Working with Strings

Many times, an application will need to manipulate (process) string data in some way. For example, it may need to look at the first character in an inventory part number to determine the part's location in the warehouse. Or, it may need to search an address to determine the street name. Or, it may need to verify that the user's input is in the expected format. In this chapter, you will learn several ways of manipulating strings in Visual Basic. You will begin by learning how to determine the number of characters in a string.

How Many Characters Are There?

If an application expects the user to enter a seven-digit phone number or a five digit ZIP code, the application's code should verify that the user's input contains the required number of characters. The number of characters contained in a string is stored as an integer in the string's **Length property**. Figure 23-1 shows the property's syntax and includes examples of using the property. In the syntax, *string* can be a String variable, a String named constant, or the Text property of a control.

Length Property

Syntax: *string*.**Length**
Purpose: stores an integer that represents the number of characters contained in the string

Example 1
```
strFullName = "Jose Espinosa"
intNumChars = strFullName.Length
```
assigns the number 13 to the `intNumChars` variable

Example 2
```
intNumChars = txtPhone.Text.Length
```
assigns the number of characters in the txtPhone control's Text property to the `intNumChars` variable

Example 3
```
Do
    strZip = InputBox("5-digit ZIP code", "ZIP")
Loop Until strZip.Length = 5
```
continues prompting the user for a ZIP code until the user enters exactly five characters

Figure 23-1 Syntax and examples of the Length property

You will use the Length property in the Part Number Validator application. The application's interface provides a text box for the user to enter a part number. To be valid, the part number must contain either four or five characters. If the part number is valid, the application displays the part number in the interface; otherwise, it displays an appropriate message to the user.

To code and then test the Part Number Validator application:

1. Start Visual Studio 2012. Open the **Part Numbers Solution (Part Numbers Solution.sln)** file contained in the ClearlyVB2012\Chap23\Part Numbers Solution folder. If necessary, open the designer window. See Figure 23-2.

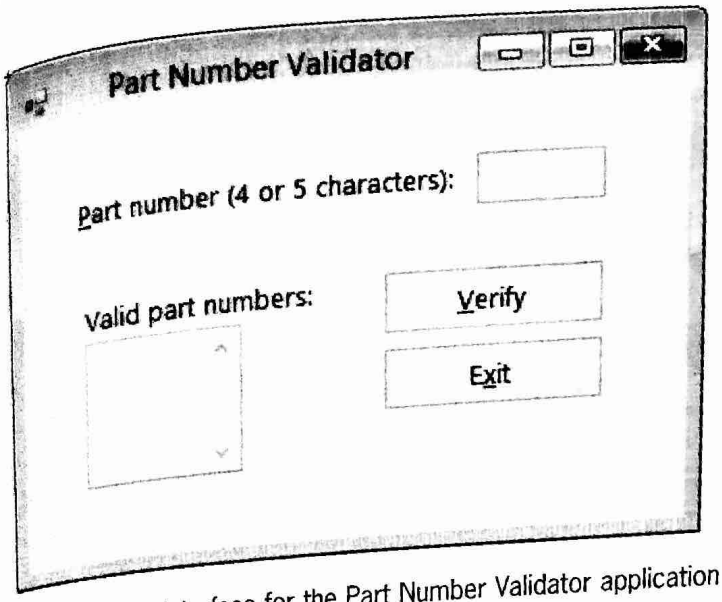


Figure 23-2 Interface for the Part Number Validator application

2. Open the Code Editor window and locate the btnVerify_Click procedure. Before verifying the part number's length, you will remove any leading and trailing spaces from the part number. You can do this using the Trim method, which you learned about in Chapter 11. Click the **blank line** below the `' remove any leading or trailing spaces` comment and then enter the following assignment statement:

```
strPartNum = txtPartNum.Text.Trim
```

3. Now you will determine whether the part number contains the appropriate number of characters, which is either four or five. Click the **blank line** below the `' verify length` comment and then enter the following If clause:

```
If strPartNum.Length = 4 OrElse strPartNum.Length = 5 Then
```

4. If the part number is valid, the selection structure's true path should display the part number in the txtValid control. Enter the following assignment statement:

```
txtValid.Text = txtValid.Text &
    strPartNum.ToUpper & ControlChars.NewLine
```

5. If the part number is not valid, the selection structure's false path should display an appropriate message. Enter the additional five lines of code indicated in Figure 23-3.

```
Private Sub btnVerify_Click(sender As Object, e As EventArgs) Handl
    ' verifies that a part number contains four
    ' or 5 characters and displays the valid ones

    Dim strPartNum As String

    ' remove any leading or trailing spaces
    strPartNum = txtPartNum.Text.Trim

    ' verify length
    If strPartNum.Length = 4 OrElse strPartNum.Length = 5 Then
        txtValid.Text = txtValid.Text &
            strPartNum.ToUpper & ControlChars.NewLine
    Else
        MessageBox.Show("Part numbers must be 4 or 5 characters.",
                        "Part Number Validator",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information)
    End If

    txtPartNum.Focus()
End Sub
```

enter these five lines of code

Figure 23-3 btnVerify_Click procedure

6. Save the solution and then start the application. First, you will enter a part number that contains three characters. Type **abc** as the part number and then click the **Verify** button. A message box opens and displays the "Part numbers must be 4 or 5 characters." message. Close the message box.
7. Now you will enter a part number that contains five characters followed by two spaces. Change the part number to **abc23** and then press the **Spacebar** twice. Click the **Verify** button. ABC23 appears in the listing of valid part numbers.
8. Next, you will enter a part number that contains four characters. Change the part number to **nrt4**. Click the **Verify** button. NRT4 appears in the listing of valid part numbers. See Figure 23-4. (Recall that you can use the Alt key to show/hide the access keys.)

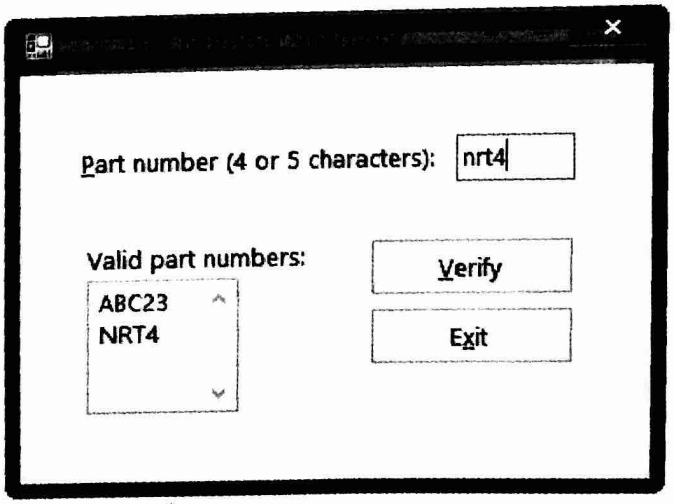


Figure 23-4 Valid part numbers displayed in the interface

9. On your own, test the application using a part number that contains nine characters. Also test it using a part number that contains both leading and trailing spaces.
10. When you are finished testing the application, click the **Exit** button. Close the Code Editor window and then close the solution.

I Need to Fit This in Somewhere

Visual Basic's **Insert method** allows you to insert characters anywhere in a string. Possible uses for the method include inserting an employee's middle initial within his or her name and inserting parentheses around the area code in a phone number. The method's syntax is shown in Figure 23-5 along with examples of using the method. In the syntax, *string* can be a String variable, a String named constant, or the Text property of a control. When processing the Insert method, the computer makes a temporary copy of the *string* in memory. It then performs the specified insertion on the copy only. The Insert method does not affect the original *string*. The *startIndex* argument in the method is an integer that specifies where in the string's copy you want the *value* inserted. The integer represents the character's index—in other words, its position in the string. The first character in a string has an index of 0, the second character has an index of 1, and so on. To insert the value at the beginning of a string, you use a startIndex of 0, as shown in Example 1 in Figure 23-5. To insert the value beginning with the ninth character in the string, you use a startIndex of 8, as shown in Example 2. The Insert method returns a string with the appropriate characters inserted.

