Visual Basic.NET Introduction

ROAD MAP

- Introduction
- Designing Windows Controls
- Enhancing Existing Controls
- Building Compound Controls
- Building User-Drawn Controls
- Designing irregular controls
- Owner-Drawn Controls

Introduction

- To design custom Windows controls for .NET
- how to build custom controls that combine multiple .NET control
  - Compound controls
- To build user-drawn controls

Designing Windows Controls

- Standard application
  - a main form
  - several (optional) auxiliary forms
  - interacts with the user through its interface
- design of a Windows control
  - similar to the design of a form
  - place controls on a Form-like object (UserControl), which is the control's surface
- a custom control is an application with a visible user interface as well as an invisible programming interface
- manipulate the control
  - At design/ run time

Enhancing Existing Controls

- The .NET Windows controls are quite functional,
- Example: TextBox control
  - Customize appearance: change its color/ contents/ control formats
- The best approach is to create a new Windows control with all the desired functionality and then reuse it in multiple projects

Building the FocusedTextBox Control

- Call our new custom control FocusedTextBox
- Start a new VB project
- On the New Project dialog box
- select the template Windows Control Library
- Name the project FocusedTextBox.

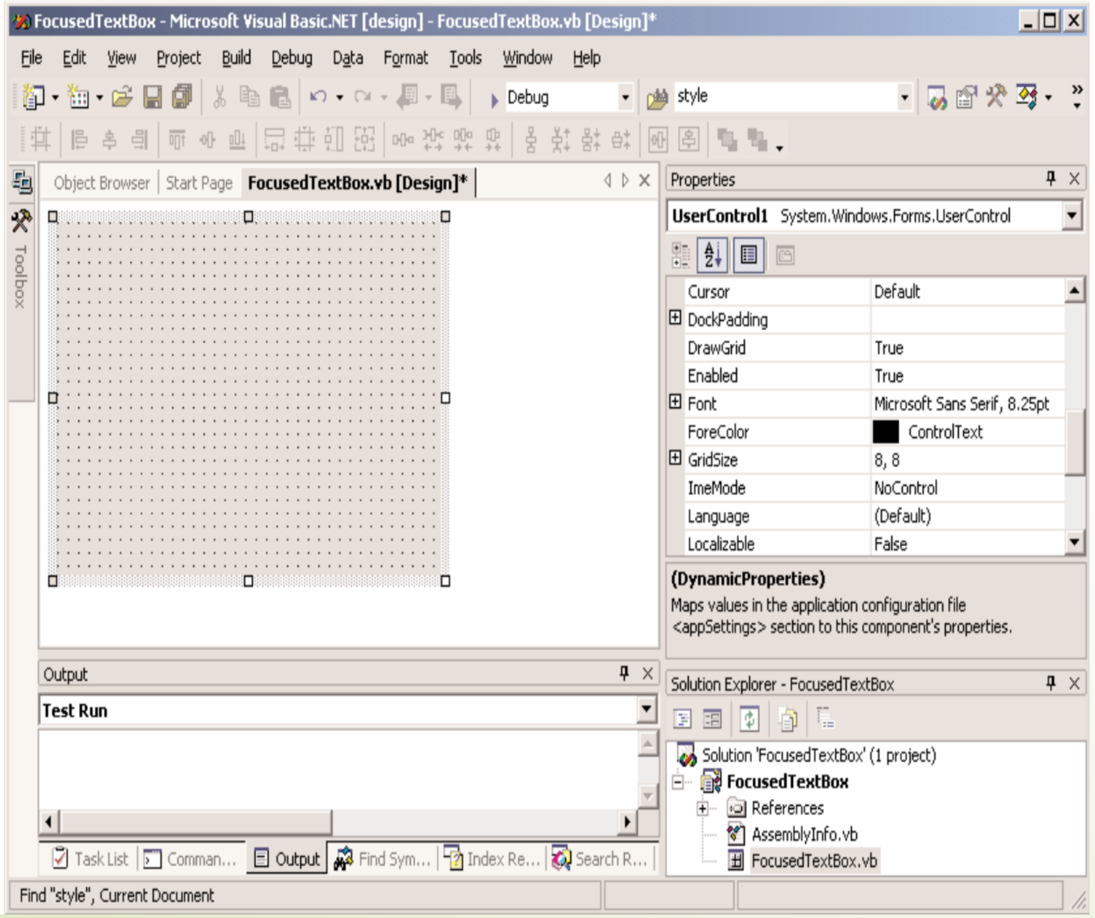


FIGURE 9.1

A custom control in design mode

FocusedTextBox - Microsoft Visual Basic.NET [design] - FocusedTextBox.vb [Design]*

Properties

UserControl1 System.Windows.Forms.UserControl

Cursor	Default
⊞ DockPadding	
DrawGrid	True
Enabled	True
⊞ Font	Microsoft Sans Serif, 8.25pt
ForeColor	■ ControlText
⊞ GridSize	8, 8
ImeMode	NoControl
Language	(Default)
Localizable	False

(DynamicProperties)
Maps values in the application configuration file <appSettings> section to this component's properties.

