C H A P T E R

2 User Interface Design

at the completion of this chapter, you will be able to . . .

1. Use text boxes, masked text boxes, rich text boxes, group boxes, check boxes, radio buttons, and picture boxes effectively.
2. Set the BorderStyle property to make controls appear flat or three-dimensional.
3. Select multiple controls and move them, align them, and set common properties.
4. Make your projects easy for the user to understand and operate by defining access keys, setting an Accept and a Cancel button, controlling the tab sequence, resetting the focus during program execution, and causing ToolTips to appear.
5. Clear the contents of text boxes and labels.
6. Make a control visible or invisible at run time by setting its Visible property.
7. Disable and enable controls at design time and run time.
8. Change text color during program execution.
9. Code multiple statements for one control using the `With` and `End With` statements.
10. Concatenate (join) strings of text.
11. Download the Line and Shape controls, add them to the toolbox, and use the controls on your forms.

Introducing More Controls

In Chapter 1 you learned to use labels and buttons. In this chapter you will learn to use several more control types: text boxes, group boxes, check boxes, radio buttons, and picture boxes. Figure 2.1 shows the toolbox, with the *Containers* and *Common Controls* tabs open, to show the tools for these new controls. Figure 2.2 shows some of these controls on a form. You will also download, install, and use the Line and Shape controls in the Visual Basic PowerPack 2.0.

Each class of controls has its own set of properties. To see a complete list of the properties for any class of control, you can (1) place a control on a form and examine the properties list in the Properties window or (2) click on a tool or a control and press F1 for context-sensitive Help. Visual Studio will display the Help page for that control, and you can view a list of the properties and an explanation of their use.

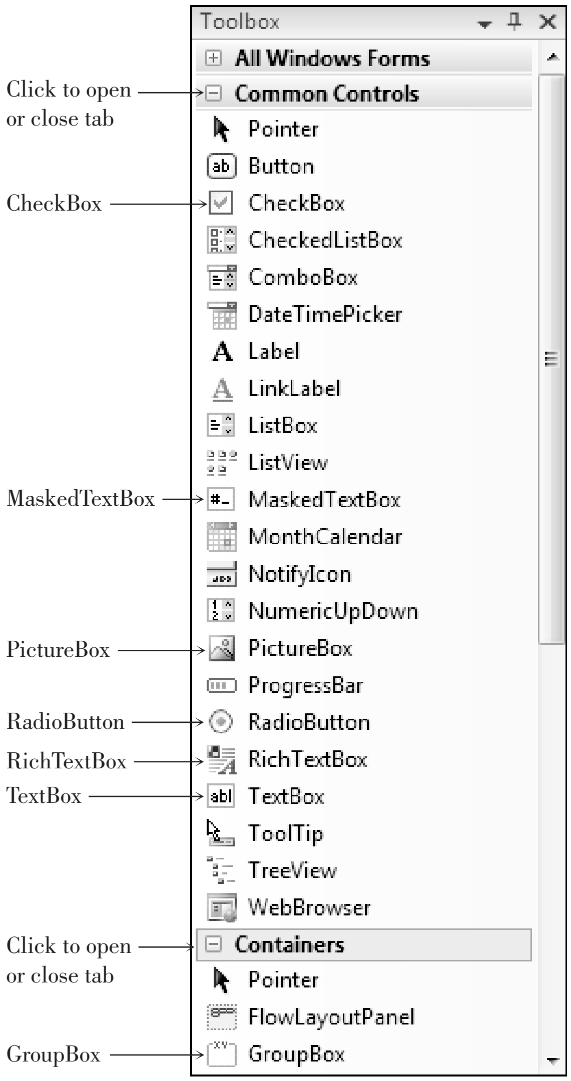
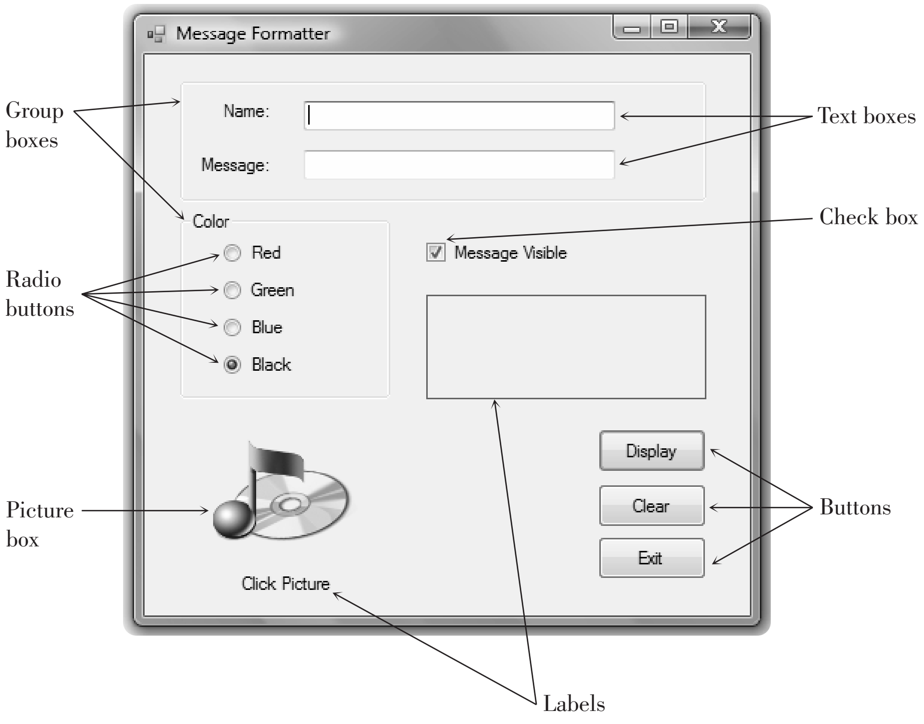


Figure 2.1

The toolbox with two tabs open, showing the new controls that are covered in this chapter. Click the plus and minus signs on the tabs to open and close each section.

Figure 2.2

This form uses labels, text boxes, a check box, radio buttons, group boxes, and a picture box.



Text Boxes

Use a **text box** control when you want the user to type some input. The form in Figure 2.2 has two text boxes. The user can move from one box to the next, make corrections, cut and paste if desired, and click the *Display* button when finished. In your program code you can use the **Text property** of each text box.

Example

```
NameLabel.Text = NameTextBox.Text
```

In this example, whatever the user enters into the text box is assigned to the Text property of NameLabel. If you want to display some text in a text box during program execution, assign a literal to the Text property:

```
MessageTextBox.Text = "Watson, come here."
```

You can set the **TextAlign property** of text boxes to change the alignment of text within the box. In the Properties window, set the property to *Left*, *Right*, or *Center*. In code, you can set the property using these values:

HorizontalAlignment.Left
HorizontalAlignment.Right
HorizontalAlignment.Center

```
MessageTextBox.TextAlign = HorizontalAlignment.Left
```

Example Names for Text Boxes

```
TitleTextBox
CompanyTextBox
```

Masked Text Boxes

A specialized form of the TextBox control is the **MaskedTextBox**. You can specify the format (the Mask property) of the data required of the user. For example, you can select a mask for a ZIP code, a date, a phone number, or a social security number. Figure 2.3 shows the *Input Mask* dialog box where you can select the mask and even try it out. At run time the user cannot enter characters that do not conform to the mask. For example, the phone number and social security number masks do not allow input other than numeric digits.

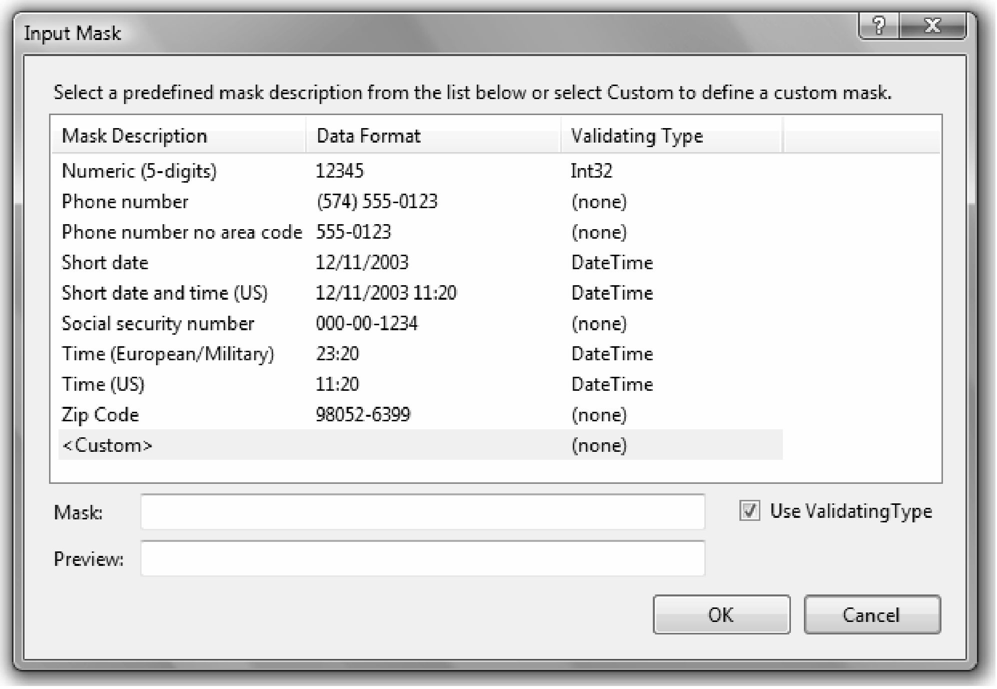
Example Names for Masked Text Boxes

```
DateMaskedTextBox
PhoneMaskedTextBox
```

Note: For a date or time mask, the user can enter only numeric digits but may possibly enter an invalid value; for example, a month or hour greater than 12. The mask will accept any numeric digits, which could possibly cause your program to generate a run-time error. You will learn to check the input values in Chapter 4.

Figure 2.3

Select a format for the input mask in the Input Mask dialog box, which supplies the Mask property of the MaskedTextBox control.



Input Mask

Select a predefined mask description from the list below or select Custom to define a custom mask.

Mask Description	Data Format	Validating Type
Numeric (5-digits)	12345	Int32
Phone number	(574) 555-0123	(none)
Phone number no area code	555-0123	(none)
Short date	12/11/2003	DateTime
Short date and time (US)	12/11/2003 11:20	DateTime
Social security number	000-00-1234	(none)
Time (European/Military)	23:20	DateTime
Time (US)	11:20	DateTime
Zip Code	98052-6399	(none)
<Custom>		(none)

Mask:

Preview:

[✓] Use ValidatingType

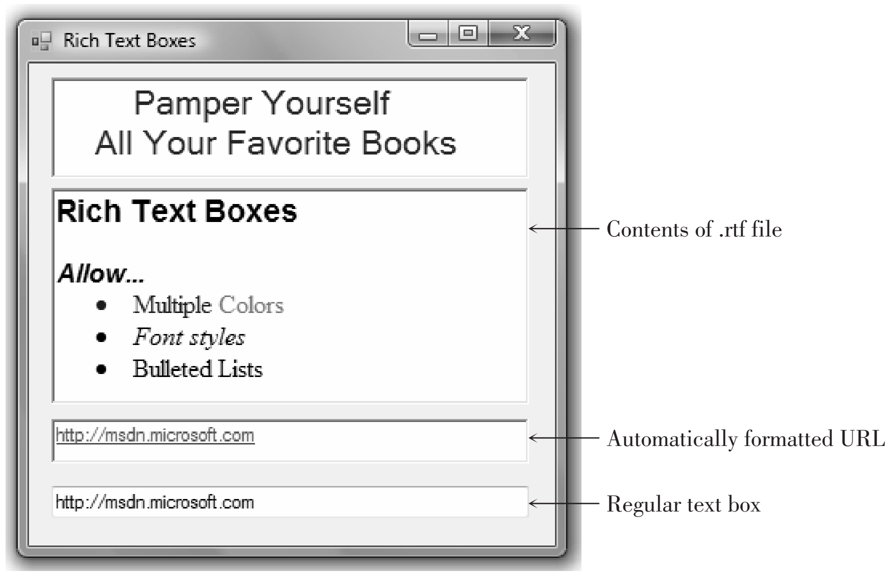
Rich Text Boxes

Another variety of text box is the **RichTextBox** control, which offers a variety of formatting features (Figure 2.4). In a regular text box, all of the text is formatted the same, but in a rich text box, the user can apply character and paragraph formatting to selected text, much like using a word processor.

One common use for a rich text box is for displaying URL addresses. In a regular text box, the address appears in the default font color, but the rich text box displays it as a link when the DetectUrl property is set to True. Note that it is not an active link, but it does have the formatting to show the URL as an address.

You also can load formatted text into a rich text box from a file stored in rich text format (rtf). Use the `LoadFile` method of the rich text box. In this example, the file "Rich Text Boxes.rtf" is stored in the bin/debug folder, but you could include the complete path to load a file from another location.

```
SampleRichTextBox.LoadFile("Rich Text Boxes.rtf")
```



Rich Text Boxes

Pamper Yourself
All Your Favorite Books

Rich Text Boxes

Allow...

