The series of step by step procedures in this chapter builds a simple ActiveX control called ShapeLabel. Although the control itself is not very interesting, building it will quickly demonstrate the major events in the life of an ActiveX control, introduce you to the intricacies of running code at design time, and show the basic steps for creating and hooking up a property page.

All of the subjects introduced in these procedures are covered in greater depth in later chapters. References to in-depth material will be found in each procedure. In addition, "Building ActiveX Controls," shows how you can use the Control Creation Wizard to make building controls even easier.

The procedures for creating the ShapeLabel control build on each other, so the sequence in which you perform the procedures is important.

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

Sample Application: CtlPlus.vbg

Fills in all the properties, methods, and events required to make ShapeLabel a functional control. Expands on the material in this chapter, showing additional control creation features, at the expense of some of the basic material covered in the step by step procedures. If you installed the sample applications, you will find them in the \CompTool\ActvComp subdirectory of the Visual Basic samples directory (\Vb\Samples\CompTool\ActvComp).

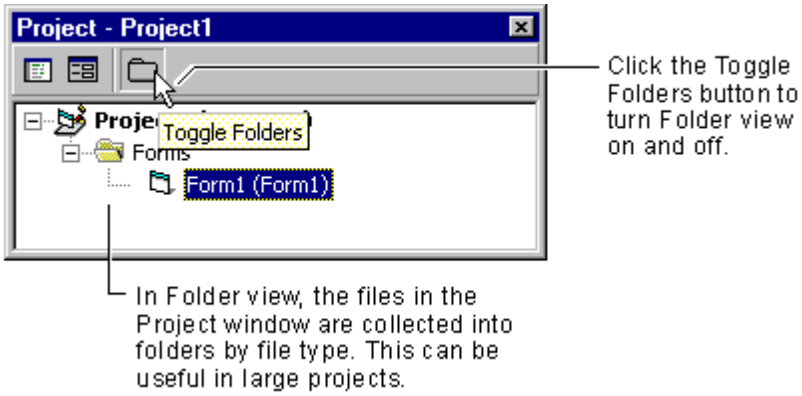
These procedures will be easier to follow if you set up your Visual Basic development environment to show the necessary windows.

Before You Begin

1 On the **View** menu, click **Toolbox** to open the Toolbox.

2 On the **View** menu, click **Project Explorer** to open the Project window. The Project window will be used extensively to switch between project files.

3 If the Project window is in Folder view, as shown below, click the **Toggle Folders** button on the Project window toolbar to turn the folders off.



1

1

4 On the **View** menu, click **Properties** window to open the **Properties** window.

5 On the **View** menu, click **Immediate** window to open the **Immediate** window. You will need this window open at design time, in order to demonstrate the control's code running at design time.

6 On the **Tools** menu, click **Options** to open the **Options** dialog box.

1Select the **Editor** tab, and make sure the **Require Variable Declaration** check box is selected. This makes it much easier to catch typing errors.

2Select the **Environment** tab. Make sure **Prompt To Save Changes** is checked, then click **OK**. This will make it easy to save the changes to the project as you go along.

2
2

Creating the ControlDemo Project

ActiveX controls can be added to any project type. When a control is compiled as part of an .exe file, however, it cannot be shared with other applications. The ShapeLabel control will be compiled into an .ocx file in a later procedure in this chapter, so it can be shared. Thus the ControlDemo project will be created as an ActiveX control project.

An ActiveX control project can contain as many controls as you like. When you build the project, the resulting .ocx file contains all the controls you've added.

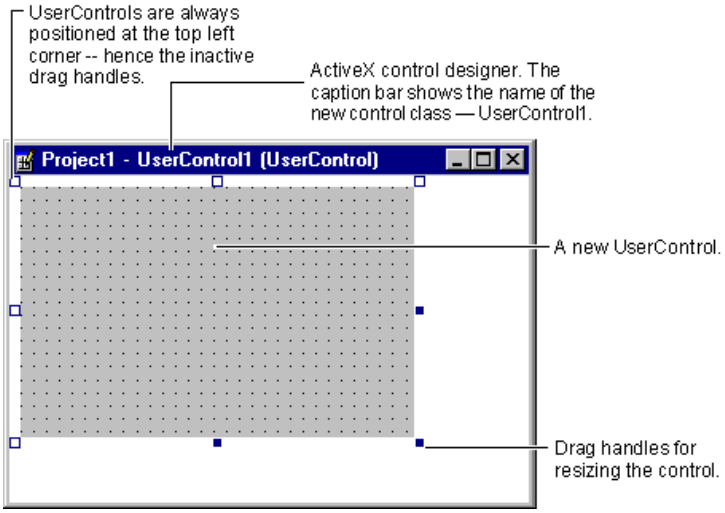
Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, “Creating an ActiveX Control.”

3

To create the ControlDemo project

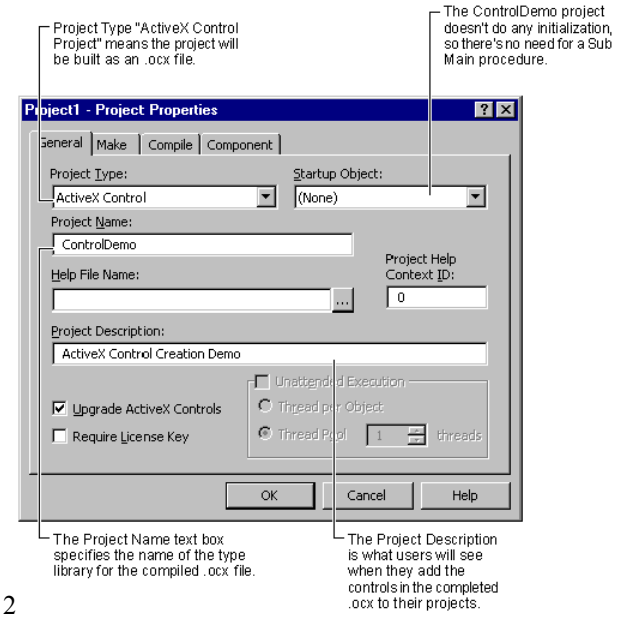
7 On the **File** menu, click **New Project** to open the **New Project** dialog box. (This will close your current project or project group; you will be prompted to save any changes you have made.) Double-click the **ActiveX Control Project** icon to create a new project.

3Visual Basic automatically adds a UserControl designer to the project. The default name, UserControl1, appears as the caption of the designer.



3

8 On the **Project** menu, click **Project1 Properties** to open the **Project Properties** dialog box. Select the **General** tab, fill out the information shown below, and then click **OK**.



2

4

9 Double-click **UserControl1** in the **Project** window to bring the designer to the front.

10 In the **Properties** window, double-click the **Name** property and change the name of the user control to ShapeLabel. The new name appears in the caption of the designer and in the Project window.

