Developing an Application in Visual Basic

It takes just a few minutes to build your first Visual Basic application. You create the user interface by "drawing" controls, such as text boxes and command buttons, on a form. Next, you set properties for the form and controls to specify such values as captions, color, and size. Finally, you write code to bring the application to life. The basic steps you take in creating your first application will show you principles that you'll use with every other application you develop.

This chapter provides an overview of the application development process, describes the terms and skills you need to use Visual Basic, and takes you step by step through several simple applications.

Note If you are using the Control Creation Edition of Visual Basic, some of the material covered in this document may not be applicable. With the full editions of Visual Basic you have the ability to create applications, ActiveX documents, and other types of components. Although some of the terminology may relate to application specific objects such as forms, in most cases the underlying concepts also apply to ActiveX control creation.

Contents

- Visual Basic Concepts
- Elements of the Integrated Development Environment
- Your First Visual Basic Application

Visual Basic Concepts

In order to understand the application development process, it is helpful to understand some of the key concepts upon which Visual Basic is built. Because Visual Basic is a Windows development language, some familiarity with the Windows environment is necessary. If you are new to Windows programming, you need to be aware of some fundamental differences between programming for Windows versus other environments.

How Windows Works: Windows, Events and Messages

A complete discussion of the inner workings of Windows would require an entire book. A deep understanding of all of the technical details isn't necessary. A simplified version of the workings of Windows involves three key concepts: windows, events and messages.

Think of a window as simply a rectangular region with its own boundaries. You are probably already aware of several different types of windows: an Explorer window in Windows 95, a document window within your word processing program, or a dialog box that pops up to remind you of an appointment. While these are the most common examples, there are actually many other types of windows. A command button is a window. Icons, text boxes, option buttons and menu bars are all windows.

The Microsoft Windows operating system manages all of these many windows by assigning each one a unique id number (window handle or hWnd). The system continually monitors each of these windows for signs of activity or events. Events can occur through user actions such as a mouse click or a key press, through programmatic control, or even as a result of another window's actions.

Each time an event occurs, it causes a message to be sent to the operating system. The system processes the message and broadcasts it to the other windows. Each window can then take the appropriate action based on its own instructions for dealing with that particular message (for example, repainting itself when it has been uncovered by another window).

As you might imagine, dealing with all of the possible combinations of windows, events and messages could be mind-boggling. Fortunately, Visual Basic insulates you from having to deal with all of the low-level message handling. Many of the messages are handled automatically by Visual Basic; others are exposed as Event procedures for your convenience. This allows you to quickly create powerful applications without having to deal with unnecessary details.

Understanding the Event-Driven Model

In traditional or "procedural" applications, the application itself controls which portions of code execute and in what sequence. Execution starts with the first line of code and follows a predefined path through the application, calling procedures as needed.

In an event-driven application, the code doesn't follow a predetermined path — it executes different code sections in response to events. Events can be triggered by the user's actions, by messages from the system or other applications, or even from the application itself. The sequence of these events determines the sequence in which the code executes, thus the path through the application's code differs each time the program runs.

Because you can't predict the sequence of events, your code must make certain assumptions about the "state of the world" when it executes. When you make assumptions (for example, that an entry field must contain a value before running a procedure to process that value), you should structure your application in such a way

as to make sure that the assumption will always be valid (for example, disabling the command button that starts the procedure until the entry field contains a value).

Your code can also trigger events during execution. For example, programmatically changing the text in a text box cause the text box's Change event to occur. This would cause the code (if any) contained in the Change event to execute. If you assumed that this event would only be triggered by user interaction, you might see unexpected results. It is for this reason that it is important to understand the event-driven model and keep it in mind when designing your application.

Interactive Development

The traditional application development process can be broken into three distinct steps: writing, compiling, and testing code. Unlike traditional languages, Visual Basic uses an interactive approach to development, blurring the distinction between the three steps.