- Multiple Colors
- *Font styles*
- Bulleted Lists

http://msdn.microsoft.com

http://msdn.microsoft.com

Figure 2.4

Using the RichTextBox control, you can apply font styles to selected text, show formatted URLs, and display text from a formatted .rtf file

Displaying Text on Multiple Lines

Both the regular text box and the rich text box have properties that allow you to display text on multiple lines. The **WordWrap property** determines whether the contents should wrap to a second line if they do not fit on a single line. The property is set to True by default. Both controls also have a **Multiline property**, which is set to False by default on a text box and True on a rich text box. Both WordWrap and Multiline must be set to True for text to wrap to a second line.

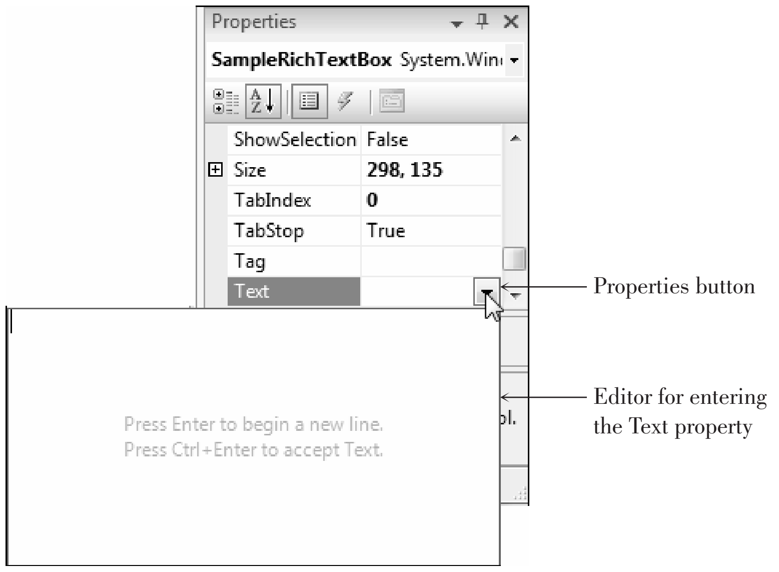
For a regular text box, you must set Multiline to True and then adjust the height to accommodate multiple lines. If Multiline is False (the default), a text box does not have resizing handles for vertical resizing. Be aware that a text box will not automatically resize to display multiple lines even though Multiline is True; you must make the height tall enough to display the lines.

You can set the Text property of a multiline text box (or rich text box) to a long value; the value will wrap to fit in the width of the box. You also can enter multiple lines and choose the location of the line breaks; the techniques differ depending on whether you set the Text property at design time or in code. At design time, click on the Text property in the Properties window and click on the Properties button (the down arrow); a small editing window pops up with instructions to press Enter at the end of each line and Ctrl + Enter to accept the text (Figure 2.5). In code, you can use a **NewLine character** (`Environment.NewLine`) in the text string where you want the line to break. Joining strings of text is called *concatenation* and is covered in "Concatenating Text" later in this chapter.

```
TitleRichTextBox.Text = " Pamper Yourself" & Environment.NewLine & _
  "All Your Favorite Books"
```

Figure 2.5

Click the Properties button for the Text property and a small editing box pops up. To enter multiple lines of text, press Enter at the end of each line and Ctrl + Enter to accept the text.



Properties
SampleRichTextBox System.Win

ShowSelection	False
Size	298, 135
TabIndex	0
TabStop	True
Tag	
Text	

Press Enter to begin a new line.
Press Ctrl+Enter to accept Text.

Group Boxes

GroupBox controls are used as **containers** for other controls. Usually, groups of radio buttons or check boxes are placed in group boxes. Using group boxes to group controls can make your forms easier to understand by separating the controls into logical groups.

Set a group box's Text property to the words you want to appear on the top edge of the box.

Example Names for Group Boxes

```
ColorGroupBox
StyleGroupBox
```

Check Boxes

Check boxes allow the user to select (or deselect) an option. In any group of check boxes, any number can be selected. The **Checked property** of a check box is set to False if unchecked or True if checked.

You can write an event procedure for the CheckedChanged event, which executes when the user clicks in the box. In Chapter 4, when you learn about `If` statements, you can take one action when the box is checked and another action when it is unchecked.

Use the Text property of a check box for the text you want to appear next to the box.

Example Names for Check Boxes

```
BoldCheckBox
ItalicCheckBox
```

Radio Buttons

Use **radio buttons** when only one button of a group may be selected. Any radio buttons that you place directly on the form (not in a group box) function as a group. A group of radio buttons inside a group box function together. The best method is to first create a group box and then create each radio button inside the group box.

When you need separate lists of radio buttons for different purposes, you must include each list in a separate group box. You can find an example program later in this chapter that demonstrates using two groups of radio buttons, one for setting the background color of the form and a second set for selecting the color of the text on the form. See "Using Radio Buttons for Selecting Colors."

The Checked property of a radio button is set to True if selected or to False if unselected. You can write an event procedure to execute when the user selects a radio button using the control's CheckedChanged event. In Chapter 4 you will learn to determine in your code whether or not a button is selected.

Set a radio button's Text property to the text you want to appear next to the button.

Example Names for Radio Buttons

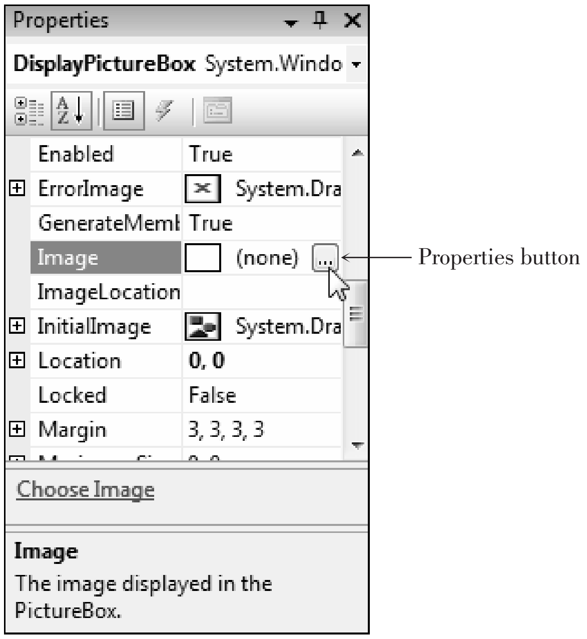
```
RedRadioButton
BlueRadioButton
```

Picture Boxes

A **PictureBox control** can hold an image. You can set a picture box's **Image property** to a graphic file with an extension of .bmp, .gif, .jpg, .jpeg, .png, .ico, .emf, or .wmf. You first add your images to the project's resources; then you can assign the resource to the Image property of a PictureBox control.

Place a PictureBox control on a form and then select its Image property in the Properties window. Click on the Properties button (Figure 2.6) to display a ***Select Resource* dialog box** where you can select images that you have already added or add new images (Figure 2.7).

Figure 2.6



Properties

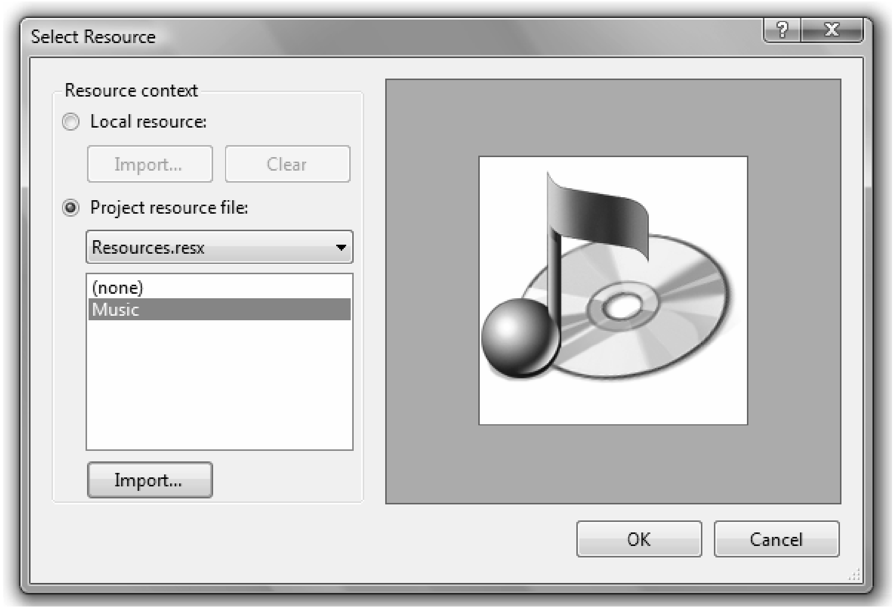
DisplayPictureBox System.Windo

Enabled	True
ErrorImage	System.Dra
GenerateMemb	True
Image	(none)
ImageLocation	
InitialImage	System.Dra
Location	0, 0
Locked	False
Margin	3, 3, 3, 3

Image
The image displayed in the PictureBox.

Click on the Image property for a PictureBox control, and a Properties button appears. Click on the Properties button to view the Select Resource dialog box.

Figure 2.7



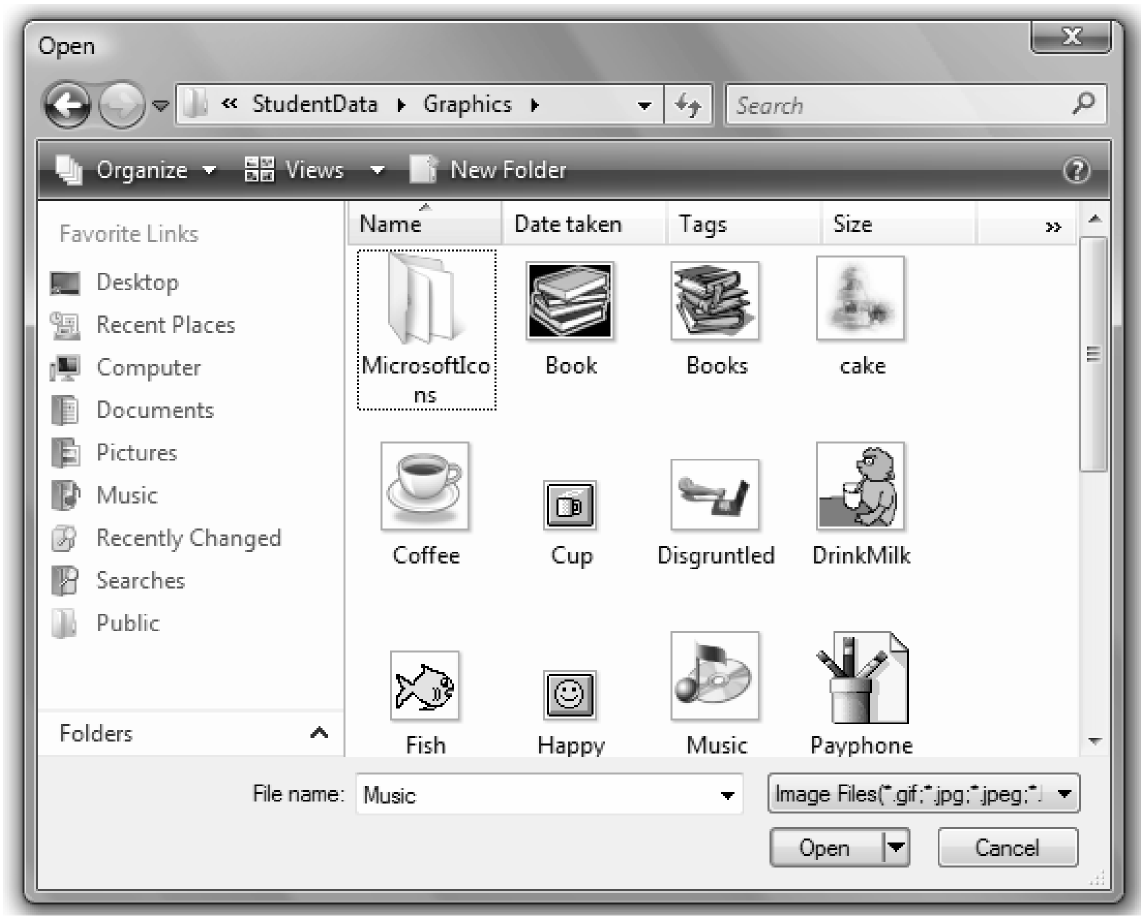
The Select Resource dialog box. Make your selection here for the graphic file you want to appear in the PictureBox control; click Import to add an image to the list.

Click on the *Import* button of the *Select Resource* dialog box to add images. An *Open* dialog box appears (Figure 2.8), where you can navigate to your image files. A preview of the image appears in the preview box.

Note: To add files with an ico extension, select *All Files* for the *Files of type* in the *Open* dialog box.

Figure 2.8

Find the text graphic files and icon files supplied by Microsoft in the Student Data\Graphics folder.