4The name you specify becomes the class name of your control, just as CommandButton is the class name for a command button. “Building ActiveX Controls” provides guidelines for choosing class names for controls.

5Notice that the **Properties** window looks much as it would for a Visual Basic form. Some properties you’re used to seeing are missing, however, and there are properties not found on ordinary Visual Basic forms. These properties are discussed in “Building ActiveX Controls.”

11 Within the control designer, resize the control using the drag handle at the lower right corner of the control, dragging up and left to make the control smaller.

6This sets the default size of the control. For convenience in later procedures, the ShapeLabel control should be of modest size.

12 On the **File** menu, click **Save Project** to save the project files. Name them as shown in the following table. Visual Basic will provide the indicated extensions automatically.

5

File	Filename	Extension
User control	ControlDemo_ShapeLabel	.ctl
Project	ControlDemo	.vbp

7Binary information in a control — such as bitmaps — will be saved in a binary file with the same name and an extension of .ctx.

1**Important** You must save the control project in order to refer to it from a test project. An unsaved component project cannot be referenced by another project.

For More Information See "Project Options for Control Components" and "Debugging Controls," in "Building ActiveX Controls."

Adding the TestCtlDemo Project

In order to test the ShapeLabel control, you need a test form. You can't just add a test form to ControlDemo and then run the project, because an .ocx project can't run all by itself. (This would be like running an .ocx file all by itself.)

To allow debugging of in-process components, Visual Basic allows you to load two or more projects into a *project group*. In addition to enabling in-process debugging, the project group makes it easier to load your .ocx project and test project.

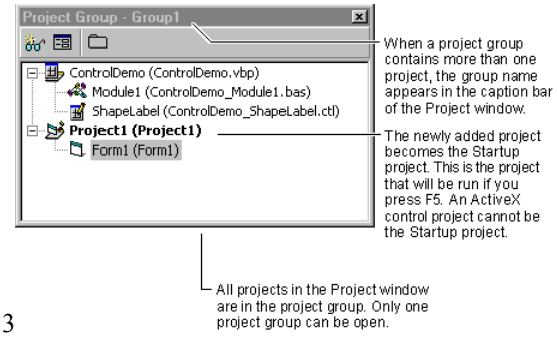
Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

To add a test project to the project group

13 On the **File** menu, click **Add Project to Group** to open the **Add Project** dialog box.

2**Important** Do not click **Open Project** or **New Project**, as these will close your control project.

14 Double-click the **EXE Project** icon to add an ordinary .exe project. You can now see both projects in the Project window, and the caption of the Project window shows the default project group name.



8 The new project immediately becomes the Startup project for the project group. The Project window identifies the Startup project by displaying its name in bold type. An ActiveX control project, like ControlDemo, cannot be the Startup project.

15 On the **File** menu, click **Save Project Group** to save the test project and the project group. Name the files as shown below. Visual Basic will provide the indicated extensions automatically.

File	Filename	Extension
Form	TestCtlDemo_Form1	.frm
Project	TestCtlDemo	.vbp
Project group	ControlDemo	.vbg

For More Information Test projects for ActiveX controls are discussed in more detail in "Debugging Controls," in "Building ActiveX Controls."

Running the ShapeLabel Control at Design Time

Unlike other programmable objects, controls have both design-time and run-time behavior. That is, some of the code in your control will execute when a developer places an instance of the control on a form at design time.

For example, the code you place in the UserControl_Resize event procedure will be executed both at design time and at run time.

In order to debug the design-time behavior of your control, you must be able to execute code in the control while the test form on which you place the control remains in design mode.

The following two procedures demonstrate this neat trick. In the first procedure, you'll add code to the Resize event of the ShapeLabel control. In the second procedure, you'll put part of ControlDemo into run mode — while the test project remains in design mode — and then add an instance of the ShapeLabel control to a form in the test project.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

To add code to the Resize event

16 In the Project window, double-click ShapeLabel to make it the active designer.

17 Double-click the ShapeLabel control to open the code window.

18 In the Procedure box, click the Resize event to go to its event procedure. Add the following code:

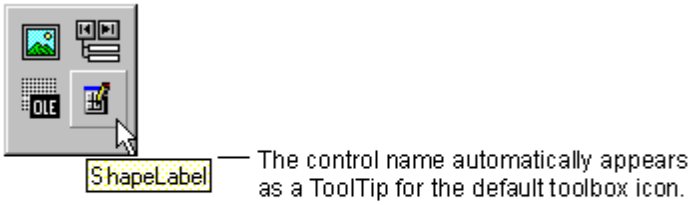
```
Private Sub UserControl_Resize()
    Static intCt As Integer
    intCt = intCt + 1
    Debug.Print "Resize " & intCt
End Sub
```

Note The name of the event procedure has the prefix “UserControl,” just as the Form_Resize event procedure for an ordinary form has the prefix “Form.”

In developing an ordinary Visual Basic application, you would now click the Start button on the toolbar, or press F5, to run your application. In order to put a ShapeLabel control on Form1, however, you have to run just the code for the control, leaving everything else in design mode.

To run the ShapeLabel control at design time

19 Click the **Close** button on the ShapeLabel designer to put the control into run mode. The default toolbox icon for a user control appears in the toolbox.



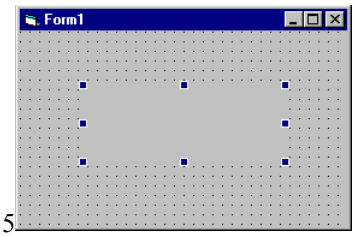
4

Important Don’t click the **Start** button on the toolbar, or press F5, because this would put the entire project group into run mode, and you would be unable to add the new control to a form.

4 If the default toolbox icon doesn’t appear on the toolbox, repeat Step 3 of “Adding the Test Project,” making sure ControlDemo is checked in the **Components** dialog box.

20 In the Project window, double-click Form1 to bring it to the front.

21 Double-click the **ShapeLabel** icon to add a ShapeLabel control to Form1. The control appears as a flat gray rectangle with grab handles:



Important If you get an error message, make sure you saved ControlDemo at the end of the procedure, “Creating the ControlDemo Project.” *You must save a control project* in order to use its controls in a test project.

In the **Properties** window you can see the default properties for a new control. The ShapeLabel control you just added to the form has been given a default name, ShapeLabel1.

Note Naming your control when you begin designing it avoids confusion. Suppose you place a control with a default name, such as UserControl1, on a form. Automatic numbering of new controls would append a number to the control name, resulting in a confusing name like UserControl11.