With most languages, if you make a mistake in writing your code, the error is caught by the compiler when you start to compile your application. You must then find and fix the error and begin the compile cycle again, repeating the process for each error found. Visual Basic interprets your code as you enter it, catching and highlighting most syntax or spelling errors on the fly. It's almost like having an expert watching over your shoulder as you enter your code.

In addition to catching errors on the fly, Visual Basic also partially compiles the code as it is entered. When you are ready to run and test your application, there is only a brief delay to finish compiling. If the compiler finds an error, it is highlighted in your code. You can fix the error and continue compiling without having to start over.

Because of the interactive nature of Visual Basic, you'll find yourself running your application frequently as you develop it. This way you can test the effects of your code as you work rather than waiting to compile later.

Elements of the Integrated Development Environment

- The working environment in Visual Basic is often referred to as the integrated development environment or IDE because it integrates many different functions such as design, editing, compiling, and debugging within a common environment. In most traditional development tools, each of these functions would operate as a separate program, each with its own interface.

Starting the Visual Basic IDE

When you run the Visual Basic Setup program, it allows you to place the program items in an existing program group or create a new program group and new program

items for Visual Basic in Windows. You are then ready to start Visual Basic from Windows.

To start Visual Basic from Windows

1 Click **Start** on the Task bar

2 Select **Programs**, and then **Visual Basic 5.0**.

1– or –

2Click **Start** on the Task bar.

3Select **Programs**.

4Use the **Windows Explorer** to find the Visual Basic executable file (VB5.exe).

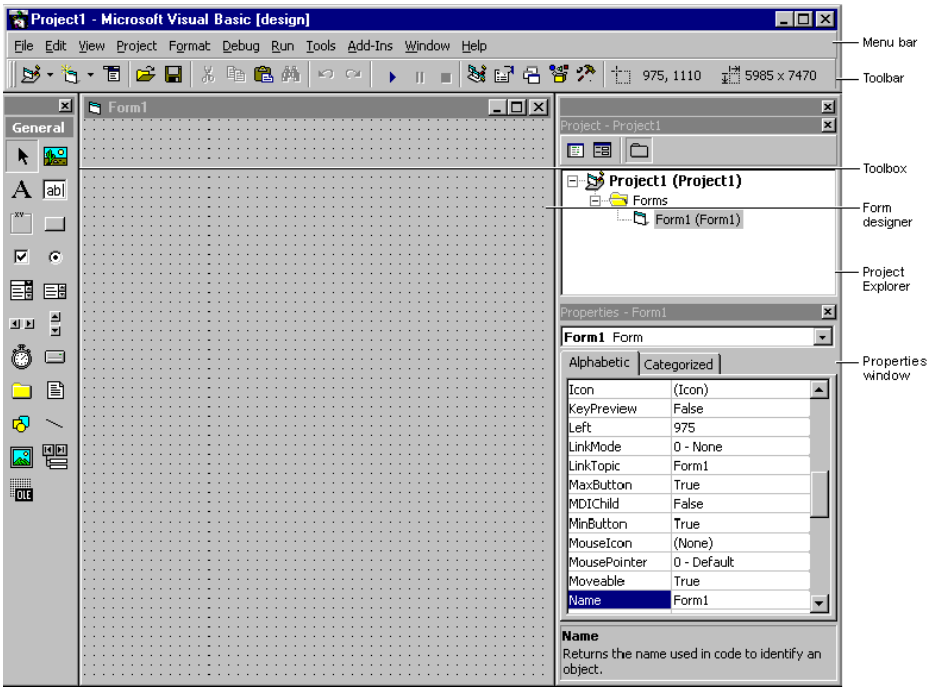
3 Double-click the Visual Basic icon.

2

You can also create a shortcut to Visual Basic, and double-click the shortcut.

When you first start Visual Basic, you see the interface of the integrated development environment, as shown in Figure 2.1.

Figure 2.1 The Visual Basic integrated development environment



3

Integrated Development Environment Elements

The Visual Basic integrated development environment (IDE) consists of the following elements.