Insert Method

Syntax
string.**Insert**(*startIndex*, *value*)

Purpose
inserts characters anywhere in a string

Example 1
```
strPhone = "111-2222"
txtPhone.Text = strPhone.Insert(0, "(877) ")
```
changes the contents of the txtPhone control's Text property to (877) 111-2222

Example 2
```
strName = "Suzanne Yonger"
strFullName = strName.Insert(8, "G. ")
```
changes the contents of the `strFullName` variable to Suzanne G. Yonger

Figure 23-5 Syntax and examples of the Insert method

You will use the Insert method in the Part Number Validator application from the previous section. Before displaying a valid part number in the txtValid control, the btnVerify_Click procedure will insert a hyphen as the third character in the part number.

To modify and then test the Part Number Validator application:

1. Use Windows to make a copy of the Part Numbers Solution folder. Rename the copy Part Numbers Solution–Insert.
2. Open the **Part Numbers Solution** (**Part Numbers Solution.sln**) file contained in the Part Numbers Solution–Insert folder. Open the designer window.
3. Open the Code Editor window and locate the btnVerify_Click procedure.
4. Insert a blank line below the If clause and then enter the additional statement shown in Figure 23-6.

enter this line of code

```
Private Sub btnVerify_Click(sender As Object, e As EventArgs) Handl
    ' verifies that a part number contains four
    ' or 5 characters and displays the valid ones

    Dim strPartNum As String

    ' remove any leading or trailing spaces
    strPartNum = txtPartNum.Text.Trim

    ' verify length
    If strPartNum.Length = 4 OrElse strPartNum.Length = 5 Then
        strPartNum = strPartNum.Insert(2, "-")
        txtValid.Text = txtValid.Text &
            strPartNum.ToUpper & ControlChars.NewLine
    Else
        MessageBox.Show("Part numbers must be 4 or 5 characters.",
                        "Part Number Validator",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information)
    End If

    txtPartNum.Focus()
End Sub
```

Figure 23-6 Modified btnVerify_Click procedure

5. Save the solution and then start the application. Type **xtr67** as the part number and then click the **Verify** button. XT-R67 appears in the Valid part numbers box. Change the part number to **bne9** and then click the **Verify** button. BN-E9 appears in the Valid part numbers box. See Figure 23-7.

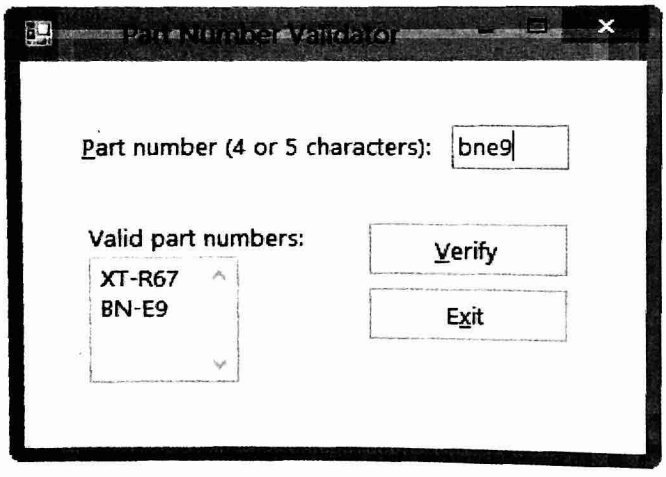


Figure 23-7 Result of using the Insert method

6. Click the **Exit** button. Close the Code Editor window and then close the solution.

The IndexOf method performs a case-sensitive search, which means the case of the subString must match the case of the string in order for both to be considered equal. The IndexOf method returns an integer: either −1 or a number that is greater than or equal to 0. The −1 indicates that the subString is not contained in the string. A number other than −1 is the character index of the subString's starting position in the string. Unless you specify otherwise, the IndexOf method starts the search with the first character in the string. To specify a different starting location, you use the optional startIndex argument. For instance, to begin the search with the second character in the string, you use a startIndex of 1.

I Just Want a Part of It

In some applications, it is necessary to access one or more characters contained in a string. For instance, you may need to display only the string's first five characters because they identify an item's location in the warehouse. Visual Basic provides the **Substring method** for accessing any number of characters in a string. Figure 23-9 shows the method's syntax and includes examples of using the method. In the syntax, *string* can be a String variable, a String named constant, or the Text property of a control. The *startIndex* argument is the index of the first character you want to access in the string. As you already know, the first character in a string has an index of 0. The optional *numCharsToAccess* argument specifies the number of characters you want to access. The Substring method returns a string that contains the number of characters specified in the numCharsToAccess argument, beginning with the character whose index is startIndex. If you omit the numCharsToAccess argument, the Substring method returns all characters from the startIndex position through the end of the string.

Substring Method

Syntax
string.**Substring(***startIndex*[, *numCharsToAccess*]**)**

Purpose
accesses any number of characters contained in a string

Example 1

character index 0 | character index 5

```
strFullName = "Jake Rubion"
strFirst = strFullName.Substring(0, 4)
strLast = strFullName.Substring(5)
```

assigns the string "Jake" to the `strFirst` variable and assigns the string "Rubion" to the `strLast` variable; you can also write the last assignment statement as `strLast = strFullName.Substring(5, 6)`

Example 2

character index 2

```
strEmployeeNum = "56F34"
strStatus = strEmployeeNum.Substring(2, 1)
```

assigns the string "F" to the `strStatus` variable

Figure 23-9 Syntax and examples of the Substring method

You will use the IndexOf and Substring methods in the Rearrange Name application, which you code in the next section.

The Rearrange Name Application

The Rearrange Name application's interface provides a text box for entering a person's first name followed by a space and the person's last name. The application rearranges the name so that the last name comes first, followed by a comma, a space, and the first name.

To code and then test the Rearrange Name application:

1. Open the **Rearrange Name Solution (Rearrange Name Solution.sln)** file contained in the ClearlyVB2012\Chap23\Rearrange Name Solution folder. If necessary, open the designer window. See Figure 23-10.

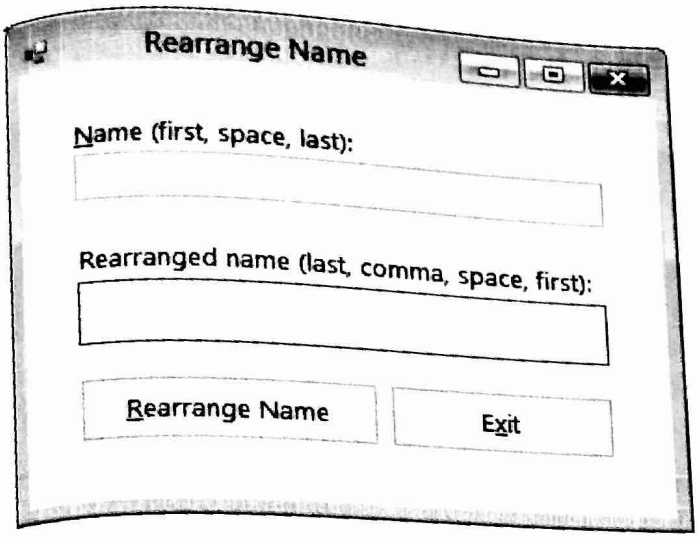


Figure 23-10 Interface for the Rearrange Name application

2. Open the Code Editor window and locate the btnRearrange_Click procedure. The procedure assigns the name entered by the user, excluding any leading and trailing spaces, to the `strName` variable.

