CHAPTER 2

Creating Applications with Visual Basic

TOPICS

2.1 Focus on Problem Solving: Building the *Directions* Application

2.2 Focus on Problem Solving: Responding to Events

2.3 Modifying the Text Property with Code

2.4 The AutoSize, BorderStyle, and TextAlign Properties

2.5 Clickable Images

2.6 Using Visual Studio Help

2.7 Debugging Your Application

In this chapter you will develop your first application, which displays a map and written directions to the Highlander Hotel. This application uses a form with labels, a PictureBox control, and buttons. You will write your first event procedures (also known as event handlers) in Visual Basic code and then you will learn to use the Label control's AutoSize, BorderStyle, and TextAlign properties. You will be introduced to clickable images, dynamic help, context-sensitive help, and the debugging process.

2.1 Focus on Problem Solving: Building the *Directions* Application

CONCEPT: In this section you create your first Visual Basic application: a window that displays a map and road directions to a hotel. In the process you learn how to place controls on a form and manipulate various properties.

The desk clerks at the historic Highlander Hotel frequently receive calls from guests requesting driving directions. Some desk clerks are not familiar with the street numbers or exits, and inadvertently give unclear or incorrect directions. The hotel manager has asked you to create an application that displays a map to the hotel. The desk clerks can

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

refer to the application when giving directions to customers over the phone. We will use the following steps to create the application:

1. Clearly define what the application is to do.
2. Visualize the application running on the computer and design its user interface.
3. Make a list of the controls needed.
4. Define the values of each control's relevant properties.
5. Start Visual Basic and create the forms and other controls.

Now we will take a closer look at each of these steps.

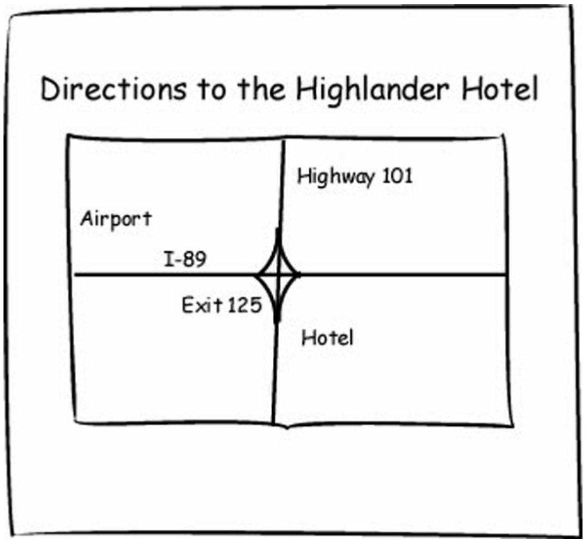
1. Clearly define what the application is to do.

Purpose:	Display a map to the Highlander Hotel
Input:	None
Process:	Display a form
Output:	Display on the form a graphic image showing a map

2. Visualize the application running on the computer and design its user interface.

Before you create an application on the computer, first you should create it in your mind. This step is the visualization of the program. Try to imagine what the computer screen will look like while the application is running. Then draw a sketch of the form or forms in the application. Figure 2-1 shows a sketch of the *Directions* form presented by this application.

Figure 2-1 Sketch of *Directions* form



3. Make a list of the controls needed.

In this step you list all the needed controls. You should assign names to all the controls that will be accessed or manipulated in the application code and provide a brief description of each control. Our application only needs three controls, listed in Table 2-1. Because none of the controls are used in code, we will keep their default names.

Table 2-1 *Directions* application controls

Control Type	Control Name	Description
Form	(Default Name: *Form1*)	A small form that will serve as the window onto which the other controls will be placed
Label	(Default Name: *Label1*)	Displays the message *Directions to the Highlander Hotel*
PictureBox	(Default Name: *PictureBox1*)	Displays the graphic image showing the map to the hotel

4. Define the values of each control's relevant properties.

Each control's property settings are listed in Table 2-2.

Table 2-2 *Directions* application control properties

Property	Value
Form	
Name	*Form1*
Text	*Directions*
Label	
Name	*Label1*
Text	*Directions to the Highlander Hotel*
TextAlign	*MiddleCenter*
Font	Microsoft sans serif, bold, 16 point
PictureBox	
Name	*PictureBox1*
Image	*HotelMap.jpg*
SizeMode	*StretchImage*

Notice that in addition to the Name and Text properties, we are setting the TextAlign and Font properties of the Label control. The **TextAlign property** determines how the text is aligned within the label. We will discuss this property in detail later.

In addition to its Name property, we are setting the PictureBox control's Image and SizeMode properties. The Image property lists the name of the file containing the graphic image. We will use *HotelMap.jpg*, which is located in the student sample programs folder named *Chap2*. The **SizeMode property** is set to *StretchImage*, which allows us to resize the image. If the image is too small, we can enlarge it (stretch it). If it is too large, we can shrink it.

5. Start Visual Basic and create the forms and other controls.

Now you are ready to construct the application's form. Tutorial 2-1 gets you started.

Tutorial 2-1:
Beginning the *Directions* application

In this tutorial you begin the *Directions* application. You will create the application's form and use the *Properties* window to set the form's Text property.

VideoNote Starting a Project

Step 1: Start Visual Studio (or Visual Basic Express), as you did in Chapter 1. Select one of the following ways to execute the *New Project* command:

- Click *File* on the menu bar and then click *New Project . . .*
- Click the *New Project* icon, the first icon on the left side of the Visual Studio toolbar

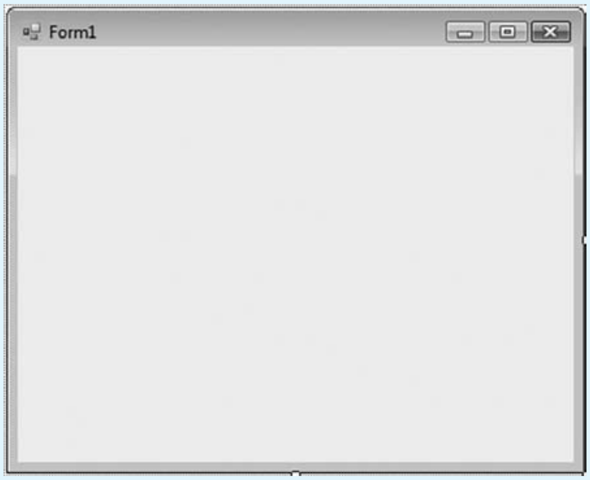
The *New Project* window will appear. If you are using Visual Studio, in the *Project types* pane, select *Windows* under *Visual Basic*. Select the *Windows Forms Application* icon in the *Templates* pane. Each project has a name. The default project name, such as *WindowsApplication1*, appears in the *Name* text box. Replace this name with **`Directions`**. Click the *OK* button to close the window.

NOTE: Your project will be saved in the location specified in the *Tools→Options* menu, in the category named *Projects and Solutions*. The *Visual Studio projects location* text box contains a directory name, which you can change.

Step 2: The Visual Basic environment should be open with a blank form named *Form1* in the *Design* window, as shown in Figure 2-2. Click the form to select it.

Step 3: Look at the *Properties* window. It should appear as shown in Figure 2-3.

Figure 2-2 *Form1* displayed in the *Design* window



Because you have selected *Form1*, the *Properties* window displays the properties for the *Form1* object. The drop-down list box at the top of the window shows the name of the selected object, *Form1*. Below that, the object's properties are displayed in two columns. The left column lists each property's name and the right column shows each property's value. Below the list of properties is a brief description of the currently selected property.

> **TIP:** The *Properties* window has two buttons near the top that control the order of names in the window. The first ▦ sorts by category and the second ↓ sorts alphabetically. We will use the alphabetical sort in our examples.

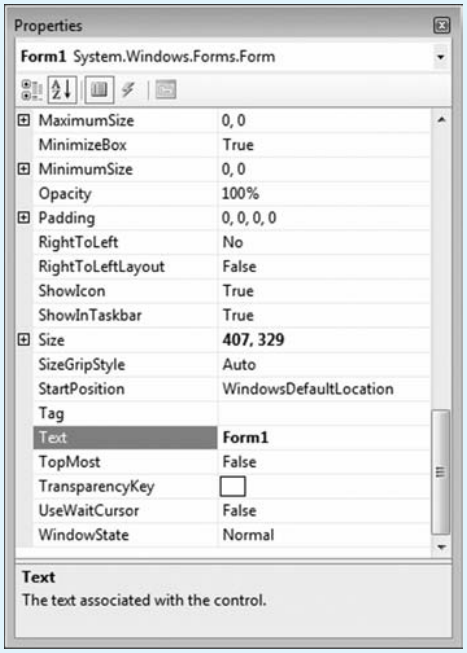
The Text property is highlighted, which means it is currently selected. A form's Text property holds the text displayed in the form's title bar. It is initially set to the same value as the form name, so this form's Text property equals *Form1*. Follow the instructions in Steps 4 and 5 to change the Text property to *Directions*.

Step 4: Double-click the word *Form1* inside the Text property.

Step 5: Delete the word *Form1* and type **`Directions`** in its place. Press the Enter key. Notice that the word *Directions* now appears in the form's title bar.

Step 6: Although you changed the form's Text property, you did not change its name. Scroll the *Properties* window up to the top of the list of properties, as shown in Figure 2-4. The Name property is still set to the default value, *Form1*.

Figure 2-3 *Properties* window showing *Form1*



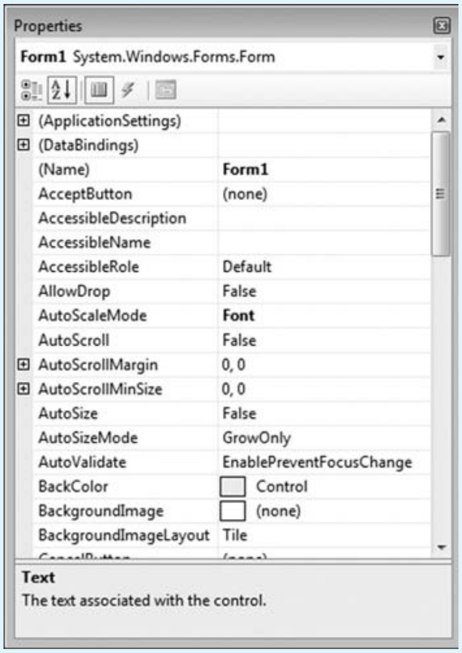
Properties

Form1 System.Windows.Forms.Form

⊞ MaximumSize	0, 0
MinimizeBox	True
⊞ MinimumSize	0, 0
Opacity	100%
⊞ Padding	0, 0, 0, 0
RightToLeft	No
RightToLeftLayout	False
ShowIcon	True
ShowInTaskbar	True
⊞ Size	**407, 329**
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	**Form1**
TopMost	False
TransparencyKey	□
UseWaitCursor	False
WindowState	Normal

Text
The text associated with the control.

Figure 2-4 *Properties* window scrolled to top



Properties

Form1 System.Windows.Forms.Form

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	**Form1**
AcceptButton	(none)
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AutoScaleMode	**Font**
AutoScroll	False
⊞ AutoScrollMargin	0, 0
⊞ AutoScrollMinSize	0, 0
AutoSize	False
AutoSizeMode	GrowOnly
AutoValidate	EnablePreventFocusChange
BackColor	□ Control
BackgroundImage	□ (none)
BackgroundImageLayout	Tile

Text
The text associated with the control.

The next step is to add a Label control to the form. Tutorial 2-2 guides you through the process.

Tutorial 2-2:
Adding a Label control to the *Directions* application

VideoNote
Using Label Controls

Step 1: Now you are ready to add the Label control to the form. Make sure the *Common Controls* tab is open in the *Toolbox* window, as shown in Figure 2-5, and double-click the *Label* control icon. The label appears on the form with a dotted line around it and a small white square in its upper left corner, as shown in Figure 2-6. The dotted-line rectangle is called a **bounding box**—it marks the tightest rectangle that contains all parts of the control.

Figure 2-5 Label control tool

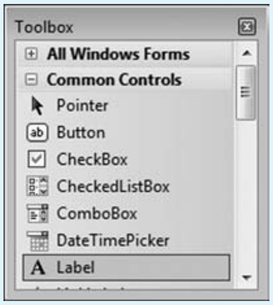
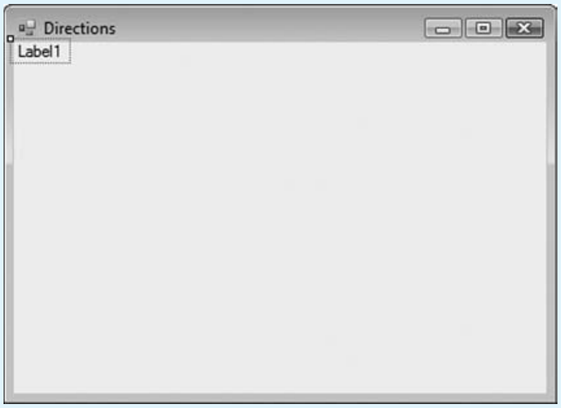
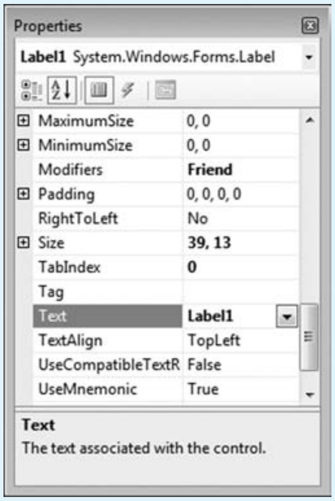


Figure 2-6 Label control on form



Step 2: Look at the *Properties* window. Because the label you just placed on the form is currently selected, the *Properties* window shows its properties (see Figure 2-7). The Text property is set, by default, to *Label1*. Double-click this value to select it, and replace its value by typing **`Directions to the Highlander Hotel`** in its place. Press the Enter key. When you have typed the new text into the Text property, the form appears, as shown in Figure 2-8. The label resizes itself to fit the contents of the Text property.

Figure 2-7 *Properties* window

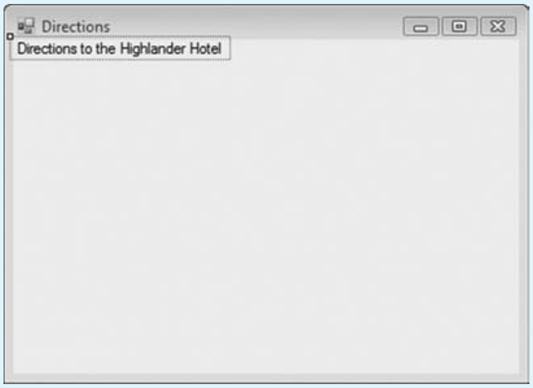


Label1 System.Windows.Forms.Label

⊞ MaximumSize	0, 0
⊞ MinimumSize	0, 0
Modifiers	**Friend**
⊞ Padding	0, 0, 0, 0
RightToLeft	No
⊞ Size	**39, 13**
TabIndex	**0**
Tag	
Text	**Label1**
TextAlign	TopLeft
UseCompatibleTextR	False
UseMnemonic	True

Text
The text associated with the control.

Figure 2-8 Label with new text property value



Step 3: Next, you will move the label to a new location on the form. Move the mouse over the label on the form, hold down the left mouse button, and drag the label to the top middle area of the form, as shown in Figure 2-9. From now on, we will refer to this type of operation as *dragging the control*.

Step 4: The AutoSize property of a label, which is *True* by default, makes the label automatically adjust its size depending on the contents of the Text property. Set the label's AutoSize property to *False* by double-clicking the AutoSize property. Figure 2-10 shows how square handles appear around the label. For practice, use the mouse to drag the handles and change the label's size.

Figure 2-9 After moving the Label control

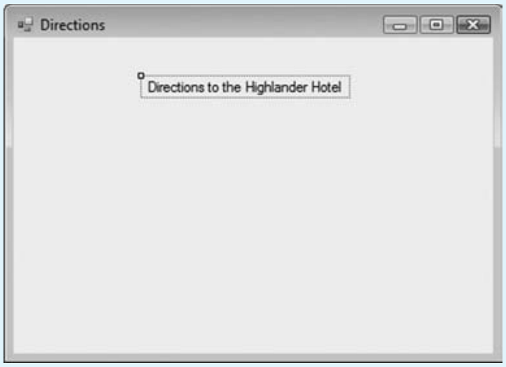
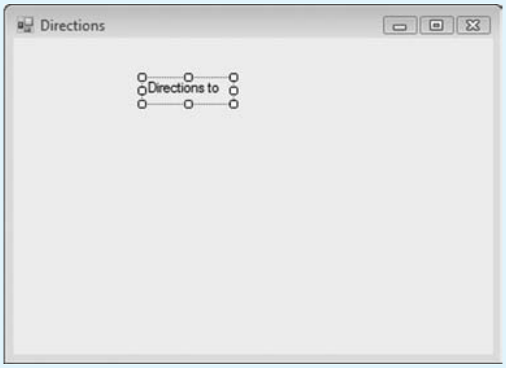
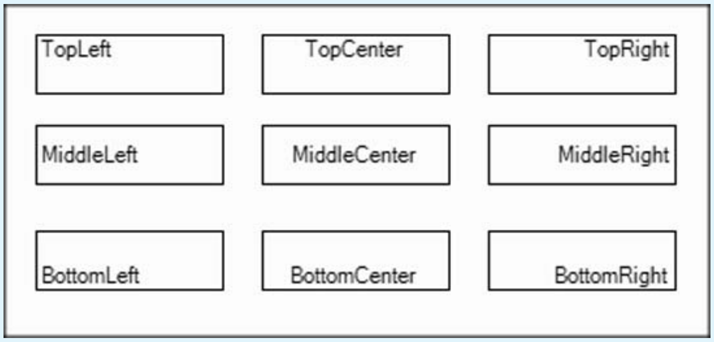


Figure 2-10 Label control, with AutoSize = *False*



By default, a label's text is aligned with the top and left edges of the label's bounding box. The position of the text within a label's bounding box is controlled by the TextAlign property, which may be set to any of the following values: *TopLeft*, *TopCenter*, *TopRight*, *MiddleLeft*, *MiddleCenter*, *MiddleRight*, *BottomLeft*, *BottomCenter*, or *BottomRight*. Figure 2-11 shows nine Label controls, each with a different TextAlign value.

Figure 2-11 Text alignments



Tutorial 2-3 takes you through the process of aligning the label's text.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Tutorial 2-3:
Setting the Label's TextAlign property

Step 1: With the Label selected, look at the *Properties* window. Notice that the value of the TextAlign property is *TopLeft*.

Step 2: Click the *TextAlign* property. Notice that a down-arrow button (▾) appears next to the property value. When you click the arrow, a small dialog box with nine buttons appears. Each of the buttons represents a TextAlign value, as shown in Figure 2-12.

Step 3: Click the *MiddleCenter* button. The label's text is now centered in the middle of the label's bounding box, as shown in Figure 2-13.