Menu Bar

Displays the commands you use to work with Visual Basic. Besides the standard File, Edit, View, Window, and Help menus, menus are provided to access functions specific to programming such as Project, Format, or Debug.

Context Menus

Contain shortcuts to frequently performed actions. To open a context menu, click the right mouse button on the object you're using. The specific list of shortcuts available from context menus depends on the part of the environment where you click the right mouse button. For example, the context menu displayed when you right click on the Toolbox lets you display the Components dialog box, hide the Toolbox, dock or undock the Toolbox, or add a custom tab to the Toolbox.

Toolbars

Provide quick access to commonly used commands in the programming environment. You click a button on the toolbar once to carry out the action represented by that button. By default, the Standard toolbar is displayed when you start Visual Basic. Additional toolbars for editing, form design, and debugging can be toggled on or off from the Toolbars command on the View menu.

Toolbars can be docked beneath the menu bar or can "float" if you select the vertical bar on the left edge and drag it away from the menu bar.

Toolbox

Provides a set of tools that you use at design time to place controls on a form. In addition to the default toolbox layout, you can create your own custom layouts by selecting Add Tab from the context menu and adding controls to the resulting tab.

For More Information To learn more about specific controls, see "Forms, Controls, and Menus" and "Using Visual Basic's Standard Controls." For information on how to add controls to the Toolbox, see "Adding Controls to a Project" in "Managing Projects."

Project Explorer Window

Lists the forms and modules in your current project. A *project* is the collection of files you use to build an application.

For More Information For information on projects, see "Managing Projects."

Properties Window

Lists the property settings for the selected form or control. A *property* is a characteristic of an object, such as size, caption, or color.

For More Information For information on properties, see "Understanding Properties, Methods, and Events" in "Forms, Controls, and Menus."

Object Browser

Lists objects available for use in your project and gives you a quick way to navigate through your code. You can use the Object Browser to explore objects in Visual Basic and other applications, see what methods and properties are available for those objects, and paste code procedures into your application.

For More Information For more information on using the Object Browser to view procedures, see "Finding Out About Objects" in "Programming with Objects."

Form Designer

Serves as a window that you customize to design the interface of your application. You add controls, graphics, and pictures to a form to create the look you want. Each form in your application has its own form designer window.

Code Editor Window

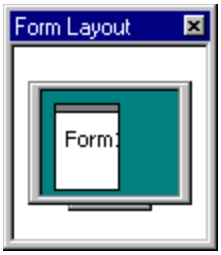
Serves as an editor for entering application code. A separate code editor window is created for each form or code module in your application.

For More Information To learn more about entering code and using the code editor, see "Programming Fundamentals."

Form Layout Window

The Form Layout window (Figure 2.2) allows you to position the forms in your application using a small graphical representation of the screen.

Figure 2.2 The Form Layout window



Immediate, Locals, and Watch Windows

These additional windows are provided for use in debugging your application. They are only available when you are running your application within the IDE.

For More Information To learn more about debugging and using the debug windows, see "Debugging Your Code and Handling Errors."

Note You can also add features to the Visual Basic interface by using a program called an *add-in*. Add-ins, which are available from Microsoft and third-party developers, can provide features like source code control, which allows you to support group development projects.

3

Environment Options

Visual Basic provides a great deal of flexibility, allowing you to configure the working environment to best suit your individual style. You can choose between a single or multiple document interface, and you can adjust the size and positioning of the various Integrated Development Environment (IDE) elements. Your layout will persist between sessions of Visual Basic.

SDI or MDI Interface

Two different styles are available for the Visual Basic IDE: single document interface (SDI) or multiple document interface (MDI). With the SDI option, all of the IDE windows are free to be moved anywhere on screen; as long as Visual Basic is the current application, they will remain on top of any other applications. With the MDI option, all of the IDE windows are contained within a single resizable parent window.

To switch between SDI and MDI modes

4 Select **Options** from the **Tools** menu.

5The **Options** dialog box is displayed.

