Chapter 8

Using TableAdapters

After completing this chapter, you will be able to:

- Create a TableAdapter class with the TableAdapter Configuration Wizard and the Server Explorer.
- Create a TableAdapter object in Visual Studio and at run time.
- Configure a TableAdapter class.
- Add a query to a TableAdapter.
- Configure a TableAdapter object.
- Execute the query methods of a TableAdapter.

In earlier chapters, you've seen the TableAdapters created by the Data Source Configuration Wizard. If you've examined the scaffolding code in some of the exercises, you've even seen them instantiated and used. In this chapter, we'll examine this new class of objects in detail.

Understanding TableAdapters

The TableAdapter is new to Microsoft ADO.NET 2.0, or more properly, to Microsoft Visual Studio 2005, because the TableAdapter class isn't technically part of ADO.NET or the Microsoft .NET Framework. Instead, TableAdapters classes are created by Visual Studio at design time as part of Typed DataSets.

TableAdapters combine the functionality of the Command and DataAdapter objects and link that functionality to the structure of a Typed DataSet. As we'll see, the link is reciprocal: When you change the SQL query that defines a TableAdapter's Fill method, the DataSet Designer alters the DataTable definition, and vice versa.

Tip To see the files generated by Visual Studio for a Typed DataSet, click the Show All Files button on the Solution Explorer toolbar.

Like a DataAdapter, the primary purpose of a TableAdapter is to synchronize the data contained in a DataSet with the data in the data source. It provides one or more methods to fill the DataSet from the data source and to update the data source with changes made to the DataSet.

The class is defined inside the <DataSetName>.Designer.vb file, where <DataSetName> is, of course, the name of the Typed DataSet. TableAdapters are defined as a separate <DataSet>TableAdapters namespace at the end of the file.

Note Partial class declarations, new to Microsoft .NET Framework 2.0, allow you to split the definition of a class across multiple files. The most immediate effect of this split is that the code generated by Visual Studio can be separated from user-written code.

Each TableAdapter is defined as a separate partial class within the TableAdapters namespace. The Partial declaration means that you can easily extend the class definition in a separate code file, insulating your extensions from the actions of the DataSet Designer.

At the simplest level, you can think of TableAdapters as sophisticated DataAdapters. Because they encapsulate the Connection object, they save you a few lines of code for instantiating, opening and closing a Connection object, and their syntax is simpler. But that's only the beginning.

As we've seen, the DataAdapter exposes only four commands, Select, Update, Insert and Delete, each of which corresponds to a single SQL command. Because the TableAdapter class is generated at design time, it can support any number of SQL statements, each exposed in a simple method call, and the DataSet Designer provides a simple interactive interface for adding queries at design time.

Furthermore, because you can extend the class definition of a TableAdapter, you can add any functionality you require. In fact, TableAdapters are the first step toward turning a Typed DataSet into a complete object that combines data and functionality.

Strict object-oriented programming (OOP) methodology defines an object as a single entity, so the DataSet/TableAdapter combination doesn't quite fit the definition. However, the two object types are much more closely linked than is typical with the .NET Framework. Remember that changes made to the DataSet will update the Fill method of the TableAdapter and vice versa.

Creating TableAdapters

There are two steps to creating a TableAdapter: First, you must create the class itself (or, more often, have Visual Studio create it for you), and then you must instantiate it, as with any .NET Framework object.

Creating TableAdapter Classes

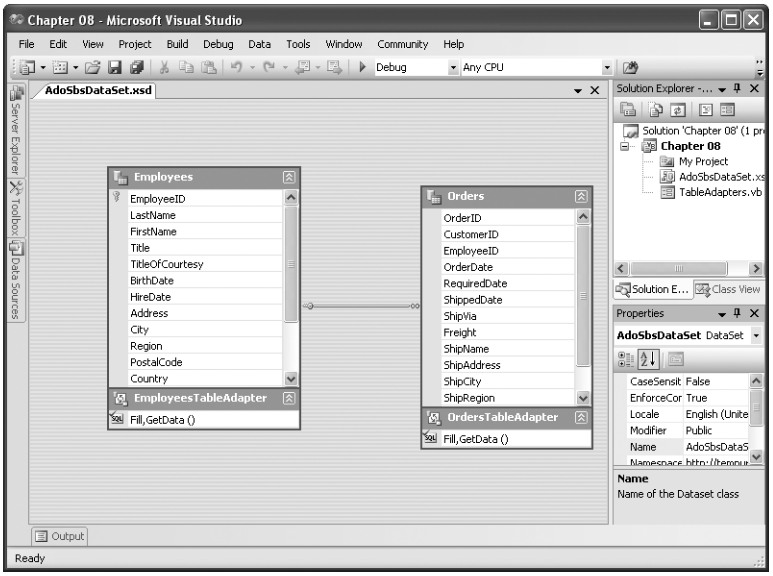
TableAdapter classes are defined by default whenever you use the Data Source Configuration Wizard to define a Typed DataSet. They can also be defined in the DataSet Designer.

Like Typed DataSets, it is theoretically possible to code a TableAdapter class from scratch, but again, it's difficult to imagine a situation in which this would be a sensible action.

Create a TableAdapter by Using the TableAdapter Configuration Wizard

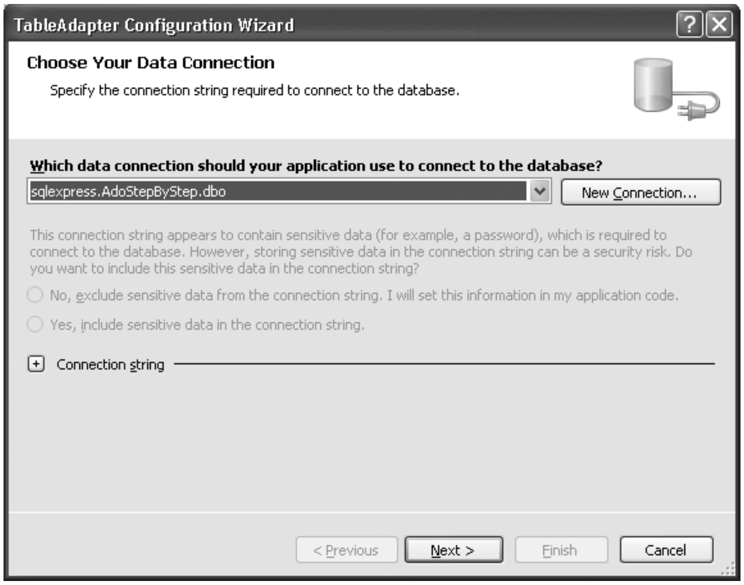
1. Open the Chapter 08 – Start project in Visual Studio, and double-click AdoSbs-DataSet.xsd in the Solution Explorer.

   Visual Studio opens the DataSet in the DataSet Designer.



2. Drag a TableAdapter from the Toolbox onto the DataSet Designer.

   The DataSet Designer opens the TableAdapter Configuration Wizard.


TableAdapter Configuration Wizard
Choose Your Data Connection
Specify the connection string required to connect to the database.