Figure 2-12 *TextAlign* drop-down dialog box

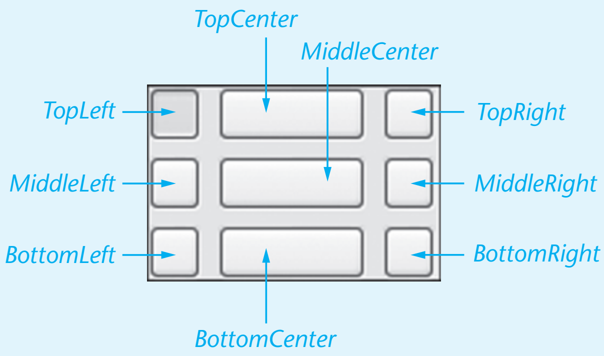
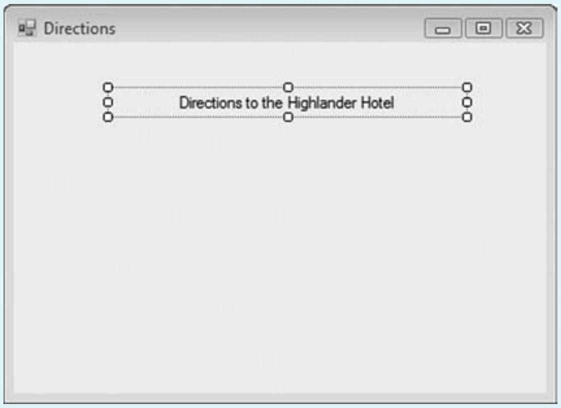


Figure 2-13 Label text centered



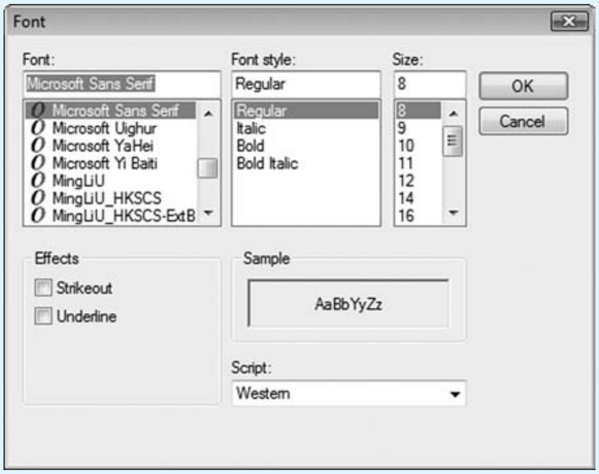
Directions to the Highlander Hotel

In the planning phase, we indicated that the label's text should be displayed in a 16-point bold Microsoft sans serif font. These characteristics are controlled by the label's Font property. The **Font property** allows you to set the font, font style, and size of the label's text. Tutorial 2-4 shows you how to change the font size and style of labels.

Tutorial 2-4:
Changing the Label's font size and style

Step 1: With the Label selected, click the *Font* property in the *Properties* window. Notice that an ellipsis button (…) appears. When you click the ellipsis button, the *Font* dialog box appears, as shown in Figure 2-14.

Figure 2-14 *Font* dialog box



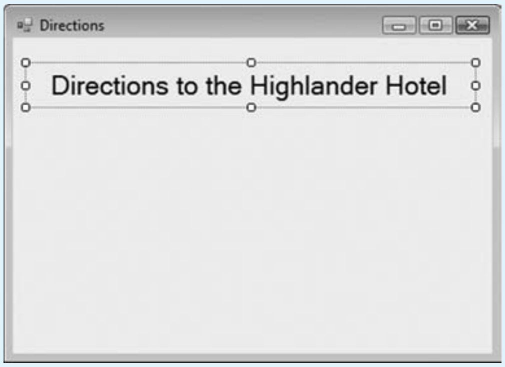
Step 2: *Microsoft Sans Serif* is already the selected font. Click *Bold* under *Font style*, and select *16* under *Size*. Notice that the text displayed in the *Sample* box changes to reflect your selections. Click the *OK* button.

The text displayed by the label is now in 16-point bold Microsoft sans serif. Unfortunately, not all the text can be seen because it is too large for the Label control. You must enlarge both the form and the Label control so all of the label text is visible.

Step 3: Select the form by clicking anywhere on it, except on the Label control. You will know you have selected the form when the **sizing handles** appear around it and the form's properties appear in the *Properties* window.

Step 4: Use the form's sizing handles to widen the form, and then select the label and enlarge it so it appears similar to the one shown in Figure 2-15.

Figure 2-15 Resized form showing all text



Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

To delete a control, select it and press the Delete key on the keyboard. In Tutorial 2-5 you add another Label control to the form (one that you will not need) and then you delete it.

Tutorial 2-5:
Deleting a control

Step 1: Double-click the *Label* tool in the *Toolbox*. Another Label control appears on the form.

Step 2: With the new Label control still selected, press the Delete key on the keyboard. The label is deleted from the form.

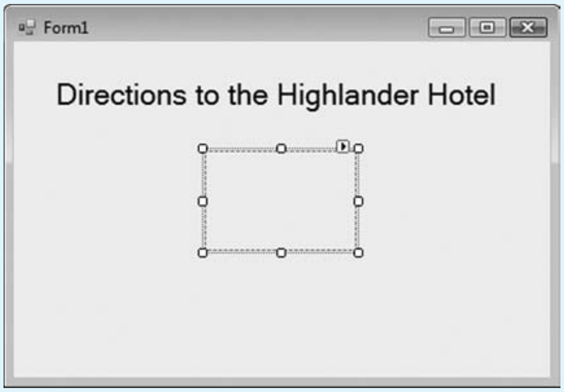
TIP: If you accidentally delete a control you can restore it with the *Undo* button (↶) on the standard toolbar.

The last step in building this application is to insert the street map. In Tutorial 2-6 you insert a **PictureBox control**, which can be used to display an image.

Tutorial 2-6:
Inserting a PictureBox control

Step 1: Double-click the *PictureBox* tool in the *Toolbox*. An empty PictureBox control appears on the form. Move the control to a position approximately in the center of the form, as shown in Figure 2-16.

Figure 2-16 PictureBox control placed



Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Step 2: The PictureBox control displays an image in a variety of ways, depending on the setting of the SizeMode property. Set the SizeMode property to *StretchImage*. This will make the image stretch or shrink so it fits within the bounding box of the PictureBox control.

The Image property is currently set to *(none)*, indicating that no image is loaded into the PictureBox control. Our next task is to load an image that was copied to your computer from the Student CD.

Step 3: Click the PictureBox's Image property and notice that a *Browse* button (...) appears next to the property value. Click the button to display the *Select Resource* dialog box, as shown in Figure 2-17. Select *Local resource* and click the *Import* button. When the *Open* dialog box appears, navigate to the file named *HotelMap.jpg* in the student sample programs directory, in the *Chap2* folder. After you click the *OK* button, the graphic shown in Figure 2-18 should appear in the PictureBox control.

Step 4: Because the SizeMode property is set to *StretchImage*, the image expands to fit the size of the PictureBox control. Use the control sizing handles to enlarge the image so its details are clearly visible (see Figure 2-19).

> **TIP:** You may want to enlarge the form again to make more room for the image.

> **NOTE:** You have now seen that properties are set in the *Properties* window in one of three ways:
>
> - Typing a value for the property
> - Selecting a value for the property from a drop-down list by clicking the down-arrow button (▾)
> - Establishing a value for the property with a dialog box, which appears when the *Browse* button (...) is clicked

Figure 2-17 Using the *Select Resource* dialog box to insert an image

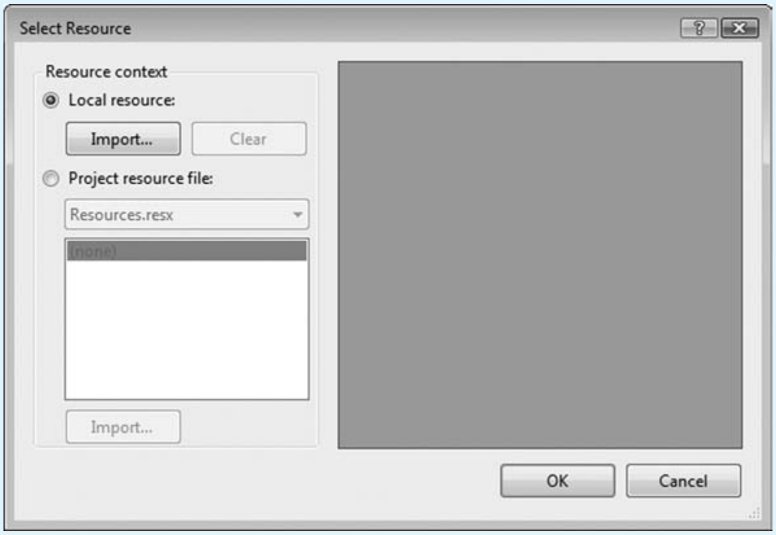


Figure 2-18 PictureBox control with image in place

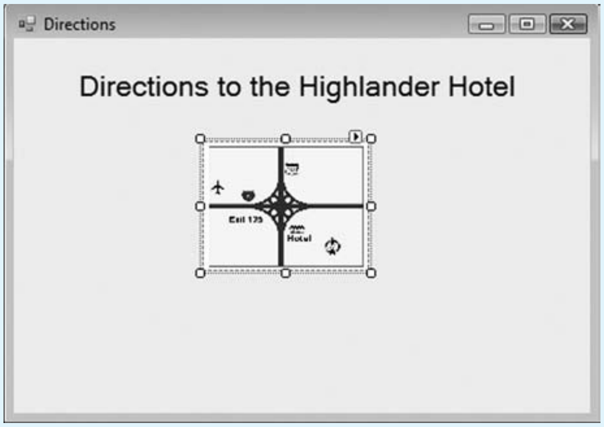
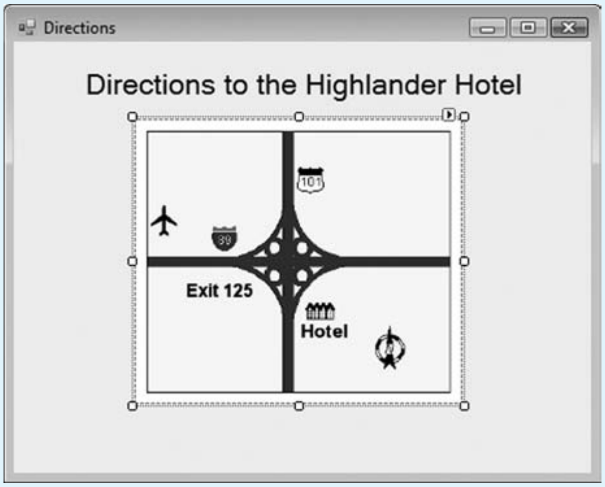


Figure 2-19 *HotelMap.jpg* enlarged



Now it's time to save the changes that you've made to your project. Tutorial 2-7 describes three different ways to save a project.

Tutorial 2-7:
Saving and running the application

Step 1: You may use any of the following methods to save a project (application).

- Click *File* on the menu bar, then click *Save All* on the *File* menu
- Press Ctrl+Shift+S on the keyboard
- Click the *Save All* button on the standard toolbar

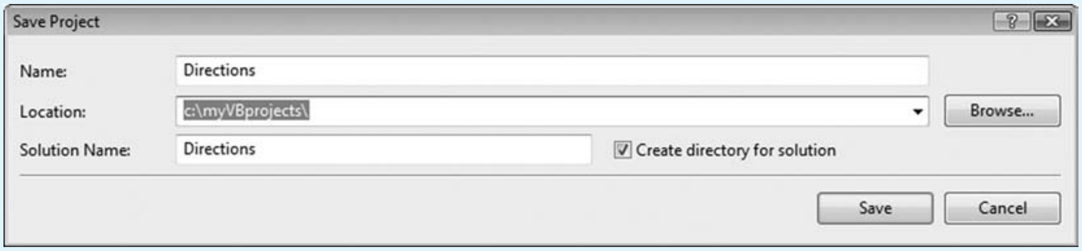
Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Use one of these methods to save your project. You will see the dialog window shown in Figure 2-20. If you wish, you can click the *Browse* button to change the directory in which your application will be saved.

Now you will run the application. It doesn't have any event procedures, so it will only display the PictureBox and Label. There are three ways to run an application in Visual Studio:

- Click the *Start Debugging* button (▶) on the toolbar
- Click *Debug* on the menu bar, then click *Start Debugging* on the *Debug* menu
- Press the F5 key

Figure 2-20 Confirming the *Name, Location,* and *New Solution Name* when saving a project



Save Project

Name: Directions

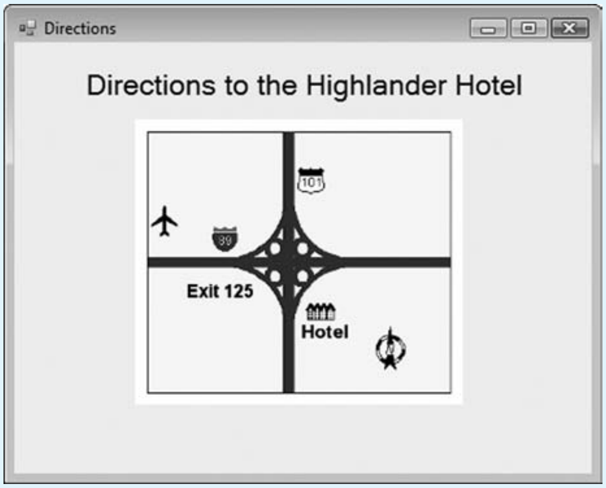
Location: c:\myVBprojects\

Solution Name: Directions

☑ Create directory for solution

Step 2: Run the application using one of the ways listed above. After a short delay, you will see the application's form display, as shown in Figure 2-21. We say that the program is now in Run mode.

Figure 2-21 Running the *Directions* application



Directions

Directions to the Highlander Hotel

TIP: Save your work often to prevent the accidental loss of changes you have made to your project.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Step 3: Now you will stop the application (end its execution). Select one of the following actions:

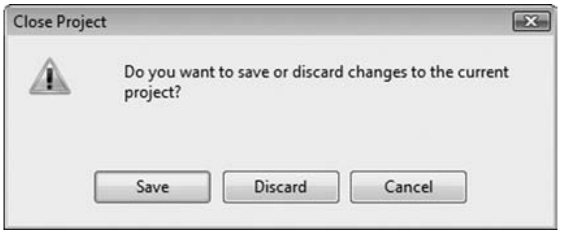
- Click the *Close* button (☒) on the application window
- Click *Debug* on the menu bar, then click *Stop Debugging* on the *Debug* menu

The application will stop and Visual Studio will return to Design mode.

Closing a Project

To close the current project, click *File* on the menu bar, and then click *Close Project*. If you have made changes to the project since the last time you saved it, you will see a dialog box asking you if you want to save your changes (see Figure 2-22).

Figure 2-22 Closing a project, confirming *Save* or *Discard*


Close Project

Do you want to save or discard changes to the current project?

Tutorial 2-8 explains how to close your Visual Basic project.

Tutorial 2-8:
Closing a Visual Basic project

Step 1: To close a Visual Basic project, click *File* on the menu bar, then click *Close Project*. If you are prompted to save changes, do so.

Step 2: You can exit Visual Studio (or Visual Basic Express) the same way you exit most other Windows applications:

- Click the *File* menu and then click the *Exit* command
- Click the *Close* button (☒) on the right edge of the title bar

Use one of these methods to exit Visual Studio.

Checkpoint

2.1 You want to change what is displayed in a form's title bar. Which of its properties do you change?

2.2 How do you insert a Label control onto a form?

2.3 What is the purpose of a control's sizing handles?

2.4 What are the possible values for a label's TextAlign property?

2.5 How do you delete a control?

2.6 What happens when you set a PictureBox control's SizeMode property to *StretchImage*?

2.7 What is the name of the dotted-line rectangle surrounding the Label control when looking at a form in Design mode?

How Solutions and Projects Are Organized on the Disk

A **solution** is a container that holds Visual Basic projects (see Figure 2-23). A project must belong to a solution. When you create a new project, Visual Studio automatically creates a solution with the same name as the project, and inserts the project in the solution. For example, when you created the *Directions* project, a solution named *Directions* was created, and the project was inserted in the *Directions* solution. In Visual Basic Express, the solution name is always the same name as the project.

Tutorial 1-4 showed that Visual Studio (or Visual Basic Express) uses a default location for saving projects and solutions. It creates a folder at this location, using the name of the solution (which is also the name of the project). Several files, and some other folders, are created and stored in this folder.

When you created the *Directions* project and the solution containing it, Visual Studio created a folder named *Directions* at the location specified in the *Save Project* dialog box. All files related to the project were stored in this folder (see Figure 2-24). The **solution file** is named *Directions.sln*. Inside the solution folder is a project folder (also named *Directions*). The contents of this folder are shown in Figure 2-25. The file named *Directions.vbproj* is called the **project file**.

Figure 2-23 Relationship between a solution and its projects

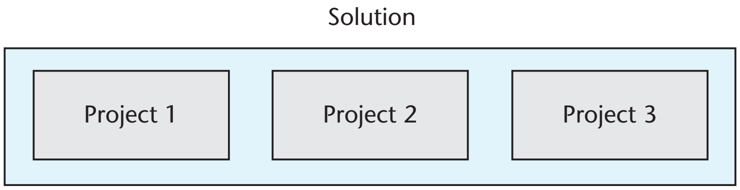
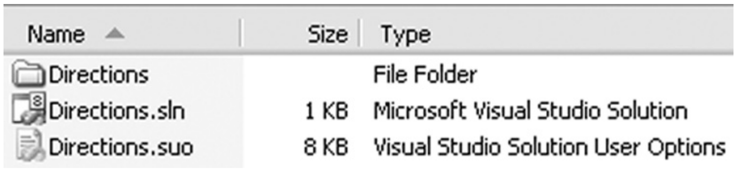
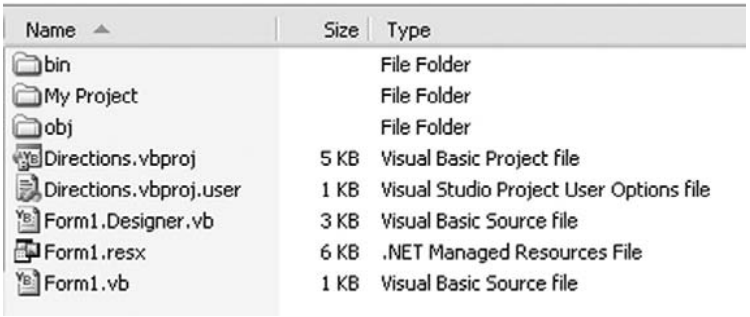


Figure 2-24 *Directions* solution folder



Name	Size	Type
Directions		File Folder
Directions.sln	1 KB	Microsoft Visual Studio Solution
Directions.suo	8 KB	Visual Studio Solution User Options

Figure 2-25 Inside the *Directions* project folder



Name	Size	Type
bin		File Folder
My Project		File Folder
obj		File Folder
Directions.vbproj	5 KB	Visual Basic Project file
Directions.vbproj.user	1 KB	Visual Studio Project User Options file
Form1.Designer.vb	3 KB	Visual Basic Source file
Form1.resx	6 KB	.NET Managed Resources File
Form1.vb	1 KB	Visual Basic Source file

Opening Existing Projects

There are three ways to open an existing project:

- Click the project name in the *Recent Projects* panel of the *Start Page*, as shown in Figure 2-26. The project should open immediately; if it doesn't, you may have moved the project to a different directory, deleted the project, or its files may have become corrupted.
- Click the *Open Project* button and browse for the project name. In Visual Studio, you will see the *Open Project* dialog box, as shown in Figure 2-27. Select the solution file (extension *.sln*) and click the *Open* button.
- Click *File* in the menu bar and then click *Open Project* in the *File* menu. Browse for the project using the *Open Project* dialog box, select the solution file, and click the *Open* button.