3. Before you can rearrange the name stored in the `strName` variable, you need to separate the first name from the last name. To do this, you first search for the space character that appears between the names. Click the **blank line** below the `' search for the space in the name` comment and then enter the following assignment statement, being sure to include a space character between the quotation marks:

   intIndex = strName.IndexOf(" ")

4. If the value in the `intIndex` variable is not –1, it means that the IndexOf method found a space character in the `strName` variable. In that case, the selection structure's true path should continue rearranging the name; otherwise, its false path should display the "Invalid name format" message. The statement to display the message is already entered in the selection structure's false path. Change the If clause in the procedure to the following:

   If intIndex <> –1 Then

5. Now you will use the value stored in the `intIndex` variable to separate the first name from the last name. Click the **blank line** below the `' separate the first and last names` comment. All of the characters to the left of the space character represent the first name, and all of the characters to the right of the space character represent the last name. Enter the following assignment statements:

   strFirstName = strName.Substring(0, intIndex)
   strLastName = strName.Substring(intIndex + 1)

Ch23-Rearrang Name

6. Finally, you will display the rearranged name in the interface. Click the **blank line** above the Else clause. Enter the additional assignment statement indicated in Figure 23-11. Be sure to include a space character after the comma.

```
Private Sub btnRearrange_Click(sender As Object,
e As EventArgs) Handles btnRearrange.Click
    ' rearranges and then displays a name

    Dim strName As String
    Dim strFirstName As String
    Dim strLastName As String
    Dim intIndex As Integer

    ' assign the input to a variable
    strName = txtName.Text.Trim

    ' search for the space in the name
    intIndex = strName.IndexOf(" ")

    ' if the input contains a space
    If intIndex <> -1 Then
        ' separate the first and last names
        strFirstName = strName.Substring(0, intIndex)
        strLastName = strName.Substring(intIndex + 1)

        ' display last name, comma, space, and first name
        lblRearrangedName.Text =
            strLastName & ", " & strFirstName

    Else    ' the name does not contain a space
        MessageBox.Show("Invalid name format",
                        "Rearrange Name",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information)
    End If
End Sub
```

enter this assignment statement

Figure 23-11 btnRearrange_Click procedure

7. Save the solution and then start the application. Type **Jonathan Crowley** as the name and then click the **Rearrange Name** button. The rearranged name appears in the interface, as shown in Figure 23-12.

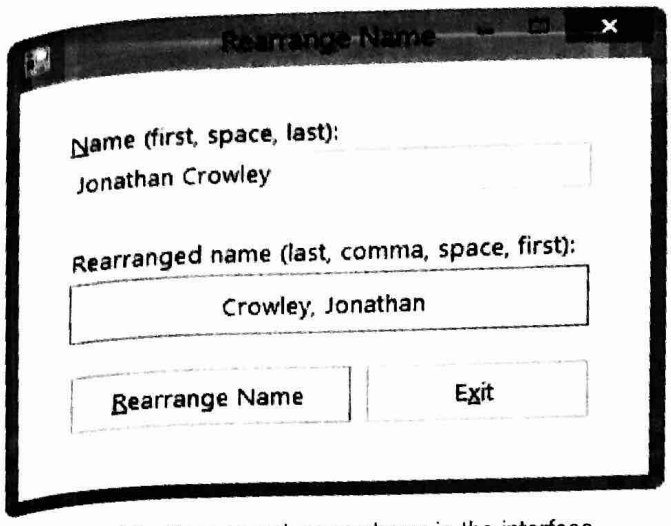


Figure 23-12 Rearranged name shown in the interface

8. Click the **Exit** button. Close the Code Editor window and then close the solution.

Throw Away Those Characters

You can use the **Remove method** to remove a specified number of characters located anywhere in a string. For example, you can use it to remove the dashes from a Social Security number, the leading zeroes from a policy number, or the middle initial from a name. Figure 23-13 shows the method's syntax and includes examples of using the method. In the syntax, *string* can be a String variable, a String named constant, or the Text property of a control. When processing the Remove method, the computer first makes a temporary copy of the *string* in memory. It then performs the specified removal on the copy only. In other words, the Remove method does not remove any characters from the original *string*. The Remove method returns a string with the appropriate characters removed.

The Remove method's *startIndex* argument is the index of the first character you want removed from the copy of the *string*. The optional *numCharsToRemove* argument is the number of characters you want removed. To remove only the first character, you use 0 as the startIndex and 1 as the numCharsToRemove. To remove the fourth through the eighth characters, you use 3 as the startIndex and 5 as the numCharsToRemove. If the numCharsToRemove argument is omitted, the Remove method removes all of the characters from the startIndex position through the end of the string, as shown in Example 2 in Figure 23-13.

10. Which of the following can be used to determine whether the `strAmount` variable contains exactly 10 characters?

    a. `If strAmount.Length = 10 Then`
    b. `If strAmount.Len = 10 Then`
    c. `If Length(strAmount) = 10 Then`
    d. none of the above

Exercises

TRY THIS

1. The `strAmount` variable contains the string "3123560". Write the Visual Basic statement to change the variable's contents to "$3,123,560". (See Appendix B for the answer.)

TRY THIS

2. Open the Zip Solution (Zip Solution.sln) file contained in the ClearlyVB2012\Chap23\Zip Solution folder. If necessary, open the designer window. The Display Shipping Charge button's Click event procedure should display the appropriate shipping charge based on the ZIP code entered by user. To be valid, the ZIP code must contain exactly five digits and the first three digits must be either "605" or "606". The shipping charge for "605" ZIP codes is $25. The shipping charge for "606" ZIP codes is $30. Display an appropriate message if the ZIP code is invalid. Code the procedure. Save the solution and then start the application. Test the application using the following ZIP codes: 60677, 60511, 60344, 6061234, and 7130. Close the Code Editor window and then close the solution. (See Appendix B for the answer.)

MODIFY THIS

3. Open the Phone Solution (Phone Solution.sln) file contained in the ClearlyVB2012\Chap23\Phone Solution folder. Open the designer and Code Editor windows. Use the comments in the btnSave_Click and btnDisplay_Click procedures to finish coding the application. Save the solution and then start the application. Test the application using the following phone numbers: 1-234-567890 and 999-888-1111. Close the Code Editor window and then close the solution.