You can use any graphic file (with the proper format) that you have available. You will find many icon files included with the Student Data files on the text Web site (www.mhhe.com/VisualBasic2008).

PictureBox controls have several useful properties that you can set at design time or run time. For example, set the **SizeMode property** to ***StretchImage*** to make the graphic resize to fill the control. You can set the **Visible property** to False to make the picture box disappear.

For example, to make a picture box invisible at run time, use this code statement:

```
LogoPictureBox.Visible = False
```

You can assign a graphic from the Resources folder at run time. You use the `My` keyword, which is a special VB keyword that gives you access to many system settings. In the following example, "Sunset" is the name of a graphic (without its extension) in the Resources folder of the project.

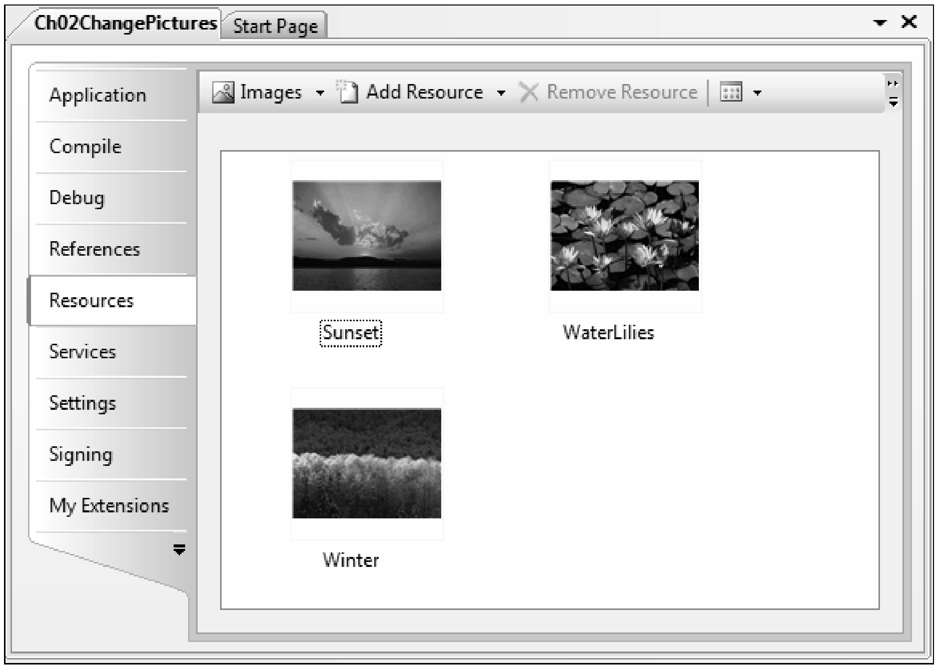
```
SamplePictureBox.Image = My.Resources.Sunset
```

Adding and Removing Resources

In Figure 2.6 you saw the easiest way to add a new graphic to the Resources folder, which you perform as you set the Image property of a PictureBox control. You also can add, remove, and rename resources using the Visual Studio **Project Designer**. From the *Project* menu, select *ProjectName Properties* (which always shows the name of the current project). The Project Designer opens in the main Document window; click on the *Resources* tab to display the project resources (Figure 2.9). You can use the buttons at the top of the window to add and remove images, or right-click an existing resource to rename or remove it.

Figure 2.9

Click on the Resources tab of the Project Designer to work with project resources. You can add, remove, and rename resources on this page.



Using Smart Tags

You can use smart tags to set the most common properties of many controls. When you add a PictureBox or a TextBox to a form, for example, you see a small arrow in the upper-right corner of the control. Click on the arrow to open the smart tag for that control (Figure 2.10). The smart tag shows a few properties that you can set from there, which is just a shortcut for making the changes from the Properties window.

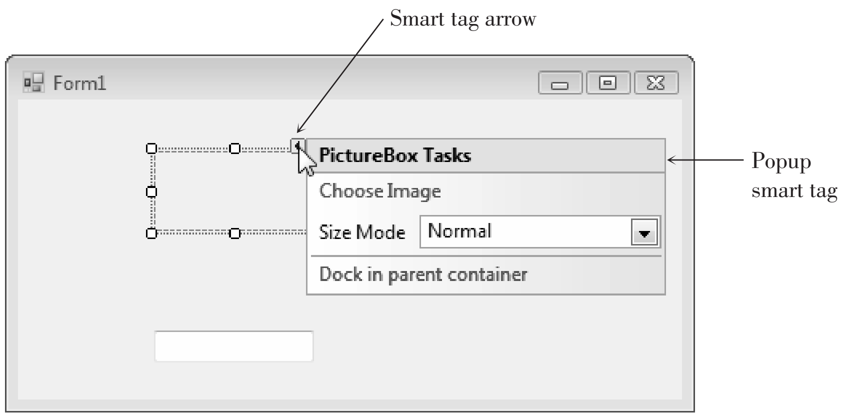


Figure 2.10

Point to the smart tag arrow to open the smart tag for a control. For this PictureBox control, you can set the Image, Size Mode, and Dock properties in the smart tag.

Using Images for Forms and Controls

You can use an image as the background of a form or a control. For a form, set the BackgroundImage property to a graphic resource; also set the form's BackgroundImageLayout property to Tile, Center, Stretch, or Zoom.

Controls such as buttons, check boxes, and radio buttons have an Image property that you can set to a graphic from the project's resources.

Setting a Border and Style

Most controls can appear to be three-dimensional or flat. Labels, text boxes, and picture boxes all have a **BorderStyle property** with choices of *None*, *FixedSingle*, or *Fixed3D*. Text boxes default to *Fixed3D*; labels and picture boxes default to *None*. Of course, you can change the property to the style of your choice.

Feedback 2.1

Create a picture box control that displays an enlarged icon and appears in a 3D box. Make up a name that conforms to this textbook's naming convention.

Property	**Setting**
Name	
BorderStyle	
SizeMode	
Visible	

Drawing a Line

You can draw a line on a form by using the Label control. You may want to include lines when creating a logo or you may simply want to divide the screen by drawing a line. To create the look of a line, set the AutoSize property of your label to False, set the Text property to blank, set the BorderStyle to *None*, and change the

Backcolor to the color you want for the line. You can control the size of the line with the Width and Height properties, located beneath the Size property.

Another way to draw a line on a form is to use the LineShape control, which you can download and install into Visual Studio. See "Downloading and Using the Line and Shape Controls" later in this chapter.

You also can draw a line on the form using the graphics methods. Drawing graphics is covered in Chapter 13.

Working with Multiple Controls

You can select more than one control at a time, which means that you can move the controls as a group, set similar properties for the group, and align the controls.

Selecting Multiple Controls

There are several methods of selecting multiple controls. If the controls are near each other, the easiest technique is to use the mouse to drag a selection box around the controls. Point to a spot that you want to be one corner of a box surrounding the controls, press the mouse button, and drag to the opposite corner (Figure 2.11). When you release the mouse button, the controls will all be selected (Figure 2.12). Note that selected labels and check boxes with AutoSize set to True do not have resizing handles; other selected controls do have resizing handles.

You also can select multiple controls, one at a time. Click on one control to select it, hold down the Ctrl key or the Shift key, and click on the next control. You can keep the Ctrl or Shift key down and continue clicking on controls you wish to select. Ctrl-click (or Shift-click) on a control a second time to deselect it without changing the rest of the group.

When you want to select most of the controls on the form, use a combination of the two methods. Drag a selection box around all of the controls to select them and then Ctrl-click on the ones you want to deselect. You also can select all of the controls using the *Select All* option on the *Edit* menu or its keyboard shortcut: Ctrl + A.

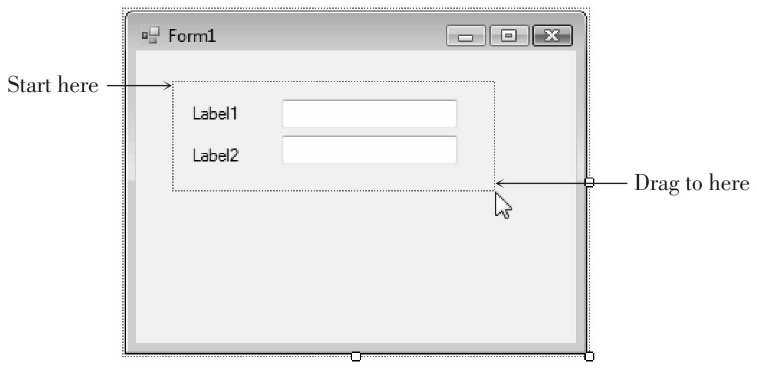


Figure 2.11

Use the pointer to drag a selection box around the controls you wish to select.

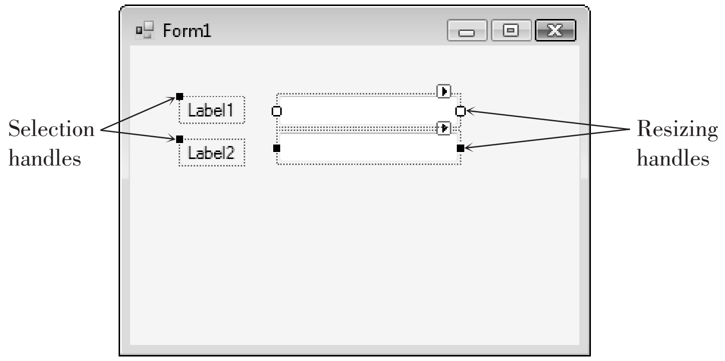


Figure 2.12

When multiple controls are selected, each has resizing handles (if resizable).

Deselecting a Group of Controls

When you are finished working with a group of controls, it's easy to deselect them. Just click anywhere on the form (not on a control) or select another previously unselected control.

Moving Controls as a Group

After selecting multiple controls, you can move them as a group. To do this, point inside one of the selected controls, press the mouse button, and drag the entire group to a new location (Figure 2.13).

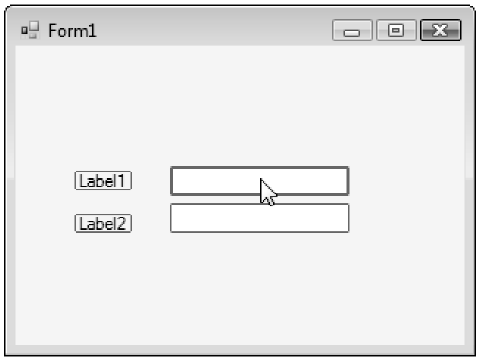


Figure 2.13

Drag a group of selected controls to move the entire group to a new location.

TIP

Make sure to read Appendix C for tips and shortcuts for working with controls. ■

Setting Properties for Multiple Controls

You can set some common properties for groups of controls. After selecting the group, look at the Properties window. Any properties that appear in the window are shared by all of the controls and can be changed all at once. For example, you may want to set the BorderStyle property for a group of controls to three-dimensional or change the font used for a group of labels. Some properties appear empty; those properties do not share a common value. You can enter a new value that will apply to all selected controls.

TIP

Setting the font for the form changes the default font for all controls on the form. ■

Aligning Controls

After you select a group of controls, it is easy to resize and align them using the buttons on the Layout toolbar (Figure 2.14) or the corresponding items on the *Format* menu. Select your group of controls and choose any of the resizing buttons. These can make the controls equal in width, height, or both. Then select another button to align the tops, bottoms, or centers of the controls. You also can move the entire group to a new location.

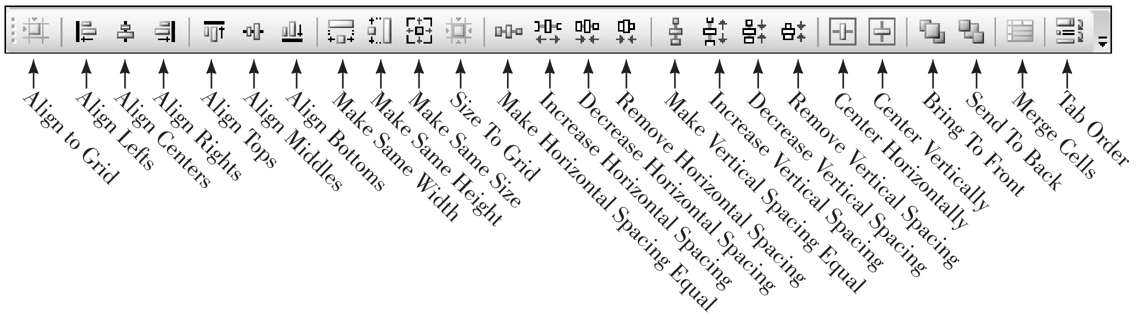
Note: The alignment options align the group of controls to the control that is active (the sizing handles are white). Referring to Figure 2.12, the upper text box is the active control. To make another selected control the active control, simply click on it.

To set the spacing between controls, use the buttons for horizontal and/or vertical spacing. These buttons enable you to create equal spacing between controls or to increase or decrease the space between controls.