Tutorial 2-9 guides you through the steps of opening an existing project.

Figure 2-26 List of recent projects on the *Start Page*

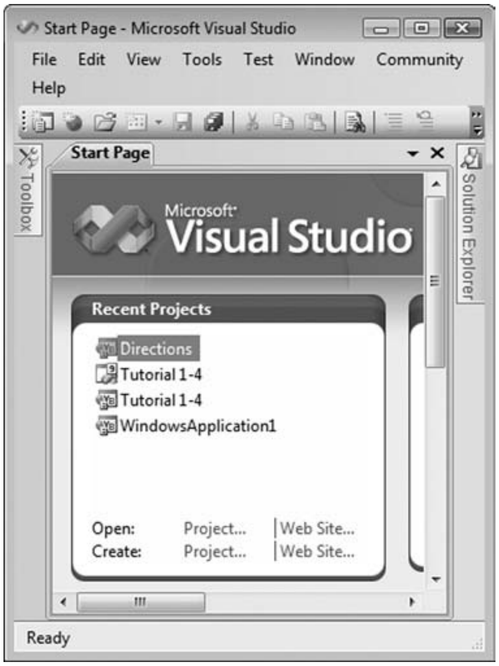
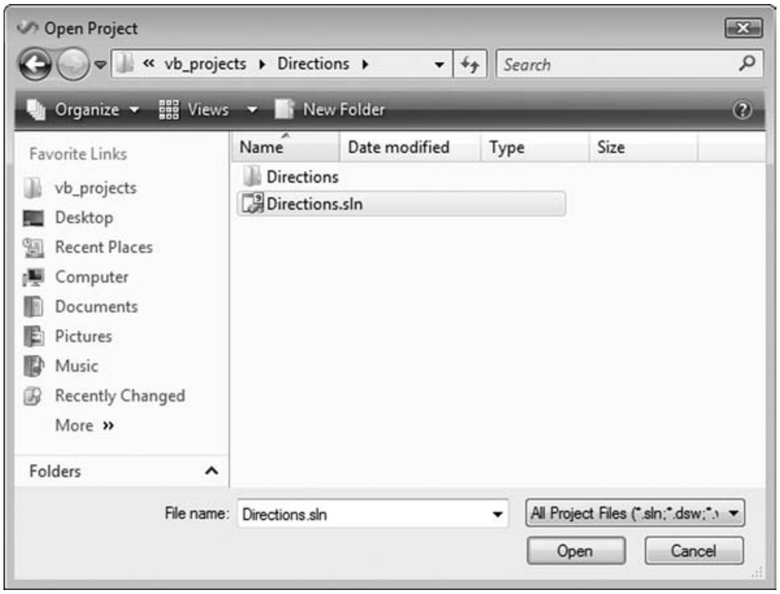

Start Page - Microsoft Visual Studio
File Edit View Tools Test Window Community
Help
Toolbox
Start Page
Solution Explorer
Microsoft
Visual Studio
Recent Projects
Directions
Tutorial 1-4
Tutorial 1-4
WindowsApplication1
Open: Project... | Web Site...
Create: Project... | Web Site...
Ready

Figure 2-27 *Open Project* dialog box with *Directions* solution folder open



Tutorial 2-9:
Opening an existing project

This tutorial assumes that the *Directions* project is saved to your disk and Visual Studio is not currently running.

Step 1: Start Visual Studio or Visual Basic Express. From the *Start Page*, open the *Directions* project by clicking its name in the *Recent Projects* panel. Another way to open the *Directions* project is to click *File* on the menu bar, and then click *Open Project*. This also causes the *Open Project* dialog box to appear. Browse to and open the *Directions* folder, select the *Directions.sln* file, and click the *Open* button.

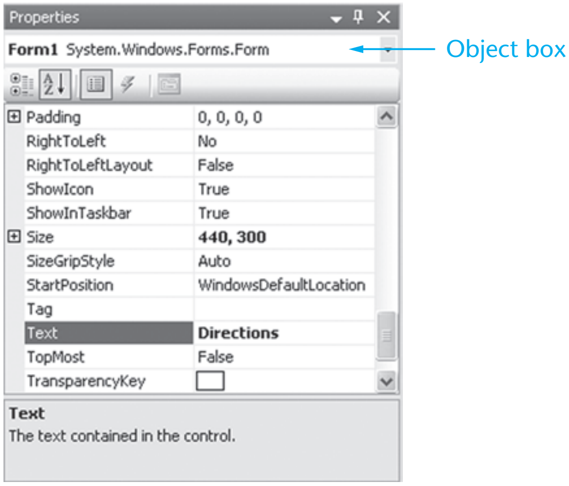
Step 2: After performing one of these actions, the *Directions* project should be open and you should see the form with the map to the hotel displayed in the *Design* window. If you do not see the form displayed, look in the *Solution Explorer* window and double-click the name *Form1.vb*.

(Leave the project open because you will use it in Tutorial 2-10.)

More about the *Properties* Window

In this section you will learn about the *Properties* window's **object box** and its *Alphabetical* and *Categorized* buttons. Figure 2-28 shows the location of the object box. The *Alphabetical* and *Categorized* buttons appear just below the object box, on a small toolbar. Figure 2-29 shows the location of the buttons on this toolbar.

Figure 2-28 Object box

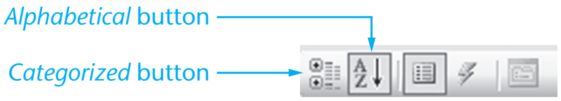


Form1 System.Windows.Forms.Form

⊞ Padding	0, 0, 0, 0
RightToLeft	No
RightToLeftLayout	False
ShowIcon	True
ShowInTaskbar	True
⊞ Size	**440, 300**
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	**Directions**
TopMost	False
TransparencyKey	□

Text
The text contained in the control.

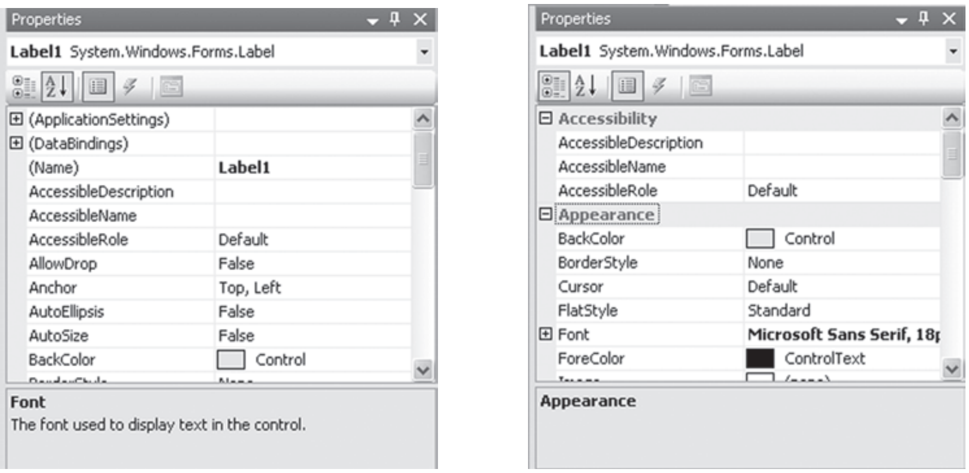
Figure 2-29 *Categorized* and *Alphabetical* buttons



In addition to clicking objects in the *Design* window, you can also use the object box on the *Properties* window to select any object in the project. The object box provides a drop-down list of the objects in the project.

The *Categorized* and *Alphabetical* buttons affect the way properties are displayed in the *Properties* window. When the ***Alphabetical* button** is selected, the properties are displayed in alphabetical order. When the ***Categorized* button** is selected, related properties are displayed together in groups. For example, in categorized view, a Label control's **BackColor property** and **BorderStyle property** are displayed within the *Appearance* group. Figure 2-30 shows examples of the *Properties* window in each view.

Figure 2-30 *Properties* window in alphabetical and categorized views



Label1 System.Windows.Forms.Label

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	**Label1**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	False
BackColor	□ Control

Font
The font used to display text in the control.

Label1 System.Windows.Forms.Label

⊟ **Accessibility**	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
⊟ **Appearance**	
BackColor	□ Control
BorderStyle	None
Cursor	Default
FlatStyle	Standard
⊞ Font	**Microsoft Sans Serif, 18p**
ForeColor	■ ControlText

Appearance

A few of the properties, including the Name property, are enclosed in parentheses. Because these properties are used so often, VB designers enclosed them in parentheses to make them appear at the top of the alphabetical list. In Tutorial 2-10 you practice using these components of the *Properties* window.

Tutorial 2-10:
Using the Object box, *Alphabetical* button, and *Categorized* button

Step 1: With the *Directions* project loaded, click the down-arrow button (▾) that appears at the right edge of the object box. You should see the names of the *Form1*, *Label1*, and *PictureBox1* controls.

Step 2: Click the name *Label1* in the list. *Label1* is now selected in the *Design* window. Repeat this procedure, selecting the *PictureBox1* control and the *Form1* control. As you select these objects, notice that they become selected in the *Design* window.

Step 3: Select the *Label1* control and click the *Categorized* button. Scroll through the list of properties displayed in the *Properties* window. Notice there are several categories of properties.

Checkpoint

2.8 Describe three ways to open an existing project.

2.9 What are the two viewing modes for the *Properties* window? How do you select either of these modes? What is the difference between the two?

2.10 Open the *Directions* project. In the *Properties* window, arrange *Form1*'s properties in categorized order. Under what category does the Text property appear? Under what category does the Name property appear?

2.11 How can you select an object using only the *Properties* window?

2.2 Focus on Problem Solving: Responding to Events

CONCEPT: An application responds to events, such as mouse clicks and keyboard input, by executing code known as *event procedures* or *event handlers*. In this section, you write event procedures for the directions application.

VideoNote
Responding to Events

The manager of the Highlander Hotel reports that the *Directions* application has been quite helpful to the desk clerks. Some clerks, however, requested that the application be modified to display written directions as well as the map. Some also requested a more obvious way to exit the application, other than clicking the standard Windows *Close* button, located on the application's title bar.

You decide to add a button to the application form that, when clicked, causes the written directions to appear. In addition, you decide to add an *Exit* button that causes the application to stop when clicked.

Figure 2-31 shows the modified sketch of the form presented by this application.

Figure 2-31 Modified *Directions* application sketch

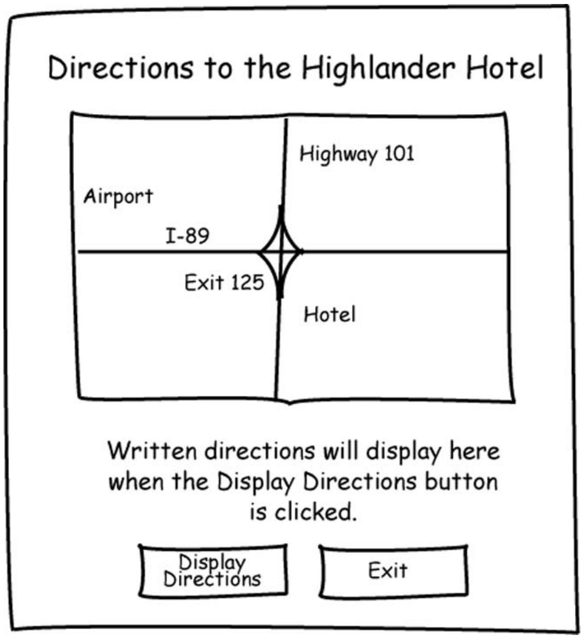

Directions to the Highlander Hotel
Highway 101
Airport
I-89
Exit 125
Hotel
Written directions will display here when the Display Directions button is clicked.
Display Directions
Exit

Table 2-3 lists the controls that will be added to the application. Because the Label control will be accessed in code and the buttons will have code associated with them, you will assign them names.

Table 2-3 Controls to be added to *Directions* application

Control Type	Control Name	Description
Label	`lblDirections`	Displays written directions to the hotel
Button	`btnDisplayDirections`	When clicked, causes the `lblDirections` control's text to appear on the form
Button	`btnExit`	Stops the application when clicked

VideoNote
The Name Property

Property settings for all controls are listed in Table 2-4. The table mentions a new property, Visible, used with the `lblDirections` control. Visible is a **Boolean property**, which means it can only hold one of two values: *True* or *False*. When a control's **Visible property** is set to *True*, the control can be seen on the form. A control is hidden, however, when its Visible property is set to *False*. In this application, we want the `lblDirections` control to be hidden until the user clicks the `btnDisplayDirections` button, so we initially set its Visible property to *False*.

Table 2-4 *Directions* application control properties

Property	Value
Label	
Name	`lblDirections`
Text	*Traveling on I-89, take Exit 125 onto Highway 101 South. The hotel is on the left, just past the I-89 intersection. Traveling on Highway 101 North, the hotel is on the right, just before the I-89 intersection.*
Visible	*False*
Button	
Name	`btnDisplayDirections`
Text	*Display Directions*
Button	
Name	`btnExit`
Text	*Exit*

Only two event procedures (event handlers) are needed in the *Directions* application, as shown in Table 2-5. `btnDisplayDirections_Click` is the name of the procedure that is invoked when the `btnDisplayDirections` button is clicked, and `btnExit_Click` is the event procedure that executes when the `btnExit` button is clicked.

Table 2-5 *Directions* application event procedures

Method	Description
`btnDisplayDirections_Click`	Causes the `lblDirections` control to become visible on the form; this is accomplished by setting the Label's Visible property to *True*
`btnExit_Click`	Terminates the application

Figure 2-32 shows a flowchart for the `btnDisplayDirections_Click` event procedure. Figure 2-33 shows a flowchart for the `btnExit_Click` event procedure.

Figure 2-32 Flowchart for `btnDisplayDirections_Click`

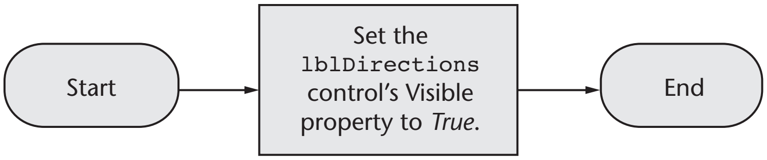
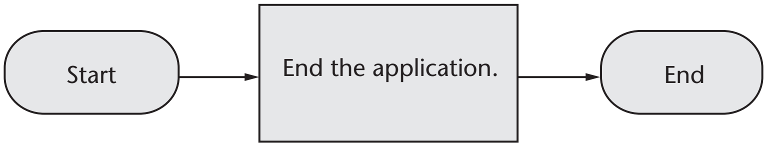


Figure 2-33 Flowchart for `btnExit_Click`



Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

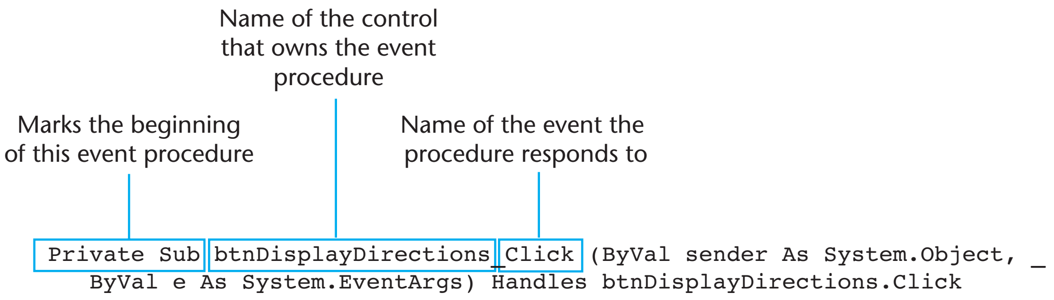
Now that you have seen flowcharts for the event procedures, let's look at the actual code you will write. The code for the `btnDisplayDirections_Click` event procedure is as follows:

```
Private Sub btnDisplayDirections_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles btnDisplayDirections.Click

  ' Make the directions visible
  lblDirections.Visible = True
End Sub
```

NOTE: The first two lines of code shown here will initially appear as one long line in the Visual Studio editor. Our example shows how to break the long line into two lines.

Event procedures are a type of Sub procedure. The word *Sub* is an abbreviation for the general term *subroutine*. This event procedure begins with the keywords `Private Sub` and ends with the keywords `End Sub`. Among other things, the first line of the event procedure (printed as two lines) identifies the control it belongs to and the event it responds to. This is illustrated in Figure 2-34.

Figure 2-34 Parts of the first line of an event procedure


Name of the control that owns the event procedure
Marks the beginning of this event procedure
Name of the event the procedure responds to

```
Private Sub btnDisplayDirections_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDisplayDirections.Click
```

In Figure 2-34, the name of the control owning the procedure appears after the words `Private Sub`. An underscore character separates the name of the owning control from the name of the event the procedure responds to (`Click`). Therefore, the lines in the figure indicate the beginning of a method belonging to the `btnDisplayDirections` control. The method responds to the `Click` event, by executing when the user clicks the button.

For now, don't be concerned with anything else that appears in the first two lines of code. As you progress through the book you will learn more about it.

The next line reads

```
' Make the directions visible
```

The apostrophe (`'`) marks the beginning of a comment. Recall from Chapter 1 that a *comment* is a note of explanation intended for people (including yourself) reading a program's source code. Comments are part of the program but do not affect the program's execution. The comment *Make the directions visible* explains in ordinary terms what action the next line of code performs. A person reading the code doesn't have to guess what it does.

Always use descriptive comments in your code that explain how and why you are performing tasks. Imagine creating a large and complex application. Once you have tested and debugged it, you give it to its user and move on to the next project. Ten months later, the user asks you to make a modification (or worse, to track down and fix an elusive bug). As you look through several hundred lines of code, you are astonished to discover that some of it makes no sense! If only you had left notes to yourself explaining the different parts of the program. Writing comments takes time, but it almost always saves time later.