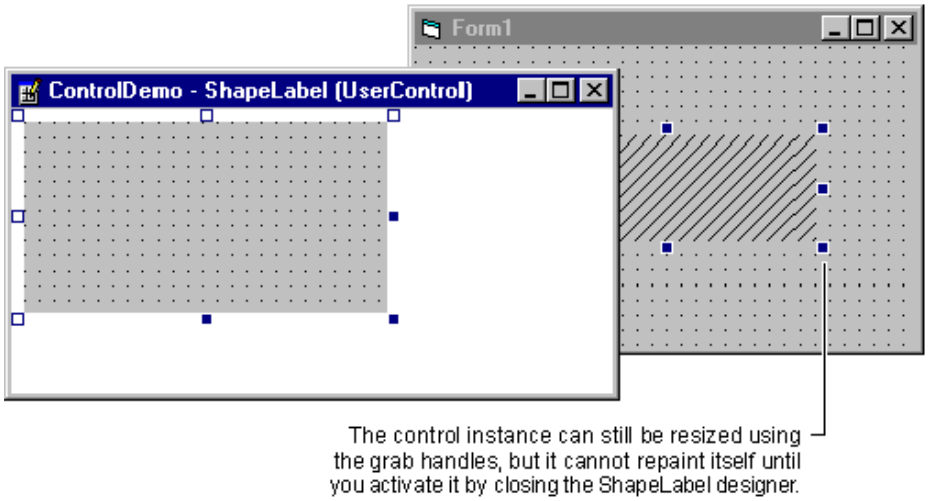
The ShapeLabel control’s Resize event occurred when it was placed on the form, as you can see by looking at the **Immediate** window. Use the grab handles to resize the control several times. Each time you resize it, the Resize event occurs again.

If you simply move the control around the form, the Resize event does not occur.

On Form1, double-click the **ShapeLabel** control to open the code window for Form1. The cursor will be on the default event procedure, ShapeLabel1_GotFocus. You can use the **Procedure** box to view the other three events Visual Basic automatically provides for your control. Close the code window when you are done.

In the Project window, double-click **ShapeLabel** to open the ShapeLabel designer. Notice that the ShapeLabel control you placed on Form1 is shaded with hatch marks to indicate that it is inactive.

6



17

11 Opening a control's designer makes all instances of the control inactive. Changing the code in the control's code window may also make control instances inactive.

25 Code in ShapeLabel's code module cannot be executed while the designer is open. Use the grab handles to resize the shaded ShapeLabel control on Form1. The Resize event doesn't fire, so no new messages appear in the **Immediate** window.

26 On the designer for the ShapeLabel control, click the **Close** button to reactivate the control instance. The shading disappears from the control on Form1, indicating that the instance is active again.

12 If the control has become inactive because of changes to its code, you can right-click the test form to bring up its context menu, and click **Update UserControls** to reactivate control instances.

18

Note Due to the number of windows required by these procedures, you may frequently find that ShapeLabel's designer has disappeared behind another form. You can double-click **ShapeLabel** in the Project window to bring the designer to the front.

9

For More Information More information about running code at design time can be found in "Debugging Controls," in "Building ActiveX Controls."

Life and Times of a UserControl Object

The life of an ordinary Visual Basic form is marked by certain key events, such as Initialize, Load, QueryUnload, and Unload. In order to create well-behaved applications, it's important to know when these events occur in the life cycle of a form.

The same is true for controls. The key events in the life cycle of a UserControl are Initialize, InitProperties, ReadProperties, WriteProperties, and Terminate. The following procedure explores these events.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

10

To observe key events for ShapeLabel

27 In the Project window, double-click **ShapeLabel** to open its designer.

28 Double-click the designer to open a code window for ShapeLabel, and enter code in the following event procedures:

```
6Private Sub UserControl_Initialize()
7    Debug.Print "Initialize"
8End Sub
9
10Private Sub UserControl_InitProperties()
11   Debug.Print "InitProperties"
12End Sub
13
14Private Sub UserControl_ReadProperties(PropBag As _
15        PropertyBag)
16   Debug.Print "ReadProperties"
17End Sub
18
19Private Sub UserControl_WriteProperties(PropBag _
20        As PropertyBag)
21   Debug.Print "WriteProperties"
22End Sub
23
24Private Sub UserControl_Terminate()
25   Debug.Print "Terminate"
26End Sub
```

19

3**Note** For UserControl objects, Load and Unload are superseded by the ReadProperties and WriteProperties events. This is discussed in more detail in "Understanding Control Lifetime and Key Events," in "Building ActiveX Controls."

20

29 On the ShapeLabel designer, click the **Close** button to put the control in run mode. Debug messages will appear in the **Immediate** window:

7

21

13 What's going on here? You haven't put another instance of the ShapeLabel control on Form1. Where did all these events come from?

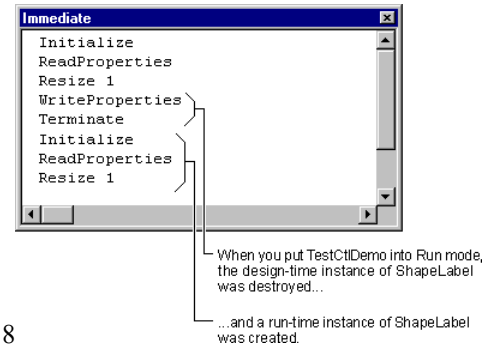
14 This illustrates an important point about controls. A user puts a control on a form, and thereafter thinks of the control as a permanent fixture of the form. From the control developer's perspective, however, controls are getting destroyed and re-created all the time.

15 When you put ShapeLabel in run mode by closing its designer, the instance of ShapeLabel on Form1 was destroyed and re-created, at which point it received an Initialize event. Why didn't you see a Terminate event first? Because the original instance of ShapeLabel you placed on Form1 was created before you added the code in the UserControl_Terminate event procedure! Welcome to the wild and woolly world of control creation.

4 **Note** Control instances are also destroyed and recreated when you click **Update UserControls** on the form's context menu.

22

30 Press F5, or click the **Start** button on the toolbar, to run TestCtlDemo. When the project is running, the grid on Form1 is gone, so you can't see the ShapeLabel, but you can see its life flash before your eyes in the **Immediate** window:



8

23

16 After a control instance is created, the ReadProperties event gives you a chance to obtain the control's saved property values from the .frm file belonging to the form that contains the control instance.

17 When the design-time instance of the control is destroyed, the WriteProperties event gives you a chance to save the property values the user set at design time. Property values are saved in the containing form's .frm file, as you'll see in "Saving the ShapeLabel Control's Property Values," later in this chapter.

18The Terminate event occurs when the control is being destroyed.

31 On Form1, click the **Close** button to return to design mode. In the **Immediate** window, you'll see a Terminate event (but not WriteProperties—why not?) as the run-time instance of ShapeLabel is torn down. Then you'll see the Initialize, ReadProperties, and Resize events, as the design-time instance of the control is created.

19The run-time instance of a control never gets a WriteProperties event, because it doesn't need to save its property values. To see why not, consider ShapeLabel's future. When it's compiled into an .ocx file, you'll add it to another project, put an instance on a form, compile the project into an .exe, and run it. When you close that .exe, the only place the ShapeLabel instance could save its property values would be in the .exe file. This sort of behavior is not tolerated by well-behaved operating systems.