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

sqlexpress.AdoStepByStep.dbo

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

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

Yes, include sensitive data in the connection string.

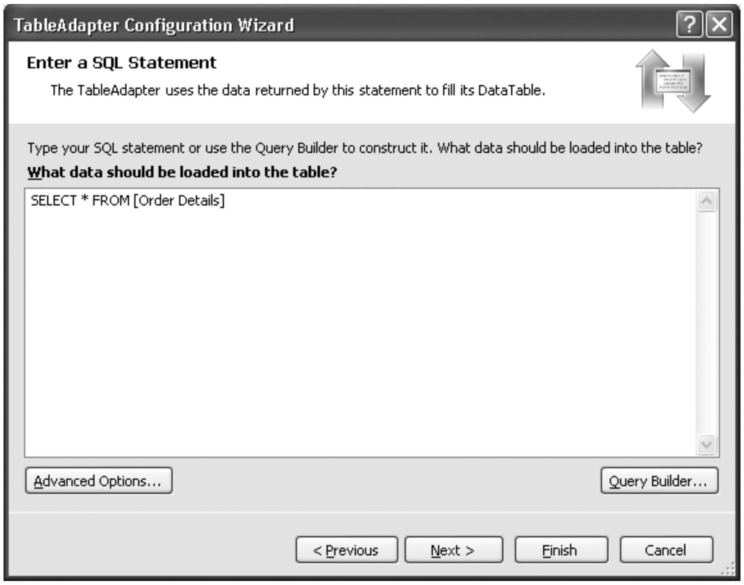
3. Make sure that the connection suggested by the wizard points to the AdoStepByStep sample database. If it doesn't, click the New Connection button, and create a new connection as described in Chapter 1, "Getting Started with ADO.Net." When you are finished, click Next.

   The wizard suggests saving the connection string to the application configuration file.
4. Clear the check box, if necessary, and then Click Next.

   The wizard displays a page requesting the command type.
5. Accept the default Use SQL Statements option, and click Next.

   The wizard displays a page requesting the SQL statement to be used to fill the table.
6. Enter the following SQL statement:

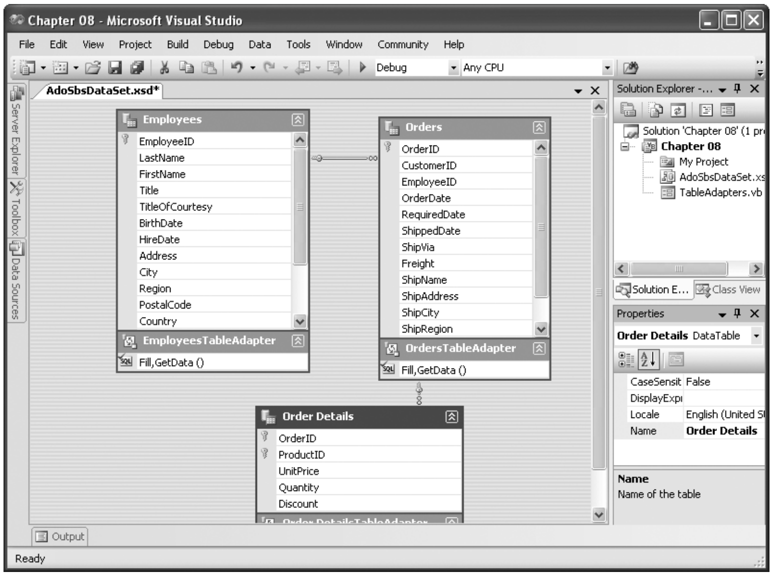
```
SELECT * FROM [Order Details]
```


TableAdapter Configuration Wizard
Enter a SQL Statement
The TableAdapter uses the data returned by this statement to fill its DataTable.
Type your SQL statement or use the Query Builder to construct it. What data should be loaded into the table?
What data should be loaded into the table?
Advanced Options...
Query Builder...
< Previous
Next >
Finish
Cancel

```
SELECT * FROM [Order Details]
```

7. Click Finish.

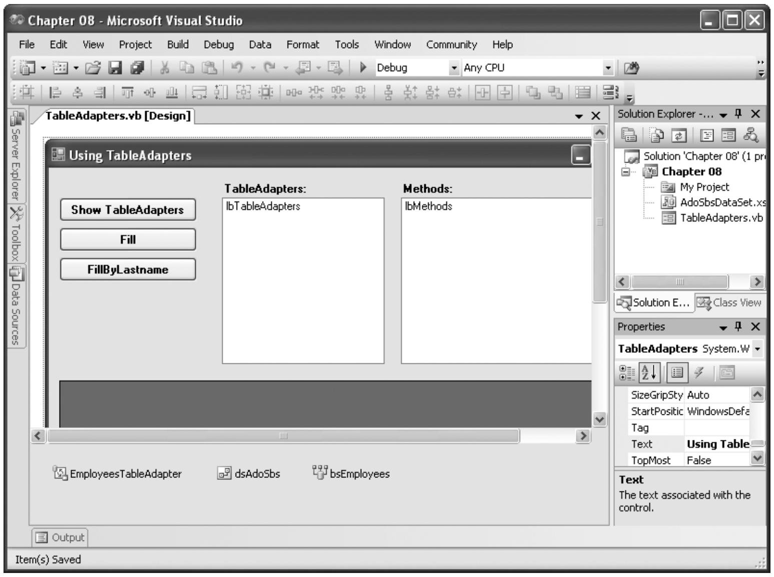
   The TableAdapter Configuration Wizard adds the DataTable and its TableAdapter to the DataSet. Notice that it also creates the Relation between Order Details and Orders.



8. Save the DataSet.

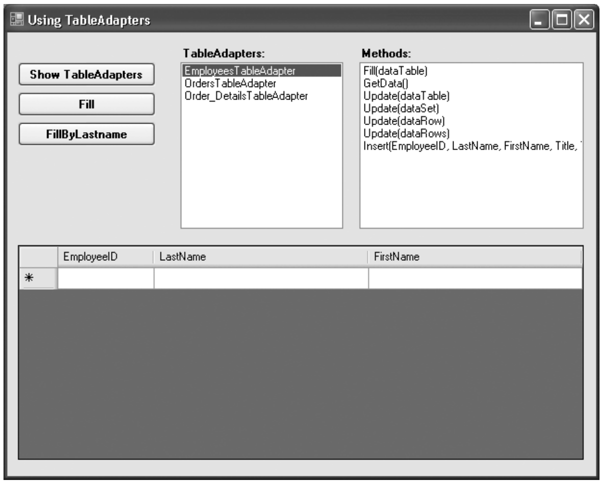
9. Double-click TableAdapters.vb (or TableAdapters.cs if you're using C#) in the Solution Explorer.

Visual Studio opens the form in the Form Designer.