The next line in our sample code reads:

```
lblDirections.Visible = True
```

This is an **assignment statement**. The equal sign, known as the **assignment operator**, copies the value on its right side into the item on its left. The item to the left of the operator is `lblDirections.Visible`. It identifies the Visible property of the `lblDirections` control. The standard notation for referring to a control's properties in code is: `ControlName.PropertyName`. The value to the right of the equal sign, `True`, is copied into the `lblDirections.Visible` property. The effect of this statement is that the `lblDirections` control becomes visible on the form.

TIP: In an assignment statement, the name of the item receiving the value must be on the left side of the = operator. The following statement, for example, is wrong:

```
True = lblDirections.Visible            'Error
```

Statements between the first and last lines of the `btnDisplayDirections` procedure are indented. Although it is not required, it is a common practice to indent the lines inside a procedure so they are visually set apart. In this book, we are careful to use correct indentation. Many programmers prefer to leave a blank line before the `End Sub` statement, for readability. While we do not follow the convention, partly to save printing space, we think it is an excellent idea.

`btnExit_Click` Procedure

Next, let's look at the `btnExit_Click` event procedure:

```
Private Sub btnExit_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnExit.Click

  'End the application by closing the window
  Me.Close()
End Sub
```

This procedure contains only one executable statement: `Me.Close()`, which closes the main window (*Form1*) and ends the program. The `Me` object is used to identify the current Form object (*Form1*). When the form closes, the application ends. In Tutorial 2-11 you add the controls required to complete this part of the application.

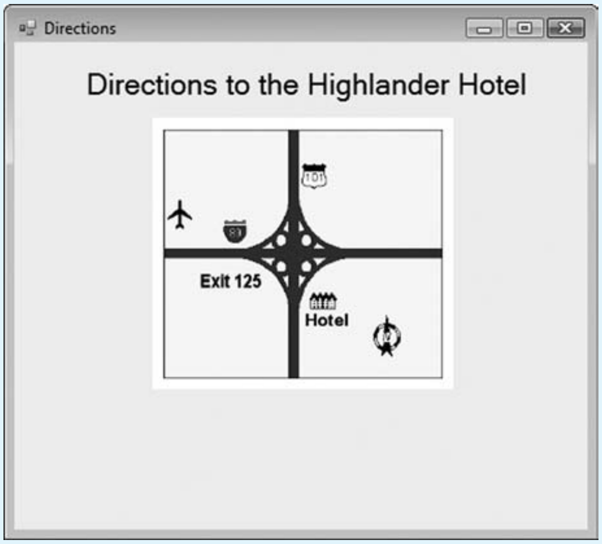
Tutorial 2-11:

Placing the `lblDirections`, `btnDisplayDirections`, and `btnExit` controls in the *Directions* application

Step 1: Start Visual Studio and open the *Directions* project. View the *Form1* form, in Design mode. If it is not visible, double-click its name in *Solution Explorer*.

Step 2: You will place the new controls at the bottom of the form, below the graphic. Because the form is too small to accommodate them, you will need to enlarge it. Drag the bottom edge of the form down until it looks something like the one shown in Figure 2-35. (Don't worry about the exact size of the form. You can adjust it again later.)

Figure 2-35 *Directions* application with enlarged form



Step 3: Create a new Label control on the form and move it to a location below the image.

Step 4: Make sure the new Label control is selected and set its AutoSize property to *False*.

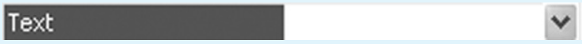
Step 5: Resize the Label control and position it so it appears similar to the *Label2* control shown in Figure 2-36.

Step 6: Change the Label control's Name property to `lblDirections`.

Step 7: You are about to enter the following text into the label's Text property:

```
Traveling on I-89, take Exit 125 onto Highway 101 South.
The hotel is on the left, just past the I-89 intersection.
Traveling on Highway 101 North, the hotel is on the right,
just before the I-89 intersection.
```

Select the label's Text property, and then click the down-arrow button (▾):



Text

When the editing box appears, as shown in Figure 2-37, type the hotel directions shown above. Type all the text on a single line before pressing the Enter key. When you are finished, click the mouse in a different property, and the text you typed will be saved.

TIP: The editing box you just used permits you to press the Enter key at the end of individual lines.

Step 8: Make sure the Label control's TextAlign property is set to *TopLeft*. The form should now appear similar to the one shown in Figure 2-38.

Step 9: Double-click the label's Visible property to set it to *False*.

Figure 2-36 Positioning a Label control

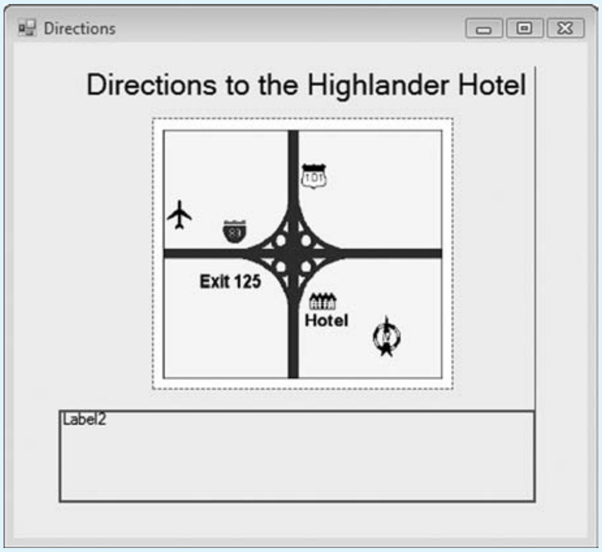
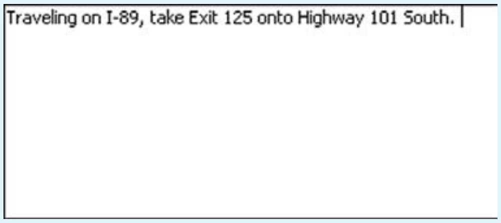


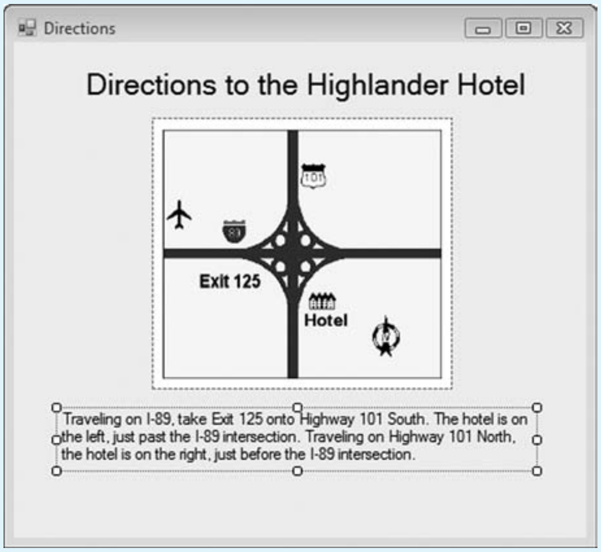
Figure 2-37 Editing box for the label's Text property



Traveling on I-89, take Exit 125 onto Highway 101 South.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.
Copyright © 2010 by Pearson Education, Inc.

Figure 2-38 Label with directions text entered



Directions to the Highlander Hotel

Traveling on I-89, take Exit 125 onto Highway 101 South. The hotel is on the left, just past the I-89 intersection. Traveling on Highway 101 North, the hotel is on the right, just before the I-89 intersection.

> **NOTE:** In Design mode, all controls are displayed, even if their Visible property is set to *False*.

Step 10: You are now ready to place the buttons. Double-click the button tool in the *Toolbox* window. A default-sized button named `Button1` appears on the form. Drag it near the form's bottom edge.

Step 11: Double-click the button tool in the *Toolbox* window again. Another button, this one named `Button2`, appears on the form.

> **NOTE:** If a control is already selected when you double-click a tool in the *Toolbox*, the new control will appear on top of the selected control. If the `Button1` control was still selected when you created `Button2`, the `Button2` control will appear on top of the `Button1` control.

Drag the `Button2` control to the bottom edge of the form and place it to the right of the `Button1` control. Arrange the two buttons, as shown in Figure 2-39.

Step 12: Select the `Button1` button.

Step 13: In the *Properties* window, change the button's Name to **`btnDisplayDirections`**.

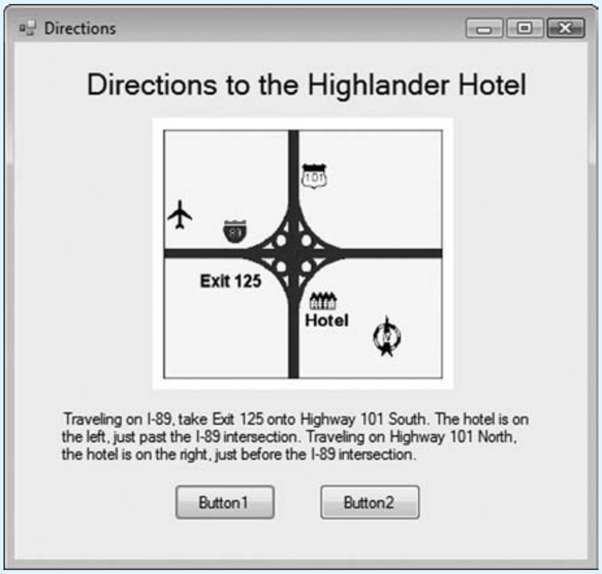
Step 14: Change the Text property to **`Display Directions`**. Because the button is not large enough to accommodate the text, use the button's sizing handles to increase its height, as shown in Figure 2-40.

Step 15: Select the `Button2` button.

Step 16: In the *Properties* window, change the button's name to **`btnExit`** and change the button's Text property to **`Exit`**.

Step 17: Resize the `btnExit` button so its size is the same as the `btnDisplayDirections` button. Your form should now resemble the one shown in Figure 2-41.

Figure 2-39 Buttons in place



Traveling on I-89, take Exit 125 onto Highway 101 South. The hotel is on the left, just past the I-89 intersection. Traveling on Highway 101 North, the hotel is on the right, just before the I-89 intersection.

Figure 2-40 Button with increased height

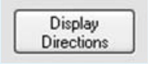
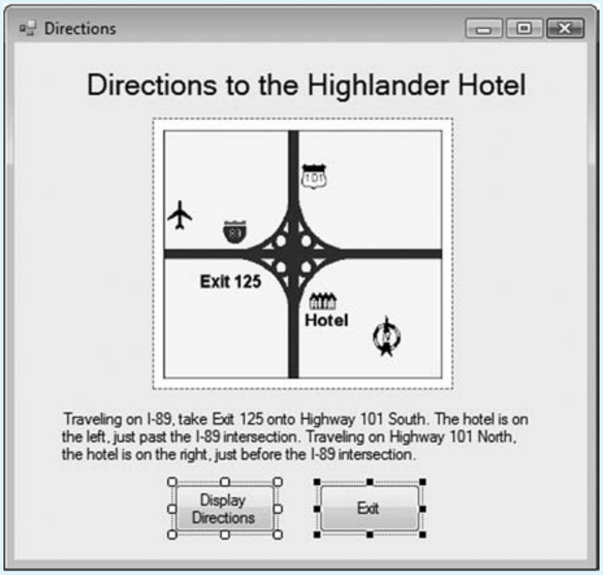


Figure 2-41 Buttons placed



Traveling on I-89, take Exit 125 onto Highway 101 South. The hotel is on the left, just past the I-89 intersection. Traveling on Highway 101 North, the hotel is on the right, just before the I-89 intersection.

In Tutorial 2-12 you write the event procedures for the buttons.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

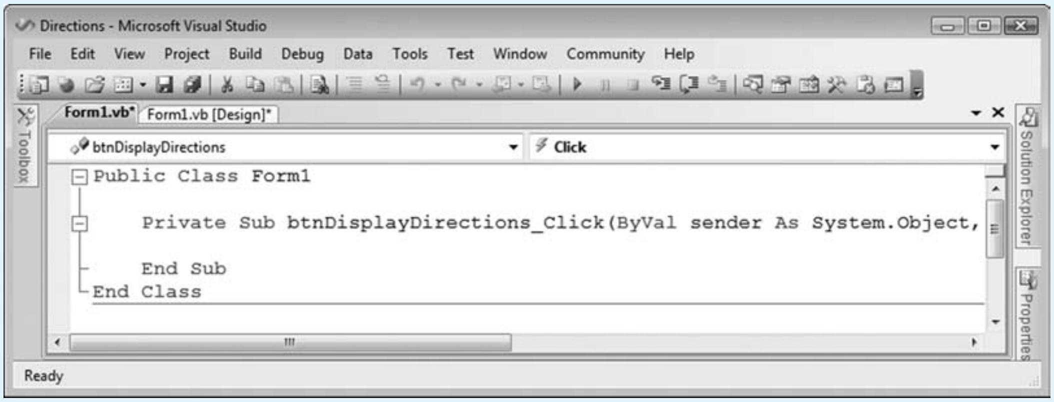
Tutorial 2-12:

Writing event procedures for the *Directions* application

Step 1: Double-click the *Display Directions* button (`btnDisplayDirections`). The **Code window** opens, as shown in Figure 2-42.

NOTE: The *Code* window is a text-editing window in which you write code. Notice that a code template appears for the `btnDisplayDirections_Click` event procedure. The template consists of the first and last lines of the procedure. You must add the code that appears between these two lines.

Figure 2-42 *Code* window showing the event handler for the `btnDisplayDirections` button


Directions - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Toolbox
Form1.vb*
Form1.vb [Design]*
btnDisplayDirections
Click
Solution Explorer
Properties
Ready

```
Public Class Form1

    Private Sub btnDisplayDirections_Click(ByVal sender As System.Object,

    End Sub
End Class
```

Step 2: Type the following code between the first and last lines of the `btnDisplayDirections_Click` procedure:

```
' Make the directions visible.
lblDirections.Visible = True
```

TIP: Make sure you type the code exactly as it appears here. Otherwise, you may encounter an error when you run the application.

Did you notice that as you entered the second line,

`lblDirections.Visible = True`,

when you typed the period the scrollable list appeared (see Figure 2-43)? This list is called an **IntelliSense** auto list box. It provides help and some automatic code completion while you are developing an application. The list displays information that may be used to complete part of a statement. It contains the name of every property and method belonging to the `lblDirections` object. When you type the letter *V*, the selector bar in the list box automatically moves to Visible. You can continue typing, or you can press the Tab key or Spacebar to select *Visible* from the list. You may then continue typing.

Another **auto list box** appears when you type the = operator, showing two values: *False* and *True* (see Figure 2-44). *False* and *True* are the only valid values you can assign to the Visible property. Either continue typing or let the auto list box help you select the code to insert.

TIP: If you did not see the auto list box, then that feature has been disabled. To enable it, click *Tools* on the menu bar, then click *Options . . .* Click the *Show all settings* option, then select *Text Editor*, *Basic*, and *General* (see Figure 2-45). Verify that the *Auto list members* option is checked.

Figure 2-43 Auto list box

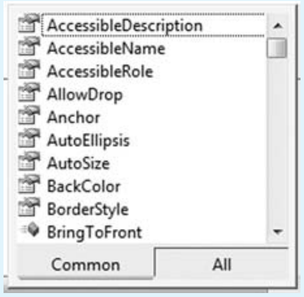


Figure 2-44 Selecting between *False* and *True*

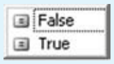
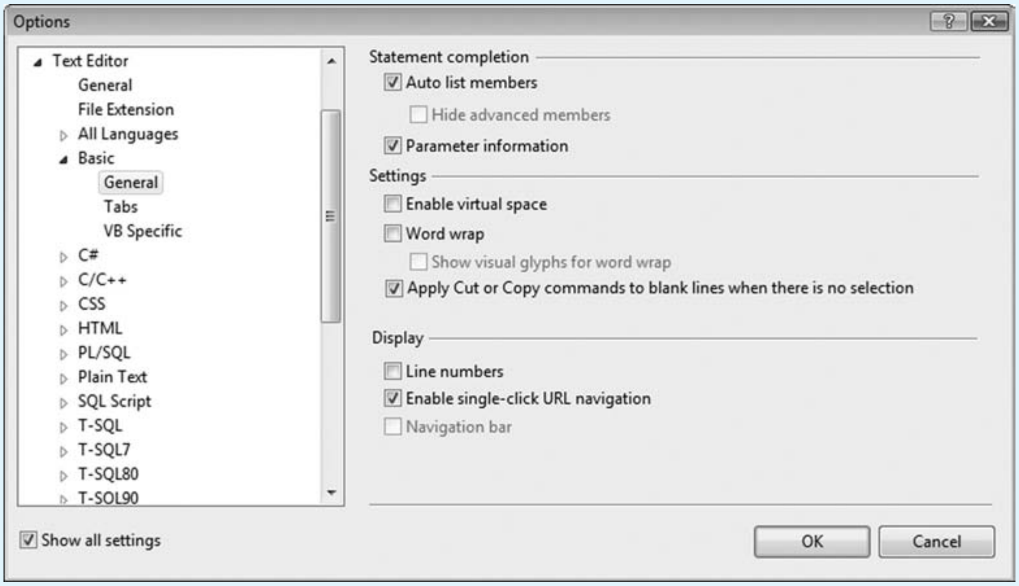


Figure 2-45 Controlling the general editor settings for Visual Basic



Statement completion

☑ Auto list members

□ Hide advanced members

☑ Parameter information

Settings

□ Enable virtual space

□ Word wrap

□ Show visual glyphs for word wrap

☑ Apply Cut or Copy commands to blank lines when there is no selection

Display

□ Line numbers

☑ Enable single-click URL navigation

□ Navigation bar

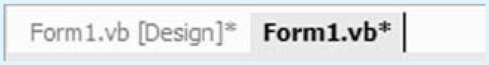
☑ Show all settings

Step 3: Now you will switch back to the *Design* window. There are four different ways to do this. Use one of these techniques to open the *Design* window.

- Notice at the top of the *Code* window the two tabs shown in Figure 2-46. You can open the *Design* window by clicking the *Form1.vb [Design]* tab, and then you can open the *Code* window again by clicking the *Form1.vb* tab.