32 Scroll to the top of the **Immediate** window, click in the top left corner, and drag to select all the text in the window. Press the DELETE key to clear the window.

33 In the Project window, double-click **Form1** to bring Form1 to the front.

34 On the **Toolbox**, double-click the **ShapeLabel** icon to add another instance of the control to Form1. You'll see a new event this time.

9

24

20When a new instance of your control is placed on a container, it gets an InitProperties event. In the UserControl_InitProperties event procedure you can place code to:

- Set the default values for each of the control's properties values
- Perform tasks whenever a user creates an instance of your control.

25

35 Close the Form1 designer by clicking its **Close** button. In the **Immediate** window, you will see two sets of WriteProperties and Terminate events, one for each instance of ShapeLabel.

36 In the Project window, double-click **Form1** to open its designer again. When the designer opens, all the controls on Form1 are created, and their Initialize events are fired. All controls then receive ReadProperties events, which allow them to retrieve their saved property values. The InitProperties event does not occur, because both instances of the ShapeLabel control already exist.

26

For More Information Control lifetime, and key events therein, are discussed in "Understanding Control Lifetime and Key Events," in "Building ActiveX Controls." "Exposing Properties of Constituent Controls," in the same chapter, explains how the ActiveX Control Wizard simplifies the creation of code to save and retrieve property values.

Drawing the ShapeLabel Control

You can use graphics methods, such as Circle and Line, to draw your control, or you can create your control's appearance using existing ActiveX controls and Visual Basic intrinsic controls. Controls you add to the UserControl to create its appearance are called *constituent controls.*

As its name suggests, ShapeLabel's appearance is created using a Shape control and a Label control.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

11

To add constituent controls to the ShapeLabel control

37 In the Project window, double-click **ShapeLabel** to open its designer.

38 In the **Toolbox**, double-click the **Visual Basic Shape** control to place a Shape control on the ShapeLabel designer. If you haven't used the Shape control before, hold the mouse over the Toolbox buttons until you find the one whose ToolTip is "Shape."

39 In the **Properties** window, set the following property values for the Shape control:

Property	Value
BorderStyle	0 - Transparent
FillColor	&H000000FF (Red)
FillStyle	0 - Solid
Name	shpBack
Shape	2 - Oval

12

5**Note** To set color properties such as FillColor and ForeColor to specific colors, select the **Palette** tab of the color selection dialog.

27

40 In the **Toolbox**, double-click the **Label** control to add a label on top of the Shape control. In the **Properties** window, set the following property values for the Label control:

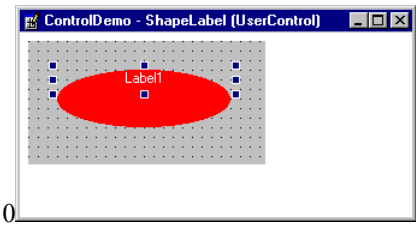
Property	Value
Alignment	2 - Center
BackStyle	0 - Transparent
ForeColor	&H00FFFFFF (White)

—13

Name lblCaption

13

41 Use the bottom grab handle to change the height of the label so that it is slightly taller than the text it contains. ShapeLabel should look something like this:



10

28

42 Double-click the **ShapeLabel** designer to bring the code window to the front, and replace the code in the UserControl_Resize event procedure with the following:

```
27Private Sub UserControl_Resize()
28  ' Size the Shape control to fill ShapeLabel's
29  '   visible surface area.
30  shpBack.Move 0, 0, ScaleWidth, ScaleHeight
31  ' Center the Label control vertically, and
32  '   make it the same width as ShapeLabel.
33  lblCaption.Move 0, (ScaleHeight _
34         - lblCaption.Height) / 2, ScaleWidth
35End Sub
```

29

43 When you're designing a user control, remember that the area you have to work with is bounded by the ScaleWidth and ScaleHeight of the control. Nothing outside this is visible to the user of your control. Furthermore, the size of the client area will change at the whim of the user. The Resize event is thus one of the most important events in control design.

44 Close the ShapeLabel designer by clicking its Close button, putting ShapeLabel in run mode. In the Project window, double-click Form1 to bring it to the front.

45 The two ShapeLabel controls should now appear as red ovals, with centered white captions that read, "Label1." Resize the ShapeLabels to test the Resize event code.

30

For More Information See "Drawing Your Control" in "Building ActiveX Controls."

Saving the ShapeLabel Control's Property Values

You can add properties and methods to an ActiveX control in the same way you add them to class modules: by creating Public procedures. Since ShapeLabel is going to be an enhanced label control, it makes sense for it to have a Caption property.

The following procedure adds a Caption property, and the support code to save and retrieve the property value. A control's property values are saved along with the other data that describes the container—in this case, Form1.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

14

To add a Caption property to the ShapeLabel control

46 In the Project window, double-click **ShapeLabel** to open its designer, then double-click on **ShapeLabel** to bring its code window to the front.

47 On the **Tools** menu, click **Add Procedure** to open the **Insert Procedure** dialog box. In the **Name** box, enter the name **Caption**. Click **Property** and **Public**, then click **OK**.

48 In the Code window, change the newly created property procedures to appear as follows:

```
36Public Property Get Caption() As String
37  Caption = lblCaption.Caption
38End Property
39
40Public Property Let Caption(NewCaption As String)
41  lblCaption.Caption = NewCaption
42End Property
```

31

6**Note** Be careful to change both property declaration lines by adding **As String**, as shown above. Property Get and Property Let declarations must match. Using specific type names speeds up execution, and provides type checking for the user of your control.

32

21The Property Let procedure is executed whenever a new value is assigned to the Caption property. It stores the new value directly in the Caption property of the lblCaption label on ShapeLabel.

22The Property Get procedure is executed whenever the property value is retrieved. It reads the value stored in the Caption property of the lblCaption label.

23Property procedures are discussed in "Adding Properties to a Class," in "Programming with Objects."

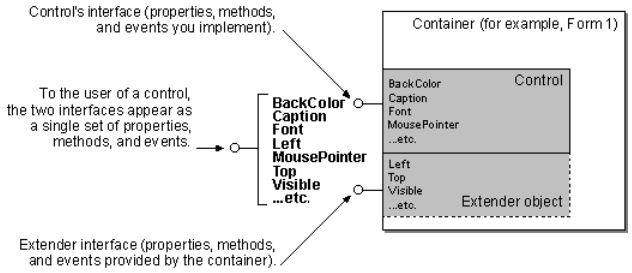
49 To initialize the Caption property, add the following code to the UserControl_InitProperties event procedure:

```
43Private Sub UserControl_InitProperties()
44  ' Let the starting value for the Caption
45  '   property be the Name given to this
46  '   instance of ShapeLabel.
47  Caption = Extender.Name
```

```
48  Debug.Print "InitProperties"
49End Sub
```

33

24What is this Extender object? To the user of a control, *extender properties* — such as Name, Top, and Left — appear to be part of your control. However, extender properties are really provided by the container your control is placed on. The Extender object of the UserControl gives you, the control designer, access to these properties from within your control.