10. Press F5 to run the application.
11. Click the Show TableAdapters button.

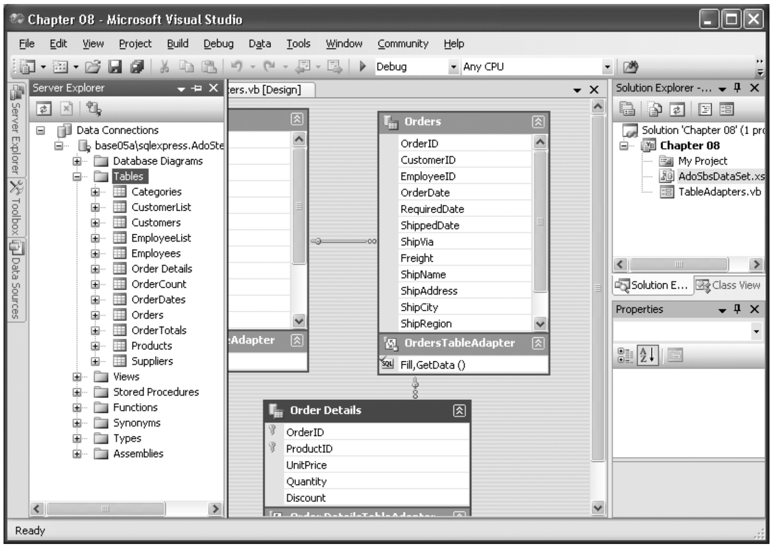
    The application displays the defined TableAdapters.
12. Select one of the TableAdapters to see the methods defined for that TableAdapter.



13. Close the application.

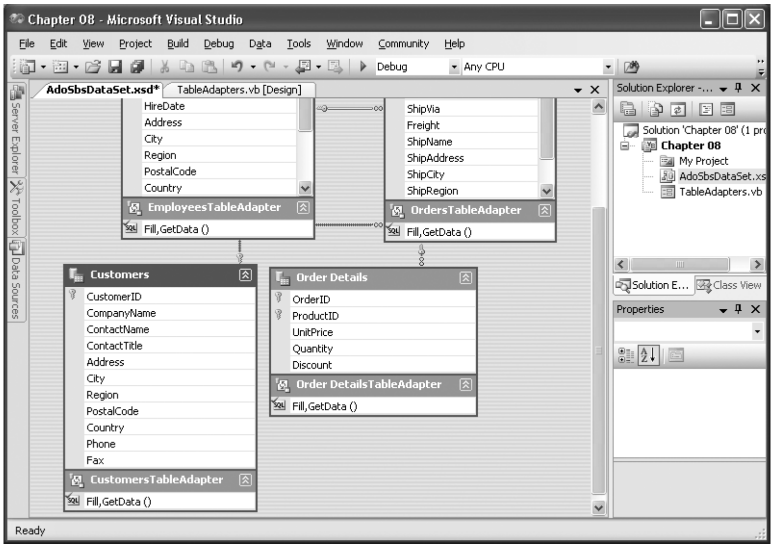
Create a TableAdapter from the Server Explorer

1. With the AdoSbsDataSet still open in the DataSet Designer (it should still be open from the last exercise), open the Server Explorer by choosing Server Explorer from the View menu.
2. Verify that the connection to the AdoStepByStep database is displayed. If it isn't, click the Connect To Database toolbar button on the Server Explorer tab, and create a connection to the AdoStepByStep SQL Server database as described in Chapter 6, "Modeling a Database by Using DataSets and DataRelations."
3. Expand the connection to the AdoStepByStep sample database, and then expand the Tables node.



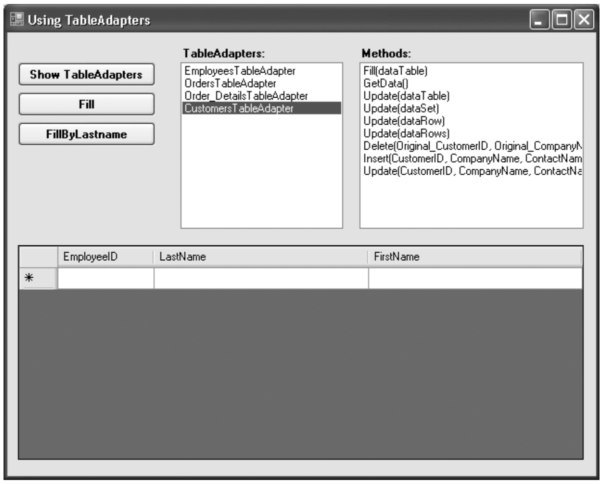
4. Drag the Customers table onto the DataSet designer.

   The DataSet Designer creates the DataTable and TableAdapter and adds them to the DataSet.



5. Save the DataSet.
6. Press F5 to run the application.
7. Click the Show TableAdapters button.

   The application shows the defined TableAdapters, including the new Customers TableAdapter.
8. Select CustomersTableAdapters to see the defined methods.



TableAdapters:
```
EmployeesTableAdapter
OrdersTableAdapter
Order_DetailsTableAdapter
CustomersTableAdapter
```

Methods:
```
Fill(dataTable)
GetData()
Update(dataTable)
Update(dataSet)
Update(dataRow)
Update(dataRows)
Delete(Original_CustomerID, Original_CompanyN
Insert(CustomerID, CompanyName, ContactNam
Update(CustomerID, CompanyName, ContactNa
```

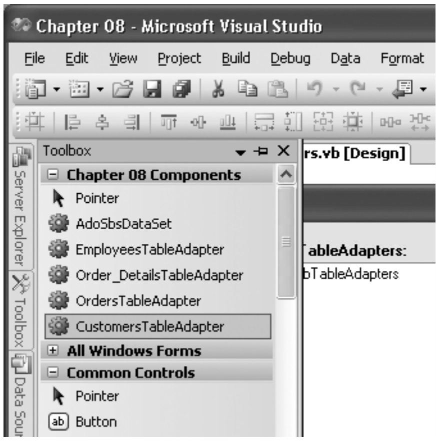
9. Close the application.

Creating TableAdapter Objects

Like any other classes in the .NET Framework, TableAdapter classes must be instantiated before use. You can create objects at design time by dragging a DataTable from the Data Sources window, or at run time by using the New constructor.

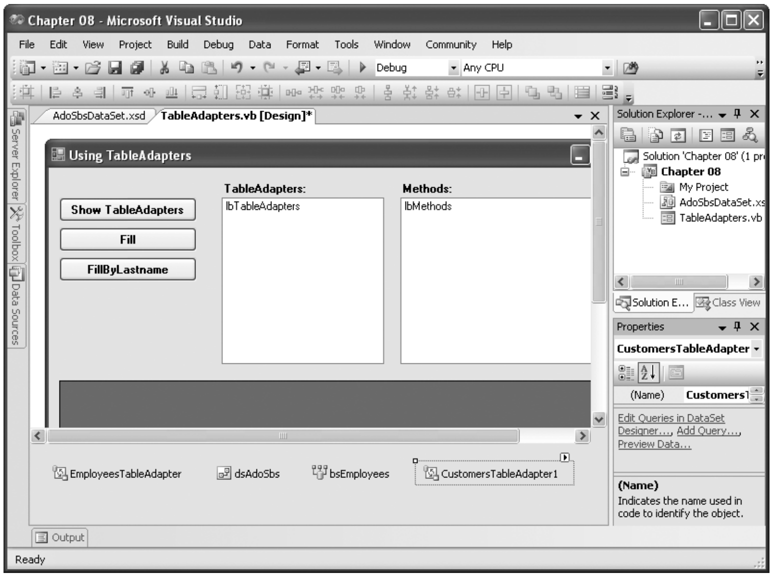
Create a TableAdapter in Visual Studio

1. Display the Form Designer, and then open the Toolbox.



2. Drag the CustomersTableAdapter from the Chapter 08 Components section onto the Form Designer.

   Visual Studio adds an instance of the TableAdapter to the components tray.