INTRODUCTORY

4. Open the CityState Solution (CityState Solution.sln) file contained in the ClearlyVB2012\Chap23\CityState Solution folder. The interface provides a text box for entering the name of a city, followed by a comma, a space, and a state name. The Display Message button's Click event procedure should display the message "*cityName* is located in *stateName*", where *cityName* and *stateName* are the names of the city and state, respectively, entered by the user. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

5. Open the First Name Solution (First Name Solution.sln) file contained in the ClearlyVB2012\Chap23\First Name Solution folder. The interface provides a text box for entering a person's last name followed by a comma, a space, and the person's first name. The Display First Name button's Click event procedure should display the person's first name only. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

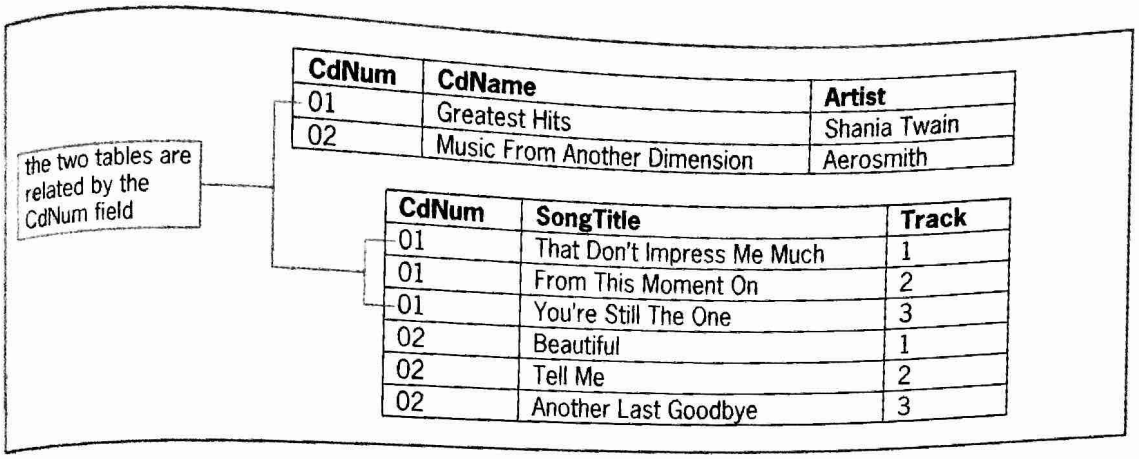
INTRODUCTORY

6. The `strAmount` variable contains the string "3,123,560". Write the Visual Basic statements to change the contents of the variable to "3123560".

INTERMEDIATE

7. Open the Sales Tax Solution (Sales Tax Solution.sln) file contained in the ClearlyVB2012\Chap23\Sales Tax Solution folder. If necessary, open the designer window. The interface allows the user to enter a sales amount and a tax rate. Open the Code Editor window. The btnCalc_Click procedure should determine whether the tax rate ends with a percent sign. If it does, the procedure should remove the percent sign from the rate. Make the appropriate modifications to the code. Save the solution and then start the application. Test the application using the following data: a sales amount of 1000 and a tax rate

Figure 24-2 shows an example of a two-table database that stores CD information. The first table is referred to as the **parent table**, and the second table is referred to as the **child table**. The CdNum field is the primary key in the parent table because it uniquely identifies each record in the table. The CdNum field in the child table is used solely to link the song title and track information to the appropriate CD in the parent table. In the child table, the CdNum field is called the **foreign key**. (Parent and child tables are also referred to as master and detail tables, respectively.)



CdNum	CdName	Artist
01	Greatest Hits	Shania Twain
02	Music From Another Dimension	Aerosmith

CdNum	SongTitle	Track
01	That Don't Impress Me Much	1
01	From This Moment On	2
01	You're Still The One	3
02	Beautiful	1
02	Tell Me	2
02	Another Last Goodbye	3

Figure 24-2 Example of a two-table relational database

Storing data in a relational database offers many advantages. The computer can retrieve data stored in a relational format both quickly and easily, and the data can be displayed in any order. The information in the CD database, for example, can be arranged by artist name, song title, and so on. You can also control the amount of information you want to view from a relational database. You can view all of the information in the CD database, only the information pertaining to a certain artist, or only the names of the songs contained on a specific CD.

Connecting...Connecting

Morgan Industries stores information about its employees in a Microsoft Access database named Employees. The database contains one table, which is named tblEmploy. The table data is shown in Figure 24-3. The table contains seven fields and 17 records. The Emp_Number field is the primary key because it uniquely identifies each record in the table. The Status field contains the employment status, which is either the letter F (for full-time) or the letter P (for part-time). The Code field identifies the employee's department: 1 for Accounting, 2 for Advertising, 3 for Personnel, and 4 for Inventory.

field names

Emp_Number	Last_Name	First_Name	Hired	Rate	Status	Code
100	Benton	Jack	3/5/2001	$15.00	F	2
101	Jones	Carol	4/2/2001	$15.60	F	2
102	Ismal	Asaad	1/15/2002	$10.00	P	1
103	Rodriguez	Carl	5/6/2002	$12.00	P	3
104	Iovanelli	Rebecca	8/15/2002	$20.00	F	1
105	Nyugen	Thomas	10/20/2002	$11.00	P	3
106	Vine	Martha	2/5/2003	$9.50	P	2
107	Smith	Jefferson	5/14/2003	$17.50	F	2
108	Gerber	Sarah	9/24/2004	$21.00	F	3
109	Jones	Samuel	1/10/2005	$13.50	F	4
110	Smith	John	5/6/2005	$9.00	P	4
111	Krutchen	Jerry	5/7/2006	$9.00	P	4
112	Smithson	Jose	6/27/2009	$14.50	F	1
113	Johnson	Leshawn	7/20/2009	$10.00	P	4
114	Jerod	James	4/9/2010	$10.00	P	4
115	Simons	Pam	6/8/2011	$9.00	P	2
116	Sorenson	Harry	3/4/2012	$15.00	F	3

records

Figure 24-3 Data contained in the tblEmploy table

Before an application can access the data stored in a database, it needs to be connected to the database. You can make the connection using the Data Source Configuration Wizard. The wizard allows you to specify the data you want to access. The computer makes a copy of the specified data and stores the copy in its internal memory. The copy of the data you want to access is called a **Dataset**. In the following set of steps, you will connect the Morgan Industries application to the Employees database.

To connect the Morgan Industries application to the Employees database:

1. Start Visual Studio 2012. Open the **Morgan Industries Solution (Morgan Industries Solution.sln)** file contained in the ClearlyVB2012\Chap24\Morgan Industries Solution-DataGridView folder. If necessary, open the designer window.
2. Auto-hide the Properties and Toolbox windows, and permanently display the Solution Explorer window.
3. If necessary, click **VIEW** on the menu bar, point to **Other Windows**, and then click **Data Sources** to open the Data Sources window. If necessary, click the window's **Auto Hide** button to permanently display the window.
4. Click **Add New Data Source** in the Data Sources window to start the Data Source Configuration Wizard. If necessary, click **Database** on the Choose a Data Source Type screen. See Figure 24-4. (If you want to show/hide the wizard's access keys, press the Alt key.)