11

34

25The read-only Name property of the Extender object returns the name the container (or the user) gives to a specific instance of your control. Using this name (for example, ShapeLabel1) as the initial value of the Caption property mimics the behavior of the Label control.

1**Tip** If your control imitates the behavior of controls that provide similar functionality, using it will be more intuitive.

35

26What would happen if you created a Name property for your control? You would be able to access it from within your control, but the only Name property your user would see would be the Name property of the Extender object.

27This introduces a recurrent theme for controls: The container determines a large portion of your control's behavior and appearance. It's the container that determines your control's Name, and your Top and Left properties are maintained relative to the container's coordinates. This theme will be taken up again in "Building ActiveX Controls."

28One last item of business: Why put this code in the InitProperties event? Why not use the Initialize event? As you have seen, Initialize is called every time the control instance is created, which happens often. InitProperties happens only when the user places the control on the container. This makes it the appropriate place to set initial values for a control instance.

29In addition, the UserControl object's Extender and Ambient objects are not yet available when the Initialize event occurs. "Understanding Control Lifetime and Key Events," in "Building ActiveX Controls," discusses appropriate uses of the Initialize event.

50 To save the value of your Caption property, add the following code to the UserControl_WriteProperties event procedure:

```
50Private Sub UserControl_WriteProperties(PropBag As _
51        PropertyBag)
52  Debug.Print "WriteProperties"
53  PropBag.WriteProperty "Caption", Caption, _
54        Extender.Name
55End Sub
```

36

30The PropertyBag is just what its name implies, a “bag” in which property values are saved. The bag is provided by the container. You can’t see into it, and you have no idea where or how the data is saved. All you can do is put values in and take them out.

31The first argument of the WriteProperty method is the name of the property, which will be used as the retrieval key. You should use the name of the property for this argument, because it will appear in the .frm text file (in Visual Basic—other containers may use other file names to save project data), and may be seen by the user of the control.

32The second argument is the value. A property value is saved as a Variant.

33The third argument, oddly enough, is a default value. Why provide a default when saving the property value? Before saving the value, the WriteProperty method compares the property value with this default. If they are the same, the property value doesn’t have to be saved, because default values will be set automatically when the control is reloaded. This keeps the .frm file from being cluttered with hundreds of default entries, a great favor to your users!

51 Place the following code in the ReadProperties event, to retrieve the persisted property value for the Caption property:

```
56Private Sub UserControl_ReadProperties(PropBag As _
57        PropertyBag)
58  Debug.Print "ReadProperties"
59  Caption = PropBag.ReadProperty("Caption", _
60        Extender.Name)
61End Sub
```

37

34The second argument of the ReadProperty method is a default value to be used if no value has been saved, if the user has deleted the property from the text file, or if the value has never been changed from the default, and therefore never saved by WriteProperty.

52 Click the **Close** button on the ShapeLabel designer, to put ShapeLabel into run mode. Like magic, the captions of the ShapeLabel controls change to match the default names of the two instances, ShapeLabel1 and ShapeLabel2.

35Use the **Properties** window to change the Caption properties of the two ShapeLabel controls on Form1, then click the **Close** button on the Form1 designer. In the Project window, double-click **Form1** to re-open the Form1 designer.

36From the messages in the **Immediate** window, you can see that the controls have been destroyed and re-created, but the values of the Caption properties have been saved and retrieved.

53 Press F5 to run TestCtlDemo, the Startup project of the project group, and observe the run-time behavior of the ShapeLabel control.

54 Click the **Close** button on Form1 to return to design mode.

38

For More Information Details of saving and retrieving property values can be found in "Adding Properties to Controls," in "Building ActiveX Controls." "Exposing Properties of Constituent Controls," in the same chapter, explains how the ActiveX Control Interface Wizard simplifies the creation of code to save and retrieve property values.

Giving the ShapeLabel Control a Property Page

Simple properties that you create using property procedures will be shown automatically in the Visual Basic Properties window. You can also connect your control to *property pages,* which display your control's properties in an alternate format.

Each property page you connect to your control becomes one tab on the tabbed Properties dialog box. Visual Basic handles all the details of presenting the pages as a tabbed dialog, and manages the OK, Cancel, and Apply buttons. All you have to do is lay out the controls that will be used to set the property values.

Property pages are useful when a group of properties interact in a complex fashion, as with the Toolbar control included with Visual Basic. They're also useful for controls that will be distributed internationally, because the captions can be localized for different languages. Property pages also allow your controls to be used with development tools that don't have a Properties window.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

15

To add a property page to the project

55 In the **Project** window, click **ControlDemo** to select the control project. On the **Project** menu, click **Add Property Page** to add a property page to the project.

56 In the **Properties** window, double-click the **Name** property, and change the name of the property page to **SLGeneral**. Double-click the **Caption** property, and change the caption to **General**.

37 The caption is what will appear on the property page's tab when it's in use.

38 Why name the page SLGeneral instead of General? You may have several controls in a project, and each one may have a General page. This is the ShapeLabel control's General page.

57 On the **File** menu, click **Save Project** to save the project. Name the property page as shown in the following table. Visual Basic will provide the indicated extension automatically.

File	Filename	Extension
Property page	ControlDemo_SLGeneral	.pag

16

39Binary information in a property page — such as bitmaps — will be saved in a binary file with the same name and an extension of .pgx.

39

The designer for a property page looks much like the designer for a control, except that the caption bar of the designer shows the Caption property of the property page, instead of the Name property.

To design the General property page for the ShapeLabel control

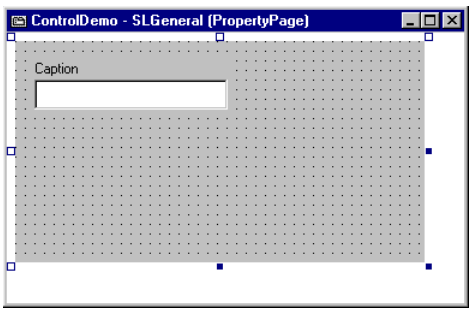
58 Place a Label control on the property page, and set the Caption property of the label to the word **Caption**.

59 Underneath the label, place a TextBox control, and assign it the following property values:

Property	Value
Name	txtCaption
Text	<empty>

40

40The property page should appear as shown below.



41

41Placing the property description label above the text box in this fashion makes it easier to localize your control component for other languages, in which the word for “Caption” may be longer or shorter. Localization of controls is discussed in detail in “Building ActiveX Controls.”

