getting from SQL Server databases, 608–609
  - Tables property of ADO.NET DataSet class, description of, 275
  - tags in HTML
    - examples of, 358
    - explanation of, 356
    - list of, 357
    - navigating with Document Outline viewer, 448
  - <td> tag, description of, 357–358
  - ternary operators, definition of, 37
  - text file databases, accessing, 633–641
  - text files
    - accessing, 637–641
    - connecting with ODBC data providers for ADO.NET, 293–294
    - defining formats and column settings for, 639
    - exporting to Access tables, 633–637
  - Text member
    - of CommandType enumeration, 305
    - of XmlNodeType enumeration, 384
  - Text node in XML, description of, 375
  - TextBox ASP.NET data-bound control, description of, 464
  - TextBox class of System.Windows.Forms, description of, 101
  - TextDB.txt file, accessing, 640
  - Thanks.aspx item, adding to MyGuestBook ASP.NET application, 484, 486–487
  - three slashes (///), using with comments in source code, 67
  - TILs (transaction isolation levels), determining for relational databases, 662–663
  - Timer class of System.Windows.Forms, description of, 101
  - <title> tag, description of, 357
  - toolbar buttons
    - adding images with Windows.Forms namespace, 107–113
    - writing event handlers with Windows.Forms namespace, 113–115
  - ToolBar controls, adding with Windows.Forms namespace, 104–105
  - toolbars, adding to applications with Windows.Forms namespace, 103–115
  - ToString method of Object class, description of, 12, 16
  - <tr> tag, description of, 357
  - Transaction classes, methods of, 342
  - transaction processing, beginning, 340
  - transactions
    - in ADO.NET, 342–343
    - introduction to, 339–341
  - Transform XSL method, functionality of, 403–404
  - trees in XSLT, definition of, 403
  - TreeView class of System.Windows.Forms, description of, 101
  - TRUNCATE TABLE statement, using with SQL, 677–678
  - try . . . catch blocks, using, 65
  - Type class
    - and inheritance, 46
    - usage of, 12
  - type conversions, performing in C#, 29–31
  - type testing operators, examples of, 38
  - typed DataSet objects, using in VS .NET, 199–207
  - typed DataSets
    - in ADO.NET, 276–277
    - advantages of, 204
    - generating from existing XML schemas, 430–432
  - typeof operator, explanation of, 39
  - types
    - definition of, 4
    - introduction to, 17–18
  - typesafety feature of C#, explanation of, 3–4
- ## U
- uint C# type alias, details of, 18
  - ulong C# type alias, details of, 18
  - unary (-) operators, using with for loop statements, 42
  - unary (++) operators, using with for loop statements, 42
  - unary operators
    - definition of, 37
    - examples of, 38
  - unboxing, definition of, 30–31
  - Unchanged member of ADO.NET DataRowState enumeration, 257
  - unchecked operators, explanation of, 38–39
  - Unique property of ADO.NET DataColumn disconnected class, description of, 244
  - UniqueConstraint class in ADO.NET System.Data namespaces, description of, 237
  - UniqueID property of ASP.NET DataGrid control, description of, 466
  - unmanaged code, definition of, 3, 594
  - Unspecified isolation level for transactions, descriptions, 343
  - untyped DataSets in ADO.NET, description of, 276–277
  - Update method
    - adding data with, 328–330
    - calling for ADO.NET events, 557–558

- Update method (*continued*)
    - deleting data with, 331
    - of OleDbDataAdapters, 323
    - using with databases, 327–331
  - UPDATE statement, using with SQL, 675–676
  - UpdateCommand property
    - of OleDbDataAdapter Command, 322
    - of OleDbDataAdapters, 322
  - UpdateDataSet method, generating with
    - Data Form Wizard, 226
  - UpdateDataSource method, generating with
    - Data Form Wizard, 226
  - URI (Universal Resource Identifier),
    - explanation of, 363
  - User ASP.NET server-side controls,
    - explanation of, 457
  - UserAppDataPath method of
    - Windows.Forms.Application class, description of, 102
  - ushort C# type alias, details of, 18
  - Using System; line of “Hello, C# World!” program, examining, 8
- V**
- valid XML documents, explanation of, 367–372
  - Validation ASP.NET server-side controls, list of, 457