Note: If the Layout toolbar is not displaying, select *View / Toolbars / Layout*.

Figure 2.14

Resize and align multiple controls using the Layout toolbar.



Designing Your Applications for User Convenience

One of the goals of good programming is to create programs that are easy to use. Your user interface should be clear and consistent. One school of thought says that if users misuse a program, it's the fault of the programmer, not the users. Because most of your users will already know how to operate Windows programs, you should strive to make your programs look and behave like other Windows programs. Some of the ways to accomplish this are to make the controls operate in the standard way, define keyboard access keys, set a default button, and make the Tab key work correctly. You also can define ToolTips, which are those small labels that pop up when the user pauses the mouse pointer over a control.

Designing the User Interface

The design of the screen should be easy to understand and "comfortable" for the user. The best way to accomplish these goals is to follow industry standards for the color, size, and placement of controls. Once users become accustomed to a screen design, they will expect (and feel more familiar with) applications that follow the same design criteria.

You should design your applications to match other Windows applications. Microsoft has done extensive program testing with users of different ages, genders, nationalities, and disabilities. We should take advantage of this research and follow their guidelines. Take some time to examine the screens and dialog boxes in Microsoft Office as well as those in Visual Studio.

One recommendation about interface design concerns color. You have probably noticed that Windows applications are predominantly gray. A reason for this choice is that many people are color blind. Also, research shows that gray is easiest for the majority of users. Although you may personally prefer brighter colors, you will stick with gray, or the system palette the user chooses, if you want your applications to look professional.

Note: By default the BackColor property of forms and controls is set to *Control*, which is a color included in the operating system's palette. If the user changes the system theme or color, your forms and controls will conform to their new settings.

Colors can indicate to the user what is expected. Use a white background for text boxes to indicate that the user should input information. Use a gray background for labels, which the user cannot change. Labels that will display a message should have a border around them; labels that provide text on the screen should have no border (the default).

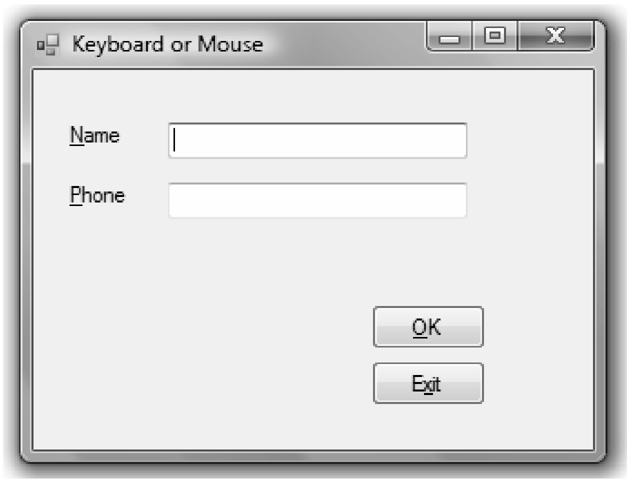
Group your controls on the form to aid the user. A good practice is to create group boxes to hold related items, especially those controls that require user input. This visual aid helps the user understand the information that is being presented or requested.

Use a sans serif font on your forms, such as the default MS Sans Serif, and do not make them boldface. Limit large font sizes to a few items, such as the company name.

Defining Keyboard Access Keys

Many people prefer to use the keyboard, rather than a mouse, for most operations. Windows is set up so that most functions can be done with either the keyboard or a mouse. You can make your projects respond to the keyboard by defining **access keys,** also called *hot keys*. For example, in Figure 2.15 you can select the *OK* button with Alt + o and the *Exit* button with Alt + x.

Figure 2.15



The underlined character defines an access key. The user can select the OK button by pressing Alt + o and the Exit button with Alt + x.

You can set access keys for buttons, radio buttons, and check boxes when you define their Text properties. Type an ampersand (&) in front of the character you want for the access key; Visual Basic underlines the character. You also can set an access key for a label; see "Setting the Tab Order for Controls" later in this chapter.

For examples of access keys on buttons, type the following for the button's Text property:

`&OK`	for	OK
`E&xit`	for	Exit

When you define access keys, you need to watch for several pitfalls. First, try to use the Windows-standard keys whenever possible. For example, use the x of Exit and the S of Save. Second, make sure you don't give two controls the same access key. It confuses the user and doesn't work correctly. Only the next control (from the currently active control) in the tab sequence is activated when the user presses the access key.

Note: To view the access keys on controls or menus in Windows 2000, Windows XP, or Windows Vista, you may have to press the Alt key, depending on your system settings. You can set Windows Vista to always show underlined shortcuts in the Control Panel's *Ease of Use* section. Select *Change how your keyboard works*, and check the box for *Underline keyboard shortcuts and access keys*.

TIP

Use two ampersands when you want to make an ampersand appear in the Text property: `&Health && Welfare` for "Health & Welfare". ■

Setting the Accept and Cancel Buttons

Are you a keyboard user? If so, do you mind having to pick up the mouse and click a button after typing text into a text box? Once a person's fingers are on the keyboard, most people prefer to press the Enter key, rather than to click the mouse. If one of the buttons on the form is the Accept button, pressing Enter is the same as clicking the button.

You can make one of your buttons the Accept button by setting the **AcceptButton property** of the form to the button name. When the user presses the Enter key, that button is automatically selected.

You also can select a *Cancel button.* The Cancel button is the button that is selected when the user presses the Esc key. You can make a button the Cancel button by setting the form's **CancelButton property**. An example of a good time to set the CancelButton property is on a form with *OK* and *Cancel* buttons. You may want to set the form's AcceptButton to OKButton and the CancelButton property to CancelButton.

Setting the Tab Order for Controls

In Windows programs, one control on the form always has the **focus**. You can see the focus change as you tab from control to control. For many controls, such as buttons, the focus appears as a thick border. Other controls indicate the focus by a dotted line or a shaded background. For text boxes, the insertion point (also called the *cursor*) appears inside the box.

Some controls can receive the focus; others cannot. For example, text boxes and buttons can receive the focus, but labels and picture boxes cannot.

The Tab Order

Two properties determine whether the focus stops on a control and the order in which the focus moves. Controls that are capable of receiving focus have a **TabStop property**, which you can set to True or False. If you do not want the focus to stop on a control when the user presses the Tab key, set the TabStop property to False.

The **TabIndex property** determines the order the focus moves as the Tab key is pressed. As you create controls on your form, Visual Studio assigns the TabIndex property in sequence. Most of the time that order is correct, but if you want to tab in some other sequence or if you add controls later, you will need to modify the TabIndex properties of your controls.

When your program begins running, the focus is on the control with the lowest TabIndex (usually 0). Since you generally want the insertion point to appear in the first control on the form, its TabIndex should be set to 0. The next control should be set to 1; the next to 2; and so forth.

You may be puzzled by the properties of labels, which have a TabIndex property but not a TabStop. A label cannot receive focus, but it has a location in the tab sequence. This fact allows you to create keyboard access keys for text boxes. When the user types an access key that is in a label, such as Alt + N, the focus jumps to the first TabIndex following the label (the text box). See Figure 2.16.

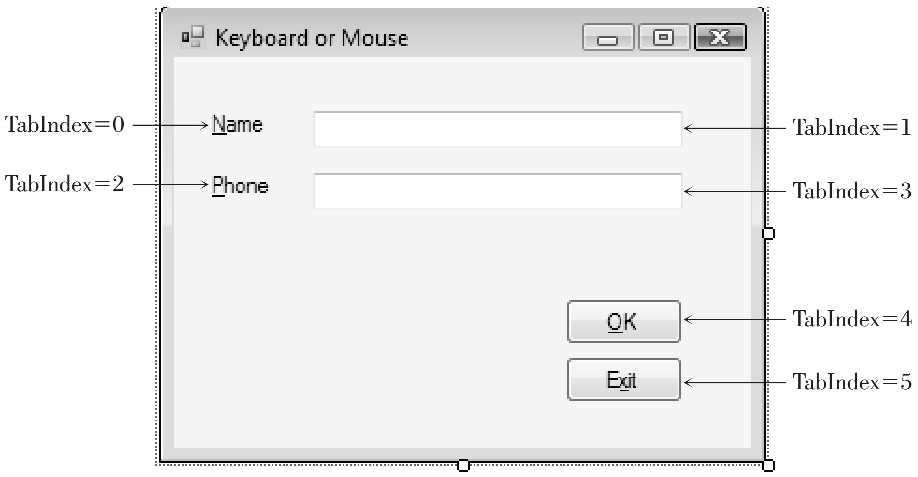


Figure 2.16

To use a keyboard access key for a text box, the TabIndex of the label must precede the TabIndex of the text box.

By default, buttons, text boxes, and radio buttons have their TabStop property set to True. Be aware that the behavior of radio buttons in the tab sequence is different from other controls: The Tab key takes you only to one radio button in a group (the selected button), even though all buttons in the group have their TabStop and TabIndex properties set. If you are using the keyboard to select radio buttons, you must tab to the group and then use your up and down arrow keys to select the correct button.

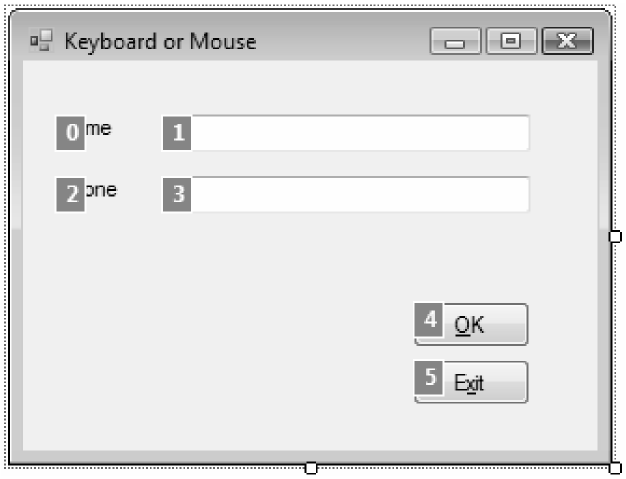
TIP

Make sure to not have duplicate numbers for the TabIndex properties or duplicate keyboard access keys. The result varies depending on the location of the focus and is very confusing. ■

Setting the Tab Order

To set the tab order for controls, you can set each control's TabIndex property in the Properties window. Or you can use Visual Studio's great feature that helps you set TabIndexes automatically. To use this feature, make sure that the Design window is active and select *View / Tab Order* or click the *Tab Order* button on the Layout toolbar. (The *Tab Order* item does not appear on the menu and is not available on the Layout toolbar unless the Design window is active.) Small numbers appear in the upper-left corner of each control; these are the current TabIndex properties of the controls. Click first in the control that you want to be TabIndex zero, then click on the control for TabIndex one, then click on the next control until you have set the TabIndex for all controls (Figure 2.17).

Figure 2.17



Click on each control, in sequence, to set the TabIndex property of the controls automatically.

When you have finished setting the TabIndex for all controls, the white numbered boxes change to blue. Select *View / Tab Order* again to hide the sequence numbers or press the Esc key. If you make a mistake and want to change the tab order, turn the option off and on again, and start over with TabIndex zero again, or you can keep clicking on the control until the number wraps around to the desired value.

TIP

To set the tab order for a group of controls, first set the TabIndex property for the group box and then set the TabIndex for controls inside the group. ■

Setting the Form's Location on the Screen

When your project runs, the form appears in the upper-left corner of the screen by default. You can set the form's screen position by setting the **StartPosition property** of the form. Figure 2.18 shows your choices for the property setting. To center your form on the user's screen, set the StartPosition property to *CenterScreen*.

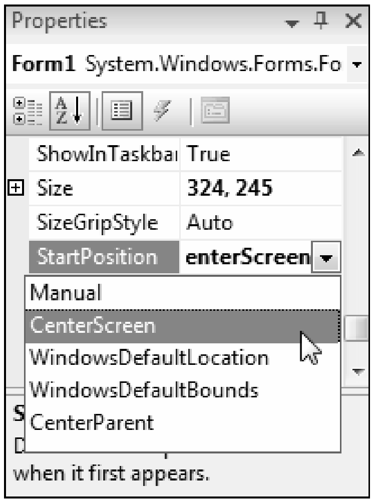


Figure 2.18

Set the StartPosition property of the form to CenterScreen to make the form appear in the center of the user's screen when the program runs.

Creating ToolTips

If you are a Windows user, you probably appreciate and rely on **ToolTips**, those small labels that pop up when you pause your mouse pointer over a toolbar button or control. You can easily add ToolTips to your projects by adding a **ToolTip component** to a form. After you add the component to your form, each of the form's controls has a new property: **ToolTip on ToolTip1**, assuming that you keep the default name, ToolTip1, for the control.

To define ToolTips, select the ToolTip tool from the toolbox (Figure 2.19) and click anywhere on the form or double-click the ToolTip tool in the toolbox. The new control appears in the component tray that opens at the bottom of the Form Designer (Figure 2.20). The component tray holds controls that do not have a visual representation at run time, such as the PrintForm control that you used in Chapter 1.