5 Select the **Advanced** tab.

6 Check or uncheck the **SDI Development Environment** check box.

6The IDE will start in the selected mode the next time you start Visual Basic.

7– or –

8Run Visual Basic from the command line with a /sdi or /mdi parameter.

5

Docking Windows

Many of the windows in the IDE can be docked, or connected, to each other or to the edge of the screen. These include the Toolbox, Form Layout Window, Project Explorer, Properties window, Color Palette, and Immediate, Locals, and Watch windows.

With the MDI option, windows can be docked to any side of the parent window; with SDI they can only be docked beneath the menu bar. Docking capabilities can be toggled on or off for a given window by selecting the appropriate check box on the Docking tab of the Options dialog box, available from the Options command on the Tools menu.

To dock or undock a window

7 Select the window you wish to dock or undock.

9Drag the window to the desired location by holding down the left mouse button.

8 The outline of the window will be displayed as you drag.

9 Release the mouse button.

6

Your First Visual Basic Application

Creating an application in Visual Basic is simple. How simple? For the answer, try out the Hello, Visual Basic and Firstapp applications that follow.

Hello, Visual Basic

There are three main steps to creating an application in Visual Basic:

1. Create the interface.
2. Set properties.
3. Write code.

7

To see how this is done, use the steps in the following procedures to create a simple application that consists of a text box and a command button. When you click the command button, the message "Hello, world!" appears in the text box.

Creating the Interface

Forms are the foundation for creating the interface of an application. You can use forms to add windows and dialog boxes to your application. You can also use them as containers for items that are not a visible part of the application's interface. For example, you might have a form in your application that serves as a container for graphics that you plan to display in other forms.

The first step in building a Visual Basic application is to create the forms that will be the basis for your application's interface. Then you draw the objects that make up the interface on the forms you create. For this first application, you'll use two controls from the Toolbox.

