Lab 8: Creating ActiveX Controls

For background information on this lab, click each of these topics:

Objectives

In this lab, you will build and test an ActiveX control.

After completing this lab, you will be able to:

- Create a basic ActiveX control with Visual Basic.
- Add properties and methods to the control.
- Save and load control properties.
- Raise an event from the control.
- Create a property page for the control.
- Create a data-bound control.

Prerequisites

Before working on this lab, you should be familiar with the following concepts:

- Creating ActiveX code components
- The contents of this chapter

Lab Setup

To complete this lab, you need the following setup:

- Visual Basic version 5.0 or later

To see a demonstration of the completed lab solution, click this icon.

Estimated time to complete this lab: **90 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 8.

Exercise 1: Creating a Control

In this exercise, you will create a simple ActiveX control and test that control from a Standard EXE project.

Exercise 2: Adding Properties and Methods

In this exercise, you will create properties and a method and add them to the Stock ActiveX control created in the previous exercise.

Exercise 3: Saving and Loading Properties

In this exercise, you will add code to save and restore the **UserControl** object property values.

Exercise 4: Raising an Event

In this exercise, you will add code to raise the TickerKeyPress event from the Stock control.

Exercise 5: Creating a Property Page

In this exercise, you will create a property page for the Stock ActiveX control.

Exercise 6: (Optional) Creating a Data-Bound Control

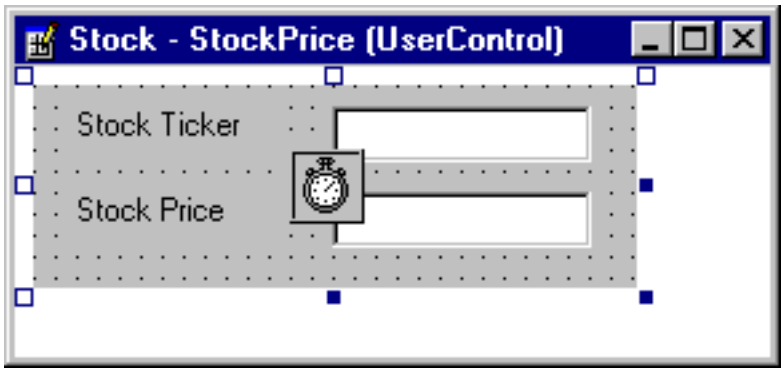
In this exercise, you will create an ActiveX control that can be bound to a data source.

Exercise 1: Creating a Control

In this exercise, you will create a simple ActiveX control and test the control from a Standard EXE project. The ActiveX control will simulate returning stock price information.

▶ Create the ActiveX control

1. Create a new ActiveX Control project.
2. Name the project **Stock**.
3. Name the **UserControl** object **StockControl**.
4. Save the **UserControl** object and project files as Stock.ctl and Stock.vbp, respectively.
5. Add two **Label** controls, two **TextBox** controls, and one **Timer** control to the **UserControl** object, as shown in the following illustration.



6. Set the **Timer** control's **Interval** property to **1000**.
7. In the **Timer** control's Timer event, add the following code:

```
If txtStockTicker = "MSFT" Then
    'Return simulated stock price.
    txtStockPrice.TEXT = Rnd() * 200
Else 'unknown ticker
    txtStockPrice.TEXT = 0
End If
```

   Your text boxes may be named differently, so you should adjust your code accordingly.

8. Save the project.

▶ Test the ActiveX control

1. Add a new Standard EXE project to the project group.
2. Save the new form and project as Contain.frm and Contain.vbp, respectively.
3. Add a **StockControl** to the form.

   If the **Toolbox** icon for the **StockControl** is disabled, make sure all of the windows associated with the Stock project are closed.

Notice that when the control is placed on the form, and the form is running, the number zero should appear in the **Stock Price** text box.

4. Run the .exe project.
5. In the **Stock Ticker** text box, enter **MSFT** (all caps).

   You should see new random values every two seconds in the **Stock Price** text box.