Output: Test Run

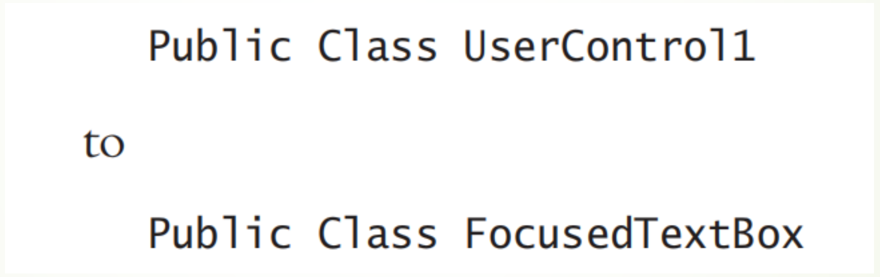
Solution Explorer - FocusedTextBox

```
Solution 'FocusedTextBox' (1 project)
  FocusedTextBox
    References
    AssemblyInfo.vb
    FocusedTextBox.vb
```

Find "style", Current Document

Building the FocusedTextBox Control

- Rename the UserControl1 object to FocusedTextBox
  - Rename the class



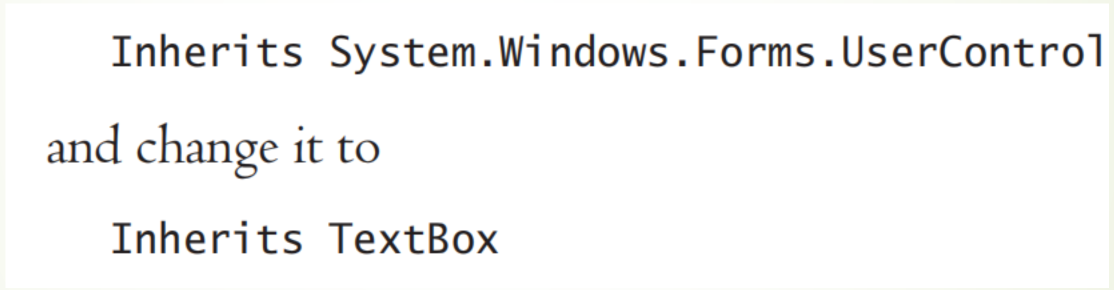
```
    Public Class UserControl1
```

to

```
    Public Class FocusedTextBox
```

Building the FocusedTextBox Control

- Inherit all the functionality of the TextBox control



```
    Inherits System.Windows.Forms.UserControl
```