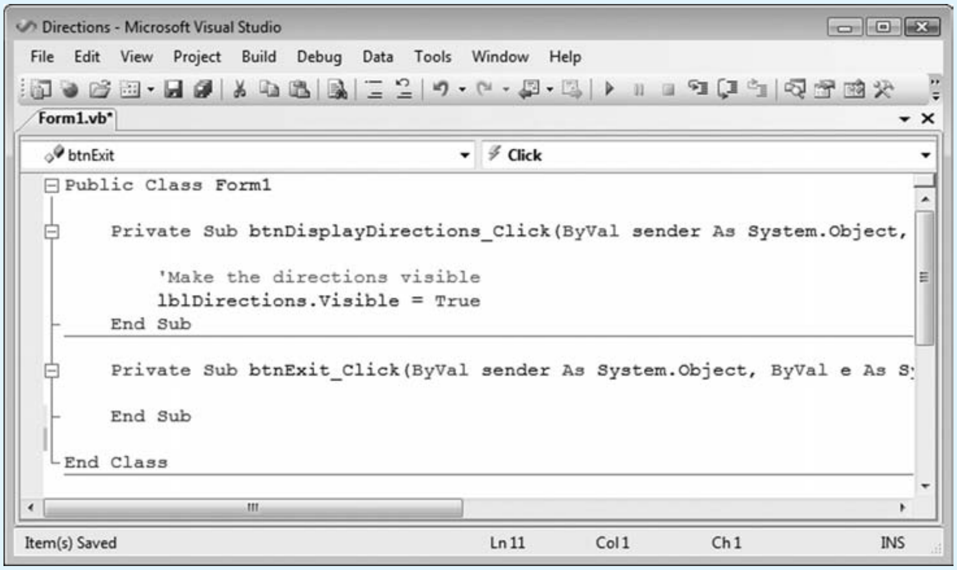
- You can click *View* on the menu bar, and then click *Designer* on the *View* menu.
- You can use the *Solution Explorer* window. Click the *View Designer* button (☰) to switch to the *Design* window, and click the *View Code* button (☰) to switch back to the *Code* window.
- You can also press Shift+F7 on the keyboard.

Figure 2-46 Design and code tabs



Step 4: Now you must write the event procedure for the other button. On the form, double-click the *Exit* button. The *Code* window reappears, as shown in Figure 2-47.

Figure 2-47 *Code* window, ready for *Exit* button code



```
Public Class Form1

    Private Sub btnDisplayDirections_Click(ByVal sender As System.Object,

        'Make the directions visible
        lblDirections.Visible = True
    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As S

    End Sub

End Class
```

Notice that Visual Studio has now provided a **code template** for the `btnExit_Click` procedure.

Step 5: Between the first and last lines of the `btnExit_Click` code template, type the following code:

```
' End the application by closing the window.
Me.Close()
```

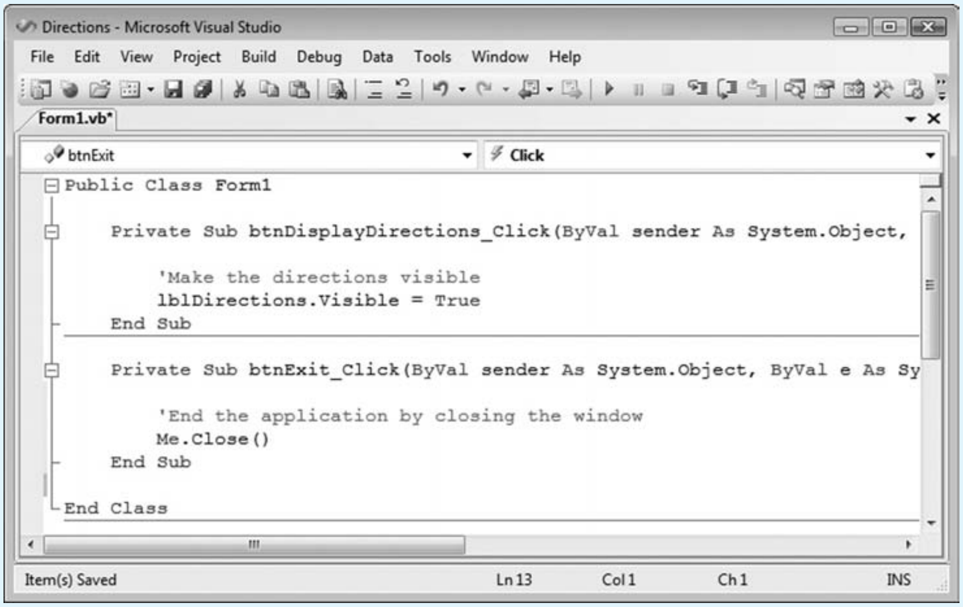
The *Code* window should now look like the one shown in Figure 2-48.

Step 6: Use the *Save All* command on the *File* menu to save the project.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.
Copyright © 2010 by Pearson Education, Inc.

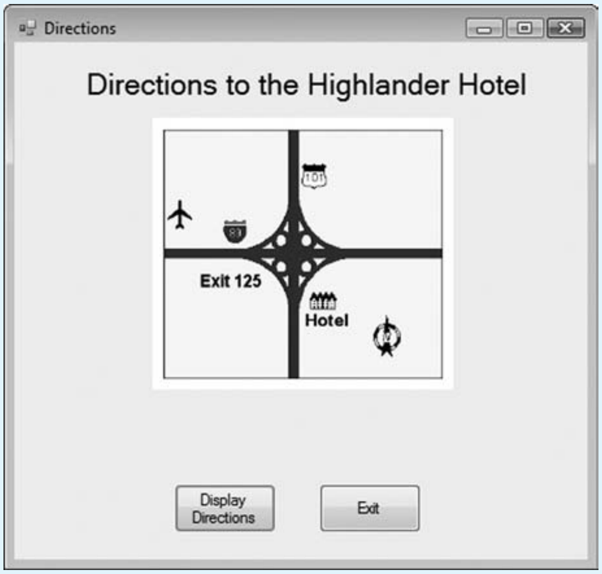
Step 7: Click the *Start* button (▶) on the Visual Studio toolbar. The application begins executing, as shown in Figure 2-49. Notice that the `lblDirections` label is not visible.

Figure 2-48 *Code* window with `btnExit_Click` procedure completed


Directions - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Help
Form1.vb*
btnExit
Click
Item(s) Saved
Ln 13
Col 1
Ch 1
INS

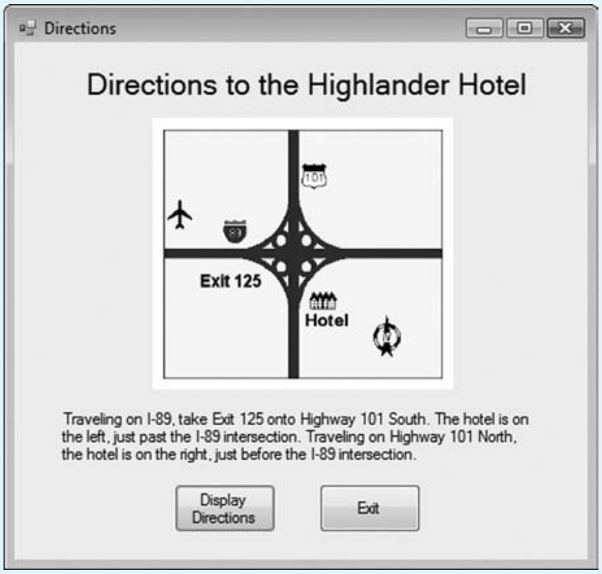
```
Public Class Form1

    Private Sub btnDisplayDirections_Click(ByVal sender As System.Object,

        'Make the directions visible
        lblDirections.Visible = True
    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As Sy

        'End the application by closing the window
        Me.Close()
    End Sub

End Class
```

Figure 2-49 *Directions* application at startup



Step 8: Click the *Display Directions* button. The written directions appear on the form, as shown in Figure 2-50.

Figure 2-50 *Directions* application with text displayed



Traveling on I-89, take Exit 125 onto Highway 101 South. The hotel is on the left, just past the I-89 intersection. Traveling on Highway 101 North, the hotel is on the right, just before the I-89 intersection.

Step 9: Click the *Exit* button. The application terminates.

Checkpoint

2.12 How do you decide if you will keep a control's default name or assign it a name?

2.13 What is a Boolean property?

2.14 Suppose an application has a button named `btnShowName`. The button has an event procedure that executes when the user clicks it. What would the event procedure be named?

2.15 Assume the following line appears in a button's `Click` event procedure. What does the line cause the application to do?

```
' Display the word "Hello"
```

2.16 What is the purpose of a remark (or comment)?

2.17 Suppose an application has a Label control named `lblSecretAnswer`. Write an assignment statement that causes the control to be hidden. (*Hint*: Use the Visible property.)

2.18 What is the purpose of the `Me.Close()` statement?

Changing Text Colors

You have already learned that the font style and size of text on a control can be easily changed through the *Properties* window. You can also change the text background and foreground color with the BackColor and ForeColor properties. Tutorial 2-13 walks you through the process.

Tutorial 2-13:
Changing the text colors

Step 1: With the *Directions* project loaded, open the *Design* window.

Step 2: Select the *Label1* control. This is the Label control whose Text property reads *Directions to the Highlander Hotel*.

Step 3: In the *Properties* window, look for the BackColor property. This is the property that establishes the background color for the label text. Click the property value. A down-arrow button (▼) appears.

Step 4: Click the down-arrow button (▼). A drop-down list of colors appears, as shown in Figure 2-51.

The drop-down list has three tabs: *Custom*, *Web*, and *System*. The *System* tab lists colors defined in the current Windows configuration. The *Web* tab lists colors displayed with consistency in Web browsers. The *Custom* tab displays a color palette, as shown in Figure 2-52.

Select a color from one of the tabs. Notice that the label text background changes to the color you selected.

Figure 2-51 BackColor drop-down list

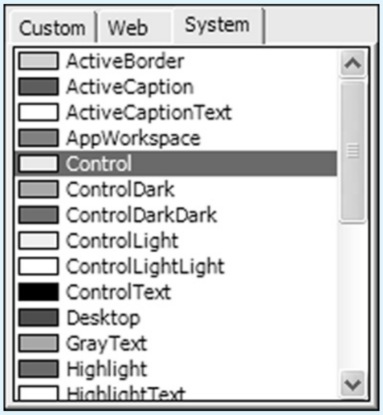
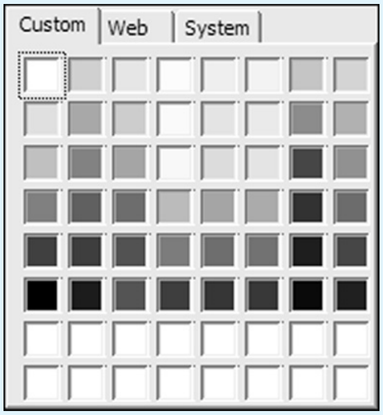


Figure 2-52 Custom color palette



Step 5: Now look for the **ForeColor property** in the *Properties* window. When you click it, a down-arrow button (▼) appears.

Step 6: Click the down-arrow button (▼) and notice the same drop-down list as you saw in Step 4. Once again, select a color from one of the tabs. Notice that the label's text foreground color changes to the color you selected.

Step 7: Start the application to test the new colors. After you close the application, save your project.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Setting the FormBorderStyle Property and Locking Controls

The FormBorderStyle Property

Sometimes you want to prevent the user from resizing, minimizing, or maximizing a window, or from closing a window using its *Close* button (☒). You can control all of these actions by selecting an appropriate value for the form's **FormBorderStyle property**. Table 2-6 shows a list of the possible values for the FormBorderStyle property.

Table 2-6 Values for FormBorderStyle

Value	Description
Fixed3D	Displays a 3D border. The form's size is fixed and displayed with *Minimize*, *Maximize*, and *Close* buttons on its title bar. Although the form may be maximized and minimized, it may not be resized by its edges or corners.
FixedDialog	This type of border shows *Minimize*, *Maximize*, and *Close* buttons on its title bar. Although the form may be maximized and minimized, it may not be resized by its edges or corners.
FixedSingle	The form's size is fixed and uses a border that is a single line. The form is displayed with *Minimize*, *Maximize*, and *Close* buttons on its title bar. Although the form may be maximized and minimized, it may not be resized by its edges or corners.
FixedToolWindow	Intended for use with floating toolbars. Only shows the title bar with a *Close* button. May not be resized.
None	The form is displayed with no border at all. Subsequently, there is no title bar, and no *Minimize*, *Maximize*, and *Close* buttons. The form may not be resized.
Sizable	This is the default value. The form is displayed with *Minimize*, *Maximize*, and *Close* buttons on its title bar. The form may be resized, but the controls on the form do not change position.
SizableToolWindow	Like *FixedToolWindow*, but resizable.

Locking Controls

Once you have placed all the controls in their proper positions on a form, it is usually a good idea to lock them. When you lock the controls on a form, they cannot be accidentally moved at design time. They must be unlocked before they can be moved.

To lock all the controls on a form, place the cursor over an empty spot on the form and right-click. A small menu pops up. One of the selections on the menu is *Lock Controls*.

In Tutorial 2-14 we modify the value of the form's FormBorderStyle property so the user cannot minimize, maximize, or resize the window. We will also lock the controls on the form.

Tutorial 2-14:
Setting the FormBorderStyle property and locking the controls in the *Directions* application

Step 1: Select the *Directions* form and find the FormBorderStyle property in the *Properties* window.

Step 2: Click the FormBorderStyle property. A down-arrow button (⌄) appears. Click the down-arrow button (⌄) to see a list of values.

Step 3: Click *FixedSingle*.

Step 4: Start the application and test the new border style. Notice that you can move the window, but you cannot resize it by its edges or its corners.

Step 5: Click the *Exit* button to end the application.

Step 6: Now you will lock the controls. Place the cursor over an empty spot on the form and right-click. A small menu pops up.

Step 7: Click the *Lock Controls* command.

Step 8: Select any control on the form and try to move it. Because the controls are locked, you cannot move them.

Step 9: Save the project.

WARNING: Be careful. Locked controls can still be deleted.

When you are ready to move the controls, just right-click over an empty spot on the form and select the *Lock Controls* command again. This toggles (reverses) the locked state of the controls.

Printing Your Code

To print a project's code, open the *Code* window, click *File* on the menu bar, and then click the *Print* command on the *File* menu.

Checkpoint

2.19 Which function key displays the *Code* window?

2.20 What three color categories are available when you edit a control's BackColor property?

2.21 What happens when you modify a Label control's BackColor property?

2.22 What happens when you modify a Label control's ForeColor property?

2.23 What property do you set in order to prevent a form from being resized when the application is running?

2.24 What happens when you lock the controls on a form?

2.25 How do you lock the controls on a form?

2.26 How do you unlock the controls on a form?

2.3 Modifying the Text Property with Code

CONCEPT: Quite often, you will need to change a control's Text property with code. This is done with an assignment statement.

While building the directions application, you learned that an assignment statement copies a value into a property while the application is running. Recall that the following statement sets the `lblDirections` control's Visible property to *True*.

```
lblDirections.Visible = True
```

You use the same technique to change the value of a control's Text property. For example, assume an application has a Label control named `lblMessage`. The following statement copies the sentence *Programming is fun!* to the control's Text property.

```
lblMessage.Text = "Programming is fun!"
```

Once the statement executes, the message displayed by the `lblMessage` control changes to

```
Programming is fun!
```

The quotation marks in the statement are not part of the message. They simply mark the beginning and end of the set of characters assigned to the property. In programming terms, a group of characters inside a set of quotation marks is called a **string literal**. You'll learn more about string literals in Chapter 3.

We usually display messages on a form by setting the value of a Label control's Text property. In Tutorial 2-15 you open and examine an application on the Student CD that demonstrates this technique.

Tutorial 2-15:
Examining an application that displays messages

Step 1: Start Visual Studio and open the *KiloConverter* project from the student sample programs folder named *Chap2\KiloConverter*.

Step 2: Open the *Design* window, which displays *Form1*, as shown in Figure 2-53.

Step 3: Click the *Start* button (▶) to run the application.

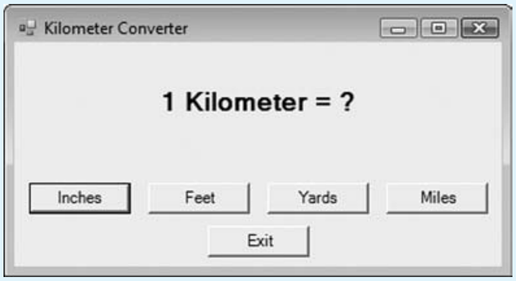
Step 4: Once the application is running, click the *Inches* button. The form displays the number of inches equivalent to a kilometer, as shown in Figure 2-54.

Step 5: Experiment with the other buttons and observe the messages that are displayed when each is clicked.

Step 6: Click the *Exit* button to exit the application. Let's examine the application code. Figure 2-55 shows the *KiloConverter* application form with its controls labeled.

The buttons `btnInches`, `btnFeet`, `btnYards`, and `btnMiles` each change the `lblMessage` Text property when clicked. Use the following step to view the *KiloConverter* event procedures.

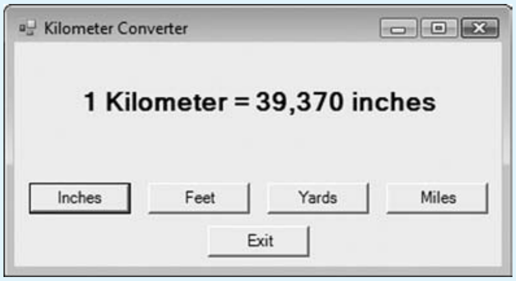
Figure 2-53 *Kilometer Converter* form



Kilometer Converter

1 Kilometer = ?

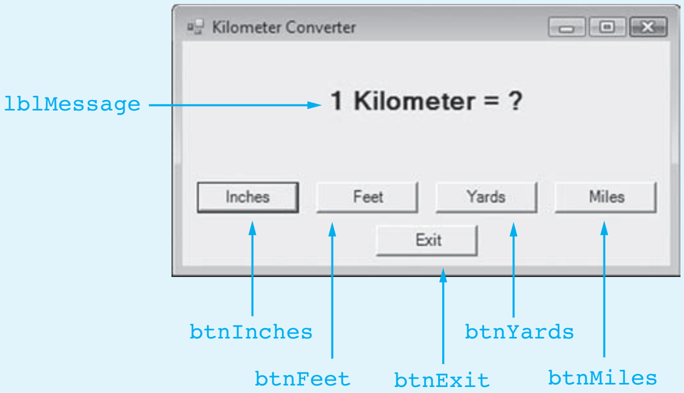
Figure 2-54 One kilometer converted to inches



Kilometer Converter

1 Kilometer = 39,370 inches

Figure 2-55 *KiloConverter* application controls



Kilometer Converter

1 Kilometer = ?

Step 7: Click the *View Code* button (☰) in the *Solution Explorer* window. The *Code* window appears. The window shows the following event procedure code:

```
Private Sub btnExit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click

  ' End the application.
  Me.Close()
End Sub
```

```
Private Sub btnFeet_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnFeet.Click

  ' Display the conversion to feet.
  lblMessage.Text = "1 Kilometer = 3,281 feet"
End Sub

Private Sub btnInches_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnInches.Click

  ' Display the conversion to inches.
  lblMessage.Text = "1 Kilometer = 39,370 inches"
End Sub

Private Sub btnMiles_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnMiles.Click

  ' Display the conversion to miles.
  lblMessage.Text = "1 Kilometer = 0.6214 miles"
End Sub

Private Sub btnYards_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnYards.Click

  ' Display the conversion to yards.
  lblMessage.Text = "1 Kilometer = 1,093.6 yards"
End Sub
```

2.4 The AutoSize, BorderStyle, and TextAlign Properties

CONCEPT: The Label control's AutoSize property allows a label to change size automatically to accommodate the amount of text in its Text property. The BorderStyle property allows you to set a border around a Label control. You previously learned to set the TextAlign property at design time. The TextAlign property can also be set with code.