After you add the ToolTip component, examine the properties list for other controls on the form, such as buttons, text boxes, labels, radio buttons, check boxes, and even the form itself. Each has a new ToolTip on ToolTip1 property.

Try this example: Add a button to any form and add a ToolTip component. Change the button's Text property to Exit and set its ToolTip on ToolTip1 property to *Close and Exit the program*. Now run the project, point to the *Exit* button, and pause; the ToolTip will appear (Figure 2.21).

You also can add multiline ToolTips. In the ToolTip on ToolTip1 property, click the drop-down arrow. This drops down a white editing box in which you enter the text of the ToolTip. Type the first line and press Enter to create a second line; press Ctrl + Enter to accept the text (or click somewhere outside the Property window).

Figure 2.19

Add a ToolTip component to your form; each of the form's controls will have a new property to hold the text of the ToolTip.

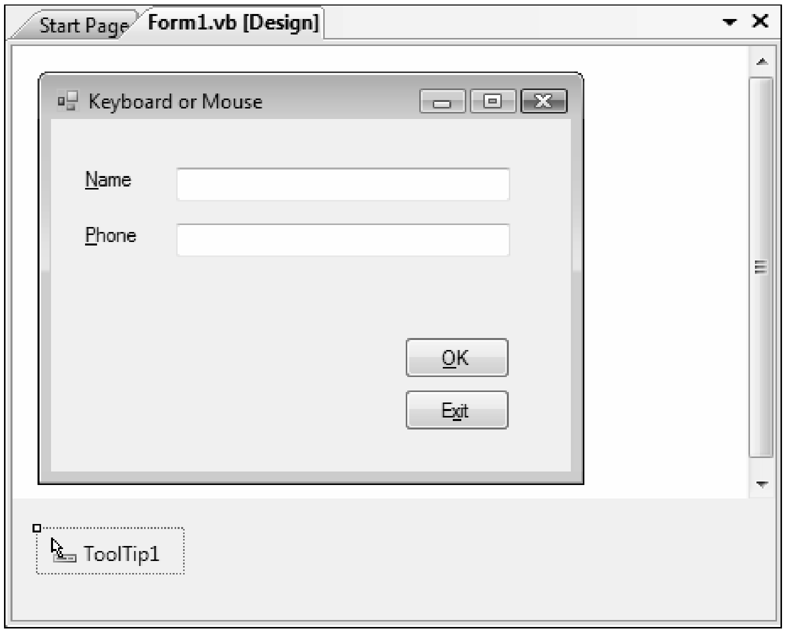


Figure 2.20

The new ToolTip component goes in the component tray at the bottom of the Form Designer window.

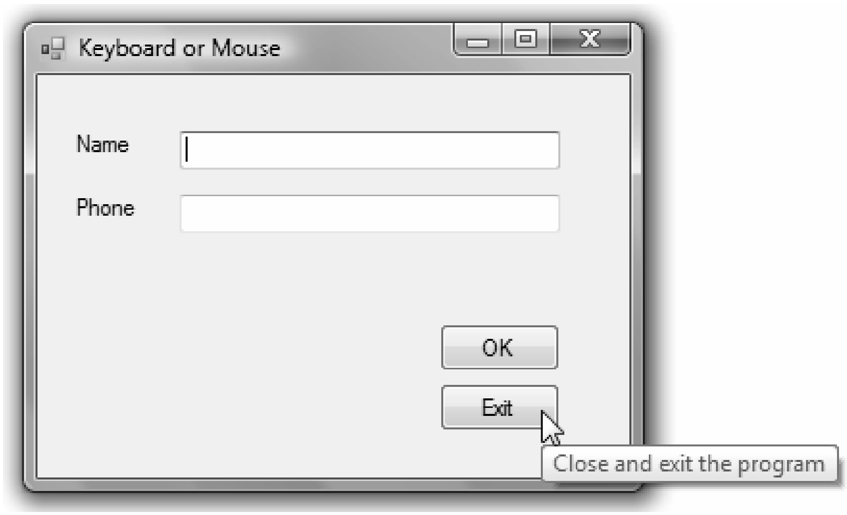


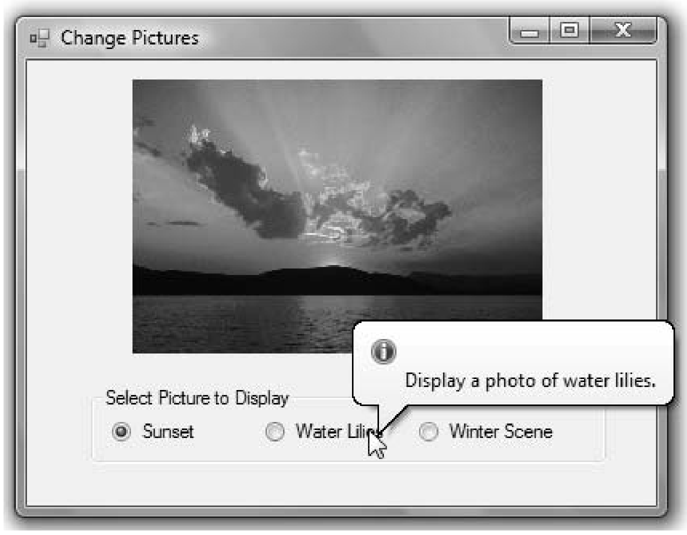
Figure 2.21

Use the ToolTip on ToolTip1 property to define a ToolTip.

You can modify the appearance of a ToolTip by setting properties of the ToolTip component. Select the ToolTip component in the component tray and try changing the BackColor and ForeColor properties. You also can set the IsBalloon property to True for a different appearance and include an icon in the ToolTips by selecting an icon for the ToolTipIcon property (Figure 2.22). Once you set properties for a ToolTip component, they apply to all ToolTips displayed with that component. If you want to create a variety of appearances, the best approach is to create multiple ToolTip components, giving each a unique name. For example, you might create three ToolTip components, in which case you would have properties for ToolTip on ToolTip1, ToolTip on ToolTip2, and ToolTip on ToolTip3 for the form and each control.

Figure 2.22

A ToolTip with properties modified for IsBalloon, ToolTipIcon, BackColor, and ForeColor.



Coding for the Controls

You already know how to set initial properties for controls at design time. You also may want to set some properties in code, as your project executes. You can clear out the contents of text boxes and labels, reset the focus (the active control), change the color of text, or change the text in a ToolTip.

Clearing Text Boxes and Labels

You can clear out the contents of a text box or label by setting the property to an **empty string**. Use "" (no space between the two quotation marks). This empty string is also called a *null string* or *zero-length string*. You also can clear out a text box using the `Clear` method or setting the Text property to `String.Empty`. Note that the `Clear` method works for text boxes but not for labels.

Examples

```
' Clear the contents of text boxes and labels.
NameTextBox.Text = ""
MessageLabel.Text = ""
DataTextBox.Clear()
MessageLabel.Text = String.Empty
```

Resetting the Focus

As your program runs, you want the insertion point to appear in the text box where the user is expected to type. The focus should therefore begin in the first text box. But what about later? If you clear the form's text boxes, you should reset the focus to the first text box. The **`Focus` method** handles this situation.

Remember, the convention is Object.Method, so the statement to set the insertion point in the text box called NameTextBox is as follows:

```
' Make the insertion point appear in this text box.
NameTextBox.Focus()
```

Note: You cannot set the focus to a control that has been disabled. See "Disabling Controls" later in this chapter.

Setting the Checked Property of Radio Buttons and Check Boxes

Of course, the purpose of radio buttons and check boxes is to allow the user to make selections. However, at times you need to select or deselect a control in code. You can select or deselect radio buttons and check boxes at design time (to set initial status) or at run time (to respond to an event).

To make a radio button or check box appear selected initially, set its Checked property to True in the Properties window. In code, assign True to its Checked property:

```
' Make button selected.
RedRadioButton.Checked = True

' Make check box checked.
DisplayCheckBox.Checked = True

' Make check box unchecked.
DisplayCheckBox.Checked = False
```

At times, you need to reset the selected radio button at run time, usually for a second request. You only need to set the Checked property to True for one button of the group; the rest of the buttons in the group will set to False automatically. Recall that only one radio button of a group can be selected at one time.

Setting Visibility at Run Time

You can set the visibility of a control at run time.

```
' Make label invisible.
MessageLabel.Visible = False
```

You may want the visibility of a control to depend on the selection a user makes in a check box or radio button. This statement makes the visibility match the check box: When the check box is checked (Checked = True), the label is visible (Visible = True).

```
' Make the visibility of the label match the setting in the check box.
MessageLabel.Visible = DisplayCheckBox.Checked
```

Disabling Controls

The **Enabled property** of a control determines whether the control is available or "grayed-out." The Enabled property for controls is set to True by default, but you can change the value either at design time or run time. You might want to disable a button or other control initially and enable it in code, depending on an action of the user. If you disable a button control (Enabled = False) at design time, you can use the following code to enable the button at run time.

```
DisplayButton.Enabled = True
```

When you have a choice to disable or hide a control, it's usually best to disable it. Having a control disabled is more understandable to a user than having it disappear.

To disable radio buttons, consider disabling the group box holding the buttons, rather than the buttons themselves. Disabling the group box grays the entire group.

```
DepartmentGroupBox.Enabled = False
```

Note: Even though a control has its TabStop property set to True and its TabIndex is in the proper order, you cannot tab to a control that has been disabled.

Feedback 2.2

1. Write the Basic statements to clear the text box called CompanyTextBox and reset the insertion point into the box.
2. Write the Basic statements to clear the label called CustomerLabel and place the insertion point into a text box called OrderTextBox.
3. What will be the effect of each of these Basic statements?
   (a) `PrintCheckBox.Checked = True`
   (b) `ColorRadioButton.Checked = True`
   (c) `DrawingPictureBox.Visible = False`
   (d) `LocationLabel.BorderStyle = BorderStyle.Fixed3D`
   (e) `CityLabel.Text = CityTextBox.Text`
   (f) `RedRadioButton.Enabled = True`

Setting Properties Based on User Actions

Often you need to change the Enabled or Visible property of a control based on an action of the user. For example, you may have controls that are disabled or invisible until the user signs in. In the following example, when the user logs in and clicks the *Sign In* button, several controls become visible, others become invisible, and a group box of radio buttons is enabled:

```
Private Sub SignInButton_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles SignInButton.Click
    ' Set up the screen to display the department promotions and the
    ' welcome message. Hide the sign-in controls.

    ' Hide and disable the sign-in controls.
    CheckInGroupBox.Visible = False
    SignInButton.Enabled = False

    ' Enable the group of radio buttons.
    DepartmentGroupBox.Enabled = True

    ' Show the promotions controls.
    PromotionTextBox.Visible = True
    ImageVisibleCheckBox.Visible = True
    WelcomeRichTextBox.Visible = True

    'Display the welcome message.
    WelcomeRichTextBox.Text = "Welcome Member #" & MemberIDMaskedTextBox.Text _
      & Environment.NewLine & NameTextBox.Text
End Sub
```

Changing the Color of Text

You can change the color of text by changing the **ForeColor property** of a control. Actually, most controls have a ForeColor and a BackColor property. The ForeColor property changes the color of the text; the BackColor property determines the color around the text.

The Color Constants

Visual Basic provides an easy way to specify a large number of colors. These **color constants** are in the Color class. If you type the keyword `Color` and a period in the editor, you can see a full list of colors. Some of the colors are listed below.

```
Color.AliceBlue
Color.AntiqueWhite
Color.Bisque
Color.BlanchedAlmond
Color.Blue
```

Examples

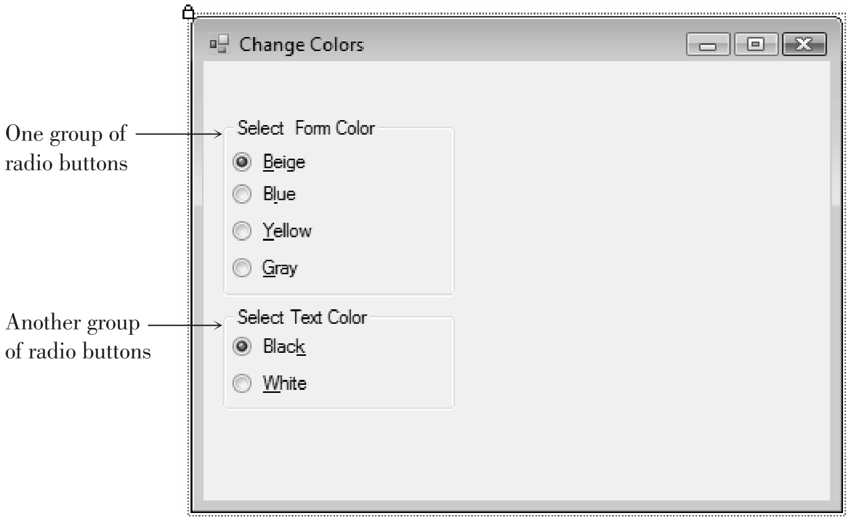
```
NameTextBox.ForeColor = Color.Red
MessageLabel.ForeColor = Color.White
```

Using Radio Buttons for Selecting Colors

Here is a small example (Figure 2.23) that demonstrates using two groups of radio buttons to change the color of the form (the form's BackColor property) and the color of the text (the form's ForeColor property). The radio buttons in each group box operate together, independently of those in the other group box.