and change it to

```
    Inherits TextBox
```

Building the FocusedTextBox Control

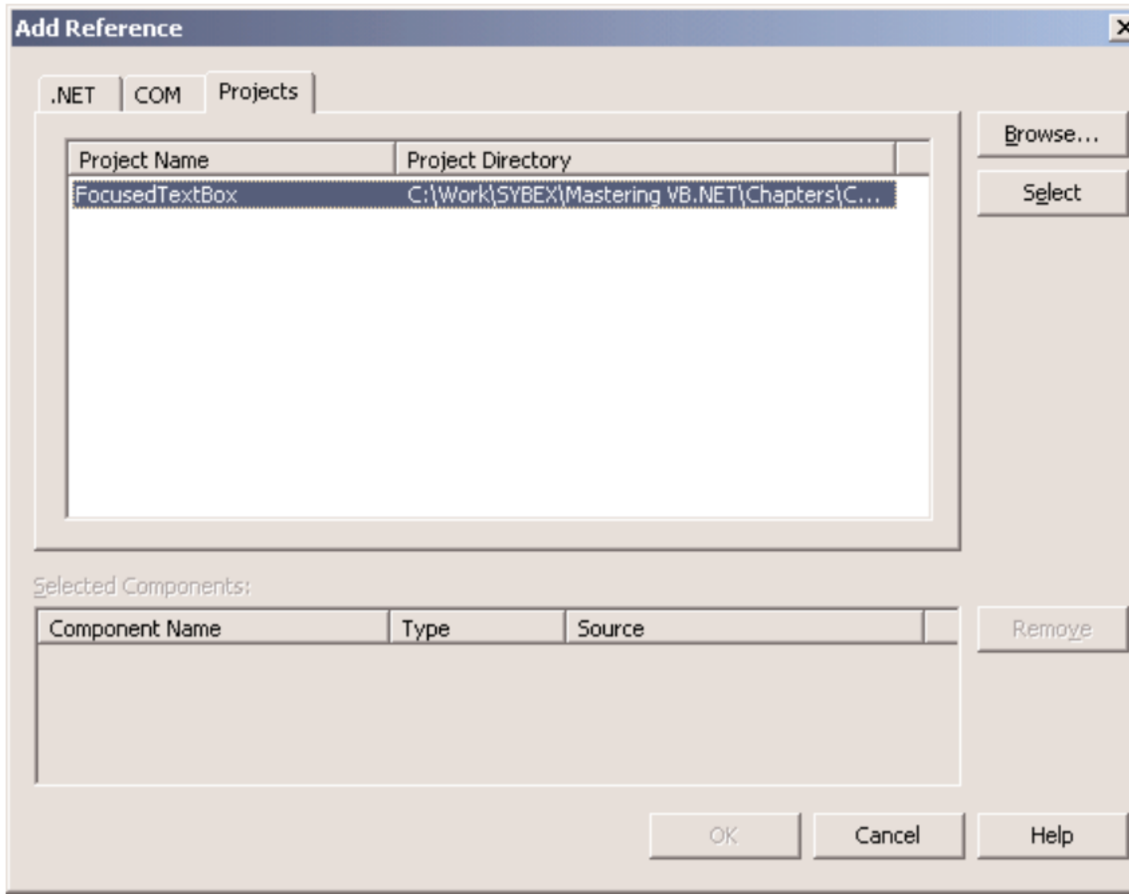
- To test the control, place an instance of the custom control on the *Form1* form of the test project
- build the control
- add a reference to this control to the test project
- Select the FocusedTextBox item in the Solution Explorer
- select the Build FocusedTextBox command

Building the FocusedTextBox Control

- Switch to the test project
  and select the Project
- Add Reference command
- switch to the Projects tab
- Your new control is now
  referenced in the test project

FIGURE 9.2

Referencing the custom control in the test project



Add Reference

.NET | COM | Projects

Project Name	Project Directory
FocusedTextBox	C:\Work\SYBEX\Mastering VB.NET\Chapters\C...

Selected Components:

Component Name	Type	Source

Building the FocusedTextBox Control

- Add some extra functionality to our custom TextBox control

```
Private Sub FocusedTextBox_Enter(ByVal sender As Object, _
               ByVal e As System.EventArgs) Handles MyBase.Enter

End Sub
```

```
Me.BackColor = Color.Cyan
```

```
Private Sub FocusedTextBox_Leave(ByVal sender As Object, _
               ByVal e As System.EventArgs) Handles MyBase.Leave
   Me.BackColor = Color.White
End Sub
```

overview of the control's custom properties

- **EnterFocusColor**
  - When the control receives the focus, its background color is set to this value
- **LeaveFocusColor**
  - When the control loses the focus, its background color is set to this value
- **Mandatory**
  - This property indicates whether the control corresponds to a required field, if Mandatory is True (Required), or an optional field, if Mandatory is False (Optional)
- **MandatoryColor**
  - This is the background color of the control if its Mandatory property is required

Building Compound Controls

- compound control
  - A visible interface that combines multiple Windows controls
  - Doesn’t inherit the functionality of any specific control
  - Must expose its properties by providing your own code

```
Property WordWrap() As Boolean
    Get
        WordWrap = TextBox1.WordWrap
    End Get
    Set(ByVal Value As Boolean)
        TextBox1.WordWrap = Value
    End Set
End Property
```

Building Compound Controls

- When your compound control contains a TextBox and a ComboBox control
- To raise the TextChanged event when the user edits the TextBox control and the (custom) SelectionChanged event
  - when the user selects another item in the ComboBox control

```
Event TextChanged
Event SelectionChanged
```