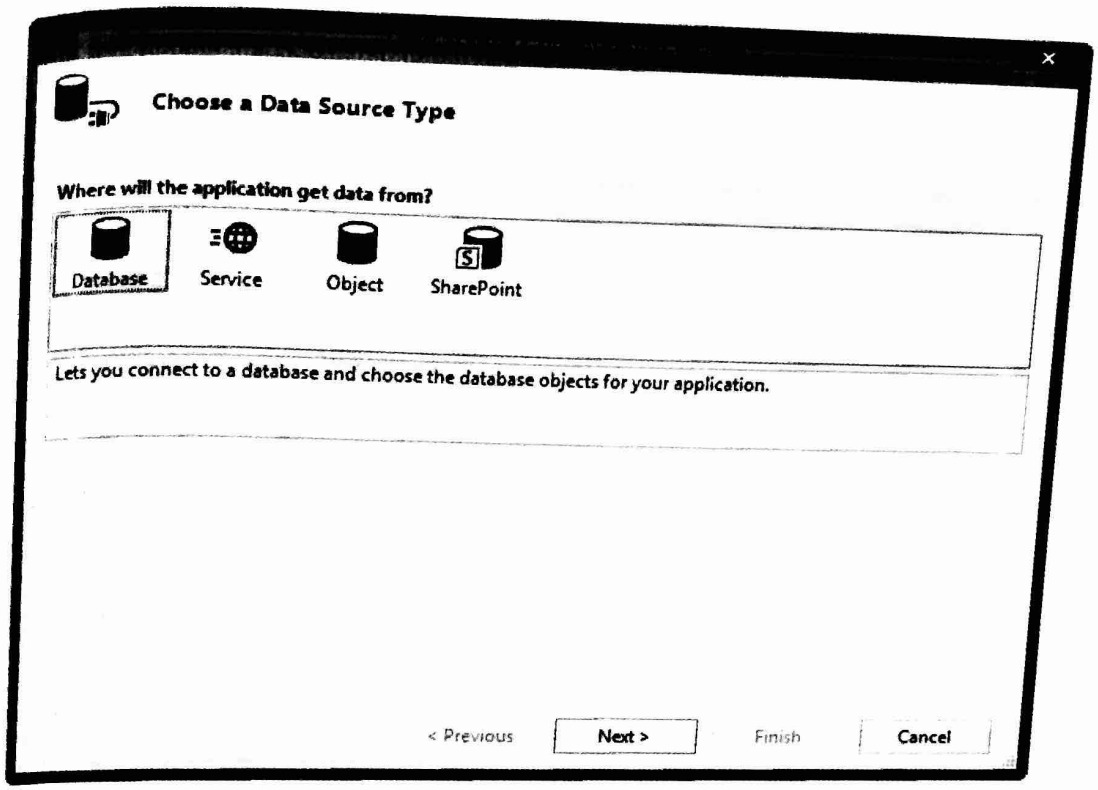


Figure 24-4 Choose a Data Source Type screen

5. Click the **Next** button to display the Choose a Database Model screen. If necessary, click **Dataset**.
6. Click the **Next** button to display the Choose Your Data Connection screen. Click the **New Connection** button to open the Add Connection dialog box. If Microsoft Access Database File (OLE DB) does not appear in the Data source box, click the **Change** button to open the Change Data Source dialog box, click **Microsoft Access Database File**, and then click the **OK** button.
7. Click the **Browse** button in the Add Connection dialog box to open the Select Microsoft Access Database File dialog box. Open the ClearlyVB2012\Chap24\Access Databases folder and then click **Employees.accdb** in the list of filenames. Click the **Open** button. Figure 24-5 shows the completed Add Connection dialog box.

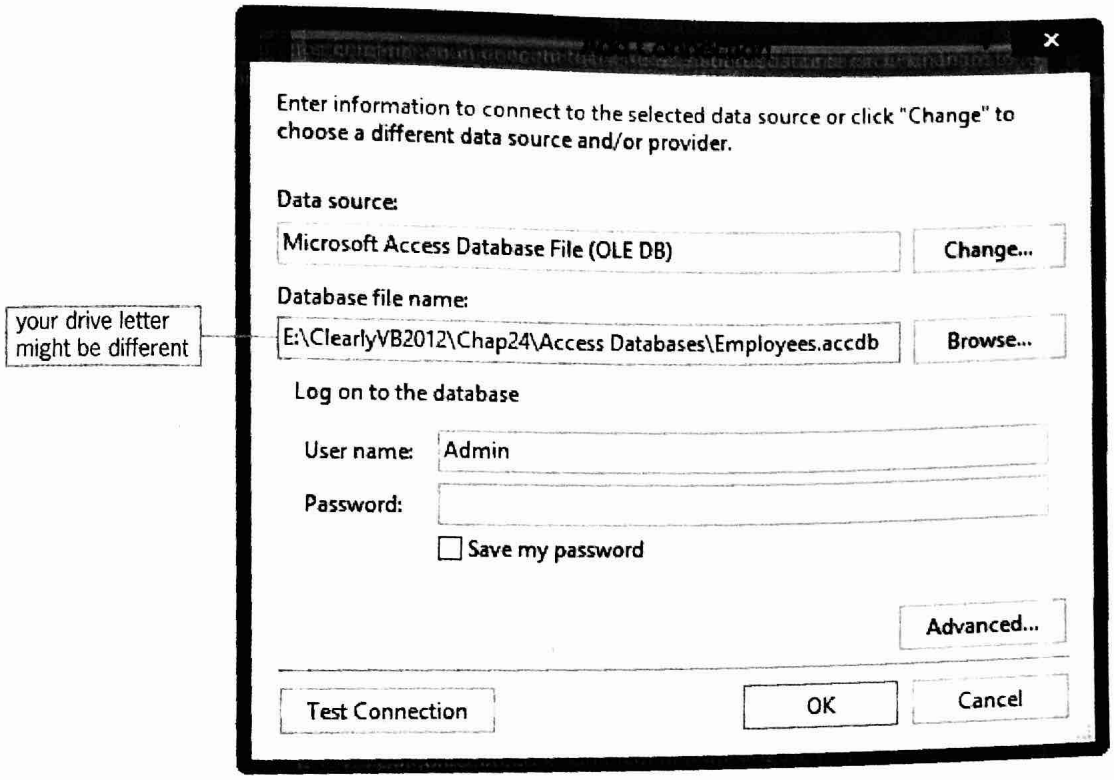


Figure 24-5 Completed Add Connection dialog box

8. Click the **Test Connection** button. The "Test connection succeeded." message appears in a message box. Close the message box.

9. Click the **OK** button to close the Add Connection dialog box. Employees.accdb appears in the Choose Your Data Connection screen. Click the **Next** button. The message box shown in Figure 24-6 opens. The message asks whether you want to include the database file in the current project. By including the file in the current project, you can more easily copy the application and its database to another computer.

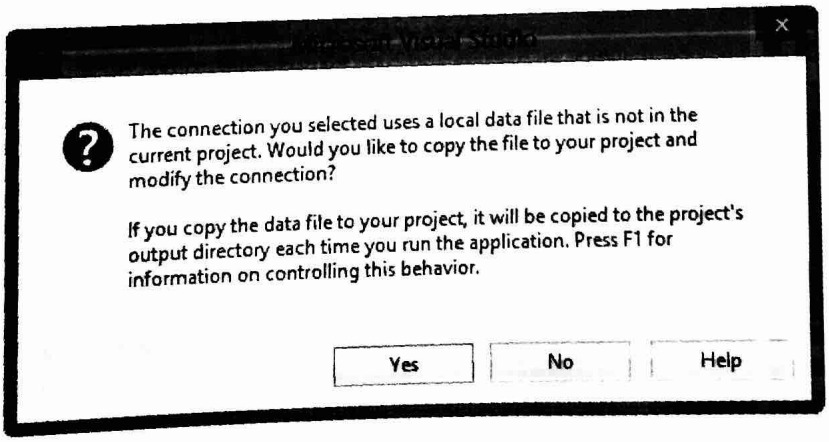


Figure 24-6 Message regarding copying the database file

10. Click the **Yes** button to add the Employees.accdb file to the application's project folder in the Solution Explorer window. The Save the Connection String to the Application Configuration File screen appears next. The name of the connection string, EmployeesConnectionString, appears on the screen. If necessary, select the **Yes, save the connection as** check box.

11. Click the **Next** button to display the Choose Your Database Objects screen. You use this screen to select the table and/or field objects to include in the dataset, which is automatically named EmployeesDataSet.

12. Expand the Tables node and then expand the tblEmploy node. (You expand a node by clicking the small triangle next to it.) In this application, you need the dataset to include all of the fields. Click the **empty box** next to tblEmploy. Doing this selects the table and field check boxes, as shown in Figure 24-7.