Figure 2.23

The radio buttons in each group box function independently of the other group. Each button changes a property of the form: BackColor changes the background of the form itself and ForeColor changes the color of the text on the form.



```
'Project:               Ch02Colors
'Programmer:            Bradley/Millspaugh
'Date:                  June 2008
'Description:           Demonstrate changing a form's background and foreground
'                       colors using two groups of radio buttons.

Public Class ColorsForm

  Private Sub BeigeRadioButton_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BeigeRadioButton.CheckedChanged
      ' Set the form color to beige.

      Me.BackColor = Color.Beige
  End Sub
```

```
  Private Sub BlueRadioButton_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BlueRadioButton.CheckedChanged
      ' Set the form color to blue.

      Me.BackColor = Color.Blue
  End Sub
```

```
  Private Sub YellowRadioButton_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles YellowRadioButton.CheckedChanged
      ' Set the form color to yellow.

      Me.BackColor = Color.Yellow
  End Sub
```

```
  Private Sub GrayRadioButton_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles GrayRadioButton.CheckedChanged
      ' Set the form color to gray.

      Me.BackColor = Color.Gray
  End Sub
  Private Sub BlackRadioButton_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BlackRadioButton.CheckedChanged
      ' Set the text color to black.

      Me.ForeColor = Color.Black
  End Sub
```

```
  Private Sub WhiteRadioButton_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles WhiteRadioButton.CheckedChanged
      ' Set the text color to white.

      Me.ForeColor = Color.White
  End Sub

End Class
```

Changing Multiple Properties of a Control

By now you can see that there are times when you will want to change several properties of a single control. In early versions of Visual Basic, you had to write out the entire name (Object.Property) for each statement.

Examples

```
TitleTextBox.Visible = True
TitleTextBox.ForeColor = Color.White
TitleTextBox.Focus()
```

Of course, you can still specify the statements this way, but Visual Basic provides a better way: the **`With` and `End With` statements**.

The With and End With Statements—General Form

General Form

```
With ObjectName
   ' Statement(s)
End With
```

You specify an object name in the `With` statement. All subsequent statements until the `End With` relate to that object.

The With and End With Statements—Example

Example

```
With TitleTextBox
  .Visible = True
  .ForeColor = Color.White
  .Focus()
End With
```

The statements beginning with `With` and ending with `End With` are called a ***With block***. The statements inside the block are indented for readability. Although indentation is not required by VB, it is required by good programming practices and aids in readability.

The real advantage of using the With statement, rather than spelling out the object for each statement, is that With is more efficient. Your Visual Basic projects will run a little faster if you use With. On a large, complicated project, the savings can be significant.

Concatenating Text

At times you need to join strings of text. For example, you may want to join a literal and a property. You can "tack" one string of characters to the end of another in the process called ***concatenation***. Use an ampersand (&), preceded and followed by a space, between the two strings.

Examples

```
MessageLabel.Text = "Your name is: " & NameTextBox.Text
NameAndAddressLabel.Text = NameTextBox.Text & " " & AddressTextBox.Text
```

Continuing Long Program Lines

Basic interprets the code on one line as one statement. You can type very long lines in the Editor window; the window scrolls sideways to allow you to keep typing. However, this method is inconvenient; it isn't easy to see the ends of the long lines.

When a Basic statement becomes too long for one line, use a **line-continuation character**. You can type a space and an underscore, press Enter, and continue the statement on the next line. It is OK to indent the continued lines. The only restriction is that the line-continuation character must appear between elements; you cannot place a continuation in the middle of a literal or split the name of an object or property.

TIP

Although in some situations Basic allows concatenation with the + operator, the practice is not advised. Depending on the contents of the text box, the compiler may interpret the + operator as an addition operator rather than a concatenation operator, giving unpredictable results or an error. ■

Example

```
GreetingsLabel.Text = "Greetings " & NameTextBox.Text & ": " & _
  "You have been selected to win a free prize. " & _
  "Just send us $100 for postage and handling."
```

Downloading and Using the Line and Shape Controls

You can add graphic shapes to your forms using a set of controls that Microsoft makes available in a PowerPack, which is a separate and free download. After you download the PowerPack, you run the installation file, which installs the controls into Visual Studio. Once installed, you can add the controls to the Visual Studio toolbox.

Download and Install the Controls

The first step is to download from Microsoft's site: http://msdn2.microsoft.com/en-us/vbasic/bb735936.aspx and follow the links to download the installation file.

It's best to download the file VisualBasicPowerPacks3Setup.exe to your hard drive (save it somewhere easy to find, such as the desktop). After the download is complete, make sure that Visual Studio is not running and double-click the setup file name to run the setup.

If you are running the Professional Edition or above and Visual Studio is closed, the new tools are automatically added to a new section of the toolbox. You can find the new section, called Visual Basic Power Packs 3.0, at the bottom of the toolbox (Figure 2.24).

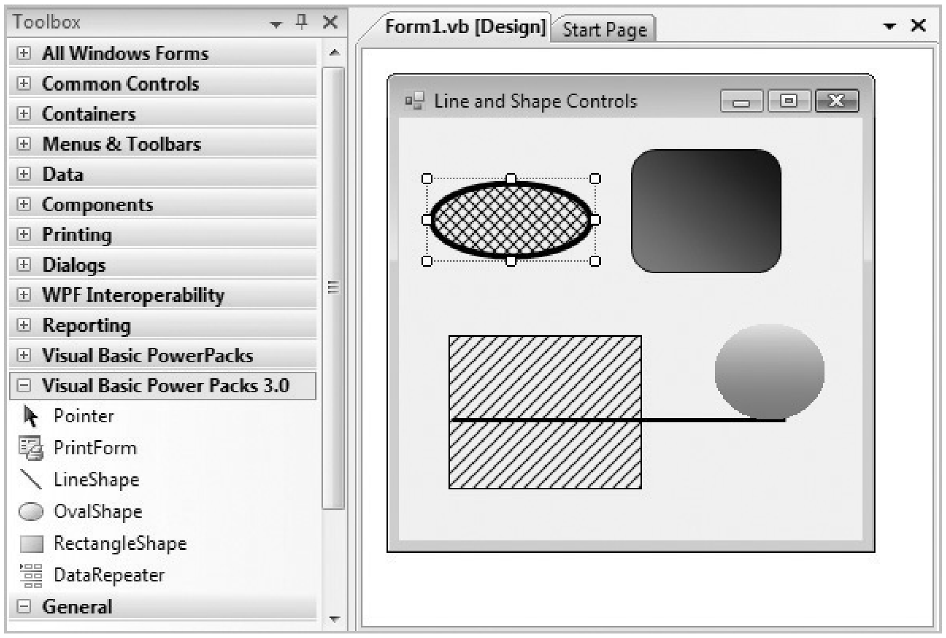
For the Express Edition, or the Professional Edition if the IDE was open when you ran setup, you must manually add the controls to the toolbox. Open Visual Studio or Visual Basic Express and start a new project so that you can see the Form Designer and the toolbox. Right-click in the toolbox and select *Add Tab*. Type "Visual Basic Power Packs 3.0" as the Tab name, then right-click on the new tab and select *Choose Items* from the context menu. The *Choose Toolbox Items* dialog box appears with the list of all available tools. You can save some time by typing "Power" in the *Filter* box, which will limit the list to the PowerPack controls. Then select *LineShape*, *OvalShape*, and *RectangleShape* (*PrintForm* should already be selected). If you have multiple versions of the controls listed, choose the highest version number (9.0.0.0 as of this writing). Then click OK to return to the toolbox.

The new tools should appear in the Visual Basic PowerPacks 3.0 tab of the toolbox.

Note: The controls appear in the section of the toolbox that is active when you select *Choose Toolbox Items*.

Figure 2.24

The line and shape controls in the toolbox, with some sample controls on the form.



Place the Controls on a Form

To place a control on the form, click on the tool in the toolbox and use the mouse pointer to draw the shape that you want on the form. Alternately, you can double-click one of the tools to create a default size control that you can move and resize as desired.

The line and shape controls have many properties that you can set, as well as events, such as Click and DoubleClick, for which you can write event procedures.

Properties of a line include BorderStyle (Solid, Dot, Dash, and a few more); BorderWidth (the width of the line, in pixels); BorderColor; and the locations for the two endpoints of the line (X1, X2, X3, X4). Of course, you can move and resize the line visually, but it sometimes is more precise to set the pixel location exactly.

The properties of the shape controls are more interesting. You can set transparency with the BackStyle property and the border with BorderColor, BorderStyle, and BorderWidth. Set the interior of the shape using FillColor, FillGradientColor, FillGradientStyle, and FillStyle. You can give a rectangle rounded corners by setting the CornerRadius, which defaults to zero for square corners.

Your Hands-On Programming Example

Create a form that allows members to log in and view special promotions for R 'n R. The member name is entered in a text box, and the member ID is entered in a masked text box that allows five numeric digits. Include three buttons: one for *Sign In*, one for *Print*, and one for *Exit*. Set the AcceptButton to the *Sign In* button, and use the *Exit* button for the CancelButton. Include keyboard shortcuts as needed.

Use a group box of radio buttons for the departments and another group box to visually separate the data entry boxes (the text box and masked text box).

A check box allows the user to choose whether an image should display for the selected department. You will need an image to display for each department. You can use any images that you have available, find them on the Web, or use images from the text Web site.

When the program begins, the Department group box should be disabled and visible, and the controls for the image, promotion, and check box should be invisible. When the user clicks on the *Sign In* button, the data entry boxes and labels should disappear, and the promotions text box, the department image, and the check box should appear. Display the user name and member ID concatenated in a rich text box. (*Hint*: If you place the data entry controls in a group box, you can simply set the group box visibility to False.)

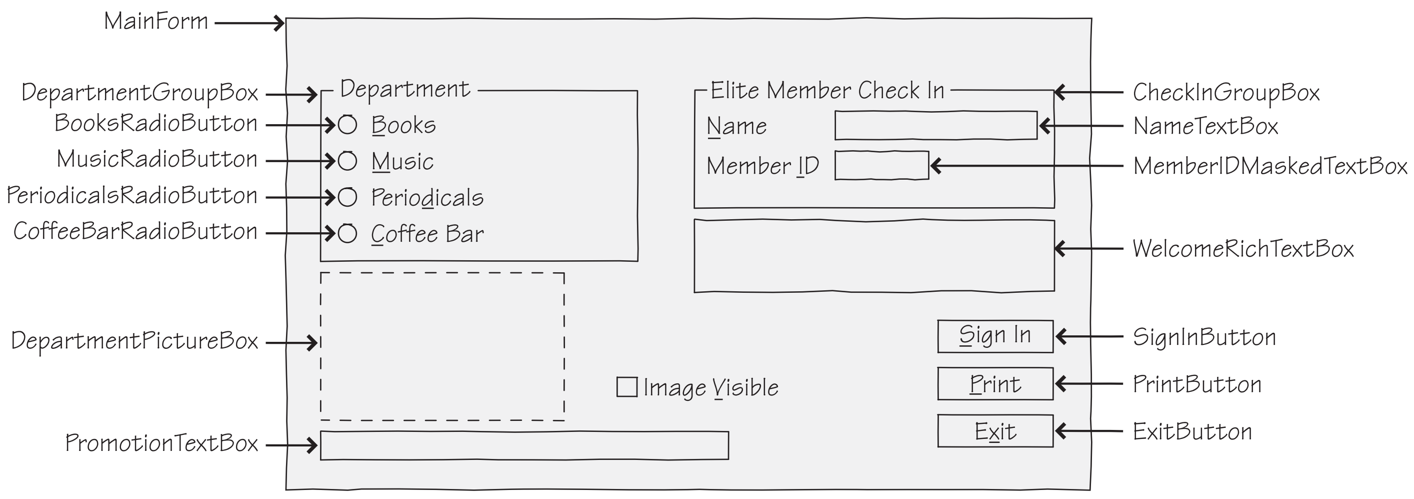
Add ToolTips as appropriate. The ToolTip for the Member ID text box should say: "Your 5 digit member number."

Planning the Project

Sketch a form (Figure 2.25), which your users sign off as meeting their needs.

Figure 2.25

A planning sketch of the form for the hands-on programming example.



Note: Although this step may seem unnecessary, having your users sign off is standard programming practice and documents that your users have been involved and have approved the design.

Plan the Objects and Properties Plan the property settings for the form and for each control.