60 Double-click the **property page**, to open a code window. In the **Events** drop down, select the **SelectionChanged** event, and add the following code:

```
62Private Sub PropertyPage_SelectionChanged()
63  ' Display the caption of the first control in
64  '   the list of currently selected controls.
65  txtCaption.Text = SelectedControls(0).Caption
```

```
End Sub
```

42

42 The purpose of this event is to get the existing property values from the ShapeLabel control *or controls* that are currently selected. That's right, there may be more than one ShapeLabel control selected. Multiple selection is a wonderful thing for the user of your control, but it means more work for you.

43 A property page receives a SelectionChanged event whenever it is opened. It also receives this event when the list of selected controls changes. This is necessary because the **Property Pages** dialog box is modeless, so a user may select additional controls while the dialog box is open.

44 You have to decide how to handle multiple selection on a property-by-property basis. For example, if your property page displays the Width property of the first control in the SelectedControls collection—that is, SelectedControls(0), as shown in the code above—it will be easy for the user to change the widths of all the selected controls to that value.

45 On the other hand, there is very little use in setting the captions of all the ShapeLabel controls on a form to the same value, so the logical thing to do with the Caption property is to disable txtCaption if the Count property of the SelectedControls collection is greater than one.

46 However, this procedure doesn't do the logical thing. For illustration purposes, the property page will be allowed to set multiple captions. Later, if you want to enable the behavior described above, you can add the following lines of code to the PropertyPage_SelectionChanged event procedure:

```
' Please don't do this yet!
If SelectedControls.Count > 1 Then
    txtCaption.Enabled = False
Else
    txtCaption.Enabled = True
End If
```

43

61 To set the property values for all currently selected controls, add the following code to the ApplyChanges event:

```
Private Sub PropertyPage_ApplyChanges()
' Use a generic Object variable, in case more
'   than one kind of control is selected.
Dim objControl As Variant
For Each objControl In SelectedControls
    objControl.Caption = txtCaption.Text
Next
End Sub
```

44

47 Your property page receives the ApplyChanges event when the user clicks the **Apply** or **Cancel** buttons of the **Property Pages** dialog box.

48 How do you know that every control in SelectedControls has a Caption property? As the designer of the control component, you determine which property pages are connected to any given control. A property page will only appear if *all* the currently selected controls have that page in their Property Pages list. The

easiest thing to do is to make sure that the pages assigned to each control don't show properties the control doesn't have.

49If you wish to use a general-purpose property page for a number of controls, and some of those controls don't have all the properties displayed on the page, you can add code to the ApplyChanges event to test the type of the control, and apply the property value as appropriate. Alternatively, you can use an On Error statement to trap and ignore errors from controls that don't have the property.

50You only need to be concerned with the controls in your component, because controls that are not part of your component will never use your component's property pages.

51"Creating Property Pages for ActiveX Controls" discusses property page layout and assignment in greater detail.

62 To enable the **Apply** button of the **Property Page** dialog box when the Caption property is changed, add the following code to the Change event of the txtCaption text box:

```
81Private Sub txtCaption_Change()
82  ' The Changed property of the property page
83  '   controls the Apply button of the Property
84  '   Pages dialog box.
85  Changed = True
86End Sub
```

45

63 Click the **Close** button on the designer for the property page to put the page in run mode. Like UserControl objects, PropertyPage objects must run while the rest of the project group is in design mode.

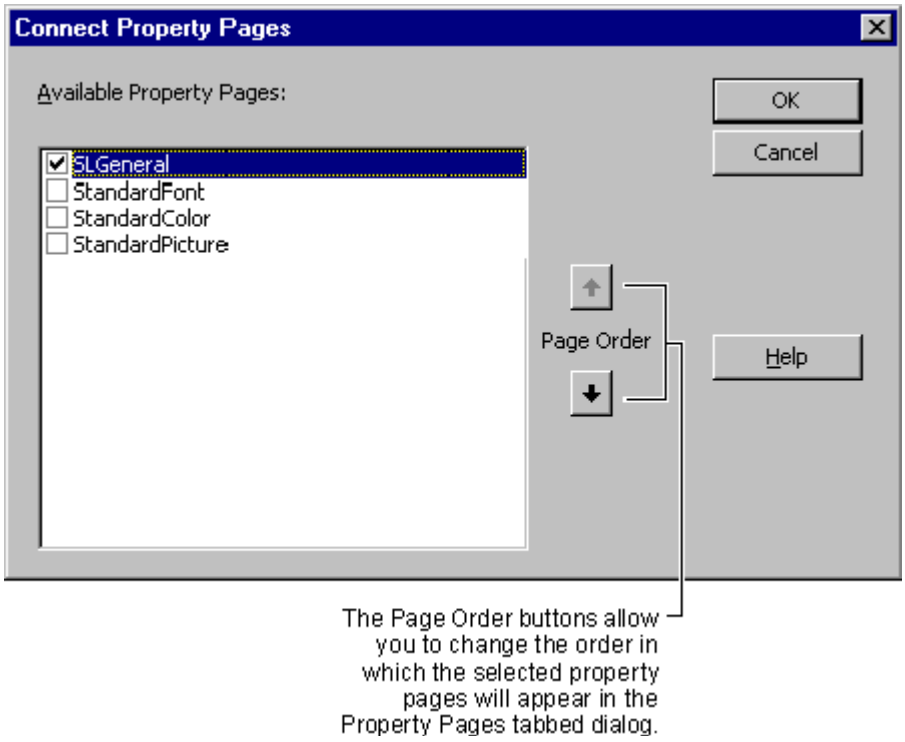
46

To connect the property page to the ShapeLabel control

64 In the Project window, double-click **ShapeLabel** to open the designer.

65 In the **Properties** window, double-click the **PropertyPages** property to display the **Connect Property Pages** dialog box.

12



47

52The Connect Property Pages dialog box can be used to connect multiple pages to a user control, and to control the display order of the tabs in the Property Pages dialog box for your control.

53Property pages can also be connected at run time. This is discussed in "Creating Property Pages for ActiveX Controls."