3. In the Properties window, change the name of CustomersTableAdapter1 to **taCustomers**.

Create a TableAdapter in Code: Visual Basic

1. Press F7 to display the Code Editor.
2. Add the following line to the class declaration, above the New constructor:

```
Dim taEmployees As AdoSbsDataSetTableAdapters.EmployeesTableAdapter
```

3. Add the following line where indicated at the end of New constructor:

```
taEmployees = New AdoSbsDataSetTableAdapters.EmployeesTableAdapter()
```

The beginning of your class file should now be:

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Reflection
Public Class TableAdapters

    Dim taEmployees As AdoSbsDataSetTableAdapters.EmployeesTableAdapter

    Public Sub New()

        ' This call is required by the Windows Form Designer.
        InitializeComponent()

        ' Add exercise code here:
        taEmployees = New AdoSbsDataSetTableAdapters.EmployeesTableAdapter()

    End Sub
```

Create a TableAdapter in Code: Visual C#

1. Press F7 to display the Code Editor.
2. Add the following line to the class declaration, above the Chapter_08 constructor:

```
AdoSbsDataSetTableAdapters.EmployeesTableAdapter taEmployees;
```

3. Add the following line to the end of the constructor where indicated:

```
taEmployees = new AdoSbsDataSetTableAdapters.EmployeesTableAdapter();
```

The beginning of your class file should now be:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Reflection;
using System.Text;
using System.Windows.Forms;

namespace Chapter_08
{
    public partial class TableAdapters : Form
    {
        AdoSbsDataSetTableAdapters.EmployeesTableAdapter taEmployees;

        public TableAdapters()
        {
            InitializeComponent();

            //Add exercise code here:
            taEmployees = new AdoSbsDataSetTableAdapters.EmployeesTableAdapter();
        }
```

Configuring TableAdapters

Because the TableAdapter exposes only two simple properties, the majority of its configuration occurs at design time, when you configure the class itself.

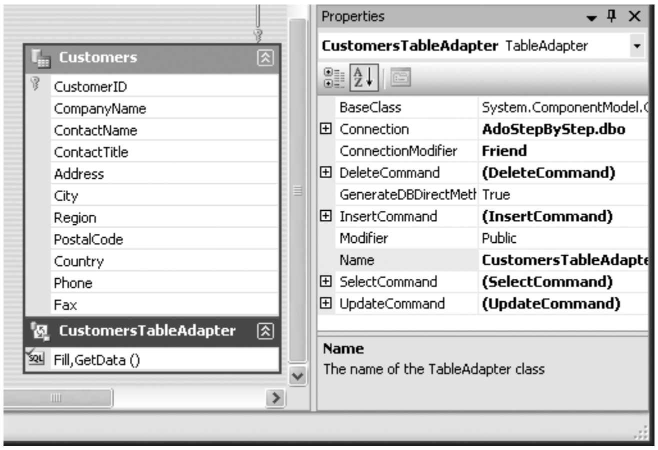
Configuring TableAdapter Classes

The TableAdapter class properties are shown in Table 8-1. The properties fall into two categories: those that determine how the class will be defined, and those that determine how data will be manipulated by the TableAdapter.

Table 8-1 TableAdapter Class Properties

Property	Description
BaseClass	The class from which the TableAdapter inherits its functionality.
ConnectionModifier	Determines the visibility of the encapsulated Connection.
Modifier	Determines the visibility of the TableAdapter.
Name	The name of the TableAdapter.
GenerateDBDirectMethods	Determines whether the TableAdapter exposes Insert, Update and Delete methods that are sent directly to the database.
DeleteCommand	The SQL statement used to delete rows.
InsertCommand	The SQL statement used to insert rows.
SelectCommand	The SQL statement used to fill the DataTable.
UpdateCommand	The SQL statement used to update rows.

You can make the TableAdapter class properties visible in the DataSet Designer by selecting a TableAdapter at the bottom of a DataTable and then viewing the Properties window.



BaseClass	System.ComponentModel.C
Connection	AdoStepByStep.dbo
ConnectionModifier	Friend
DeleteCommand	(DeleteCommand)
GenerateDBDirectMetl	True
InsertCommand	(InsertCommand)
Modifier	Public
Name	CustomersTableAdapte
SelectCommand	(SelectCommand)
UpdateCommand	(UpdateCommand)

Name
The name of the TableAdapter class

The BaseClass property determines the .NET Framework class from which the TableAdapter is derived. By default, this property is set to System.ComponentModel.Component. You can change this setting to simply extended the TableAdapter, but there is rarely a need to do so.

The ConnectionModifier property determines the visibility of the encapsulated Connection object. All of the standard visibility modifiers are available, with Friend being the default.

The Modifier property determines whether the TableAdapter exposes the DataTable itself through its GetData method. The options are Public and Friend, with Public being the default.

The Name property, as you might expect, determines the name by which the TableAdapter is referred to in code. The default is <DataTable>TableAdapter, but you can change it to any string that conforms to the rules for naming objects in the .NET Framework.

The GenerateDBDirectMethods property determines whether the TableAdapter generates Insert, Update and Delete commands that are executed directly against the Data Source, by-passing the DataTable. This property is True by default.

Finally, the four Command properties determine the SQL statements to be executed by the TableAdapter. These are similar to the corresponding Command properties in a DataAdapter, but they expose fewer Command properties. The TableAdapter commands expose only CommandText, CommandType and Parameters properties, while the commands encapsulated by a DataAdapter expose additional properties such as the CommandTimeout.

Important The smaller property set of the TableAdapter is something you need to consider when you decide whether to use a TableAdapter. If you need finer control over command behavior than is available in a TableAdapter, you must use a DataAdapter instead.