Using the AutoSize and BorderStyle Properties in a Label Control

The Label control has two additional properties, AutoSize and BorderStyle, which give you greater control over the label's appearance.

The AutoSize Property

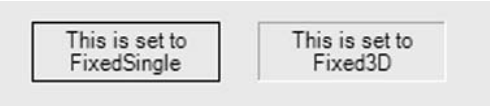
The **AutoSize property** is a Boolean property, which is set to *True* by default. When AutoSize is set to *False*, the bounding box of the Label control remains, at runtime, the size that it was given at design time. If the text copied into the label's Text property is too large to fit in the control's bounding box, the text is only partially displayed. When a label's AutoSize property is set to *True*, however, the label's bounding box will automatically resize to accommodate the text in the label's Text property.

The BorderStyle Property

The Label control's BorderStyle property may have one of three values: *None*, *FixedSingle*, and *Fixed3D*. The property is set to *None* by default, which means the label will have no border. If BorderStyle is set to *FixedSingle*, the label will be outlined with a border that is a single pixel wide. If BorderStyle is set to *Fixed3D*, the label will have a recessed 3D appearance. Figure 2-56 shows an example of two Label controls: one with BorderStyle set to *FixedSingle* and the other with BorderStyle set to *Fixed3D*.

Quite often you will want to display output, such as the results of a calculation, in a Label control with a border. You will see many example applications in this book that use this approach.

Figure 2-56 *FixedSingle* and *Fixed3D* BorderStyle examples


This is set to FixedSingle

This is set to Fixed3D

Changing a Label's TextAlign Property with Code

The Label control's TextAlign property establishes the alignment, or justification of the control's displayed text. It can equal one of the following values: *TopLeft*, *TopCenter*, *TopRight*, *MiddleLeft*, *MiddleCenter*, *MiddleRight*, *BottomLeft*, *BottomCenter*, or *BottomRight*. At design time, you establish the TextAlign property's value with the *Properties* window. You can also set the property with code at runtime. You do this by using an assignment statement to store one of the following values in the property:

```
ContentAlignment.TopLeft
ContentAlignment.TopCenter
ContentAlignment.TopRight
ContentAlignment.MiddleLeft
ContentAlignment.MiddleCenter
ContentAlignment.MiddleRight
ContentAlignment.BottomLeft
ContentAlignment.BottomCenter
ContentAlignment.BottomRight
```

For example, assume an application uses a Label control named `lblReportTitle`. The following statement aligns the control's text with the middle and center of the control's bounding box.

```
lblReportTitle.TextAlign = ContentAlignment.MiddleCenter
```

TIP: When you write an assignment statement that stores a value in the TextAlign property, an auto list box appears showing all the valid values.

Checkpoint

2.27 Suppose an application has a Label control named `lblTemperature`. Write a code statement that causes the label's Text property to display the message *48 degrees*.

2.28 What results when a Label control's AutoSize property is set to *False*? When it is set to *True*?

2.29 What are the possible values for the Label control's BorderStyle property, and what result does each value produce?

2.30 Suppose an application has a Label control named `lblName`. Write the programming statements described by the following:

- A statement that aligns the label's text in the top right.
- A statement that aligns the label's text in the bottom left.
- A statement that aligns the label's text in the top center.

2.5 Clickable Images

CONCEPT: Controls other than buttons have `Click` event procedures. In this section, you learn to create PictureBox controls that respond to mouse clicks.

In this chapter, you learned that buttons have `Click` event procedures. A `Click` event procedure is executed when the user clicks the button. Other controls, such as PictureBoxes and labels, may also have `Click` event procedures. In Tutorial 2-16 you write `Click` event procedures for a group of PictureBox controls.

Tutorial 2-16:
Writing `Click` event procedures for PictureBox controls

Step 1: Open the *Flags* project from the student sample programs folder named *Chap2\Flags*.

Step 2: Open the *Design* window and look at *Form1*. Figure 2-57 shows the form and the names of the controls on the form.

Step 3: Select the `lblMessage` control and set its TextAlign property to *MiddleCenter*, and its Font property to *Microsoft sans serif, bold, 10 points*.

Step 4: Change the form's Text property to **`Flags`**.

Step 5: `PictureBox1` shows the flag of the United States. Rename this control **`picUSA`**.

Step 6: `PictureBox2` shows the flag of Canada. Rename this control **`picCanada`**.

Step 7: `PictureBox3` shows the flag of the United Kingdom. Rename this control **`picUK`**.

Step 8: `PictureBox4` shows the flag of Australia. Rename this control **`picAustralia`**.

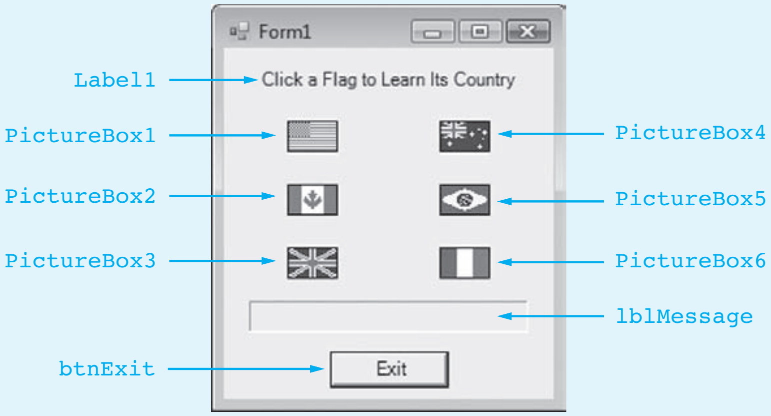
Step 9: `PictureBox5` shows the flag of Brazil. Rename this control **`picBrazil`**.

Step 10: `PictureBox6` shows the flag of Italy. Rename this control **`picItaly`**.

Step 11: Double-click the `picUSA` control. The *Code* window appears with a code template for the `picUSA_Click` procedure. Write the following code shown here in color, which copies the string `"United States of America"` into the `lblMessage` Text property.

```
Private Sub picUSA_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles picUSA.Click

  ' Display the country name
  lblMessage.Text = "United States of America"
End Sub
```

Figure 2-57 *Form1* of the *Flags* project

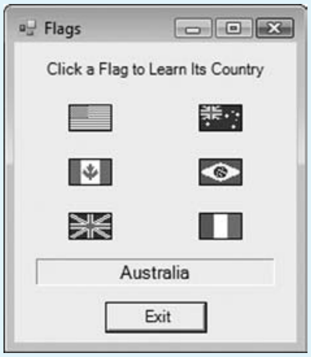


Step 12: Repeat the process outlined in Step 11 for each of the other PictureBox controls. The `Click` procedure for each PictureBox control should copy the name of its flag's country into the `lblMessage.Text` property.

Step 13: Save the application.

Step 14: Run the application. When you click any of the flags, you should see the name of the flag's country appear in the `lblMessage` Label control. For example, when you click the flag of Australia, the form should appear similar to the one shown in Figure 2-58.

Figure 2-58 *Flags* application identifying flag of Australia



Step 15: Exit the application.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.
Copyright © 2010 by Pearson Education, Inc.

2.6 Using Visual Studio Help

Microsoft Document Explorer

When you click the Visual Studio *Help* menu, a list of selections shown in Figure 2-59 appears. The Help system is also called the Microsoft Document Explorer. Table 2-7 contains a brief summary of each menu selection in the upper section of the *Help* menu. The contents of the entire Help system are dynamic because it downloads new information from Microsoft over the Internet.

> **TIP:** When it comes to displaying Help information, Visual Basic Express is very different from Visual Studio. To avoid confusion between the two products, the following discussion will focus only on Visual Studio. If you are using Visual Basic Express, note that many help functions discussed here are directly accessible from the *Help* menu. Also, when you need help on a control or keyword in Visual Basic Express, you can press the F1 function key to view relevant help information.

Figure 2-59 Upper section of the Visual Studio *Help* menu

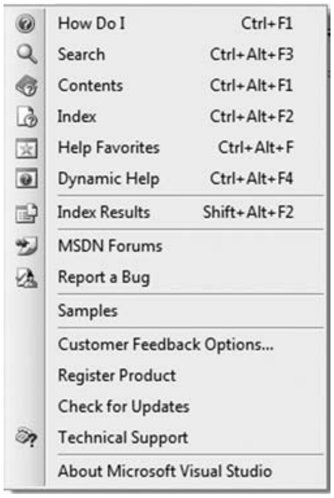


Table 2-7 Microsoft Document Explorer options

How Do I	Contains a list of task-based topics related to Visual Basic programming and application development. The list is divided into categories, each of which is a hyperlink that takes you to a help page with lots of specific tasks.
Search	Lets you search the full text of help topics for words or phrases.
Contents	Displays a top-level table of contents for the help library.
Index	Lets you search for topics using predefined keywords. As you type a keyword, Document Explorer automatically moves to the right position in the list of topics.
Help Favorites	When you view individual help topics, you can bookmark them and add them to your list of favorite help topics. Then when you click *Help Favorites*, you can see your list of bookmarks.
Dynamic Help	Provides links to information found in local help, based on the task you are trying to accomplish in Visual Studio. Not available in Visual Basic Express.

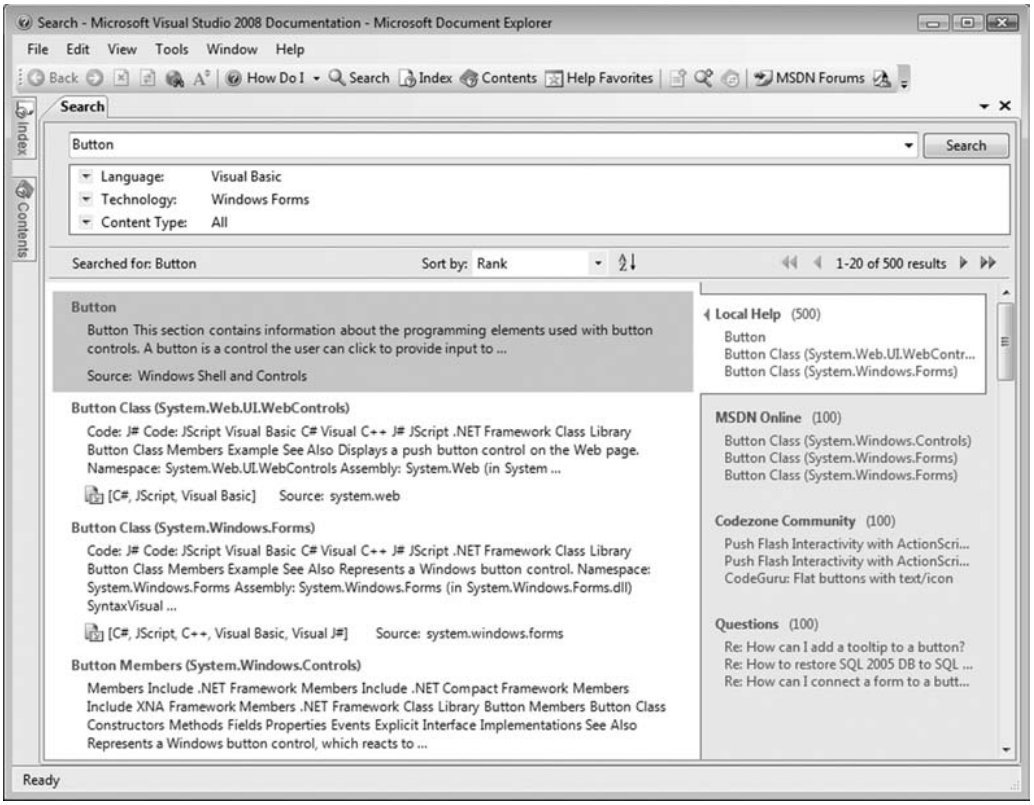
The *How Do I* selection contains a long list of topics that changes often as new entries are added by Microsoft. You might think of these as short tutorials on Windows and Web development.

A help *Search* lets you search for words and phrases within the complete text of Help topics. You can use three types of filters to narrow or widen the search:

- *Language*—programming languages such as Visual Basic, C#, or C++
- *Technology*—select from Windows Forms, Web forms, and other types of applications
- *Content Type*—narrow the search to include only specific types of information

A help *Search* example is shown in Figure 2-60. We have completed a search for help relating to the *Button* topic. The following filters are set: Language = *Visual Basic*, Technology = *Windows Forms*, and Content Type = *All*. In the *Content Type* category, you click on individual check boxes to select different types of content to be searched. The search results are displayed in the lower part of the window, containing both text and hyperlinks. If you click on any of the headings or links, you are taken to help pages that display complete information.

Figure 2-60 Using the *Search* window in Visual Studio Help



Button

Language: Visual Basic
Technology: Windows Forms
Content Type: All

Searched for: Button   Sort by: Rank   1-20 of 500 results

Button
Button This section contains information about the programming elements used with button controls. A button is a control the user can click to provide input to ...
Source: Windows Shell and Controls

Button Class (System.Web.UI.WebControls)
Code: J# Code: JScript Visual Basic C# Visual C++ J# JScript .NET Framework Class Library Button Class Members Example See Also Displays a push button control on the Web page. Namespace: System.Web.UI.WebControls Assembly: System.Web (in System ...
[C#, JScript, Visual Basic]   Source: system.web

Button Class (System.Windows.Forms)
Code: J# Code: JScript Visual Basic C# Visual C++ J# JScript .NET Framework Class Library Button Class Members Example See Also Represents a Windows button control. Namespace: System.Windows.Forms Assembly: System.Windows.Forms (in System.Windows.Forms.dll) SyntaxVisual ...
[C#, JScript, C++, Visual Basic, Visual J#]   Source: system.windows.forms

Button Members (System.Windows.Controls)
Members Include .NET Framework Members Include .NET Compact Framework Members Include XNA Framework Members .NET Framework Class Library Button Members Button Class Constructors Methods Fields Properties Events Explicit Interface Implementations See Also Represents a Windows button control, which reacts to ...

Local Help (500)
Button
Button Class (System.Web.UI.WebContr...
Button Class (System.Windows.Forms)

MSDN Online (100)
Button Class (System.Windows.Controls)
Button Class (System.Windows.Forms)
Button Class (System.Windows.Forms)

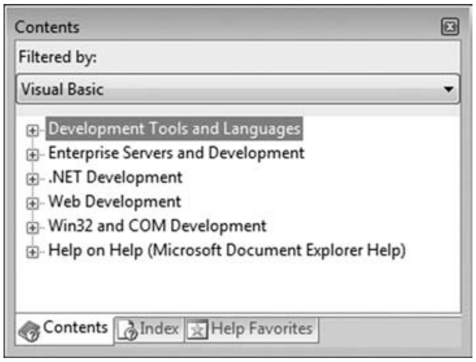
Codezone Community (100)
Push Flash Interactivity with ActionScri...
Push Flash Interactivity with ActionScri...
CodeGuru: Flat buttons with text/icon

Questions (100)
Re: How can I add a tooltip to a button?
Re: How to restore SQL 2005 DB to SQL ...
Re: How can I connect a form to a butt...

A help *Contents* example for Visual Studio is shown in Figure 2-61. The first topic, *Development Tools and Languages*, includes reference information about Visual Basic. You can expand any high-level topic by clicking the + to the left of the topic name. (This list does not contain the same items in Visual Basic Express.)

An *Index* lookup is shown in Figure 2-62, where we have typed the *Button* keyword. The list of topics shows all index entries using that keyword. When you click on any of these entries, the text window in the right-hand pane shows detailed information about the selected topic.

Figure 2-61 Help *Contents*



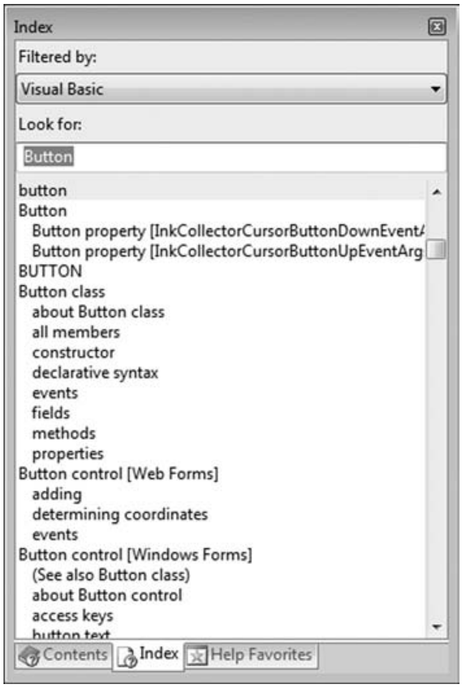
Contents

Filtered by:

Visual Basic

- Development Tools and Languages
- Enterprise Servers and Development
- .NET Development
- Web Development
- Win32 and COM Development
- Help on Help (Microsoft Document Explorer Help)

Figure 2-62 Searching the *Index* for the *Button* keyword



Index

Filtered by:

Visual Basic

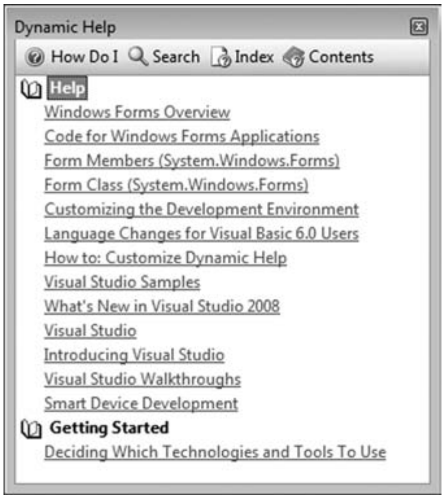
Look for:

Button

```
button
Button
  Button property [InkCollectorCursorButtonDownEventA
  Button property [InkCollectorCursorButtonUpEventArg
BUTTON
Button class
  about Button class
  all members
  constructor
  declarative syntax
  events
  fields
  methods
  properties
Button control [Web Forms]
  adding
  determining coordinates
  events
Button control [Windows Forms]
  (See also Button class)
  about Button control
  access keys
  button text
```

Dynamic Help adapts its list of topics to the task you are trying to accomplish in Visual Studio. Figure 2-63 shows the list of help topics after the user selected a form in *Design* view before activating *Dynamic Help*. The first two topics in the displayed list relate to forms. The last two topics, *Visual Studio*, and *Smart Device Development*, appear in every *Dynamic Help* window, regardless of the current task.