Exercise 2: Adding Properties and Methods

In this exercise, you will create properties and a method, and add them to the **Stock** ActiveX control created in the previous exercise.

▶ Create properties and a method for a control

1. Using property procedures, add the property **Active** to the **UserControl** object. Set the property type to **Boolean**.
2. In the **Active Property Get** procedure, return the value of the **Timer** control's **Enabled** property.
3. In the **Active Property Let** procedure, set the **Enabled** property of the **Timer** control to the value passed to the **Active** property.

   This will let you control the **Stock** control's refresh behavior.
4. Using property procedures, add the property **Font** to the **UserControl** object. Set the property type to **Font**.

   Because this is an object property, use a **Property Set** procedure instead of a **Property Let** procedure. This property will be used to set the **Font** object for all of the **TextBox** and **Label** controls on the **UserControl** object.
5. In the **Font Property Get** procedure, return the **Font** object of the **Stock Price** text box.

   When returning the **Font** object, be sure to use a **Set** statement to set **Font** is an object.
6. In the **Font Property Set** procedure, assign the **Font** value to the **Font** property for all of the **TextBox** and **Label** controls on the **UserControl** object.
7. Using a **Public Sub** procedure, add the method **Refresh** to the **UserControl** object.
8. In the **Refresh** method, call the **Timer** control's Timer event procedure.

   This will update the stock price when the **Refresh** method is called.

▶ Enable a property page for the Font property

1. In the **Procedure Attributes** dialog box, set the Property Page attribute for the **Font** property to StandardFont.

   This will let you use the **Standard Font** dialog box to change the property.
2. In the **Connect Property Page** dialog box, select the StandardFont property page.

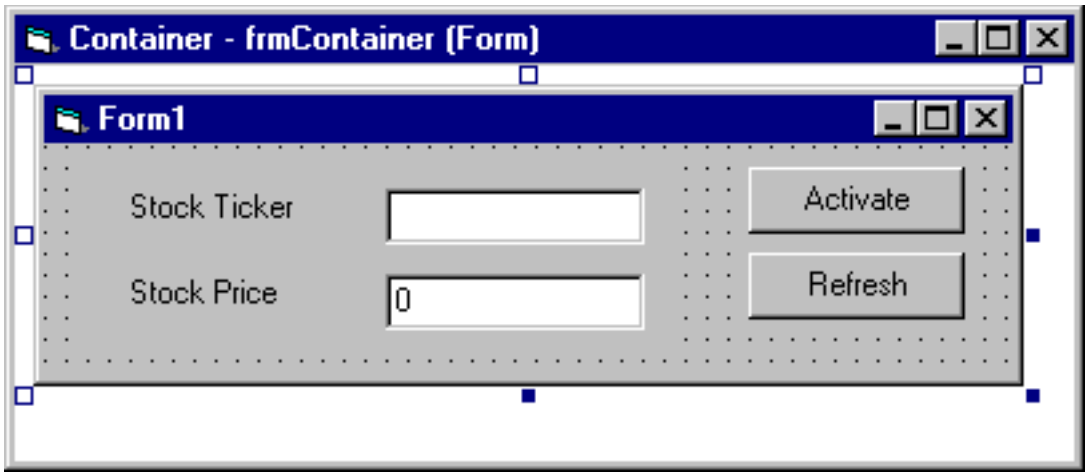
   Your **Font** property can now be set by using the StandardFont property page.
3. Save the project.

▶ Test the control

1. Close all of the windows associated with the Stock project.
2. Open the Container form and select the **StockPrice** control.
3. In the **Properties** window, select the **Font** property and click the ellipsis (**…**)button.

   This should display the **Font** property page.
4. Change the font to **Bold**, and then click **OK**.

   The **TextBox** and **Label** controls should be bold.
5. Add two command buttons to the Container form, as shown in the following illustration.