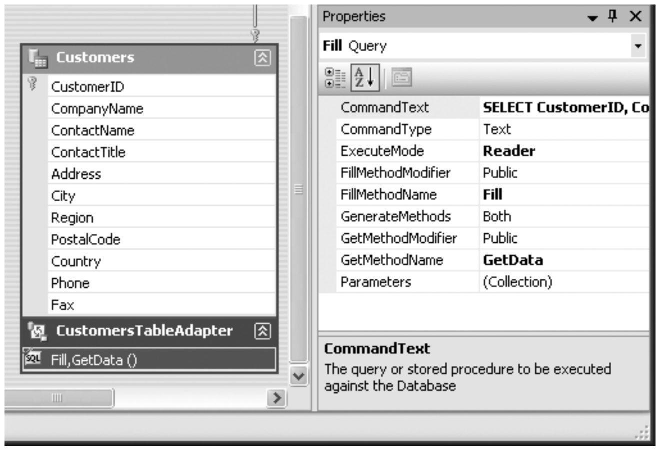
By default, a TableAdapter created by Visual Studio or the DataSet Designer exposes a single query that corresponds to the Fill and GetData methods. (We'll examine TableAdapter methods later in this chapter.) You can add additional queries by using the TableAdapter Query Configuration Wizard.

Each TableAdapter query exposes the properties shown in Table 8-2.

Table 8-2 TableAdapter Query Properties

Property	Description
FillMethodModifier	Determines the visibility of the Fill method.
FillMethodName	The name of the method.
GenerateMethods	Determines whether the TableAdapter generates Fill or GetData methods, or both.
GetMethodModifier	Determines the visibility of the GetData method.
GetMethodName	The name of the method.
CommandText	The SQL statement executed by the methods.
CommandType	The command type of the SQL statement.
ExecuteMode	The return type of the SQL statement.
Parameters	The Parameters collection of the SQL statement.

You can make the TableAdapter query properties visible in the DataSet Designer by selecting a TableAdapter at the bottom of a DataTable, selecting the TableAdapter query, and then viewing the Properties window.



CommandText	SELECT CustomerID, Co
CommandType	Text
ExecuteMode	Reader
FillMethodModifier	Public
FillMethodName	Fill
GenerateMethods	Both
GetMethodModifier	Public
GetMethodName	GetData
Parameters	(Collection)

CommandText
The query or stored procedure to be executed against the Database

The GenerateMethods property, which can be set to Fill, Get or Both, determines what methods will be defined in the TableAdapter class. The default is Both.

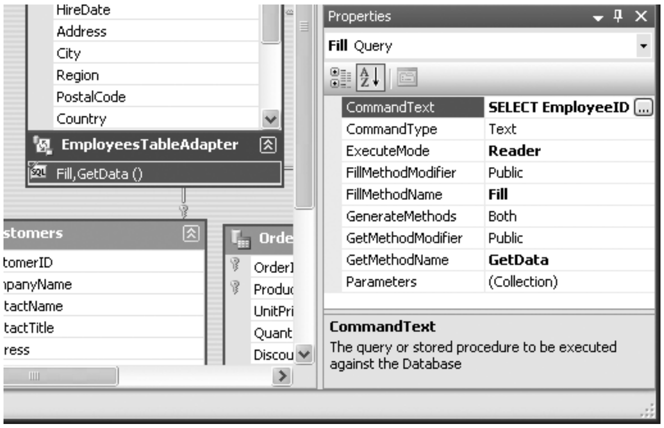
For each method selected, the <method>MethodModifier property allows you to set the visibility, while the <method>MethodName property determines the method name. The Properties window exposes only the applicable properties, based on the setting of the GenerateMethods property.

The CommandText, CommandType and Parameters properties are common to all data commands in the .NET Framework.

The ExecuteMode property specifies the return type of the SQL statement. This property can be set to Reader, indicating that the statement returns DataRows; Scalar, indicating that the query returns a single value; or NonQuery, indicating that the query returns no values.

Edit the Fill Method of a TableAdapter

1. Display the DataSet Designer, and select the Fill,GetData() method of the Employees TableAdapter.

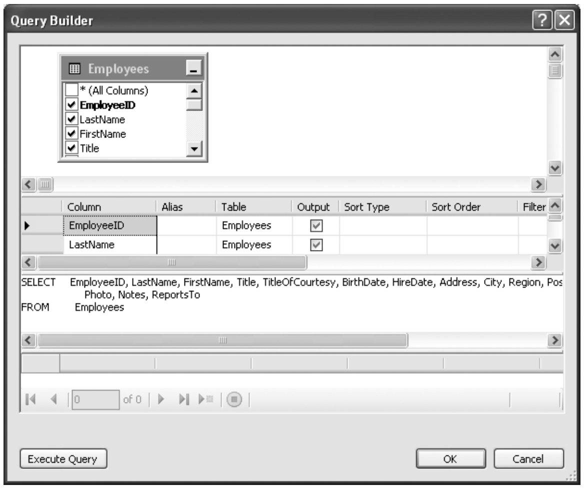


CommandText	SELECT EmployeeID
CommandType	Text
ExecuteMode	Reader
FillMethodModifier	Public
FillMethodName	Fill
GenerateMethods	Both
GetMethodModifier	Public
GetMethodName	GetData
Parameters	(Collection)

CommandText
The query or stored procedure to be executed against the Database

2. In the Properties window, click the ellipsis button next to the CommandText property.

   The DataSet Designer opens the Query Builder.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter
EmployeeID		Employees	✓			
LastName		Employees	✓			

```
SELECT    EmployeeID, LastName, FirstName, Title, TitleOfCourtesy, BirthDate, HireDate, Address, City, Region, Pos
              Photo, Notes, ReportsTo
FROM      Employees
```

3. Change the SQL statement to:

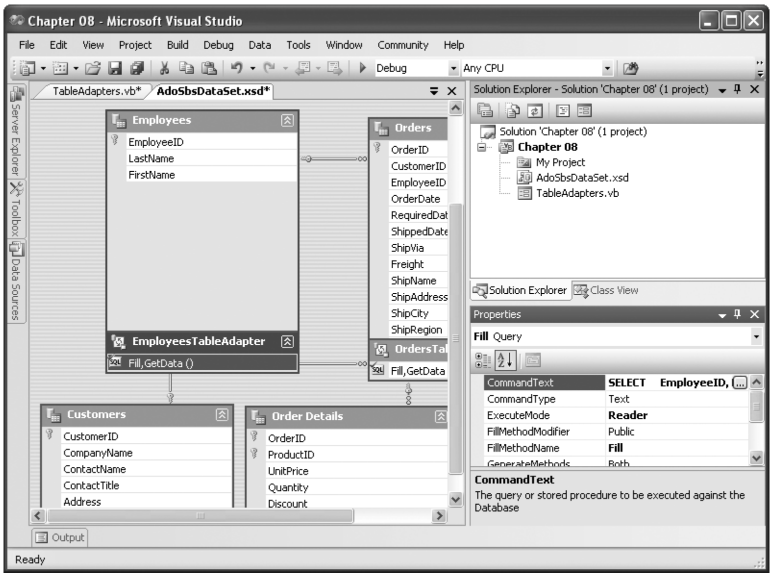
```
SELECT EmployeeID, LastName, FirstName
FROM Employees
```

4. Click OK.

   The DataSet Designer displays a message box asking whether to regenerate updating commands.

5. Click Yes.

   The DataSet Designer updates the Fill method and the DataTable column list.