Object	**Property**	**Setting**
MainForm	Name	MainForm
	Text	R 'n R--For Reading and Refreshment"
	AcceptButton	SignInButton
	CancelButton	ExitButton
	StartPosition	CenterScreen
DepartmentGroupBox	Name	DepartmentGroupBox
	Text	Department
BooksRadioButton	Name	BooksRadioButton
	Text	&Books
MusicRadioButton	Name	MusicRadioButton
	Text	&Music
PeriodicalsRadioButton	Name	PeriodicalsRadioButton
	Text	Perio&dicals
CoffeeBarRadioButton	Name	CoffeeBarRadioButton
	Text	&Coffee Bar
SignInButton	Name	SignInButton
	Text	&Sign In
PrintButton	Name	PrintButton
	Text	&Print
ExitButton	Name	ExitButton
	Text	E&xit
DepartmentPictureBox	Name	DepartmentPictureBox
	Visible	False
	SizeMode	StretchImage
ImageVisibleCheckBox	Name	ImageVisibleCheckBox
	Text	Image &Visible
	Visible	False
PromotionTextBox	Name	PromotionTextBox
	Text	(blank)

Object	Property	Setting
CheckInGroupBox	Name	CheckInGroupBox
	Text	Elite Member Check In
Label1	Text	&Name
Label2	Text	Member &ID
NameTextBox	Name	NameTextBox
	Text	(blank)
MemberIDMaskedTextBox	Name	MemberIDMaskedTextBox
	Text	(blank)
	Mask	00000
WelcomeRichTextBox	Name	WelcomeRichTextBox
	Text	(blank)
PrintForm1	Name	PrintForm1
ToolTip1	Name	ToolTip1

Plan the Event Procedures You will need event procedures for each button, radio button, and check box.

Procedure	Actions-Pseudocode
BooksRadioButton_CheckedChanged	Display "Buy two, get the second one for half price" and the books image.
MusicRadioButton_CheckedChanged	Display "Get a free MP3 download with purchase of a CD" and the music image.
PeriodicalsRadioButton_CheckedChanged	Display "Elite members receive 10% off every purchase" and the periodicals image.
CoffeeBarRadioButton_CheckedChanged	Display "Celebrate the season with 20% off specialty drinks" and the coffee image.
ImageVisibleCheckBox_CheckedChanged	Set the visibility of the department image.
SignInButton_Click	Hide the check-in group box.
	Disable the sign-in button
	Enable the Department group box.
	Make the check box, promotion text box, and the welcome text box visible.
	Concatenate the name and member ID and display them in the welcome rich text box.
PrintButton_Click	Set the print action to Print Preview.
	Call the Print method.
ExitButton_Click	End the project.

Write the Project Follow the sketch in Figure 2.25 to create the form. Figure 2.26 shows the completed form.

- Set the properties of each object, as you have planned. Make sure to set the tab order of the controls.
- Working from the pseudocode, write each event procedure.
- When you complete the code, thoroughly test the project.

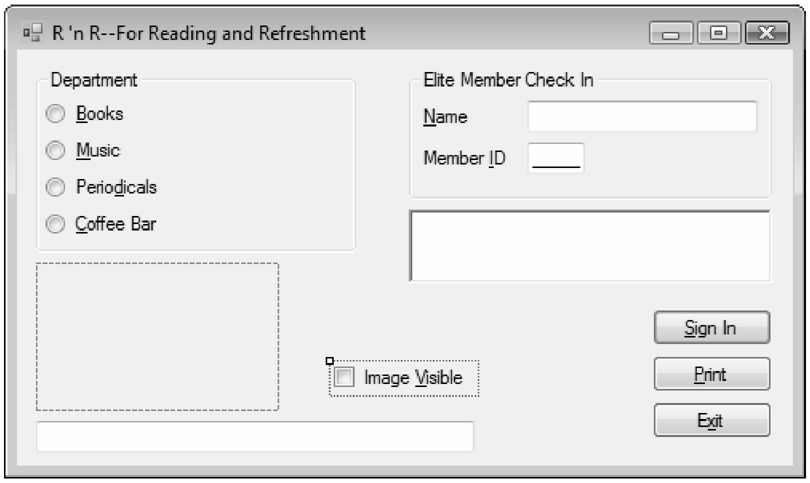


Figure 2.26

The form for the hands-on programming example.

The Project Coding Solution

```
'Project:      Ch02HandsOn
'Programmer:   Bradley/Millspaugh
'Date:         June 2008
'Description:  Allow the user to sign in and display current sales promotion.

Public Class MainForm

   Private Sub BooksRadioButton_CheckedChanged(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles BooksRadioButton.CheckedChanged
       ' Display the image and promotion for the Books department.

       DepartmentPictureBox.Image = My.Resources.Books
       PromotionTextBox.Text = "Buy two, get the second one for half price."
   End Sub

   Private Sub CoffeeBarRadioButton_CheckedChanged(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles CoffeeBarRadioButton.CheckedChanged
       ' Display the image and promotion for the Coffee Bar.

       DepartmentPictureBox.Image = My.Resources.Coffee
       PromotionTextBox.Text = _
         "Celebrate the season with 20% off specialty drinks."
   End Sub
```

```
  Private Sub MusicRadioButton_CheckedChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MusicRadioButton.CheckedChanged
      ' Display the image and promotion for the Music department.

      DepartmentPictureBox.Image = My.Resources.Music
      PromotionTextBox.Text = "Get a free MP3 download with purchase of a CD."
  End Sub

  Private Sub PeriodicalsRadioButton_CheckedChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles PeriodicalsRadioButton.CheckedChanged
      ' Display the image and promotion for the Periodicals department.

      DepartmentPictureBox.Image = My.Resources.Periodicals
      PromotionTextBox.Text = "Elite members receive 10% off every purchase."
  End Sub

  Private Sub ImageVisibleCheckBox_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ImageVisibleCheckBox.CheckedChanged
      ' Set the visibility of the department image.

      DepartmentPictureBox.Visible = ImageVisibleCheckBox.Checked
  End Sub

  Private Sub SignInButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles SignInButton.Click
      ' Set up the screen to display the department promotions and the
      ' welcome message. Hide the sign-in controls.

      ' Hide and disable the sign-in controls.
      CheckInGroupBox.Visible = False
      SignInButton.Enabled = False

      ' Enable the group of radio buttons.
      DepartmentGroupBox.Enabled = True

      ' Show the promotions controls.
      PromotionTextBox.Visible = True
      ImageVisibleCheckBox.Visible = True
      WelcomeRichTextBox.Visible = True

      ' Display the welcome message.
      WelcomeRichTextBox.Text = "Welcome Member #" & MemberIDMaskedTextBox.Text _
        & Environment.NewLine & NameTextBox.Text
  End Sub

  Private Sub PrintButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles PrintButton.Click
      ' Print the form in the Print Preview window.

      PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
      PrintForm1.Print()
  End Sub

  Private Sub ExitButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ExitButton.Click
      ' End the program.

      Me.Close()
  End Sub
End Class
```

Good Programming Habits

1. Always test the tab order on your forms. Fix it if necessary by changing the TabIndex properties of the controls.
2. Provide visual separation for input fields and output fields and always make it clear to the user which are which.
3. Make sure that your forms can be navigated and entered from the keyboard. Always set an Accept button (AcceptButton property) for every form.
4. To make a label maintain its size regardless of the value of the Text property, set AutoSize to False.
5. To make the text in a text box right justified or centered, set the TextAlign property.
6. You can use the Checked property of a check box to set other properties that must be True or False.

Summary

1. Text boxes are used primarily for user input. The Text property holds the value input by the user. You also can assign a literal to the Text property during design time or run time.
2. A MaskedTextBox has a Mask property that allows you to specify the data type and format of the input data.
3. A RichTextBox is a specialized text box that allows additional formatting of the text.
4. Both text boxes and rich text boxes have Multiline and WordWrap properties that can allow a long Text property to wrap to multiple lines. The text will wrap to the width of the control, which must be tall enough to display multiple lines.
5. Group boxes are used as containers for other controls and to group like items on a form.
6. Check boxes and radio buttons allow the user to make choices. In a group of radio buttons, only one can be selected; but in a group of check boxes, any number of the boxes may be selected.
7. The current state of check boxes and radio buttons is stored in the Checked property; the CheckedChanged event occurs when the user clicks on one of the controls.
8. Picture box controls hold a graphic, which is assigned to the Image property. Set the SizeMode property to *StretchImage* to make the image resize to fit the control.
9. The *Resources* tab of the Project Designer can be used to add, remove, and rename images in the project Resources folder.
10. Forms and controls can display images from the project's resources. Use the form's BackgroundImage property and a control's Image property.
11. The BorderStyle property of many controls can be set to *None*, *FixedSingle*, or *Fixed3D* to determine whether the control appears flat or three-dimensional.
12. To create a line on a form, you can use a Label control, or use the new LineShape control included in the Visual Basic Power Packs.

13. You can select multiple controls and treat them as a group, including setting common properties at once, moving them, or aligning them.
14. Make your programs easier to use by following Windows standard guidelines for colors, control size and placement, access keys, default and Cancel buttons, and tab order.
15. Define keyboard access keys by including an ampersand in the Text property of buttons, radio buttons, check boxes, and labels.
16. Set the AcceptButton property of the form to the desired button so that the user can press Enter to select the button. If you set the form's CancelButton property to a button, that button will be selected when the user presses the Esc key.
17. The focus moves from control to control as the user presses the Tab key. The sequence for tabbing is determined by the TabIndex properties of the controls. The Tab key stops only on controls that have their TabStop property set to True and are enabled.
18. Set the form's location on the screen by setting the StartPosition property.
19. Add a ToolTip control to a form and then set the ToolTip on ToolTip1 property of a control to make a ToolTip appear when the user pauses the mouse pointer over the control. You can set properties of the ToolTip component to modify the background, foreground, shape, and an icon for the ToolTips.
20. Clear the Text property of a text box or a label by setting it to an empty string. Text boxes also can be cleared using the `Clear` method.
21. To make a control have the focus, which makes it the active control, use the `Focus` method. Using the `Focus` method of a text box makes the insertion point appear in the text box. You cannot set the focus to a disabled control.
22. You can set the Checked property of a radio button or check box at run time and also set the Visible property of controls in code.
23. Controls can be disabled by setting the Enabled property to False.
24. Change the color of text in a control by changing its ForeColor property.
25. You can use the color constants to change colors during run time.
26. The `With` and `End With` statements provide an easy way to refer to an object multiple times without repeating the object's name.
27. Joining two strings of text is called *concatenation* and is accomplished by placing an ampersand between the two elements. (A space must precede and follow the ampersand.)
28. Use a space and an underscore to continue a long statement on another line.

Key Terms

AcceptButton property *82*
access key *81*
BorderStyle property *77*
CancelButton property *82*
check box *73*
Checked property *73*
color constant *90*
concatenation *93*
container *72*
empty string *87*
Enabled property *89*
focus *82*
`Focus` method *87*
ForeColor property *90*
GroupBox control *72*
Image property *73*
line-continuation character *93*
MaskedTextBox *70*

Review Questions

1. You can display program output in a text box or a label. When should you use a text box? When is a label appropriate?
2. What would be the advantage of using a masked text box rather than a text box?
3. When would it be appropriate to use a rich text box instead of a text box?
4. What properties of a TextBox and RichTextBox must be set to allow a long Text property to wrap to multiple lines?
5. How does the behavior of radio buttons differ from the behavior of check boxes?
6. If you want two groups of radio buttons on a form, how can you make the groups operate independently?
7. Explain how to make a graphic appear in a picture box control.
8. Describe how to select several labels and set them all to 12-point font size at once.
9. What is the purpose of keyboard access keys? How can you define them in your project? How do they operate at run time?
10. Explain the purpose of the AcceptButton and CancelButton properties of the form. Give an example of a good use for each.
11. What is a ToolTip? How can you make a ToolTip appear?
12. What is the focus? How can you control which object has the focus?
13. Assume you are testing your project and don't like the initial position of the insertion point. Explain how to make the insertion point appear in a different text box when the program begins.
14. During program execution, you want to return the insertion point to a text box called AddressTextBox. What Basic statement will you use to make that happen?
15. What Basic statements will clear the current contents of a text box and a label?
16. How are the `With` and `End With` statements used? Give an example.
17. What is concatenation and when would it be useful?
18. Explain how to continue a very long Basic statement onto another line.

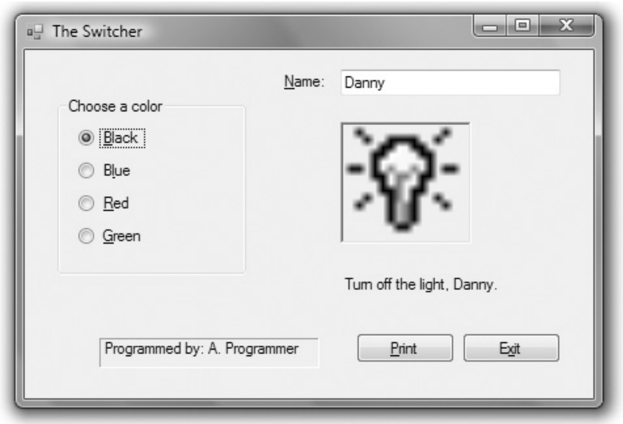
Programming Exercises

Graphics Files: You can find the icon files in the StudentData\Graphics folder on the text Web site (www.mhhe.com/VB2008).