6. Add code to the command buttons, as shown in the following code.

```
Private Sub cmdActivate_Click()
    'toggle Active property
    StockPrice1.Active = Not StockPrice1.Active
End Sub
Private Sub cmdRefresh_Click()
    StockPrice1.Refresh
End Sub
```

7. Run the Stock project, and type **MSFT** in the **Stock Ticker** text box.

   When you run the project, notice that the **Label** and **TextBox** controls are no longer bold. This is because the properties were not saved for the form.

8. Test each of the command buttons.

   The **Activate** button should refresh the **Stock** controls on and off.

   The **Refresh** button should refresh the stock price.

   If you are having problems with your solution, click this icon to see the **UserControl** object code from the lab solution.

```
Public Property Get Active() As Boolean
    Active = Timer1.Enabled
End Property
Public Property Let Active(ByVal New_Active As Boolean)
    Timer1.Enabled = New_Active
    PropertyChanged "Active"
End Property
Public Property Get Font() As Font
    Set Font = txtStockPrice.Font
End Property
Public Property Let Font(ByVal New_Font As Font)
    Dim objCtl As Control
    'loop through all controls
    For Each objCtl In Controls
        'if control is a label or textbox
        If (TypeOf objCtl Is Label) Or _
                (TypeOf objCtl Is TextBox) Then
            Set objCtl.Font = New_Font
        End If
    Next
    PropertyChanged "Font"
End Property
```

```
Public Sub Refresh()
    Timer1_Timer
End Sub
```

Exercise 3: Saving and Loading Properties

In this exercise, you will add code to save and restore the **UserControl** object's property values.

▶ Save and load property values

1. In the **UserControl_WriteProperties** event procedure, use the **WriteProperty** method to save the property values, as shown in the following table.

Property	Value	Default value
Active	Timer1.Enabled	**True**
Font	txtStockPrice.Font	(None)

2. In the **Property Let** procedures for the **Active** and **Font** procedures, call the **PropertyChanged** method.

   This will ensure that the properties are correctly saved.

3. In the **UserControl_ReadProperties** procedure, use the **ReadProperty** method to load the **Active** and **Font** properties.

   For more information about saving and loading properties, see Storing and Retrieving Property Values.

4. Save the project.

▶ Test property save and load functionality

1. Close all of the windows associated with the Stock project.
2. Open the Container form and select the **StockPrice** control.
3. Set the **Active** property to **False**.
4. Set the **Font** property to **Bold**.
5. Run the project.

   The **Active** and **Font** properties should maintain their values.

   If you are having problems with your solution, click this icon to see the **UserControl** object code from the lab solution.

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Active", Timer1.Enabled, True
    PropBag.WriteProperty "Font", txtStockPrice.Font
End Sub
Public Property Let Active(ByVal New_Active As Boolean)
    Timer1.Enabled = New_Active
    PropertyChanged "Active"
End Property
Public Property Let Font(ByVal New_Font As Font)
    Dim objCtl As Control
    'loop through all controls
    For Each objCtl In Controls
        'if control is a label or textbox
        If (TypeOf objCtl Is Label) Or _
                (TypeOf objCtl Is TextBox) Then
            Set objCtl.Font = New_Font
        End If
    Next
    PropertyChanged "Font"
```

```
End Property
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    Active = PropBag.ReadProperty("Active", True)
    Font = PropBag.ReadProperty("Font")
End Sub
```

Exercise 4: Raising an Event

In this exercise, you will add code to raise the TickerKeyPress event from the **Stock** control. This event will be raised from the KeyPress event of the ticker text box to enable developers to add code to the event.

▶ Raise a control event

1. In the **UserControl** object code window, declare the event TickerKeyPress to take the single argument **KeyAscii** of type **Integer**.
2. In the KeyPress event of the ticker text box, raise the TickerKeyPress event and pass the **KeyAscii** argument.

   This will simulate the KeyPress event for the **UserControl** object.
3. Save the project.

▶ Test the event

1. Close all of the windows associated with the Stock project.
2. Open the Container form.

   The **StockPrice** control should now contain the TickerKeyPress event.
3. In the TickerKeyPress event, add the following line of code:

   ```
   KeyAscii = Asc(UCase(Chr(KeyAscii)))
   ```

   This code will convert any character type into the uppercase value of that character.
4. Run the project.
5. Type **msft** (lowercase) into the ticker text box.

   The text should be converted to uppercase as you type.

   If you are having problems with your solution, click this icon to see the **UserControl** object code from the lab solution.