CommandText	SELECT EmployeeID,
CommandType	Text
ExecuteMode	Reader
FillMethodModifier	Public
FillMethodName	Fill
GenerateMethods	Both

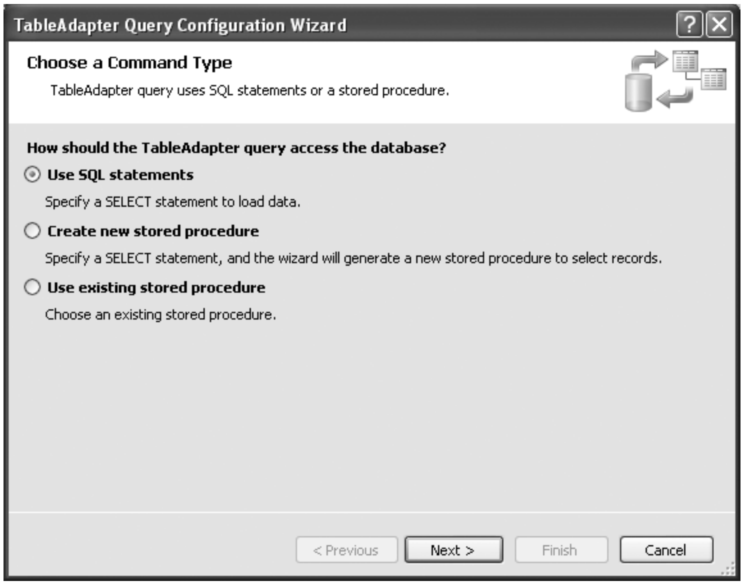
CommandText
The query or stored procedure to be executed against the Database

6. Save the DataSet.

Add a Query to a TableAdapter

1. In the DataSet Designer, right-click the EmployeesTableAdapter and choose Add Query.

   The DataSet Designer opens the TableAdapter Query Configuration Wizard.


TableAdapter Query Configuration Wizard

Choose a Command Type

TableAdapter query uses SQL statements or a stored procedure.

How should the TableAdapter query access the database?

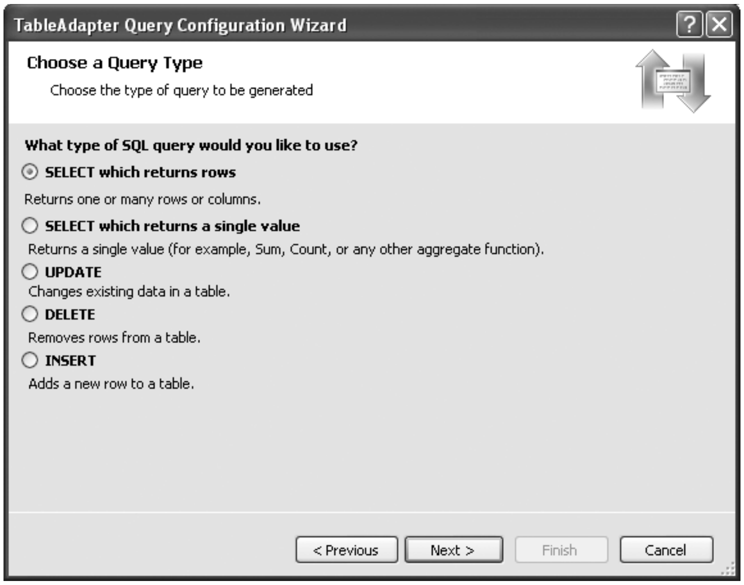
[✓ Use SQL statements]
Specify a SELECT statement to load data.

Create new stored procedure
Specify a SELECT statement, and the wizard will generate a new stored procedure to select records.

Use existing stored procedure
Choose an existing stored procedure.

2. Accept the default command type, Use SQL Statements, and then click Next.

   The TableAdapter Query Configuration Wizard displays a page requesting the type of query.


TableAdapter Query Configuration Wizard

Choose a Query Type

Choose the type of query to be generated

What type of SQL query would you like to use?

[✓ SELECT which returns rows]
Returns one or many rows or columns.

SELECT which returns a single value
Returns a single value (for example, Sum, Count, or any other aggregate function).

UPDATE
Changes existing data in a table.

DELETE
Removes rows from a table.

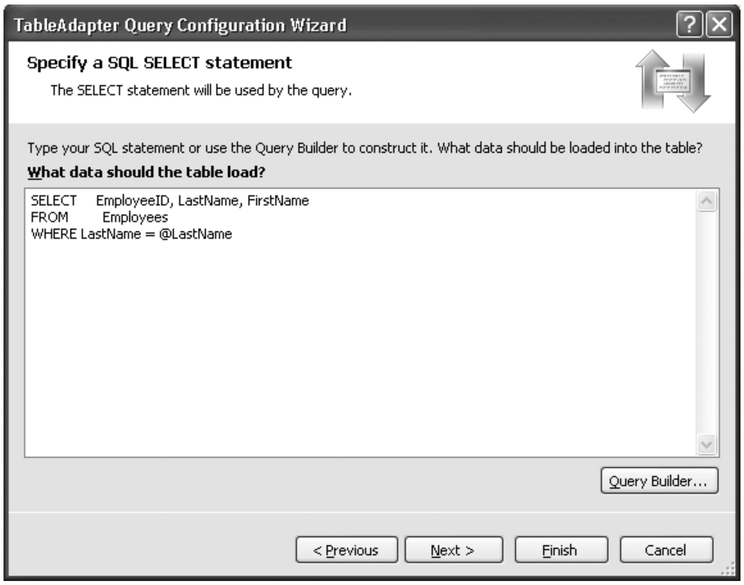
INSERT
Adds a new row to a table.

3. Accept the default of SELECT Which Returns Rows, and then click Next.

   The TableAdapter Query Configuration Wizard displays a page requesting the SQL statement.

4. Add a WHERE clause to the statement:

```
SELECT     EmployeeID, LastName, FirstName
FROM         Employees
WHERE LastName = @LastName
```

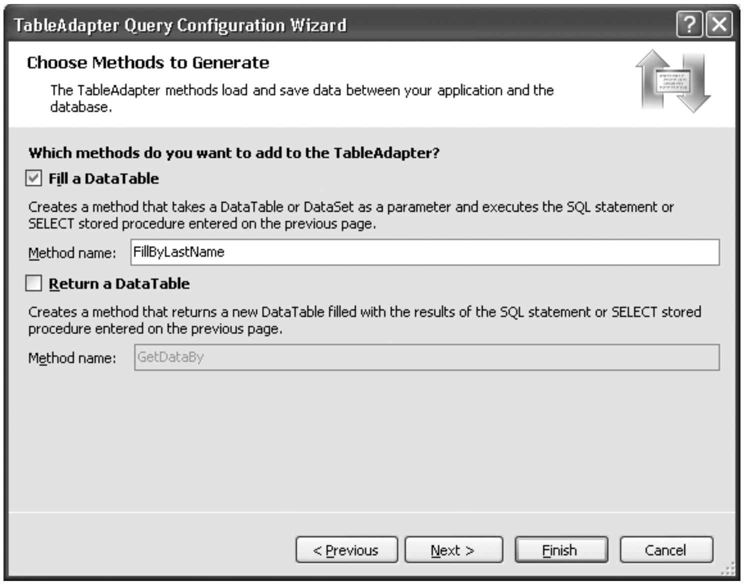

TableAdapter Query Configuration Wizard
Specify a SQL SELECT statement
The SELECT statement will be used by the query.
Type your SQL statement or use the Query Builder to construct it. What data should be loaded into the table?
What data should the table load?
Query Builder...
< Previous
Next >
Finish
Cancel

```
SELECT    EmployeeID, LastName, FirstName
FROM       Employees
WHERE LastName = @LastName
```

5. Click Next.

   The TableAdapter Query Configuration Wizard displays a page requesting the methods to be added.