  - Value property
    - of OleDbParameter class, 338
    - role in getting XML node information, 381–382
    - of XmlReader class, 387
  - value types, explanation of, 4, 17
  - values
    - assigning to variables, 32
    - representing with enum data types, 20–21
  - Values property of FillEventArgs,
    - description of, 553
  - variable access modifiers, explanation of, 32–33
  - variables
    - explanation of, 32–36
    - using with property members, 56
  - versioning control in C#, explanation of, 4–5
  - View Code option in Form Designer, using, 84–85
  - View Wizard Results page, displaying for SQL adapters, 179
  - ViewGuestBook.aspx item, adding to MyGuestBook ASP.NET application, 484–486
  - ViewGuestBook.aspx, source code for opening of, 482
  - views
    - choosing with Data Form Wizard, 212
    - creating, 589–592
    - executing from VS.NET, 592
    - retrieving data from programmatically, 593–594
    - saving, 591
    - using with ADO.NET, 588–594
    - using with SQL, 680–682
  - VirtualItemCount property of ASP.NET
    - DataGrid control, description of, 466
  - visual data components, 162–166
  - VS .NET IDE Windows Forms
    - application
      - adding classes to, 92–96
      - adding controls to, 79–82
      - adding event handlers to, 89–91
      - adding methods and properties to, 97–100
      - building and running, 91–92
      - examining code in, 84–89
      - selecting project template for, 78–79
      - setting properties in, 82–84
  - VS (Visual Studio) .NET
    - adding connection events at design-time with, 549
    - adding OleDbDataAdapters to, 192–197
    - adding server-side controls with, 460
    - benefits of using with ASP.NET, 436
    - creating ADO.NET projects in, 153–154
    - creating connection components with, 166–167
    - creating console-based applications with, 142–145
    - creating stored procedures with, 573–574
    - creating Web services in, 512–521
    - developing ASP.NET applications with, 450–455
    - executing stored procedures from, 580–583
    - executing views from, 592
    - filling DataGrid controls in, 197–198
    - generating typed DataSets with, 199–204
    - populating DataSet objects in, 197–198
    - returning values from stored procedures with, 586
    - role of connection strings in, 167–169
    - role of data connections in, 166–169
    - role of DataView objects in, 207–208
    - understanding typed DataSets in, 199–207
    - using Data Form Wizard with, 208–223
    - using DataSet components with, 199–208
    - using DataView components with, 199–208
    - using OleDb data provider with, 165
    - using Server Explorer with, 154–159
    - using visual data components with, 162–166
    - writing ADO.NET applications with, 145–151
- W**
- W3C, Web site for, 357
  - WaitHandle method, making asynchronous calls to Web services with, 541–542
  - warnings, listening to, 352–353
  - Web applications
    - creating with ASP.NET, 438–442
    - definition of, 438
  - Web-based client applications, creating, 525

- Web Forms
  - adding, editing, and deleting data in, 496–503
  - adding new records to, 499–500
  - adding server-side controls to, 460
  - adding Web controls to, 442–444
  - binding at design-time, 472–478
  - executing SQL queries in, 497–499
  - explanation of, 437–438
- Web Forms controls as server-side controls, explanation of, 459–462
- Web methods, adding to Web services, 519
- Web services
  - accessing, 530
  - adding database support to, 516–517
  - adding Web methods to, 519
  - adding Web references to, 526–527
  - attributes of, 511–512
  - creating clients for, 525–539
  - creating consumers for, 525–539
  - creating in VS .NET, 512–521
  - creating method for populating database orders in, 532
  - displaying client event handler for execution of, 538
  - displaying client event handlers for execution of, 535
  - displaying files in, 514
  - displaying for local servers, 528
  - executing asynchronously, 539–542
  - explanation of, 438, 511
  - increasing functionality of, 531–536
  - and .NET Framework, 512
  - order retrieval example of, 530
  - passing XML to, 537–539
  - testing, 521–524