```
Public Event TickerKepPress(KeyAscii As Integer)
Private Sub txtStockTicker_KeyPress(KeyAscii As Integer)
    RaiseEvent TickerKepPress(KeyAscii)
End Sub
```

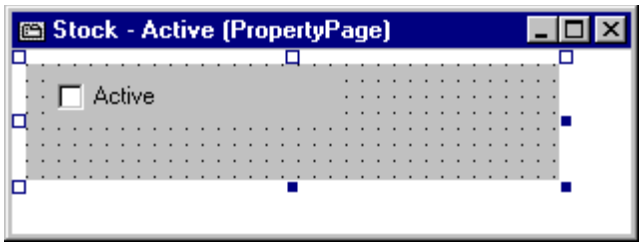
Exercise 5: Creating a Property Page

In this exercise, you will create a property page for the **Stock** ActiveX control.

▶ Add a custom property page

1. Add a new property page to the Stock project.

   Do not use the Property Page Wizard.
2. Set the property page **Name** property to **Active**.
3. Add a **CheckBox** control to the property page, as shown in the following illustration.



4. In the Click event of the **CheckBox** control, set the **Changed** property to **True**.
5. In the **PropertyPage_SelectionChanged** event procedure, set the **Value** property of the **CheckBox** control, as shown in the following table.

If SelectedControls(0).Active =	The Value property will be
True	vbChecked
False	vbUnchecked

   This will set the state of the **CheckBox** control, based on the value of the **Active** property of the selected control.

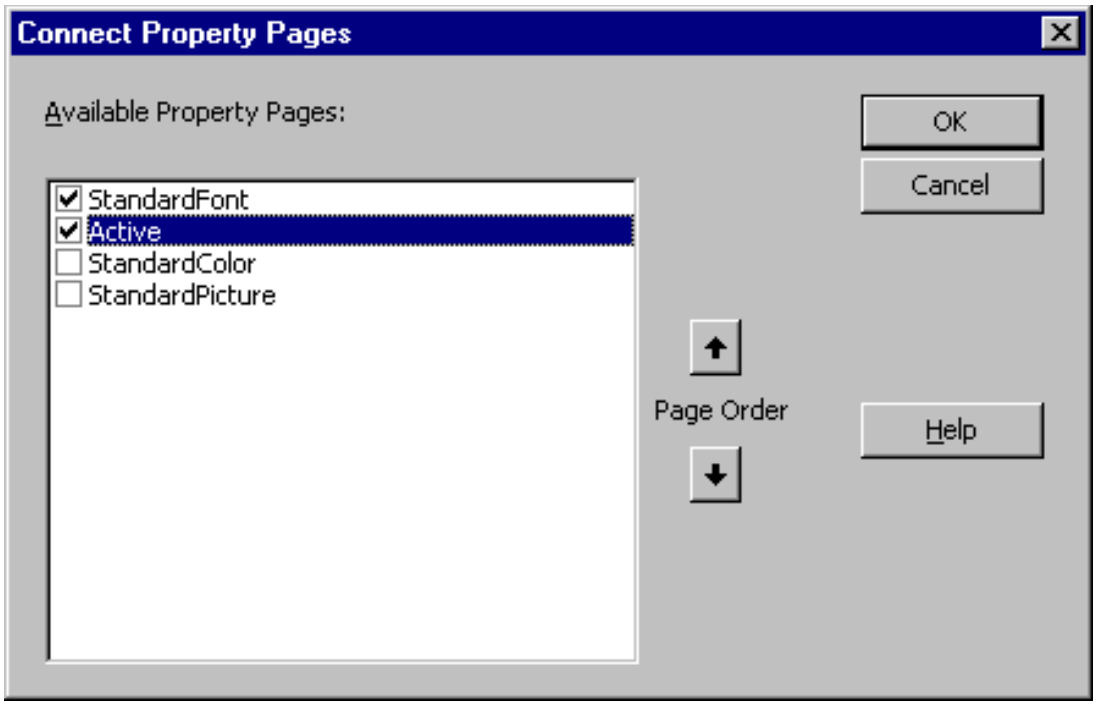
   For information about property pages, see Coding Property Page Behavior.

6. In the **PropertyPage_ApplyChanges** event procedure, set the **Active** property of SelectedControls(0), as shown in the following table.

If CheckBox Value property =	The Active property will be
vbChecked	**True**
vbUnchecked	**False**

   This will set the **Active** property of the **Stock** control, based on the selection made in the **Property** dialog box.

7. In the **Connect Property Pages** dialog box of the **Stock** control, add the **Active** property page, as shown in the following illustration.