Button	Control
	Text box
	Command button

4

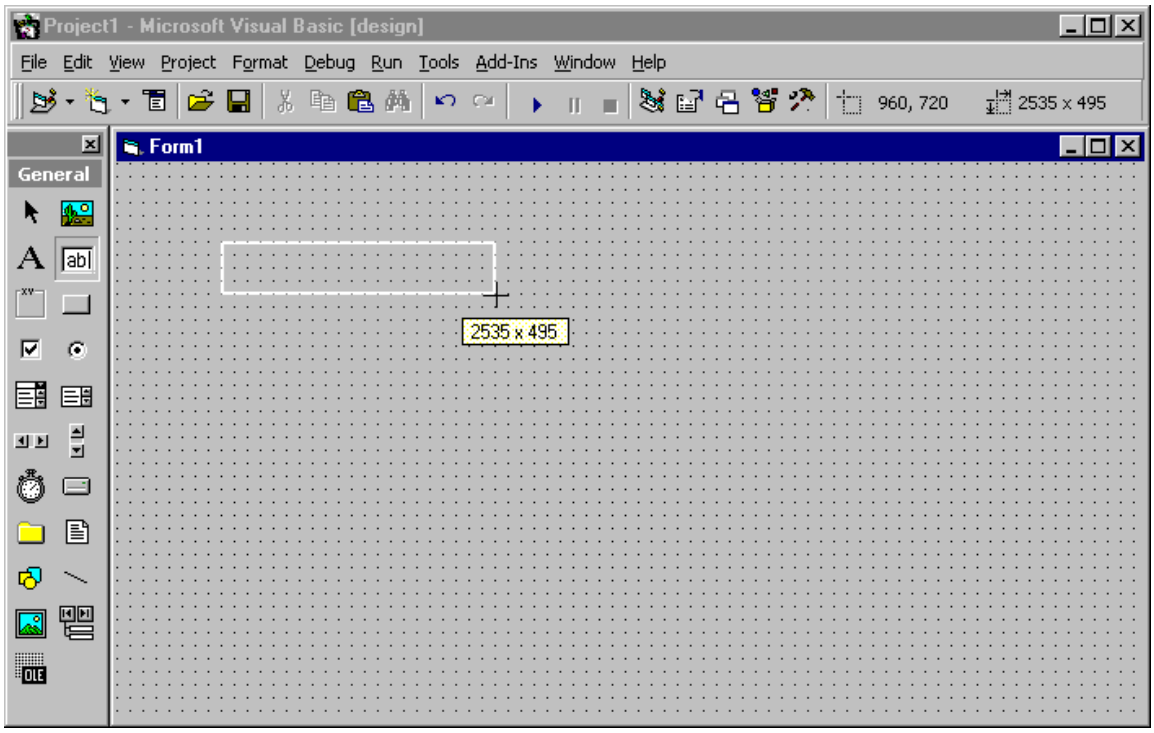
To draw a control using the Toolbox

10 Click the tool for the control you choose to draw — in this case, the **text box**.

11 Move the pointer onto your form. The pointer becomes a cross hair, as shown in Figure 2.3.

1Figure 2.3 Drawing a text box with the Toolbox

1



8

12 Place the cross hair where you want the upper-left corner of the control.

13 Drag the cross hair until the control is the size you want. (*Dragging* means holding the left mouse button down while you move an object with the mouse.)

14 Release the mouse button.

10The control appears on the form.

9

Another simple way to add a control to a form is to double-click the button for that control in the Toolbox. This creates a default-size control located in the center of the form; then you can move the control to another location on the form.

Resizing, Moving, and Locking Controls

Notice that small rectangular boxes called *sizing handles* appear at the corners of the control; you'll use these sizing handles in the next step as you resize the control. You can also use the mouse, keyboard, and menu commands to move controls, lock and unlock control positions, and adjust their positions.

To resize a control

15 Select the control you intend to resize by clicking it with the mouse.

11Sizing handles appear on the control.

16 Position the mouse pointer on a sizing handle, and drag it until the control is the size you choose.

12The corner handles resize controls horizontally and vertically, while the side handles resize in only one direction.

17 Release the mouse button.

13– or –

14Use SHIFT with the arrow keys to resize the selected control.

10

To move a control

- Use the mouse to drag the control to a new location on the form.

15– or –

16Use the Properties window to change the **Top** and **Left** properties.

11

When a control is selected, you can use CTRL with the arrow keys to move the control one grid unit at a time. If the grid is turned off, the control moves one pixel at a time.

To lock all control positions

- From the **Format** menu, choose **Lock Controls**.

17– or –

18Click the **Lock Controls Toggle** button on the **Form Editor** toolbar.

12

This will lock all controls on the form in their current positions so that you don't inadvertently move them once you have them in the desired location. This will lock controls only on the selected form; controls on other forms are untouched. This is a toggle command, so you can also use it to unlock control positions.

To adjust the position of locked controls

- You can "nudge" the control that has the focus by holding CTRL down and pressing the appropriate arrow key.

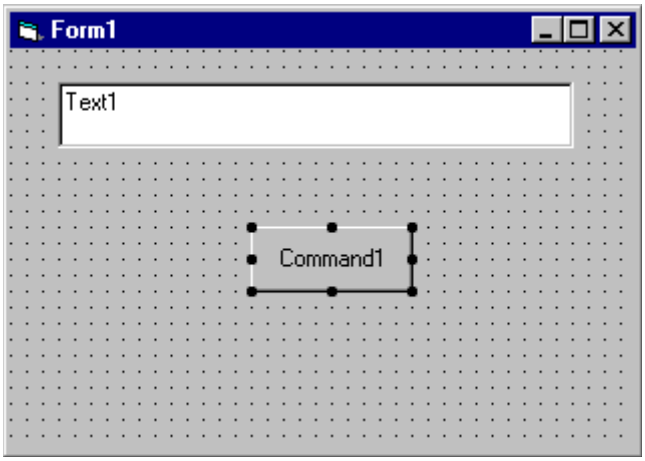
19– or –

20You can change the control's **Top** and **Left** properties in the Property window.