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles FocusedTextBox1.TextChanged
    RaiseEvent TextChanged()
End Sub
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
    RaiseEvent SelectionChanged()
End Sub
```

Building User-Drawn Controls

- A user-drawn control
  - consists of a UserControl object with no constituent controls

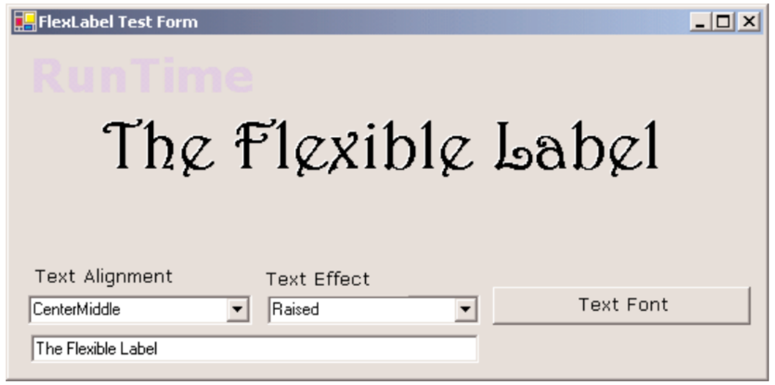
- Updating the control's visible area with the appropriate code, which must appear in the control's OnPaint method
  - called right before the OnPaint event is fired
  - if you override it, you can take control of the repaint process

Building User-Drawn Controls

- Develop the Label3D control
  - an enhanced Label control
- Provide all the members of the Label control plus a few highly desirable new features
  - the ability to align the text in all possible ways on the control, as well as in three-dimensional type