  - using discovery files with, 529
  - using proxy files with, 529
  - using WSDL files with, 529
  - viewing code in, 515
  - viewing OrderRetrievalService reference for, 528
  - writing to databases with, 531
- Web sites
  - C# Corner, 3
  - C# editors, 5
  - for DOM, 372
  - locking and concurrency levels, 665
  - Mono Project, 2
  - MySQL database, 621
  - normal forms, 660
  - ODBC .NET data providers, 613
  - Odbc .NET Software Development Kit, 162
  - relational databases, 666
  - role of design-time versus run-time development in, 69–70
  - SQL (Structured Query Language), 682–683
  - W3C, 357
- Web.config file in Web services, explanation of, 514
- WebForm1.aspx page, displaying, 440
- WebService and WebMethod attributes, adding descriptions to, 522
- WebService attribute, setting namespace and description properties for, 524
- Weight property of ASP.NET DataGrid control, description of, 466
- well-formed XML documents, explanation of, 361, 367
- while loop statement, functionality of, 42
- white spaces, role in XML documents, 372
- Whitespace member of XmlNodeType enumeration, description of, 384
- Windows applications
  - adding menu items with Windows.Forms namespace, 115–118
  - creating for toolbar example in Windows.Forms namespace, 103–104
- Windows Forms
  - adding controls to, 73–74
  - adding events to, 74–75
  - advantages of, 69
  - creating, 71–72
  - creating final code for example of, 75–78
  - sample output of, 77–78
  - setting properties for, 72–78
  - writing applications from command line, 70–78
- Windows Forms application written with VS .NET IDE
  - adding classes to, 92–96
  - adding controls to, 79–82
  - adding event handlers to, 89–91
  - adding methods and properties to, 97–100
  - building and running, 91–92
  - examining code in, 84–89
  - selecting project template for, 78–79
  - setting properties in, 82–84
- Windows.Forms namespace
  - adding menu items to applications with, 115–118
  - adding toolbars to applications with, 103–115
  - common Control class in, 103
  - creating dialogs with, 118–121
  - functionality of, 100
- Windows.Forms.Application class, methods of, 102
- wiring protocol, role in Web services, 512
- WorkStationId property of ADO.NET Connection objects, description of, 284
- Write method of System.Console class, description of, 10
- WriteLine method of System.Console class, description of, 10
- WriteNode XML method, functionality of, 389
- WriteXml method
  - of ADO.NET DataSet class, 276
  - functionality of, 406–408

- WriteXmlSchema method
  - of ADO.NET DataSet class, 276
  - functionality of, 408–409
- WSDL (Web Services Description Language)
  - files, using with Web services, 529
- X**
- XHTML (Extensible Hypertext Markup Language), explanation of, 366
- XLang property of XmlReader class,
  - description of, 387
- XML data
  - displaying in DataSet format, 411–413
  - loading with DataSet objects, 411
- XML Designer
  - generating XML with, 426–427
  - using Generate Dataset option of, 431–432
- XML Designer, launching, 422
- XML documents
  - adding nodes, 398–399
  - adding to projects, 420–421
  - closing, 387–388
  - components of, 367–372
  - generating ADO.NET typed DataSets
    - from, 430–432
  - getting node information about, 381–384
  - getting root nodes for, 399
  - inserting fragments of, 395–396
  - inserting XML fragments into, 401–402
  - loading with Load and LoadXml methods, 411
  - navigating nodes in, 384–385
  - reading, 412
  - reading with DataSet class, 405–406
  - reading with XmlReader class, 381
  - reading with XPathNavigator class, 418–419
  - removing and replacing nodes in, 399–401
  - role of attributes in, 371
  - role of CDATA sections in, 369
  - role of character and entity references in, 370
  - role of DOCTYPE declaration in, 368
  - role of empty elements in, 370–371
  - role of processing instructions in, 371
  - role of prologs in, 367–368
  - role of start and end tags in, 368–369
  - role white spaces in, 372
  - sample tree structure implement-ation of, 373
  - saving data from DataSets to, 413–414
  - transforming, 404
  - using Close method with, 392
  - validating with DTDs, 364–366
  - writing attributes for, 389