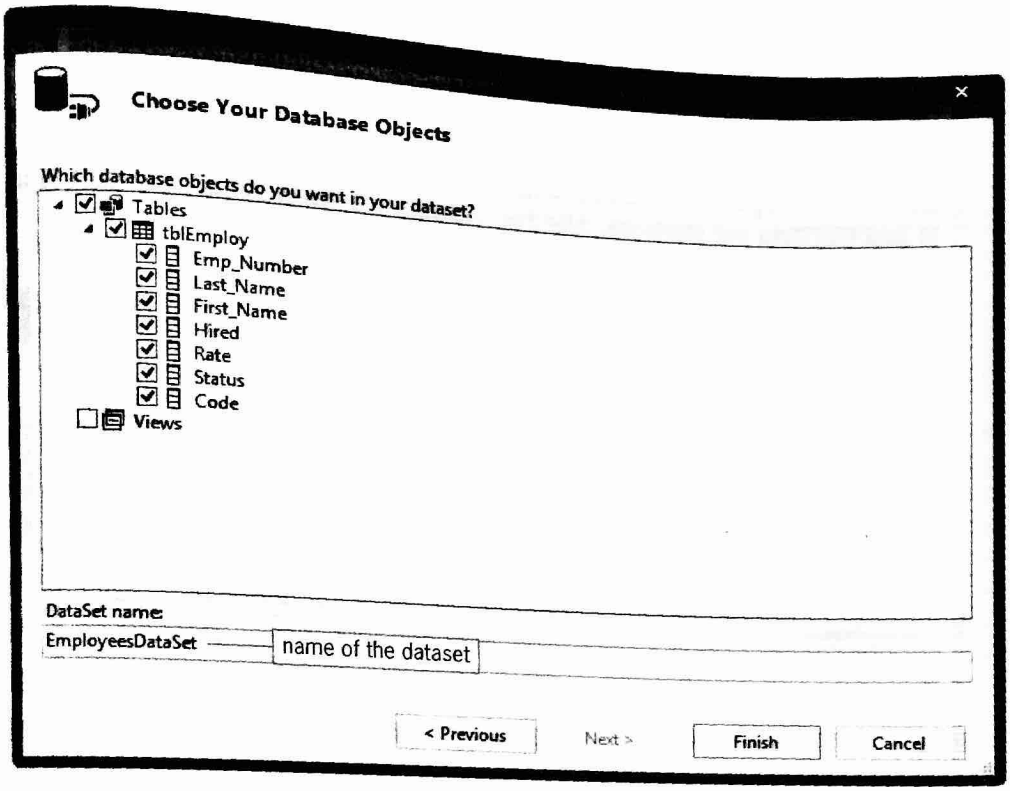


Figure 24-7 Objects selected in the Choose Your Database Objects screen

13. Click the **Finish** button and then save the solution. The computer adds the EmployeesDataSet to the Data Sources and Solution Explorer windows. Expand the tblEmploy node in the Data Sources window. As shown in Figure 24-8, the dataset contains one table and seven field objects.

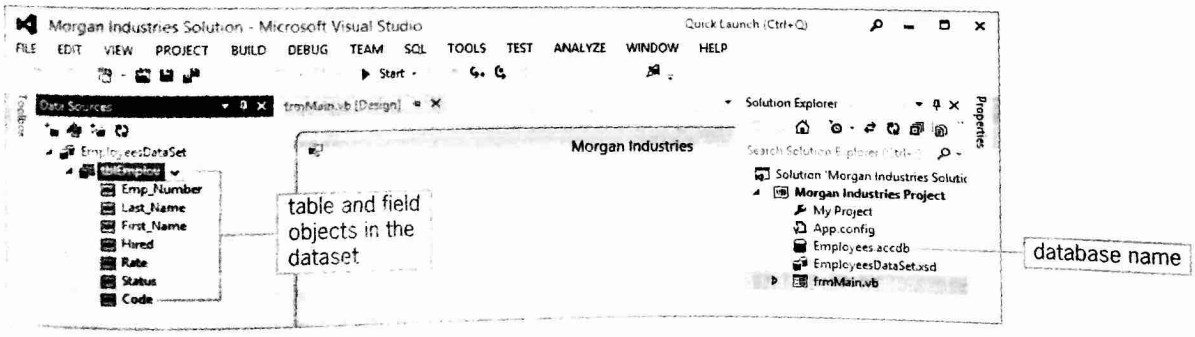


Figure 24-8 Result of running the Data Source Configuration Wizard

14. Save the solution.

What Fields and Records?

You can view fields and records contained in a dataset by right-clicking the dataset's name in the Data Sources window and then clicking Preview Data.

To view the contents of the EmployeesDataSet:

1. Right-click **EmployeesDataSet** in the Data Sources window, and then click **Preview Data** to open the Preview Data dialog box.
2. Click the **Preview** button. As Figure 24-9 shows, the EmployeesDataSet contains 17 records (rows), each having seven fields (columns). Notice the information that appears in the Select an object to preview box in the figure. EmployeesDataSet is the name of the dataset in the application, and tblEmploy is the name of the table included in the dataset. Fill and GetData are methods. The Fill method populates an existing table with data, while the GetData method creates a new table and populates it with data.

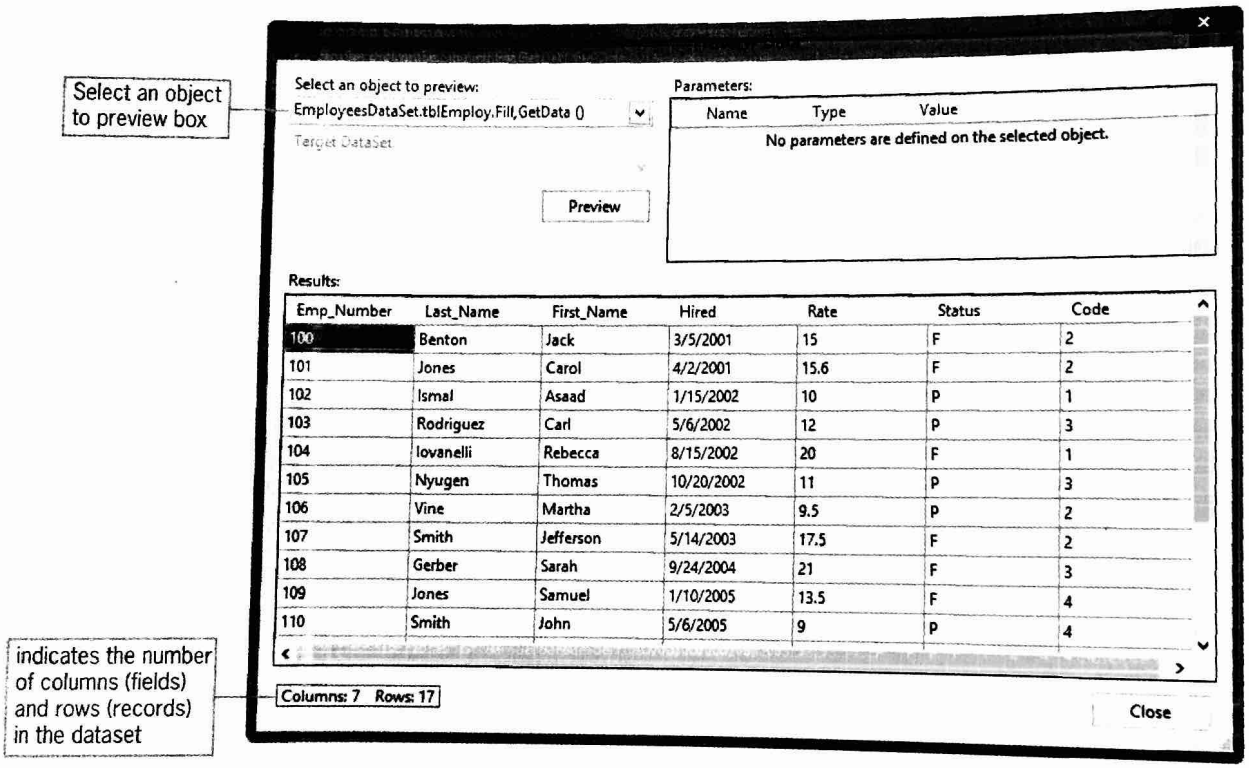


Figure 24-9 Data displayed in the Preview Data dialog box

3. Click the **Close** button to close the Preview Data dialog box, and then auto-hide the Solution Explorer window.

It's a Binding Contract

For the user to view the contents of a dataset while an application is running, you need to connect one or more objects in the dataset to one or more controls in the interface. Connecting an object to a control is called **binding**, and the connected controls are called **bound controls** (or data-aware controls). As indicated in Figure 24-10, you can bind the object either to a control that the computer creates for you or to an existing control in the interface. First, you will learn how to have the computer create a bound control.