6. Change the name of the Fill A DataTable method to **FillByLastName**, and clear the Return A DataTable check box.


TableAdapter Query Configuration Wizard

Choose Methods to Generate

The TableAdapter methods load and save data between your application and the database.

Which methods do you want to add to the TableAdapter?

☑ Fill a DataTable

Creates a method that takes a DataTable or DataSet as a parameter and executes the SQL statement or SELECT stored procedure entered on the previous page.

Method name: FillByLastName

☐ Return a DataTable

Creates a method that returns a new DataTable filled with the results of the SQL statement or SELECT stored procedure entered on the previous page.

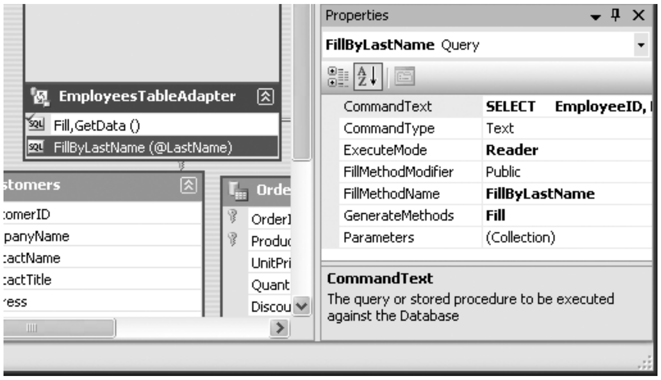
Method name: GetDataBy

7. Click Next.

   The TableAdapter Query Configuration Wizard displays a results page indicating that the SELECT statement and the Fill method were generated.

8. Click Finish.

   The TableAdapter Query Configuration Wizard adds the new query to the Table-Adapter.


EmployeesTableAdapter

Fill,GetData ()

FillByLastName (@LastName)

Properties

FillByLastName Query

CommandText	SELECT EmployeeID,
CommandType	Text
ExecuteMode	Reader
FillMethodModifier	Public
FillMethodName	FillByLastName
GenerateMethods	Fill
Parameters	(Collection)

CommandText

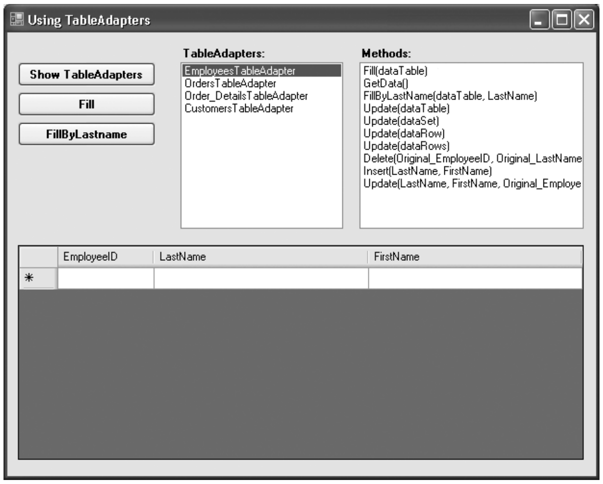
The query or stored procedure to be executed against the Database

9. Save and close the DataSet.
10. Press F5 to run the application.
11. Click the Show TableAdapters button.

    The application displays the defined TableAdapter.

12. Select EmployeesTableAdapter in the list box.

    The application displays the query functions, which include FillByLastName.



TableAdapters:

```
EmployeesTableAdapter
OrdersTableAdapter
Order_DetailsTableAdapter
CustomersTableAdapter
```

Methods:

```
Fill(dataTable)
GetData()
FillByLastName(dataTable, LastName)
Update(dataTable)
Update(dataSet)
Update(dataRow)
Update(dataRows)
Delete(Original_EmployeeID, Original_LastName
Insert(LastName, FirstName)
Update(LastName, FirstName, Original_Employe
```

	EmployeeID	LastName	FirstName
*			

13. Close the application.

Configuring TableAdapter Objects

After the TableAdapter class is defined, the TableAdapter object itself is very simple, exposing only the two properties shown in Table 8-3. (You can, of course, add additional properties by extending the class definition.)

Table 8-3 TableAdapter Properties

Property	Description
ClearBeforeFill	Determines whether a DataTable is emptied before data is loaded.
Connection	Returns the encapsulated Connection object.

The ClearBeforeFill property determines whether a DataTable will be emptied of DataRows before data is loaded into it.

The Connection property returns the Connection object encapsulated by the DataAdapter, allowing you to fine-tune its behavior. The visibility of this property is, however, controlled by the value of the ConnectionModifier property of the TableAdapter (set in the DataSet Designer). If the ConnectionModifier property is set to Private, for example, the Connection will not be exposed by the TableAdapter.

Using TableAdapter Methods

By default, the TableAdapter exposes the methods shown in Table 8-4. As we've seen, you can change the names of each of these methods in the DataSet Designer.

Table 8-4 Default TableAdapter Methods

Method	Description
Delete(col0...colN)	Deletes rows in the data source.
Fill(dataTable)	Loads data into the specified DataTable.
FillBy(dataTable, param0...paramN)	Loads data into the specified DataTable based on the specified parameters.
GetData	Returns a DataTable.
GetDataBy(dataTable, param0...paramN)	Returns a DataTable based on the specified parameters.
Insert(col0...colN)	Inserts rows in the data source.
Update(dataTable)	Synchronizes the specified DataTable with a data source.
Update(dataSet)	Synchronizes the DataTable in the specified DataSet with the data source.
Update(dataRow)	Synchronizes the specified DataRow with the data source.
Update(dataRows())	Synchronizes the specified array of DataRows with the data source.
Update(col0...colN)	Updates the rows in the data source.

The Delete, Insert and Update(col0...colN) methods are only available if the GenerateDB DirectMethods property of the DataAdapter class is True. They operate directly on the data source, by-passing the DataSet.

The Fill method corresponds to the Fill method of the DataAdapter, but supports only one version, Fill(dataTable).

The GetData method returns a new copy of the DataTable. Internally, this method creates the DataTable and calls the Fill method.

The FillBy and GetDataBy methods are generated only if the SELECT query has parameters. They will have a signature matching the query parameter list specified in the DataSet Designer.