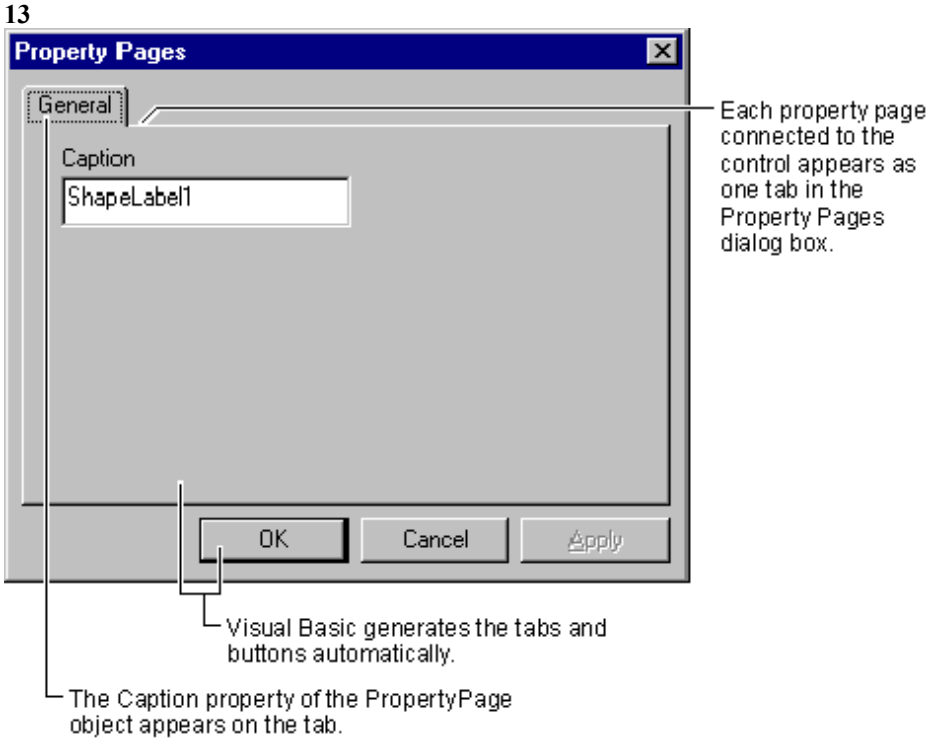
66 Check **SLGeneral**, and then click **OK**.

67 Click the **Close** button on the **ShapeLabel** designer to put the ShapeLabel control in run mode.

68 In the Project window, double-click **Form1** to open its designer.

69 Right-click on one of the ShapeLabel controls on Form1, to show the context menu, and click **Properties** to show the **Property Pages** dialog box.

13



48

70 In the **Caption** box on the **General** tab, replace the current caption with a new value. When you do this, the **Apply** button is enabled. Click the **Apply** button to change the caption of the control.

7**Note** You could also change the caption by pressing **OK**, but this would close the **Property Pages** dialog box. The dialog box should stay open for the next step.

17

71 Hold down the CTRL key and click the second **ShapeLabel** control on Form1, so that both ShapeLabels are selected. Change the caption and click the **Apply** button to set both captions to the same value.

54You may want to try adding other controls, such as command buttons, to Form1, and observing the effects of different multiple selections on the **Property Pages** dialog box.

72 When you're done experimenting, click **OK** to close the **Property Pages** dialog box.

49

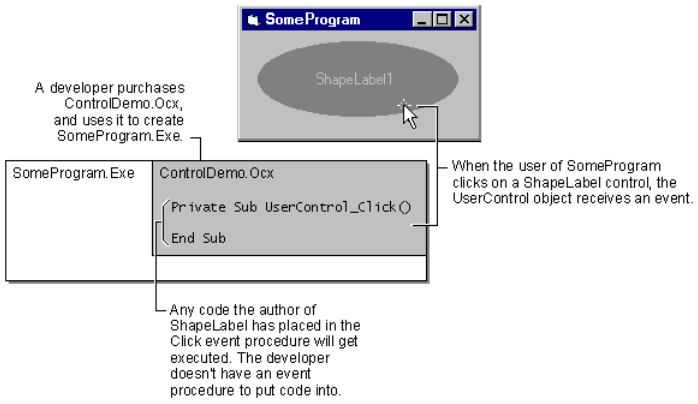
For More Information Property pages are discussed in detail in "Creating Property Pages for ActiveX Controls."

Adding an Event to the ShapeLabel Control

It's important to distinguish between the events received by your UserControl object (or by the controls it contains) and the events your control raises. Events your control *receives* are opportunities for you to do something interesting; events your control *raises* provide opportunities for the developer who uses your control to do something interesting.

Figure 4.1 shows what happens when a control author simply uses the events received by the UserControl object, and doesn't raise any events for the developer who buys the control.

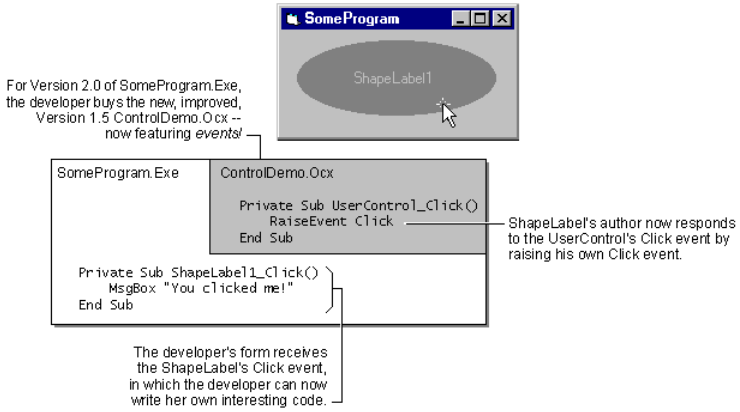
Figure 4.1 An ActiveX control that simply uses events



50

Figure 4.2 shows what happens when the author of ControlDemo.ocx — no doubt tired of developer complaints — improves the ShapeLabel control by raising a Click event for the developer to respond to.

Figure 4.2 A control that raises events for the developer to use



There are many events that might be of interest to the user of the ShapeLabel control. The Visual Basic Label control raises a Click event, and ShapeLabel is just a fancy label, so the following procedure will add a Click event. To make the event more interesting, it will be raised only if the user clicks on the oval background.

Being compatible with other controls of the same type is an important reason to add a particular event to your control. Other criteria for choosing what events to raise can be found in "Raising Events from Controls," in "Building ActiveX Controls."

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

To add a Click event to the ShapeLabel control

73 In the Project window, click **ShapeLabel** to select it, then press F7 or click the **Code** button on the Project window toolbar, to open the **Code** window.

74 In the **Object** box, select (**General**). In the **Procedure** box, select (**Declarations**) to position yourself at the top of the code module. Add the following code:

```
Option Explicit
' Declare a public Click event with no arguments.
Public Event Click()
```

75 In the **Object** box, select **lblCaption**. In the **Procedure** box, select the **Click event** for the label control. Add the following code to the lblCaption_Click event procedure:

```
Private Sub lblCaption_Click()
  ' Raise a Click event whenever the user clicks
  '   on the label.
  RaiseEvent Click
End Sub
```

The code above raises a Click event only if the user clicks on the constituent control lblCaption. It will seem more natural to users to be able to click anywhere on ShapeLabel's oval background, so the next step shows how to raise the click event if the user clicks on the colored oval.