8. In the **Procedure Attributes** dialog box, associate the **Active** property page with the **Active** property.

   For information about associating a property page with a property, see Establishing Property Page Relationships.

9. Save the project.

▶ Test the control

1. Close all of the windows associated with the Stock project.
2. Open the Container form.
3. Right-click the **StockPrice** control, and then click **Properties**.
4. In the **Property Pages** dialog box, click the **Active** tab.

   The active check box should match the value of the **Active** property.

5. Change the active check box, and then click OK.

   The **Active** property should appear changed in the **Properties** window.

6. In the **Properties** window, click the ellipsis (**…**) button next to the **Active** property.

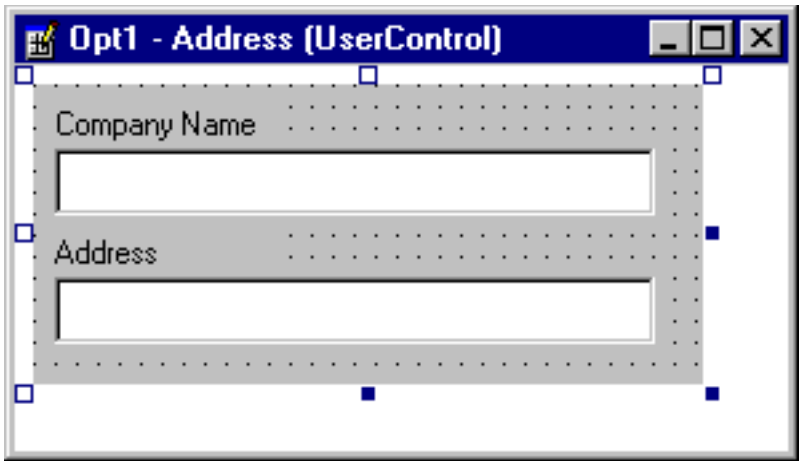
   Clicking this button will also display the **Property Pages** dialog box. Notice, however, that only the **Active** tab is displayed in the **Property Pages** dialog box.

Exercise 6: (Optional) Creating a Data-Bound Control

In this exercise, you will create an ActiveX control that can be bound to a data source.

▶ Create the control

1. Create a new ActiveX control, as shown in the following illustration.



2. Create **CompanyName** and **Address** properties for the control, and make them bindable.
3. Save the project.

▶ Test the control

1. Create a client project to test the control.
2. Bind the **CompanyName** and **Address** properties to the Company Name and Address fields of the Publishers table located in the Biblio.mdb database.

   The file Biblio.mdb is located in the Visual Basic install directory.
3. Test to see if the data is bound, and that it updates correctly.