  - writing comments to, 389
  - writing elements to, 389
  - writing strings to, 389
  - writing to console, 391
  - writing with DataSet class, 406–409
- XML DOM tree representation, diagram of, 374
- XML elements, explanation of, 365–366, 369
- XML (eXtensible Markup Language)
  - attributes in, 361–362
  - case-sensitivity of, 361
  - characteristics of, 361–362
  - example of, 359–360
  - functionality of, 359
  - and Microsoft .NET, 375–380
  - navigating in, 415–420
  - passing to Web services, 537–539
  - role in ADO.NET, 124, 127
  - role of comments in, 369
  - role of DTDs and schemas in, 364–366
  - role of URIs in, 363
- XML items, writing, 389–390
- XML names, transferring to ADO.NET
  - objects, 393
- XML namespaces, explanation of, 363–364
- XML .NET API, explanation of, 378–379
- XML nodes
  - adding attributes to, 402
  - description of, 374–375
  - navigating, 384–385
  - reading and displaying, 419–420
  - reading for XML documents, 382–383
  - removing and replacing, 399–401
  - searching for, 386
  - using GetAttributes method with, 385–386
- XML parser, explanation of, 363–366
- XML-related technology, defining, 355–358
- XML schema items
  - adding to projects, 422–427
  - deleting from projects, 424
- XML schema toolbox, displaying, 422–423
- XML schemas
  - browsing for, 431
  - creating, 406
  - explanation of, 364–366
  - generating, 420
  - generating DataSet objects from, 430–432
  - generating for database tables, 429
- XML source trees, definition of, 402–403
- XML support, management by ADO.NET ver-  
sus ADO, 129
- XML trees, opening documents as, 402–403
- XML versions of documents, defining, 360
- XmlConvert class in System.Xml namespace,
  - explanation of, 377
- XmlConvert class, example of, 393
- XmlDataDocument class
  - diagram of, 409
  - explanation of, 376
  - functionality of, 415
  - loading data from, 411
  - reading and writing data with, 410
- XmlDataDocument events, using with  
ADO.NET events, 565–568
- XmlDataDocumentSample.cs, code and out-  
put for, 412–413



- XmlDeclaration member of XmlNodeType enumeration, description of, 384
- XmlDocument class
  - role in DOM implementation, 394–395
  - role in XML .NET API, 378
  - in System.Xml namespace, 376
- XmlDocumentFragment class
  - role in DOM implementation, 395–396
  - in System.Xml namespace, 376
- XmlDocumentType class in System.Xml namespace, explanation of, 376
- XmlElement class, role in DOM implementation, 397–402
- XmlException class, role in System.Xml namespace, 377
- XmlLinkedNode class, role in System.Xml namespace, 377
- XmlNamespaceManager class, role in System.Xml namespace, 377
- XmlNode class in System.Xml namespace, explanation of, 375–376
- XmlNode class
  - functionality of, 401
  - role in DOM implementation, 394
- XmlNode methods, functionality of, 396
- XmlNodeChangedEventHandler, code for, 565–566
- XmlNodeList class, role in System.Xml namespace, 377
- XmlNodeType enumeration, role in getting XML node information, 382–384
- XmlReader and XmlWriter classes, role in System.Xml namespace, 376
- XmlReader class
  - diagram of, 380
  - properties of, 387–388
  - role in XML .NET API, 378
- XmlSpace property of XmlReader class, description of, 387
- XmlTextWriter method, example of, 389–390
- XmlWriter class
  - diagram of, 388
  - example of, 390–392
  - properties of, 389
  - role in XML .NET API, 378
- XmlWriterSample.cs class, output of, 391–392
- xmlWriterTest document, code for, 390–392
- XPathDocument class in System.Xml.XPath namespace, explanation of, 378
- XPathExpression class in System.Xml.XPath namespace, explanation of, 378
- XPathIterator class in System.Xml.XPath namespace, explanation of, 378
- XPathNavigator class
  - functionality of, 416
  - Move methods of, 416–417
  - reading XML documents with, 418–419
  - searching with, 419–420
  - in System.Xml.XPath namespace, 378
- XSD (XML Schema Definition) types versus CLR data types, 393
- xsd:schema statements, explanation of, 364–365
- XSL (Extensible Stylesheet Language), functionality of, 402
- XSLT (XSL Transformation), explanation of, 402–404
- XslTransformation class in .NET Framework, functionality of, 403–404