8. Click the **Save Data** button. The "Changes saved" message appears in a message box. Close the message box, and then click the **Close** button on the form's title bar to stop the application.

9. Start the application again to verify that your changes were saved, and then stop the application. Close the Code Editor window and then close the solution.

I'll Use My Own Controls, Thank You

As indicated earlier in Figure 24-10, you can bind an object in a dataset to an existing control on the form. The easiest way to do this is by dragging the object from the Data Sources window to the control. However, you can also click the control and then set one or more properties in the Properties window. The appropriate property (or properties) to set depends on the control you are binding. For example, you use the DataSource property to bind a DataGridView control. However, you use the DataBindings/Text property to bind label and text box controls.

When you drag an object from the Data Sources window to an existing control, the computer does not create a new control; instead, it binds the object to the existing control. Because a new control does not need to be created, the computer ignores the control type specified for the object in the Data Sources window. Therefore, it is not necessary to change the control type in the Data Sources window to match the existing control's type. In other words, you can drag an object that is associated with a text box in the Data Sources window to a label control on the form. The computer will bind the object to the label, but it will not change the label to a text box.

In the next set of steps, you will open a different version of the Morgan Industries application. The application is already connected to the Employees database, so you will just need to bind the objects in the dataset to the existing label controls in the interface. In this version of the application, you will not need to change the database file's Copy to Output Directory property to Copy if newer because the user will not be adding, deleting, or editing the records in the dataset.

To bind an object in the dataset to an existing control:

1. Open the **Morgan Industries Solution** (**Morgan Industries Solution.sln**) file contained in the ClearlyVB2012\Chap24\Morgan Industries Solution–Labels folder. If necessary, open the designer window.

2. Permanently display the Data Sources window. If necessary, expand the EmployeesDataSet and tblEmploy nodes. As in the previous version of the application, the dataset in this version contains the tblEmploy table; however, this version contains only four of the seven field objects. See Figure 24-27.

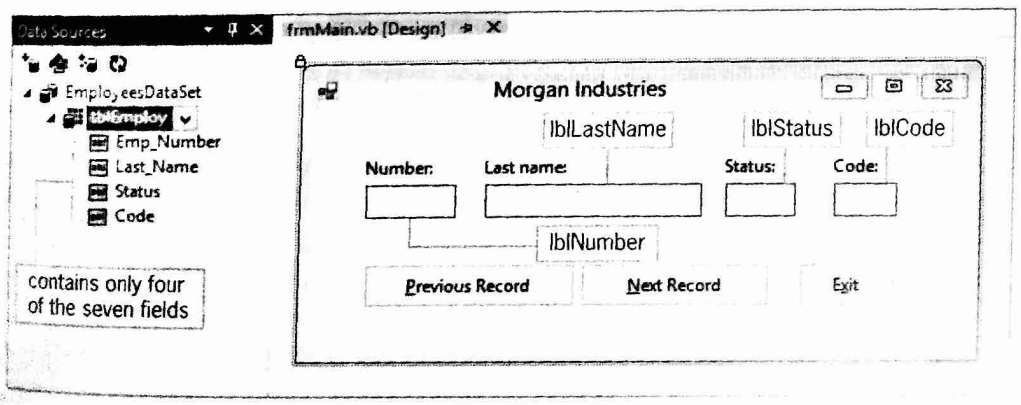


Figure 24-27 Dataset in this version of the application

Important note: Recall that you use the Choose Your Database Objects screen, shown earlier in Figure 24-7, to select the fields you want to include in the dataset. For this version of the application, only four of the fields were selected on that screen.

3. Click **Emp_Number** in the Data Sources window and then drag the field object to the lblNumber control. Release the mouse button. The computer binds the control and adds the DataSet, BindingSource, TableAdapter, and TableAdapterManager objects to the component tray. See Figure 24-28.

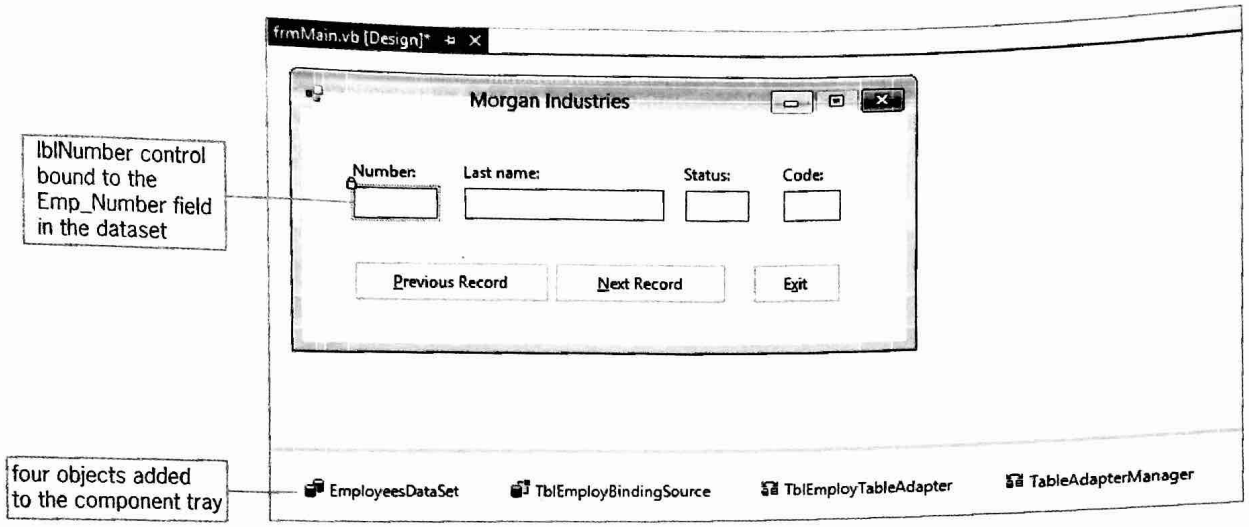


Figure 24-28 Result of binding a field to an existing control

Notice that when you drag an object from the Data Sources window to an existing control, the computer does not add a BindingNavigator object to the component tray, nor does it add a BindingNavigator control to the form. You can use the BindingNavigator tool, which is located in the Data section of the toolbox, to add a BindingNavigator control and object to the application. You then would set the control's DataSource property to the name of the BindingSource object (in this case, TblEmployBindingSource).

Besides adding the objects shown in Figure 24-28 to the component tray, the computer also enters (in the Code Editor window) the Load event procedure shown earlier in Figure 24-21. Recall that the procedure uses the TableAdapter object's Fill method to retrieve the data from the database and store it in the DataSet object.