76 In the **Object** box, select **UserControl**. In the **Procedure** box, select the **UserControl's MouseUp** event. Add the following code to the UserControl_MouseUp event procedure:

```
Private Sub UserControl_MouseUp(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
  ' Raise a Click event only if the color of the
  '   point that was clicked on matches the color
  '   of the Shape control.  Ignore clicks that are
  '   outside the oval.
If Point(X, Y) = shpBack.FillColor Then
      RaiseEvent Click
End If
End Sub
```

56Determining whether an event occurred in a particular location is called *hit testing*.

57You might expect to put the hit test code in the shpBack_Click event procedure, because shpBack is always resized to cover the entire surface of the ShapeLabel control. However, Shape controls don't receive Click events. Instead, the Click event is received by the object that contains the Shape — in this case, the UserControl object.

58"Drawing Your Control," in "Building ActiveX Controls," discusses the use of transparent backgrounds to create irregularly shaped controls.

77 In the Project window, click **Form1** to select it, then press F7 or click the **Code** button on the Project window toolbar, to open the Code window.

78 In the **Object** box, select one of the **ShapeLabel** controls you added to Form1. In the **Procedure** box, select the **Click** event.

8**Note** If the Click event does not appear, make sure the ShapeLabel designer is closed.

55

59Add the following code to the ShapeLabel1_Click event procedure:

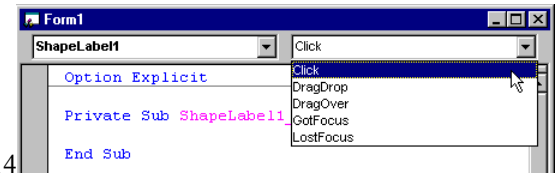
```
105Private Sub ShapeLabel1_Click()
106 MsgBox "Thanks for clicking! My caption is: " _
107       & ShapeLabel1.Caption
108End Sub
```

56

9**Note** If the ShapeLabel you selected is not named ShapeLabel1, use the appropriate name when entering the code above.

57

60You can click the arrow on the Procedure box to view all of the events for the ShapeLabel control. In addition to your Click event, there are four events — DragDrop, DragOver, GotFocus, and LostFocus — that are automatically provided for you by the container, Form1.



14

58

79 On the toolbar, click the **Start** button, or press F5 to run TestCtlDemo. Try clicking various places on the form and on the ShapeLabel control, to verify that the Click event is being raised only when you click inside the oval background.

80 There's a subtle bug in the hit testing for ShapeLabel's click event. To see this, press the mouse button while the mouse pointer is in the lower half of the red oval. Holding the mouse button down, carefully move the mouse pointer until the tip of the arrow is on the white text of ShapeLabel's caption, then release the mouse button. The message box doesn't appear!

61 The lblCaption_Click event procedure doesn't get executed, because the MouseDown event occurred over the UserControl. Therefore, when the MouseUp event occurs, it is received by the UserControl — even if the mouse has been moved completely off Form1.

62 The hit test code in the MouseUp event works if the mouse button is released over the red background that shows through lblCaption's transparent background, but not if the button is released over the white foreground color of the text. (If the button is released outside ShapeLabel, the Point function returns -1, so releasing the mouse button over some random red spot will not raise the Click event.)

63 Fixing this bug is left as an exercise for the reader. (Hint: Moving the hit test to the Click event of the UserControl won't help, because the Click event doesn't occur when the MouseUp event is over a different object from the MouseDown.)

59

For More Information See "Adding Events to Controls" in "Building ActiveX Controls."

60

Compiling the ControlDemo Component

Once you have created an ActiveX control project containing one or more UserControl objects, you can compile it into an .ocx file and use the controls in other applications. The following procedures demonstrate this.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

19

Compiling the ControlDemo project

81 If the TestCtlDemo project is still in run mode, click the **Close** button on Form1 to return to design mode.

82 In the Project window, click **ControlDemo** to select the project.

83 On the **File** menu, click **Make ControlDemo.ocx** to open the **Make Project** dialog box. Click **OK** to build the .ocx file.

84 On the **File** menu, click **Remove Project** to remove ControlDemo from the project group, so that Visual Basic will use the compiled binary component (.ocx file) instead of the project.

64 Visual Basic displays a warning message, because the TestCtlDemo project contains a reference to ControlDemo. Click **Yes** to remove ControlDemo anyway.

65 When you remove ControlDemo from the project group, Visual Basic looks for ControlDemo.ocx in the Windows Registry. If the .ocx file exists, Visual Basic automatically updates the reference you set in the procedure "Adding the TestCtlDemo Project."

66To switch back to using the project instead of the binary component, you can click **Add Project to Group**, on the **File** menu, and add the ControlDemo project back to the project group.

85 Press F5 to run TestCtlDemo using the .ocx file.

61

When ControlDemo is running from source code, you cannot access the ShapeLabel control from other applications, or from another copy of Visual Basic. This is because ActiveX control components must run in process. Once you have compiled a .ocx component, you can test it from other applications.

To use ControlDemo.ocx in another copy of Visual Basic

86 Open a new instance of Visual Basic. In the **New Project** dialog box, double-click the **EXE Project** icon to open an .exe project.

87 On the **Project** menu, click **Components** to open the **Components** dialog box. On the **Controls** tab, check **ActiveX Control Creation Demo,** and then click **OK**.

67The icon for ShapeLabel appears on the Toolbox. You can now add ShapeLabel controls to the default form, and use the **Properties** window to set their properties. You can also right-click on an instance of ShapeLabel, and choose **Properties** from the Context menu to edit the control's properties with the property page.

88 Press F5 to run the project.

68You can also compile the project and run the .exe.

62

For More Information An .ocx file can contain multiple controls and property pages. "Distributing ActiveX Controls," in "Building ActiveX Controls," discusses control packaging and distribution.

63

Control Creation Recap

In order to introduce new concepts in the most natural order, the procedures in this chapter have not followed the normal sequence of steps for creating a new control.

Note This topic is part of a series that walks you through creating a sample ActiveX control. It begins with the topic, "Creating an ActiveX Control."

20

When you create a new control, the steps you'll generally follow are these:

1. Determine the features your control will provide.
2. Design the appearance of your control.
3. Design the interface for your control — that is, the properties, methods, and events your control will expose.
4. Create a project group consisting of your control project and a test project.
5. Implement the appearance of your control by adding controls and/or code to the UserControl object.

6. Implement the interface and features of your control.
7. As you add each interface element or feature, add features to your test project to exercise the new functionality.
8. Design and implement property pages for your control.
9. Compile your control component (.ocx file) and test it with all potential target applications.

64

If your control component will provide more than one control, you should begin by deciding what controls the package will include. Your test project should have separate test forms for the individual controls, and at least one form that tests the controls together.

For More Information General design issues for ActiveX components are discussed in "General Principles of Component Design" and "Debugging, Testing, and Deploying Components." Issues exclusive to ActiveX control creation, testing, packaging, and deployment are discussed in "Building ActiveX Controls" and "Creating Property Pages for ActiveX Controls"