Figure 2-63 *Dynamic Help* activated after the user selected a form in Design view



Dynamic Help

Help

- Windows Forms Overview
- Code for Windows Forms Applications
- Form Members (System.Windows.Forms)
- Form Class (System.Windows.Forms)
- Customizing the Development Environment
- Language Changes for Visual Basic 6.0 Users
- How to: Customize Dynamic Help
- Visual Studio Samples
- What's New in Visual Studio 2008
- Visual Studio
- Introducing Visual Studio
- Visual Studio Walkthroughs
- Smart Device Development

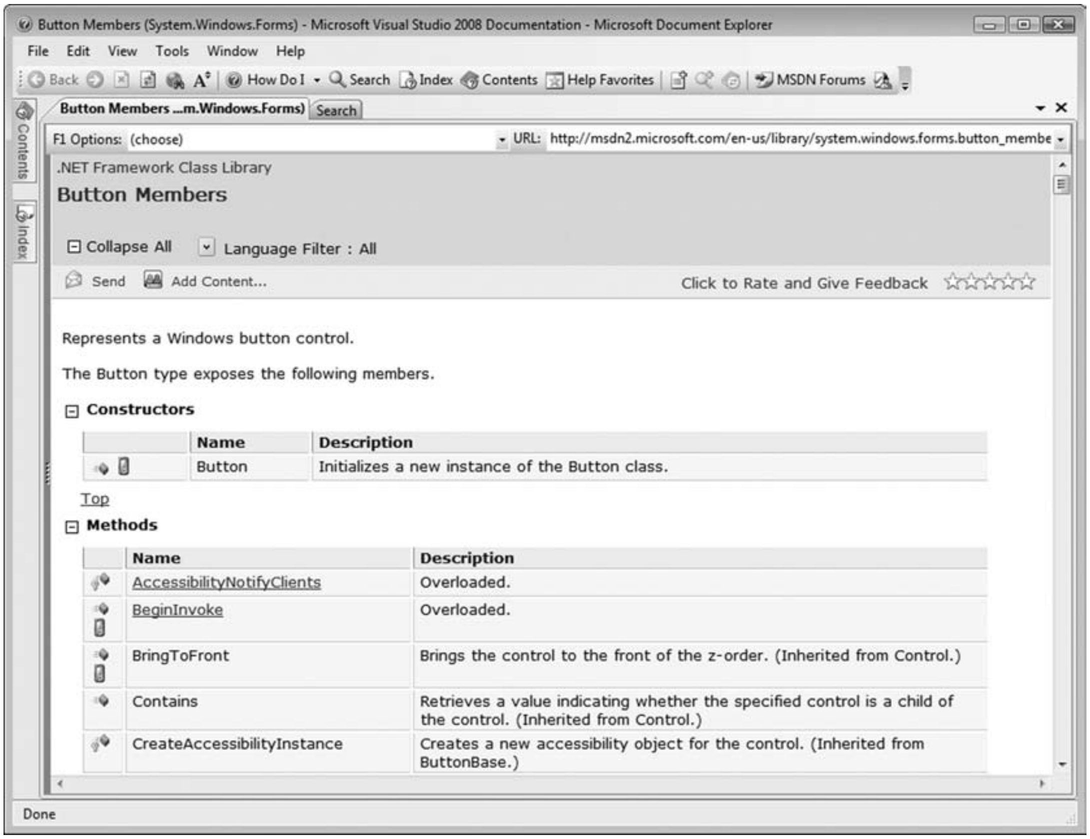
Getting Started

- Deciding Which Technologies and Tools To Use

Context-Sensitive Help (F1 key)

Finally, an easy way to display help on a single topic is to press the F1 key. This is called **context-sensitive help**. In Figure 2-65, the user selected a Button control on a form and then pressed F1. The *Help* window explains all about the Button control. If more than one help topic is found, you can select from a drop-down list labeled *F1 Options* in the upper left corner of the *Help* window's text area. Look for it in Figure 2-64.

Figure 2-64 Displaying a single topic after pressing the F1 key


Button Members (System.Windows.Forms) - Microsoft Visual Studio 2008 Documentation - Microsoft Document Explorer
File Edit View Tools Window Help
Back How Do I Search Index Contents Help Favorites MSDN Forums
Contents
Index
Button Members ...m.Windows.Forms) Search
F1 Options: (choose)
URL: http://msdn2.microsoft.com/en-us/library/system.windows.forms.button_membe
Collapse All
Language Filter : All
Send
Add Content...
Click to Rate and Give Feedback
Top
Done

.NET Framework Class Library

Button Members

Represents a Windows button control.

The Button type exposes the following members.

Constructors

	Name	Description
	Button	Initializes a new instance of the Button class.

Methods

	Name	Description
	AccessibilityNotifyClients	Overloaded.
	BeginInvoke	Overloaded.
	BringToFront	Brings the control to the front of the z-order. (Inherited from Control.)
	Contains	Retrieves a value indicating whether the specified control is a child of the control. (Inherited from Control.)
	CreateAccessibilityInstance	Creates a new accessibility object for the control. (Inherited from ButtonBase.)

In Tutorial 2-17, you will use *Dynamic Help* in Visual Studio.

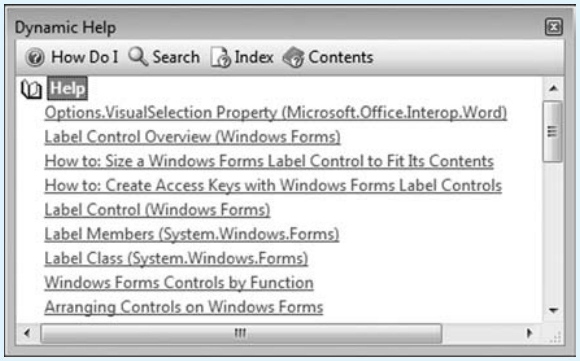
Tutorial 2-17:
Using *Dynamic Help* in Visual Studio

Step 1: Start Visual Studio and load the *Directions* project. Make sure the *Design* window is open.

Step 2: Select *Dynamic Help* from the *Help* menu. The *Dynamic Help* window should open.

Step 3: In the *Design* window, select one of the Label controls on the form. Notice that the content of the *Dynamic Help* window changes. It should look similar to Figure 2-65, showing topics relating to Label controls. *(The exact list of topics may be different in your window because Microsoft updates its Help Web site regularly.)*

Figure 2-65 *Dynamic Help* activated after the user selects a Label control

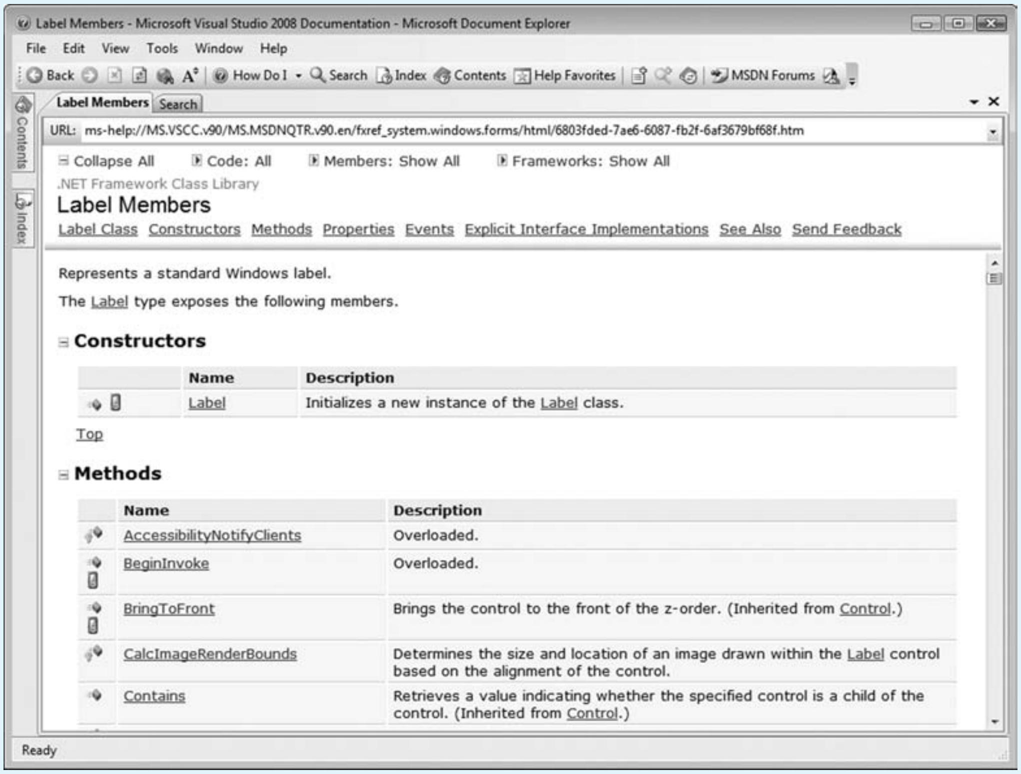


Help

- Options.VisualSelection Property (Microsoft.Office.Interop.Word)
- Label Control Overview (Windows Forms)
- How to: Size a Windows Forms Label Control to Fit Its Contents
- How to: Create Access Keys with Windows Forms Label Controls
- Label Control (Windows Forms)
- Label Members (System.Windows.Forms)
- Label Class (System.Windows.Forms)
- Windows Forms Controls by Function
- Arranging Controls on Windows Forms

Step 4: Click the *Label Members* help topic, which should cause the *Help* window shown in Figure 2-66 to display. This window provides help on all the properties and methods of the Label control.

Figure 2-66 Displaying the *Label Members* Help topic



URL: ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/6803fded-7ae6-6087-fb2f-6af3679bf68f.htm

.NET Framework Class Library

Label Members

Label Class Constructors Methods Properties Events Explicit Interface Implementations See Also Send Feedback

Represents a standard Windows label.

The Label type exposes the following members.

Constructors

	Name	Description
	Label	Initializes a new instance of the Label class.

Top

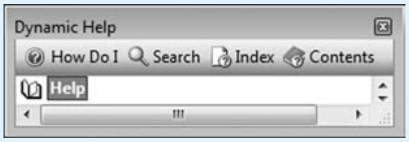
Methods

	Name	Description
	AccessibilityNotifyClients	Overloaded.
	BeginInvoke	Overloaded.
	BringToFront	Brings the control to the front of the z-order. (Inherited from Control.)
	CalcImageRenderBounds	Determines the size and location of an image drawn within the Label control based on the alignment of the control.
	Contains	Retrieves a value indicating whether the specified control is a child of the control. (Inherited from Control.)

Step 5: Click the *Close* button (☒) in the upper right corner of the *Label Members* Help window.

Step 6: Look at the *Dynamic Help* window again. Figure 2-67 shows the locations of the *Contents*, *Index*, and *Search* buttons. When any of these buttons are clicked, another Help window with additional functionality appears in the same location as the *Solution Explorer*. The ***Contents* button** displays a table of contents in which related topics are organized into groups. The ***Index* button** displays a searchable alphabetized index of all the help topics. The ***Search* button** allows you to search for help using keywords.

Figure 2-67 *Dynamic Help* window buttons

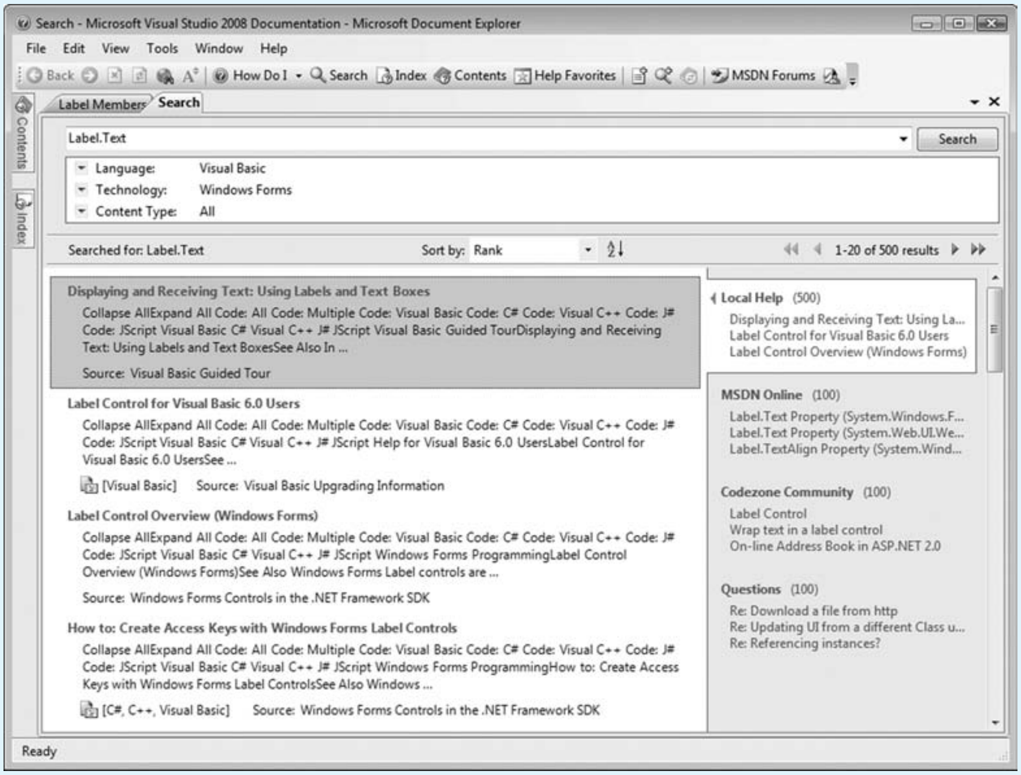


Step 7: In the *Dynamic Help* window's toolbar, click the *Search* button. The *Search* window should appear. There are so many topics in the MSDN help system that you need to narrow down your search using the following criteria: Language, Technology, and Content Type.

- In the Language list, select only *Visual Basic*.
- In the Technology list, select only *Windows Forms*.
- In the Content Type list, select only *Documentation & Articles*.

Enter *Label.Text* into the input text line and click the *Search* button. Figure 2-68 shows a sample of the type of help information you should see. *(The exact list of topics may be different in your window because Microsoft updates its Help Web site regularly.)*

Figure 2-68 Searching for the Label.Text property



Label.Text

Language: Visual Basic
Technology: Windows Forms
Content Type: All

Searched for: Label.Text   Sort by: Rank   1-20 of 500 results

Displaying and Receiving Text: Using Labels and Text Boxes
Collapse AllExpand All Code: All Code: Multiple Code: Visual Basic Code: C# Code: Visual C++ Code: J# Code: JScript Visual Basic C# Visual C++ J# JScript Visual Basic Guided TourDisplaying and Receiving Text: Using Labels and Text BoxesSee Also In ...
Source: Visual Basic Guided Tour

Label Control for Visual Basic 6.0 Users
Collapse AllExpand All Code: All Code: Multiple Code: Visual Basic Code: C# Code: Visual C++ Code: J# Code: JScript Visual Basic C# Visual C++ J# JScript Help for Visual Basic 6.0 UsersLabel Control for Visual Basic 6.0 UsersSee ...
[Visual Basic]   Source: Visual Basic Upgrading Information

Label Control Overview (Windows Forms)
Collapse AllExpand All Code: All Code: Multiple Code: Visual Basic Code: C# Code: Visual C++ Code: J# Code: JScript Visual Basic C# Visual C++ J# JScript Windows Forms ProgrammingLabel Control Overview (Windows Forms)See Also Windows Forms Label controls are ...
Source: Windows Forms Controls in the .NET Framework SDK

How to: Create Access Keys with Windows Forms Label Controls
Collapse AllExpand All Code: All Code: Multiple Code: Visual Basic Code: C# Code: Visual C++ Code: J# Code: JScript Visual Basic C# Visual C++ J# JScript Windows Forms ProgrammingHow to: Create Access Keys with Windows Forms Label ControlsSee Also Windows ...
[C#, C++, Visual Basic]   Source: Windows Forms Controls in the .NET Framework SDK

Local Help (500)
Displaying and Receiving Text: Using La...
Label Control for Visual Basic 6.0 Users
Label Control Overview (Windows Forms)

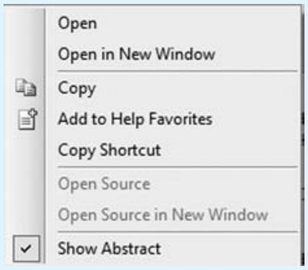
MSDN Online (100)
Label.Text Property (System.Windows.F...
Label.Text Property (System.Web.UI.We...
Label.TextAlign Property (System.Wind...

Codezone Community (100)
Label Control
Wrap text in a label control
On-line Address Book in ASP.NET 2.0

Questions (100)
Re: Download a file from http
Re: Updating UI from a different Class u...
Re: Referencing instances?

Step 8: You may find it helpful to keep a list of favorite help topics you can find quickly in the future. For example, right-click the help topic you selected in Step 7. The context menu, as shown in Figure 2-69 lets you add this topic to your list of Help Favorites. Add several help topics to your Help Favorites. When you are done, close the *Search* window.

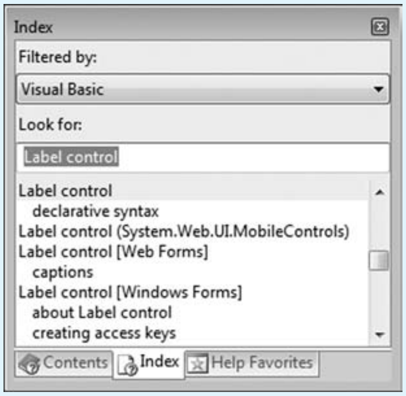
Figure 2-69 Adding a help topic to your Help Favorites



Step 9: In the *Dynamic Help* window, click the *Index* button. The Index lookup panel appears in the left side of the *Search* window. Set the *Filtered by* selection to *Visual Basic*.

Step 10: Type *Label control* into the box near the top labeled as *Look for:* (see Figure 2-70). Notice how Label control appears in different contexts. When you select the entry relating to Windows Forms, the help topic's text should appear.

Figure 2-70 Using the *Index* window



Index

Filtered by:

Visual Basic

Look for:

Label control

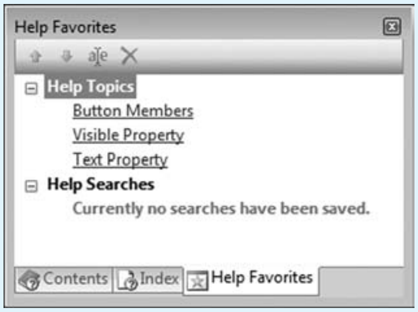
Label control
  declarative syntax
Label control (System.Web.UI.MobileControls)
Label control [Web Forms]
  captions
Label control [Windows Forms]
  about Label control
  creating access keys

Step 11: Click the *Help Favorites* tab at the bottom of the *Search* window, as shown in Figure 2-71. You will see a list of help topics you previously added to your Help Favorites. This list is saved when you exit Visual Studio.