13

You now have the interface for the "Hello, world!" application, as shown in Figure 2.4.

Figure 2.4 The interface for the "Hello, world!" application

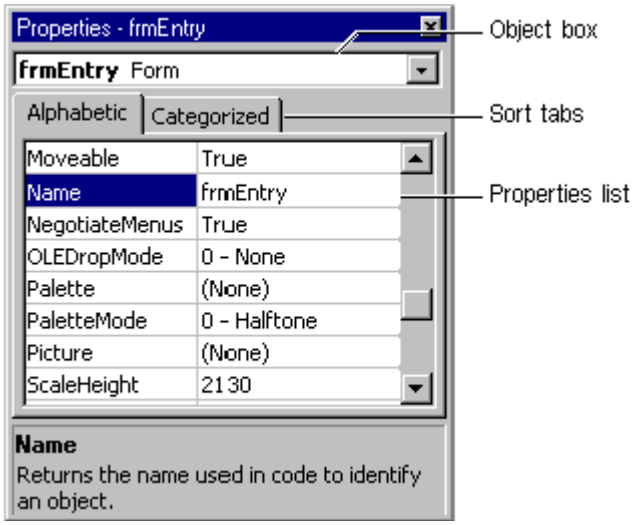


14

Setting Properties

The next step is to set properties for the objects you've created. The Properties window (Figure 2.5) provides an easy way to set properties for all objects on a form. To open the Properties window, choose the Properties Window command from the View menu, click the Properties Window button on the toolbar, or use the context menu for the control.

Figure 2.5 The Properties window



15

The Properties window consists of the following elements:

- Object box — Displays the name of the object for which you can set properties. Click the arrow to the right of the object box to display the list of objects for the current form.
- Sort tabs — Choose between an alphabetic listing of properties or a hierarchical view divided by logical categories, such as those dealing with appearance, fonts, or position.
- Properties list — The left column displays all of the properties for the selected object. You can edit and view settings in the right column.

16

To set properties from the Properties window

18 From the **View** menu, choose **Properties**, or click the **Properties** button on the toolbar.

21 The **Properties** window displays the settings for the selected form or control.

19 From the **Properties** list, select the name of a property.

20 In the right column, type or select the new property setting.

22 Enumerated properties have a predefined list of settings. You can display the list by clicking the down arrow at the right of the Settings box, or you can cycle through the list by double-clicking a list item.

17

For the "Hello, world!" example, you'll need to change three property settings. Use the default settings for all other properties.

Object	Property	Setting
Form	Caption	Hello, world!
Text box	Text	(Empty)
Command button	Caption	OK

5

Setting the Icon Property

All forms in Visual Basic have a generic, default icon that appears when you minimize that form. However, you will probably change this icon to one that illustrates the use of the form or your application. To assign an icon to a form, set the Icon property for that form. You can use 32 x 32 pixel icons that were standard in 16-bit versions of Microsoft Windows and are also used in Windows 95 and Windows NT, as well as the 16 x 16 pixel icons used in Windows 95.

Writing Code

The *Code Editor window* is where you write Visual Basic code for your application. Code consists of language statements, constants, and declarations. Using the Code Editor window, you can quickly view and edit any of the code in your application.

To open the Code window

- Double-click the form or control for which you choose to write code.