2.1 Create a project that will switch a light bulb on and off, using the user interface shown below as a guide.
Form: Include a text box for the user to enter his or her name. Create two picture boxes, one on top of the other. Only one will be visible at a time. Use radio buttons to select the color of the text in the label beneath the light bulb picture box.
Include keyboard access keys for the radio buttons and the buttons. Make the *Exit* button the Cancel button. Create ToolTips for both light bulb picture boxes; make the ToolTips say "Click here to turn the light on or off."
Project Operation: The user will enter a name and click a radio button for the color (not necessarily in that order). When the light bulb is clicked, display the other picture box and change the message below it. Concatenate the user name to the end of the message.
The two icon files are Lightoff.ico and Lighton.ico and are found in the following folder:

```
Graphics\MicrosoftIcons
```

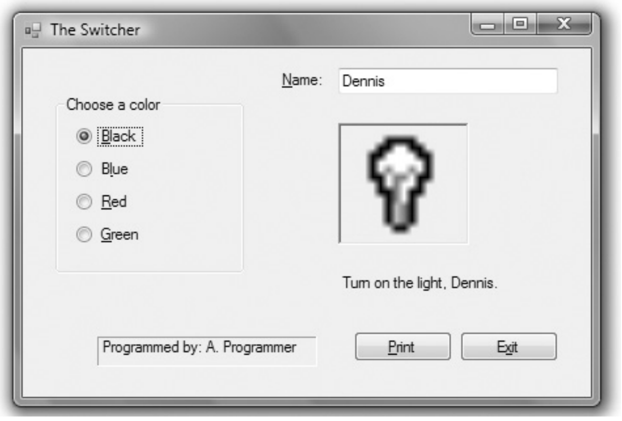
(See the note at the top of the exercises for graphic file locations.)
Coding: In the click event procedure for each Color radio button, change the color of the message below the light bulb.



Danny

Turn off the light, Danny.

Programmed by: A. Programmer



Dennis

Turn on the light, Dennis.

Programmed by: A. Programmer

2.2 Write a project to display the flags of four different countries, depending on the setting of the radio buttons. In addition, display the name of the country in the large label under the flag picture box. The user also can choose to display or hide the form's title, the country name, and the name of the programmer. Use check boxes for the display/hide choices.
Include keyboard access keys for all radio buttons, check boxes, and buttons. Make the *Exit* button the Cancel button. Include ToolTips.

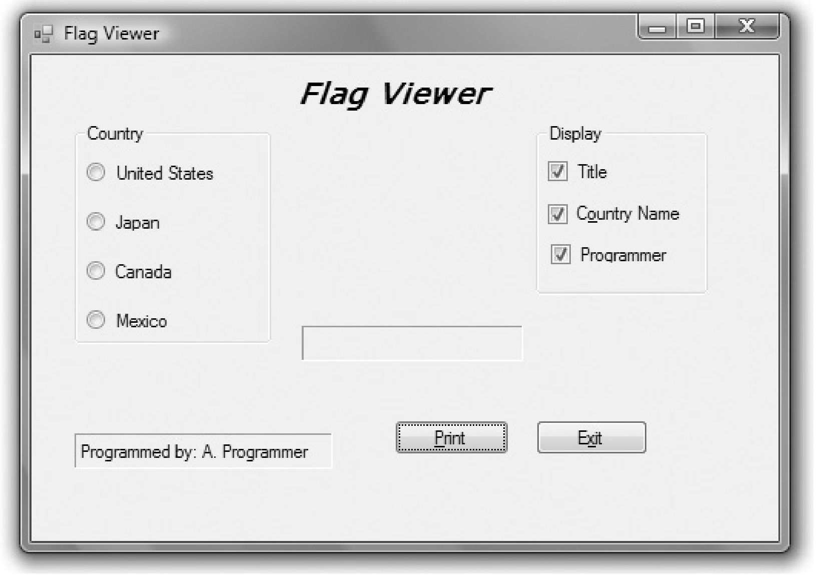
You can choose the countries and flags.

Hints: When a project begins running, the focus goes to the control with the lowest TabIndex. Because that control likely is a radio button, one button will appear selected. You must either display the first flag to match the radio button or make the focus begin in a different control. You might consider beginning the focus on the button.

Set the Visible property of a control to the Checked property of the corresponding check box. That way when the check box is selected, the control becomes visible.

Because all three selectable controls will be visible when the project begins, set the Checked property of the three check boxes to True at design time. Set the flag picture box to `Visible = False` so it won't appear at startup. (If you plan to display the picture box at startup, its Visible property must be set to True.)

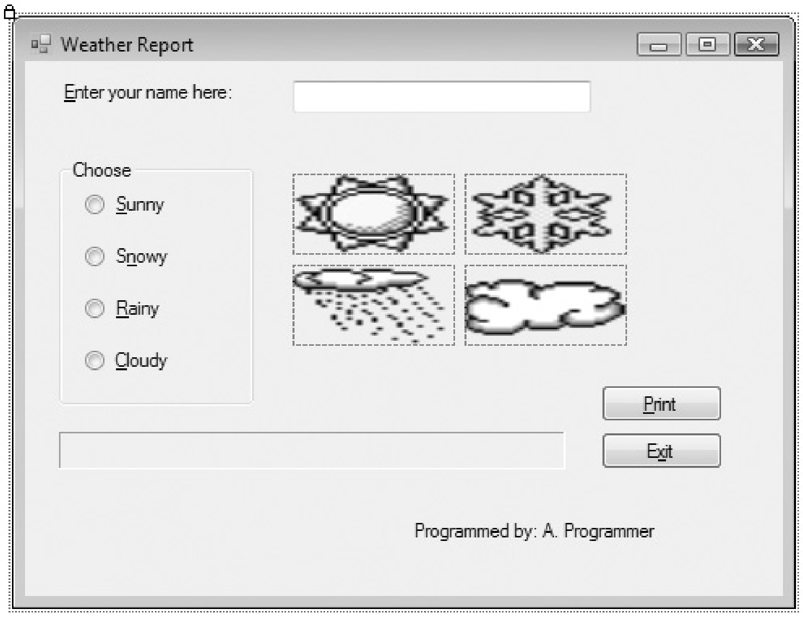
Make sure to set the SizeMode property of the picture box control to *StretchImage*.



2.3 Write a project to display a weather report. The user can choose one of the radio buttons and display an icon and a message. The message should give the weather report in words and include the person's name (taken from the text box at the top of the form). For example, if the user chooses the Sunny button, you might display "It looks like sunny weather today, John" (assuming that the user entered *John* in the text box).

Include keyboard access keys for the button and radio buttons. Make the *Exit* button the Cancel button and include ToolTips.

The four icons displayed are called Cloud.ico, Rain.ico, Snow.ico, and Sun.ico. (See the note at the top of the exercises for graphic file locations.)



2.4 Write a project that will input the user name and display a message of the day in a label, along with the user's name. Include buttons (with keyboard access keys) for *Display*, *Clear*, and *Exit*. Make the *Display* button the Accept button and the *Clear* button the Cancel button. Include ToolTips where appropriate.

Include a group of radio buttons for users to select the color of the message. Give them a choice of four different colors.

Make your form display a changeable picture box. You can use the happy face icon files or any other images you have available (Face01.ico, Face02.ico, and Face03.ico). (See the note at the top of the exercises for graphic file locations.)

You may choose to have only one message of the day, or you can have several that the user can select with radio buttons. You might want to choose messages that go with the different face icons.

2.5 Create a project that allows the user to input information and then display the lines of output for a mailing label.

Remember that fields to be input by the user require text boxes, but display the output in labels. Use text boxes for the first name, last name, street address, city, state, and ZIP code; give meaningful names to the text boxes and set the initial Text properties to blank. Add appropriate labels to each text box to tell the user which data will be entered into each box and also provide ToolTips.

Use buttons for *Display Label Info*, Clear, and *Exit*. Make the *Display* button the Accept button and the *Clear* button the Cancel button.

Use three labels for displaying the information for Line 1, Line 2, and Line 3.

Use a masked text box for the ZIP code.

A click event on the *Display Label Info* button will display the following:

Line 1—The first name and last name concatenated together with a space between.
Line 2—The street address.
Line 3—The city, state, and ZIP code concatenated together. (Make sure to concatenate a comma and a space between the city and state, using ", " and two spaces between the state and ZIP code.)

Case Studies

VB Mail Order

Design and code a project that displays shipping information.

Use an appropriate image in a picture box in the upper-left corner of the form.

Use text boxes with identifying labels for Catalog Code, Page Number, and Part Number.

Use two groups of radio buttons on the form; enclose each group in a group box. The first group box should have a Text property of *Shipping* and contain radio buttons for Express and Ground. Make the second group box have a Text property of *Payment Type* and include radio buttons for Charge, COD, and Money Order.

Use a check box for New Customer.

Add buttons for *Print*, *Clear*, and *Exit*. Make the *Clear* button the Cancel button.

Add ToolTips as appropriate.

VB Auto Center

Modify the project from the Chapter 1 VB Auto Center case study, replacing the buttons with images in picture boxes. (See “Copy and Move Projects” in Appendix C for help in making a copy of the Chapter 1 project to use for this project.) Above each picture box, place a label that indicates which department or command the graphic represents. A click on a picture box will produce the appropriate information in the special notices label.

Add an image in a picture box that clears the special notices label. Include a ToolTip for each picture box to help the user understand the purpose of the graphic.

Add radio buttons that will allow the user to view the special notices label in different colors.

Include a check box labeled Hours. When the check box is selected, a new label will display the message “Open 24 Hours—7 days a week”.

By default, the images are all stored in the Icons folder. (See the note at the top of the exercises for graphic file locations.)

Department/Command	**Image for Picture box**
Auto Sales	Cars.ico
Service Center	Wrench.ico
Detail Shop	Water.ico
Employment Opportunities	MAIL12.ico
Exit	MSGBOX01.ico

Video Bonanza

Design and code a project that displays the location of videos using radio buttons. Use a radio button for each of the movie categories and a label to display the aisle number. A check box will allow the user to display or hide a message for members. When the check box is selected, a message stating "All Members Receive a 10% Discount" will appear.

Include buttons (with keyboard access keys) for *Clear* and *Exit*. The *Clear* button should be set as the Accept button and the *Exit* as the Cancel button.

Place a label on the form in a 24-point font that reads "Video Bonanza". Use a line to separate the label from the rest of the interface. Include an image in a picture box.

Radio Button	Location
Comedy	Aisle 1
Drama	Aisle 2
Action	Aisle 3
Sci-Fi	Aisle 4
Horror	Aisle 5

Very Very Boards

Create a project to display an advertising screen for Very Very Boards. Include the company name, a slogan (use "The very best in boards" or make up your own slogan), and a graphic image for a logo. You may use the graphic included with the text materials (Skateboard.gif) or one of your own.

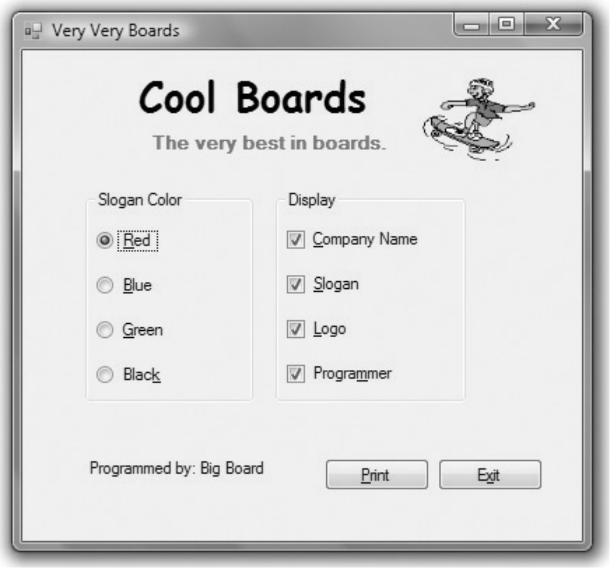
Allow the user to select the color for the slogan text using radio buttons. Additionally, the user may choose to display or hide the company name, the slogan, and the logo. Use check boxes for the display options so that the user can select each option independently.

Include keyboard access keys for the radio buttons and the buttons. Make the *Exit* button the Cancel button. Create ToolTips for the company name ("Our company name"), the slogan ("Our slogan"), and the logo ("Our logo").

When the project begins execution, the slogan text should be red and the Red radio button selected. When the user selects a new color, change the color of the slogan text to match.

Each of the check boxes must appear selected initially, since the company name, slogan, logo, and programmer name display when the form appears. Each time the user selects or deselects a check box, make the corresponding item display or hide.

Make the form appear in the center of the screen.


Very Very Boards
Cool Boards
The very best in boards.
Slogan Color
Red
Blue
Green
Black
Display
Company Name
Slogan
Logo
Programmer
Programmed by: Big Board
Print
Exit