Step 12: Close the project.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Figure 2-71 *Help Favorites* window



2.7 Debugging Your Application

CONCEPT: At some point, most applications contain bugs (errors) that prevent the application from operating properly. In this section, you learn fundamental debugging techniques.

Visual Basic reports errors in your project as soon as it finds them. In general, there are two types of errors: compile errors and runtime errors.

Compile errors are syntax errors, such as misspelled keywords or the incorrect use of operators or punctuation. Visual Basic checks each line of code for compile errors as soon as you enter it. When a compile error is found, it is underlined with a jagged blue line. A description of the error is also displayed in the *Error List* window. You display it by clicking *View* on the menu bar, and then selecting *Error List*.

Runtime errors are errors found while an application is running. They are not syntax errors but result if there is an attempt to perform an operation that Visual Basic cannot execute.

Tutorial 2-18 demonstrates how Visual Basic reports compile errors.

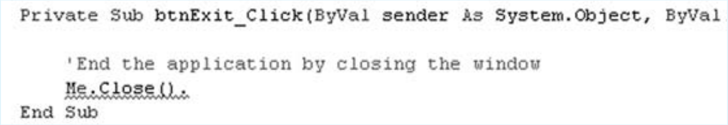
Tutorial 2-18:
Locating a compile error in Design mode

Step 1: Open the *Directions* project and open the *Code* window.

Step 2: You will modify the `btnExit_Click` procedure so it contains a syntax error. Position the text editing cursor at the end of the line that reads `Me.Close()`.

Step 3: Type a period at the end of the line and press Enter. Notice that the statement is now underlined with a jagged blue line, as shown in Figure 2-72.

Figure 2-72 Error underlined

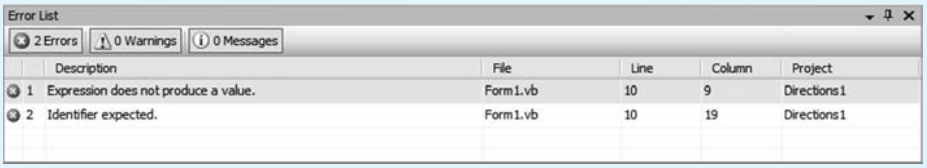


```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal

    'End the application by closing the window
    Me.Close().
End Sub
```

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Step 4: Look at the *Error List* window, which should appear similar to Figure 2-73. It shows an error message indicating that the statement is not valid.

Figure 2-73 *Error List* window with error message



	Description	File	Line	Column	Project
1	Expression does not produce a value.	Form1.vb	10	9	Directions1
2	Identifier expected.	Form1.vb	10	19	Directions1

Step 5: The *Error List* window also helps you find the statement that contains the error. If you do not see it, select *Error List* from the *View* menu. Double-click the error message in the *Error List* window. Notice that the erroneous statement is highlighted in the *Code* window.

TIP: Sometimes you will see an error message you do not fully understand. When this happens, look at the highlighted area of the line containing the error and try to determine the problem. When you figure out what caused the error, consider how it relates to the error message.

Step 6: Erase the period to correct the error.

Checkpoint

2.31 In Visual Studio, if the *Dynamic Help* window is not visible, how do you display it?

2.32 Describe two ways to display a searchable alphabetized index of help topics.

2.33 How do you cause the *Contents*, *Index*, and *Search* windows to display only help topics related to Visual Basic?

2.34 What is context-sensitive help?

2.35 What are the two general types of errors in a Visual Basic project?

Summary

2.1 Focus on Problem Solving: Building the *Directions* Application

- The *Properties* window is used at design time to set control property values. The PictureBox control's SizeMode property determines how the control will place and size its image.
- Small squares (sizing handles) are used to enlarge or shrink the control. Once you place a control on a form, you can move it by clicking and dragging. To delete a control, select it and then press the Delete key.
- To run an application click the *Start Debugging* button on the toolbar, click the *Start Debugging* command on the *Debug* menu, or press the F5 key. To end a running application, click the *Stop Debugging* command on the *Debug* menu or click the *Close* button on the application window.
- When you create a new project, Visual Studio automatically creates a solution with the same name as the project, and adds the project to the solution. The solution and the project are stored on the disk in a folder with the same name as the solution. The solution file ends with the *.sln* extension and the project file ends with the *.vbproj* extension.
- To open an existing project, click its name on the *Start Page*, click the *Open Project* button on the toolbar, or click *File* on the menu bar, then click *Open Project*.
- The *Properties* window's object box provides a drop-down list of the objects in the project. The *Alphabetical* button causes the properties to be listed alphabetically. The *Categorized* button causes related properties to be displayed in groups.

2.2 Focus on Problem Solving: Responding to Events

- The apostrophe (`'`) marks the beginning of a comment in code. A comment is a note of explanation that is ignored by Visual Basic.
- The `Me.Close()` statement causes the current form to close. If the current form is the application's startup form, the application ends.
- To display the *Code* window, you click the *View Code* button on the *Solution Explorer* window, click *View* on the menu bar, then *Code*, or press the F7 key on the keyboard.
- The color of a label's text is changed with the BackColor and ForeColor properties.
- Once you have placed all the controls in their proper positions on a form, it is a good idea to lock them.

2.3 Modifying the Text Property with Code

- You may change a control's Text property with code. An assignment statement copies a value into the property at runtime.

2.4 The AutoSize, BorderStyle, and TextAlign Properties

- You can set the TextAlign property with code at runtime by using an assignment statement to store a valid value in the property.

2.5 Clickable Images

- Buttons are not the only controls with `Click` event procedures. Other controls, such as PictureBoxes and Labels, also have `Click` event procedures.

2.6 Using Visual Studio Help

- In Visual Studio (but not Visual Basic Express), the *Dynamic Help* window displays a list of help topics that changes as you perform operations. The window also has buttons to view the help contents, view the index, and search for a help topic. The same commands are available in the *Help* menu of Visual Basic Express.
- In Visual Basic Express, you can select a control or keyword and press the F1 key to view immediate help.

2.7 Debugging Your Application

- Visual Basic checks each line of code for syntax errors as soon as you enter it. When a compile error is found, it is underlined with a jagged blue line.
- Runtime errors occur while an application is running. They are not syntax errors, but result if there is an attempt to perform an operation that Visual Basic cannot execute.

Key Terms

Alphabetical button
assignment operator
assignment statement
auto list box
AutoSize property
BackColor property
Boolean property
BorderStyle property
bounding box
Categorized button
code template
Code window
compile errors
Contents button
context-sensitive help
Font property
ForeColor property
FormBorderStyle property
Index button
IntelliSense
object box
PictureBox control
project file
runtime errors
Search button
SizeMode property
sizing handles
solution (container)
solution file
string literal
TextAlign property
Visible property

Review Questions and Exercises

Fill-in-the-Blank

1. The ______________________ property determines how a Label control's text is aligned.

2. A PictureBox control's ______________________ property lists the name of the file containing the graphic image.

3. A PictureBox control's ______________________ property determines how the graphic image will be positioned and scaled to fit the control's bounding box.

4. When set to ______________________, the TextAlign property causes text to appear in the bottom right area of a Label control.

5. The contents of a form's Text property is displayed on the form's ______________________.

6. Anytime you select an existing control, ______________________ appear, which you use to resize the control.

7. A control's ________________________ is a transparent rectangular area that defines the control's size.

8. A Label control's ________________________ property establishes the font, style, and size of the label's displayed text.

9. To delete a control during design time, select it and press the ___________________ key.

10. The ________________________ control is used to display graphic images.

11. The SizeMode property is set to ________________________ by default.

12. When the ________________________ button is selected on the *Solution Explorer* window, it opens the *Code* window.

13. Clicking the ________________________ button in the *Properties* window causes related properties to be listed in groups.

14. Visible is a ________________________ property, which means it can only hold one of two values: *True* or *False*.

15. An apostrophe (`'`) in code marks the beginning of a ________________________.

16. The equal sign (=) is known as the ________________________ operator. It copies the value on its right into the item on its left.

17. In an assignment statement, the name of the item receiving the value must be on the ________________________ side of the = operator.

18. Visual Basic automatically provides a code ________________________ , which is the first and last lines of an event procedure.

19. The ________________________ statement causes the application to end.

20. The ________________________ property establishes the background color for a Label control's text.

21. The ________________________ property establishes the color of the type for a Label control's text.

22. The ________________________ property allows you to prevent the user from resizing, minimizing, or maximizing a form, or closing a form using its *Close* button.

23. When you ________________________ the controls on a form, they cannot be accidentally moved at design time.

24. You display text in a form's title bar by setting the value of the form's ________________________ property.

25. You commonly display messages on a form by setting the value of a Label control's ________________________ property.

26. The ________________________ property causes the Label control to resize automatically to accommodate the amount of text in the Text property.

27. ________________________ is a help screen displayed for the currently selected item.

28. ________________________ errors are errors found while an application is running.

True or False

Indicate whether the following statements are true or false.

1. T F: Sizing handles appear around the control that is currently selected.

2. T F: The PictureBox control has an ImageStretch property.

3. T F: The Visible property is Boolean.

4. T F: A control is hidden at design time if its Visible property is set to *False*.

5. T F: You can delete a locked control.

6. T F: A control is accessed in code by its Text property.

7. T F: The TextAlign property causes a control to be aligned with other controls on the same form.

8. T F: Text is frequently the first property that the programmer changes, so it is listed in parentheses in the *Properties* window. This causes it to be displayed at the top of the alphabetized list of properties.

9. T F: Resizing handles are positioned along the edges of a control's bounding box.

10. T F: A label's text is *MiddleCenter* aligned by default.

11. T F: You can run an application in the Visual Studio environment by pressing the F5 key.

12. T F: The first line of an event procedure identifies its name and event it handles.

13. T F: You should be very cautious about, and even avoid, placing comments in your code.

14. T F: In an assignment statement, the name of the item receiving the value must be on the right side of the = operator.

15. T F: You can bring up the *Code* window by pressing the F7 key when the *Design* window is visible.

16. T F: The BackColor property establishes the color of a Label control's text.

17. T F: A form's BorderStyle property can be used to prevent the user from resizing, minimizing, or maximizing a window, or closing the window using its *Close* button.

18. T F: When you lock the controls on a form, the user must enter a password before the application will run.

19. T F: You cannot modify a control's Text property with code.

20. T F: The *Properties* window only shows a control's properties that may be changed at design time.

21. T F: The AutoSize property is set to *True* by default.

22. T F: PictureBox controls have a `Click` event procedure.

23. T F: Context-sensitive help is a help screen displayed for the currently selected item.

24. T F: You access context-sensitive help by pressing the F1 key.

Short Answer

1. Explain the difference between an object's Text and its Name.
2. List three ways to run an application within the Visual Studio environment.
3. List three ways to display the *Code* window.
4. Why is the code between the first and last lines of an event procedure usually indented?
5. How do you make a PictureBox control respond to mouse clicks?

What Do You Think?

1. Why, in the *Properties* window, do you change some properties with a drop-down list or a dialog box, while you change others by typing a value?
2. Why is it a good idea to equip a form with a button that terminates the application, if the form already has a standard Windows *Close* button in the upper right corner?
3. What is the benefit of creating PictureBox controls that respond to mouse clicks?

Find the Error

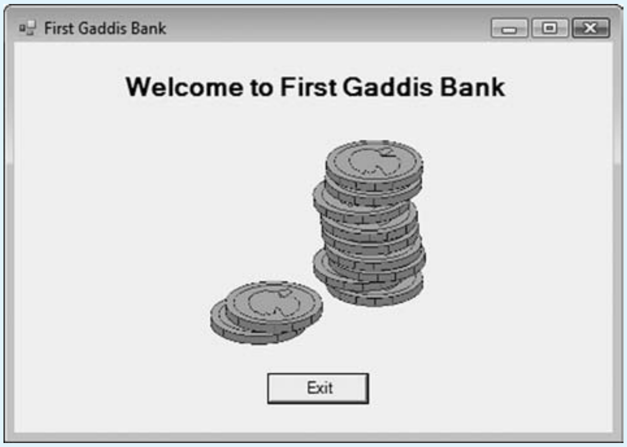
1. Open the *Error1* project from the student sample programs folder named *Chap2**Error1*. Run the application. When Visual Basic reports an error, find and fix the error.
2. Open the *Error2* project from the student sample programs folder named *Chap2**Error2*. Run the application. When Visual Basic reports an error, find and fix the error.

Programming Challenges

1. **Welcome Screen Modification**

   For this exercise, you will modify an application that displays a welcome screen for the First Gaddis Bank. After starting Visual Studio, open the *First Gaddis* project from the student sample programs folder named *Chap2\First Gaddis*. Figure 2-74 shows the application's form.

Figure 2-74 *First Gaddis* welcome screen, *Form1*



Welcome to First Gaddis Bank

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

Make the following modifications to the application:

a. Change the form's title bar text to read *First Gaddis Bank*.
b. Change the label's Font property to *Microsoft sans serif, bold, 14.25 point*.
c. Change the label's text alignment to middle center.
d. Change the button's name to `btnExit`.
e. Change the button's text to read *Exit*.
f. Change the label's background color to a shade of light blue, or another color of your choice.
g. Change the form's background color to the same color you chose for the label.
h. Write a `Click` event procedure for the button that closes the application window.

VideoNote
The Name and Address Problem

2. **Name and Address**

Create an application that displays your name and address when a button is clicked. The application's form should appear as shown in Figure 2-75 when it first runs. Once the *Show Info* button is clicked, the form should appear similar to the one shown in Figure 2-76.

Figure 2-75 Intitial *Name and Address* form

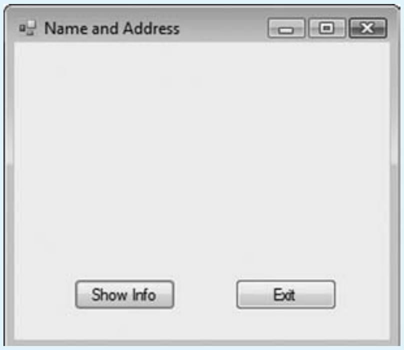
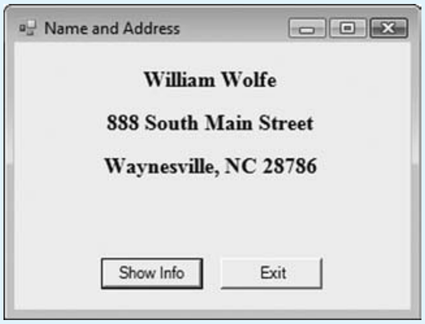


Figure 2-76 *Name and Address* form after *Show Info* button is clicked



William Wolfe

888 South Main Street

Waynesville, NC 28786

Here are the detailed property specifications:

a. The button that displays the name and address should be named `btnShowInfo`. Its text should read *Show Info*.
b. The button that closes the application should be named `btnExit`. Its text should read *Exit*.
c. The form should have three Label controls. The first will hold your name, the second will hold your street address, and the third will hold your city, state, and ZIP code. The labels should be named `lblName`, `lblStreet`, and `lblCityStateZip`, respectively. The labels' Font property should be set to *Times New Roman, bold, 12 point*. The labels' TextAlign property should be set to *MiddleCenter*.
d. The form's title bar should read *Name and Address*.

3. **Math Tutor Application**

You are to create a *Math Tutor* application. The application should display a simple math problem in a Label control. The form should have a button that displays the answer to the math problem in a second label, when clicked. It should also have a button that closes the application. Figure 2-77 shows an example of the application's form before the button is clicked to display the answer. Figure 2-78 shows the

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.
Copyright © 2010 by Pearson Education, Inc.

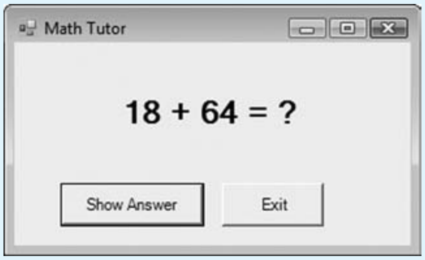


Figure 2-77 Initial *Math Tutor* application

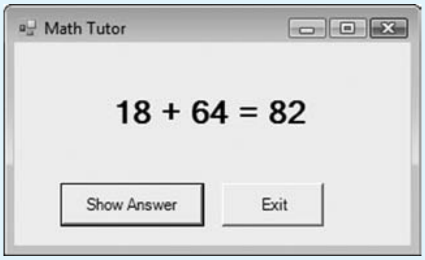


Figure 2-78 *Math Tutor* application after *Show Answer* button is clicked

form after the *Show Answer* button is clicked. Here are the detailed property specifications:

a. The button that displays the answer should be named `btnShowAnswer`. Its Text property should read *Show Answer*.

b. The button that closes the application should be named `btnExit`. Its Text property should read *Exit*.

c. The label that displays the answer to the math problem should be named `lblAnswer`.

d. The form's title bar should read *Math Tutor*.

Design Your Own Forms

4. State Abbreviations

The following table shows lists of six states and their official abbreviations.

State	Abbreviation
Virginia	VA
North Carolina	NC
South Carolina	SC
Georgia	GA
Alabama	AL
Florida	FL

Create an application that allows the user to select a state, and then displays that state's official abbreviation. The form should have six buttons, one for each state. When the user clicks a button, the application displays the state's abbreviation in a Label control.

5. Latin Translator

Look at the following list of Latin words and their meanings.

Latin	English
sinister	left
dexter	right
medium	center

Create an application that translates the Latin words to English. The form should have three buttons, one for each Latin word. When the user clicks a button, the application should display the English translation in a Label control. When the user clicks the *sinister* button, the translation should appear with middle left alignment. When the user clicks the *dexter* button, the translation should appear with middle right alignment. When the user clicks the *medium* button, the translation should appear with middle center alignment.

Starting out with Visual Basic® 2008, Updated Fourth Edition, by Tony Gaddis and Kip Irvine. Published by Addison-Wesley.

6. **Clickable Image**

Create an application with a PictureBox control. For the Image property, use one of Visual Basic's graphics files. If Visual Studio is installed, graphics files may be stored in the following file on your hard drive: *C:\Program Files\Microsoft Visual Studio 9.0\Common7\VS2008ImageLibrary\1033\VS2008ImageLibrary.zip*. In this file, you will find several folders of graphics files. Explore them and select a file you want to use. If you are using Visual Basic Express, you can obtain graphic files from the Internet or from your instructor.

Once you have selected a graphic file, place a Label control on the form. Then, write code in the PictureBox control's `Click` event procedure that displays the name of your school in the Label control.

7. **Joke and Punch line**

A joke typically has two parts: a setup and a punch line. For example, this might be the setup for a joke:

> *How many programmers does it take to change a light bulb?*

And this is the punch line:

> *None. That's a hardware problem.*

Think of your favorite joke and identify its setup and punch line. Then, create an application that has a Label and two buttons on a form. One of the buttons should read "Setup" and the other button should read "Punch line." When the *Setup* button is clicked, display the joke's setup in the Label. When the *Punch line* button is clicked, display the joke's punch line in the Label.