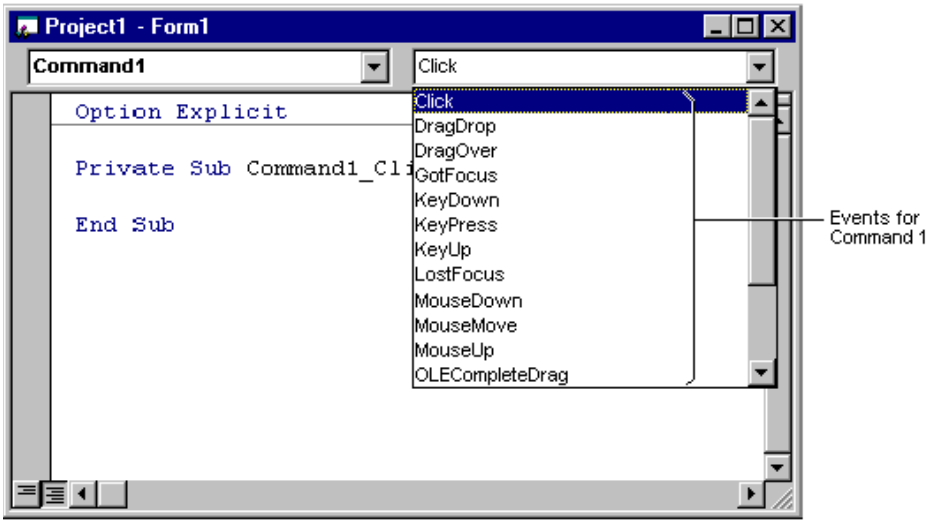
23– or –

24From the Project Explorer window, select the name of a form or module, and choose the **View Code** button.

18

Figure 2.6 shows the Code Editor window that appears when you double-click the Command button control, and the events for that command.

Figure 2.6 The Code Editor window



19

You can choose to display all procedures in the same Code window, or display a single procedure at a time.

To display all procedures in the same Code window

21 From the **Tools** menu, select the **Options** dialog box.

22 On the **Editor** tab in the **Options** dialog box, select the check box to the left of **Default to Full Module View**. The check box to the left of **Procedure Separator** adds or removes a separator line between procedures.

25– or –

26Click the **Full Module View** button in the lower left corner of the Code Editor window.

20

To display one procedure at a time in the Code window

23 From the **Tools** menu, select the **Options** dialog box.

24 On the **Editor** tab in the **Options** dialog box, clear the check box to the left of **Default to Full Module View**.

27– or –

28Click the **Procedure View** button in the lower left corner of the Code Editor window.

21

The Code window includes the following elements:

- Object list box — Displays the name of the selected object. Click the arrow to the right of the list box to display a list of all objects associated with the form.
- Procedure list box — Lists the procedures, or events, for an object. The box displays the name of the selected procedure — in this case, Click. Choose the arrow to the right of the box to display all the procedures for the object.

22

Creating Event Procedures

Code in a Visual Basic application is divided into smaller blocks called *procedures*. An *event procedure*, such as those you'll create here, contains code that is executed when an event occurs (such as when a user clicks a button). An event procedure for a control combines the control's actual name (specified in the Name property), an underscore (_), and the event name. For example, if you want a command button named Command1 to invoke an event procedure when it is clicked, use the procedure Command1_Click.

To create an event procedure

25 In the **Object** list box, select the name of an object in the active form. (The *active* form is the form that currently has the focus.)

29For this example, choose the command button, Command1.

26 In the **Procedure** list box, select the name of an event for the selected object.

30Here, the **Click** procedure is already selected, because it's the default procedure for a command button. Note that a *template* for the event procedure is now displayed in the Code window.

27 Type the following code between the **Sub** and **End Sub** statements:

```
1Text1.Text = "Hello, world!"
```

23

31The event procedure should look like this:

```
2Private Sub Command1_Click ()
3    Text1.Text = "Hello, world!"
4End Sub
```

24

You'll note here that the code is simply changing the Text property of the control named Text1 to read "Hello, world!" The syntax for this example takes the form of *object.property,* where Text1 is the object and Text is the property. You can use this syntax to change property settings for any form or control in response to events that occur while your application is running.

For More Information For information on creating other types of procedures, see "Introduction to Procedures" in "Programming Fundamentals."

Running the Application

To run the application, choose Start from the Run menu, or click the Start button on the toolbar, or press F5. Click the command button you've created on the form, and you'll see "Hello, world!" displayed in the text box.