To bind the remaining objects in the dataset to existing controls:

1. On your own, drag the Last_Name, Status, and Code field objects to the lblLastName, lblStatus, and lblCode controls, respectively.
2. Auto-hide the Data Sources window and then save the solution. Start the application. Only the first record in the dataset appears in the interface. See Figure 24-29.

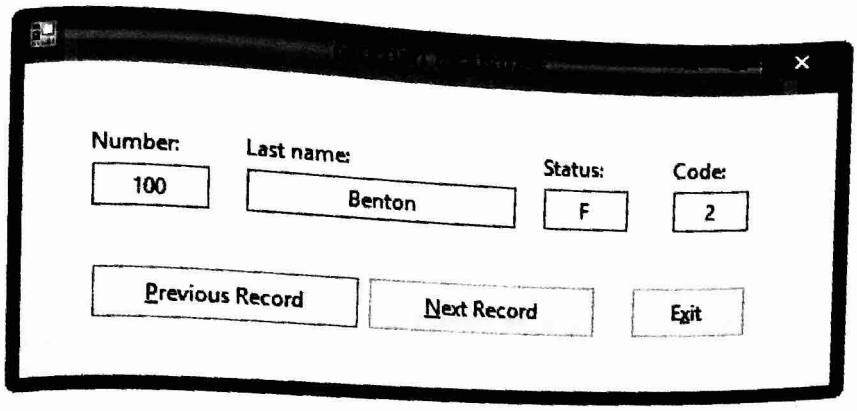


Figure 24-29 First record displayed in the interface

3. Because the interface does not contain a BindingNavigator control, which would allow you to move from one record to the next, you will need to code the Next Record and Previous Record buttons to view the remaining records. Click the **Exit** button to stop the application.

Coding the Next Record and Previous Record Buttons

The BindingSource object uses an invisible record pointer to keep track of the current record in the dataset. It stores the position of the record pointer in its **Position property**. The first record is in position 0, the second record is in position 1, and so on. Figure 24-30 shows the Position property's syntax and includes examples of using the property.

BindingSource Object's Position Property

Syntax
bindingSourceName.**Position**

Example 1
`intRecordNum = TblEmployBindingSource.Position`
assigns the current record's position to the `intRecordNum` variable

Example 2
`TblEmployBindingSource.Position = 4`
moves the record pointer to the fifth record in the dataset

Example 3
`TblEmployBindingSource.Position += 1`
moves the record pointer to the next record in the dataset; you can also write the statement as follows: `TblEmployBindingSource.Position = TblEmployBindingSource.Position + 1`

Figure 24-30 Syntax and examples of the BindingSource object's Position property

Rather than using the Position property to position the record pointer in a dataset, you can use the BindingSource object's Move methods. The **Move methods** move the record pointer to the first, last, next, or previous record in the dataset. Figure 24-31 shows each Move method's syntax and includes examples of using two of the methods.

BindingSource Object's Move Methods

Syntax

bindingSourceName.**MoveFirst()**
bindingSourceName.**MoveLast()**
bindingSourceName.**MoveNext()**
bindingSourceName.**MovePrevious()**

Example 1

```
TblEmployBindingSource.MoveFirst()
```

moves the record pointer to the first record in the dataset

Example 2

```
TblEmployBindingSource.MoveNext()
```

moves the record pointer to the next record in the dataset

Figure 24-31 Syntax and examples of the BindingSource object's Move methods

To code the Next Record and Previous Record buttons, and then test the code:

1. Open the Code Editor window. When the user clicks the Next Record button, the button's Click event procedure should move the record pointer to the next record in the dataset. Similarly, when the user clicks the Previous Record button, the button's Click event procedure should move the record pointer to the previous record in the dataset. Open the code templates for the btnNext_Click and btnPrevious_Click procedures. Enter the comments and Move methods shown in Figure 24-32.

enter this comment and Move method

```
Private Sub btnNext_Click(sender As Object, e As EventArgs)
    ' moves the record pointer to the next record

    TblEmployBindingSource.MoveNext()
End Sub
```

enter this comment and Move method

```
Private Sub btnPrevious_Click(sender As Object, e As EventAr
    ' moves the record pointer to the previous record

    TblEmployBindingSource.MovePrevious()
End Sub
```

Figure 24-32 btnNext_Click and btnPrevious_Click procedures

2. Save the solution and then start the application. Click the **Next Record** button to display the second record. Continue clicking the **Next Record** button until the last record appears in the interface.
3. Click the **Previous Record** button until the first record appears in the interface, and then click the **Exit** button. Close the Code Editor window and then close the solution.

Exercises

TRY THIS

1. Open the Morgan Industries Solution (Morgan Industries Solution.sln) file contained in the ClearlyVB2012\Chap24\Morgan Industries Solution–Details folder. If necessary, open the designer window. Connect the application to the Employees database, which is stored in the ClearlyVB2012\Chap24\Access Databases\Employees.accdb file. After connecting the application to the database, click tblEmploy in the Data Sources window and then click the down arrow that appears next to tblEmploy. Click Details in the list. Drag the tblEmploy object to the form and then release the mouse button. Click FORMAT on the menu bar, point to Center in Form, and then click Horizontally. Click FORMAT on the menu bar, point to Center in Form, and then click Vertically. Add a Try...Catch statement to the Save Data button's Click event procedure. Change the database file's Copy to Output Directory property to Copy if newer. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. (See Appendix B for the answer.)

MODIFY THIS

2. In this exercise, you modify one of the Morgan Industries applications from the chapter. Use Windows to make a copy of the Morgan Industries Solution-Labels folder. Save the copy in the ClearlyVB2012\Chap24 folder. Rename the copy Modified Morgan Industries Solution-Labels. Open the Morgan Industries Solution (Morgan Industries Solution.sln) file contained in the Modified Morgan Industries Solution-Labels folder. Open the designer and Code Editor windows. Modify the btnNext_Click and btnPrevious_Click procedures to use the Position property rather than the MoveNext and MovePrevious methods. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

3. Open the Cartwright Solution (Cartwright Solution.sln) file contained in the ClearlyVB2012\Chap24\Cartwright Solution folder. Connect the application to the Items database, which is stored in the ClearlyVB2012\Chap24\Access Databases\Items.accdb file. The database contains one table named tblItems. The table contains 10 records, each composed of three fields. The ItemNum and ItemName fields contain text; the Price field contains numbers. Display the records in a DataGridView control. Add a Try...Catch statement to the Save Data button's Click event procedure; display appropriate messages. Change the database file's Copy to Output Directory property appropriately. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

4. Sydney Industries records the item number, name, and price of each of its products in a database named Products. The Products database is stored in the ClearlyVB2012\Chap24\Access Databases\Products.accdb file. The database contains a table named tblProducts. The table contains 10 records, each composed of three fields. The ItemNum and ItemName fields contain text; the Price field contains numbers. Open the Sydney Solution (Sydney Solution.sln) file contained in the ClearlyVB2012\Chap24\Sydney Solution folder. Connect the application to the Products database. Bind the appropriate objects to the existing label controls. Open the Code Editor window. Code the Click event procedures for the Next Record and Previous Record buttons. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

5. The MusicBox database is stored in the ClearlyVB2012\Chap24\Access Databases\MusicBox.accdb file. The database contains a table named tblBox. The table contains 10 records, each composed of four text fields. Open the MusicBox Solution (MusicBox Solution.sln) file contained in the ClearlyVB2012\Chap24\MusicBox Solution-DataGrid View folder. Connect the application to the MusicBox database. Change the database file's Copy to Output Directory property to Copy if newer. Bind the table to a DataGridView control and then make the necessary modifications to the control. Open the Code Editor window and enter the Try...Catch statement in the Save Data button's Click event procedure. Include appropriate messages. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.