Finally, the Update(dataTable) method is identical to the Update(dataTable) method of the DataAdapter.

Executing Query Methods

As we've seen, the queries that you add to the TableAdapter in the DataSet Designer are exposed as methods of the TableAdapter class. For each query, you can choose to expose a Fill method that receives a DataTable as a parameter and fills it with data (optionally clearing it first), or a Get method that returns a new DataTable, or both.

This is one of the primary advantages of the TableAdapter over the earlier Connection/Data-Adapter object combination. The TableAdapter is extensible–you can add additional queries to the TableAdapter in the DataSet Designer and expose them as methods of the TableAdapter class.

Execute a Default Query Method: Visual Basic

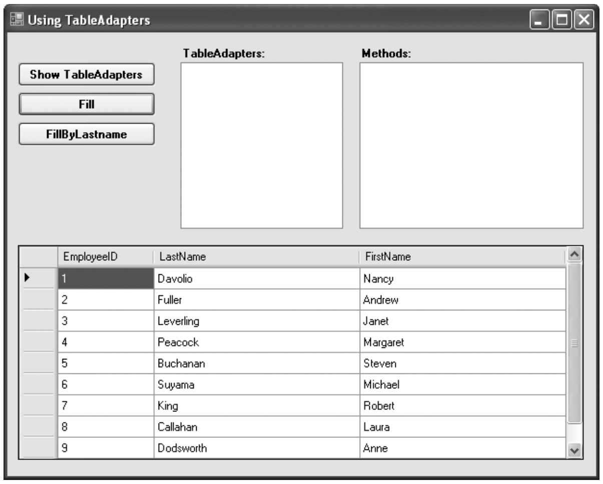
1. In the Code Editor, add the following line to the btnFill_Click event handler:

   ```
   taEmployees.Fill(dsAdoSbs.Employees)
   ```

   This line calls the default Fill method of the taEmployees TableAdapter we created earlier.

2. Press F5 to run the application.
3. Click the Fill button.

   The application fills the DataGridView.



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

4. Close the application.

Execute a Default Query Method: Visual C#

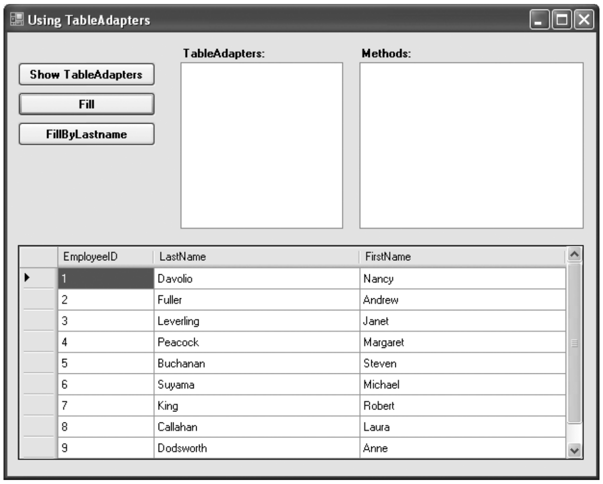
1. In the Code Editor, add the following line to the btnFill_Click event handler:

   ```
   taEmployees.Fill(dsAdoSbs.Employees);
   ```

   This line calls the default Fill method of the taEmployees TableAdapter we created earlier.

2. Press F5 to run the application.
3. Click the Fill button.

   The application fills the DataGridView.



	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

4. Close the application.

Execute a Parameterized Query Method: Visual Basic

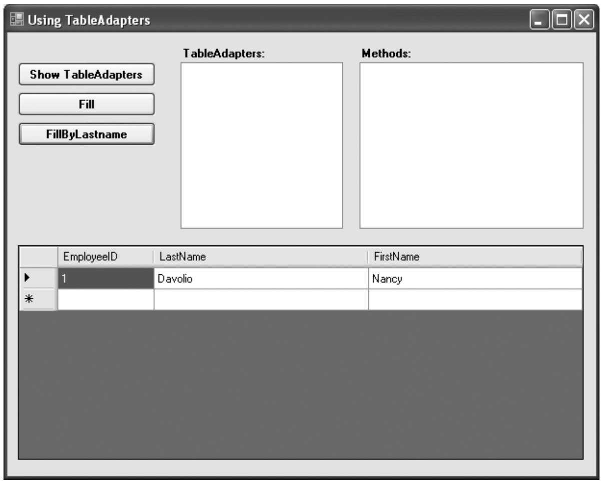
1. In the Code Editor, add the following line to the btnGet_Click event handler:

   ```
   taEmployees.FillByLastName(dsAdoSbs.Employees, "Davolio")
   ```

   This line calls the default FillByLastName method of the taEmployees TableAdapter.

2. Press F5 to run the application.
3. Click the FillByLastName button.

   The application fills the DataGridView with the matching record.



	EmployeeID	LastName	FirstName
▸	1	Davolio	Nancy
*			

4. Close the application.

Execute a Parameterized Query Method:Visual C#

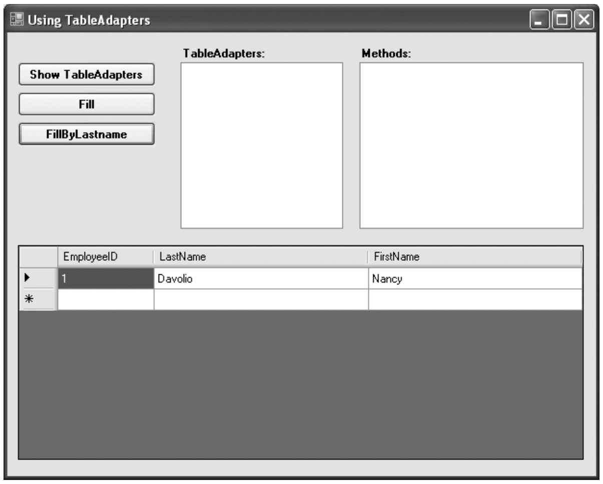
1. In the Code Editor, add the following line to the btnGet_Click event handler:

```
taEmployees.FillByLastName(dsAdoSbs.Employees, "Davolio");
```

   This line calls the default FillByLastName method of the taEmployees TableAdapter.

2. Press F5 to run the application.
3. Click the FillByLastName button.

   The application fills the DataGridView with the matching record.



	EmployeeID	LastName	FirstName
▸	1	Davolio	Nancy
*			

4. Close the application.

Summary

In this chapter we examined TableAdapters, which combine and extend the functionality of a DataAdapter and a Connection. TableAdapters are specific to a DataTable and are generated at design time by Visual Studio 2005.

We examined the technique for creating a TableAdapter class in the DataSet Designer and saw an example of adding a parameterized query to the TableAdapter class. We then looked at the straight-forward process of instantiating a TableAdapter at run time and calling its methods. Along the way, we examined the properties exposed by both the TableAdapter class and the instantiated TableAdapter object, which are closely related to the properties exposed by the DataAdapter and Connection objects that the TableAdapter encapsulates.