- New Custom Control: Label 3 D
  - the FlexLabel project
  - the Label3D project & the usual test project

FIGURE 9.5

The Label3D control is an enhanced Label control.



The Label3D Control's Specifications

- Label 3D Control
- Must provide the Caption and Font properties
- Align its caption both vertically and horizontally

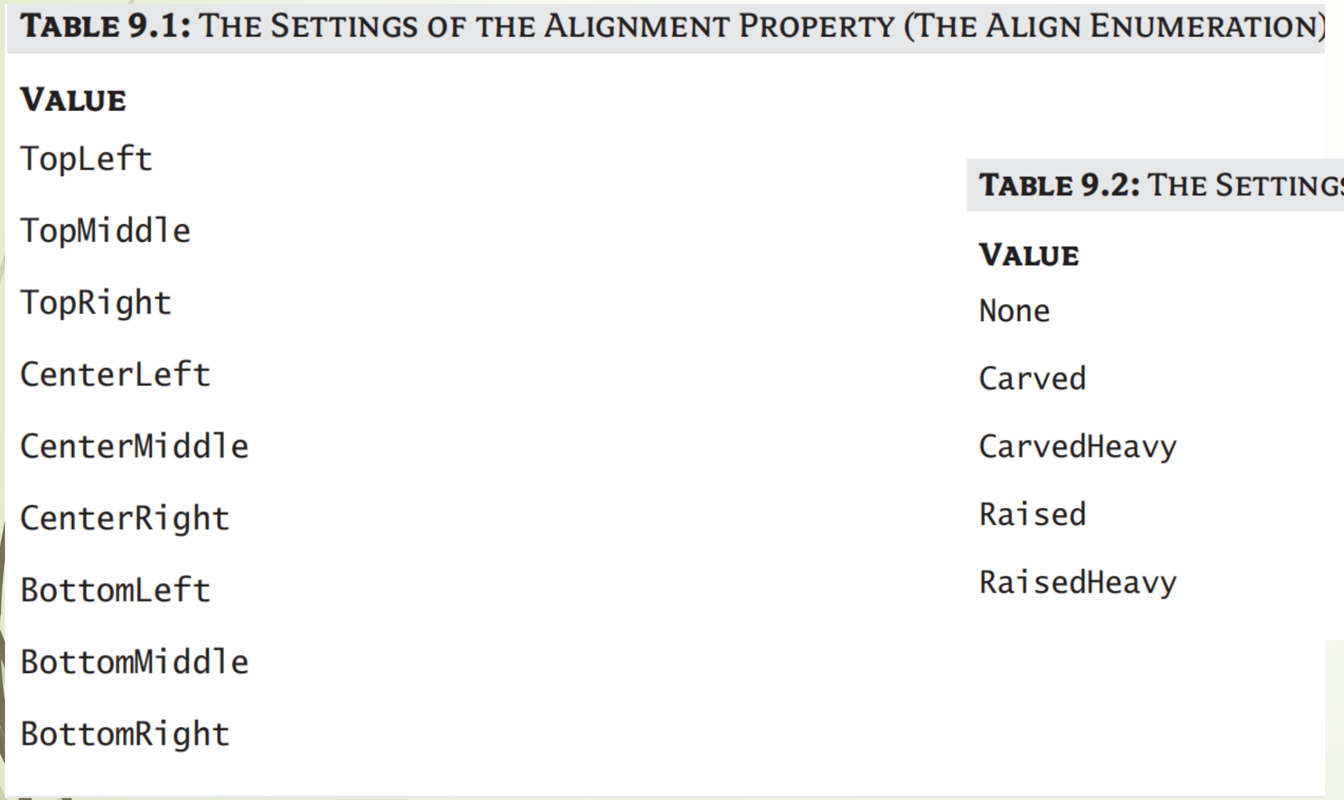


TABLE 9.1: THE SETTINGS OF THE ALIGNMENT PROPERTY (THE ALIGN ENUMERATION)

VALUE
TopLeft
TopMiddle
TopRight
CenterLeft
CenterMiddle
CenterRight
BottomLeft
BottomMiddle
BottomRight

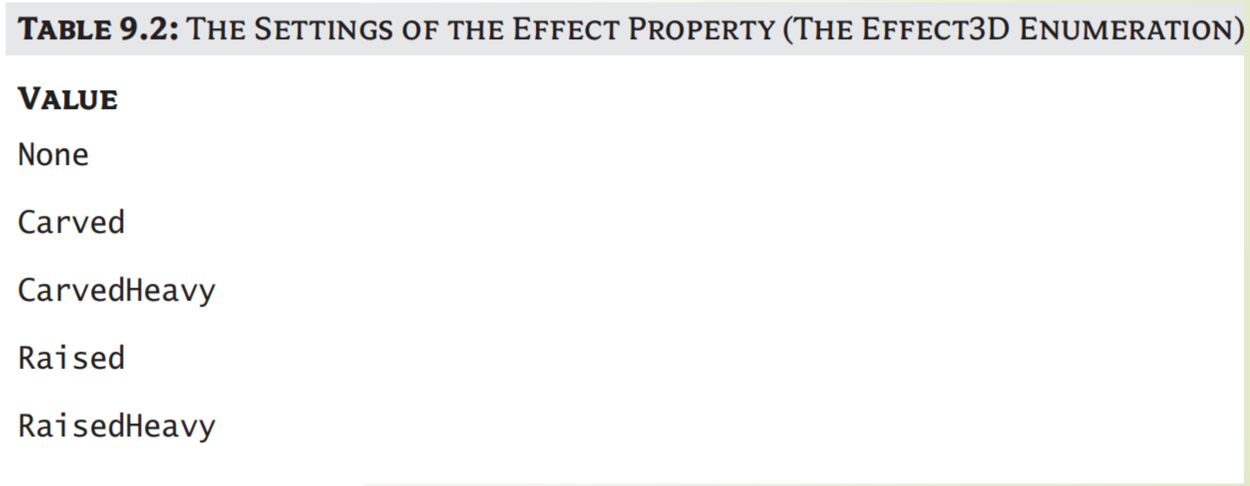
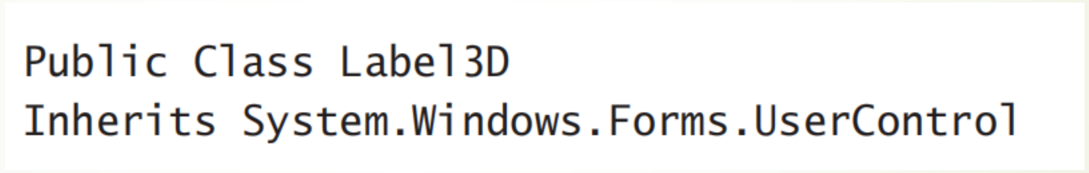


TABLE 9.2: THE SETTINGS OF THE EFFECT PROPERTY (THE EFFECT3D ENUMERATION)

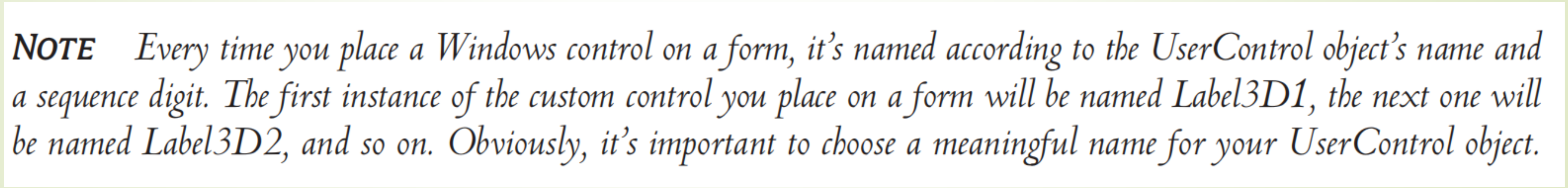
VALUE
None
Carved
CarvedHeavy
Raised
RaisedHeavy

Designing the Custom Control

- New Project: FlexLabel
- Rename the UserControl1 object to Label3D



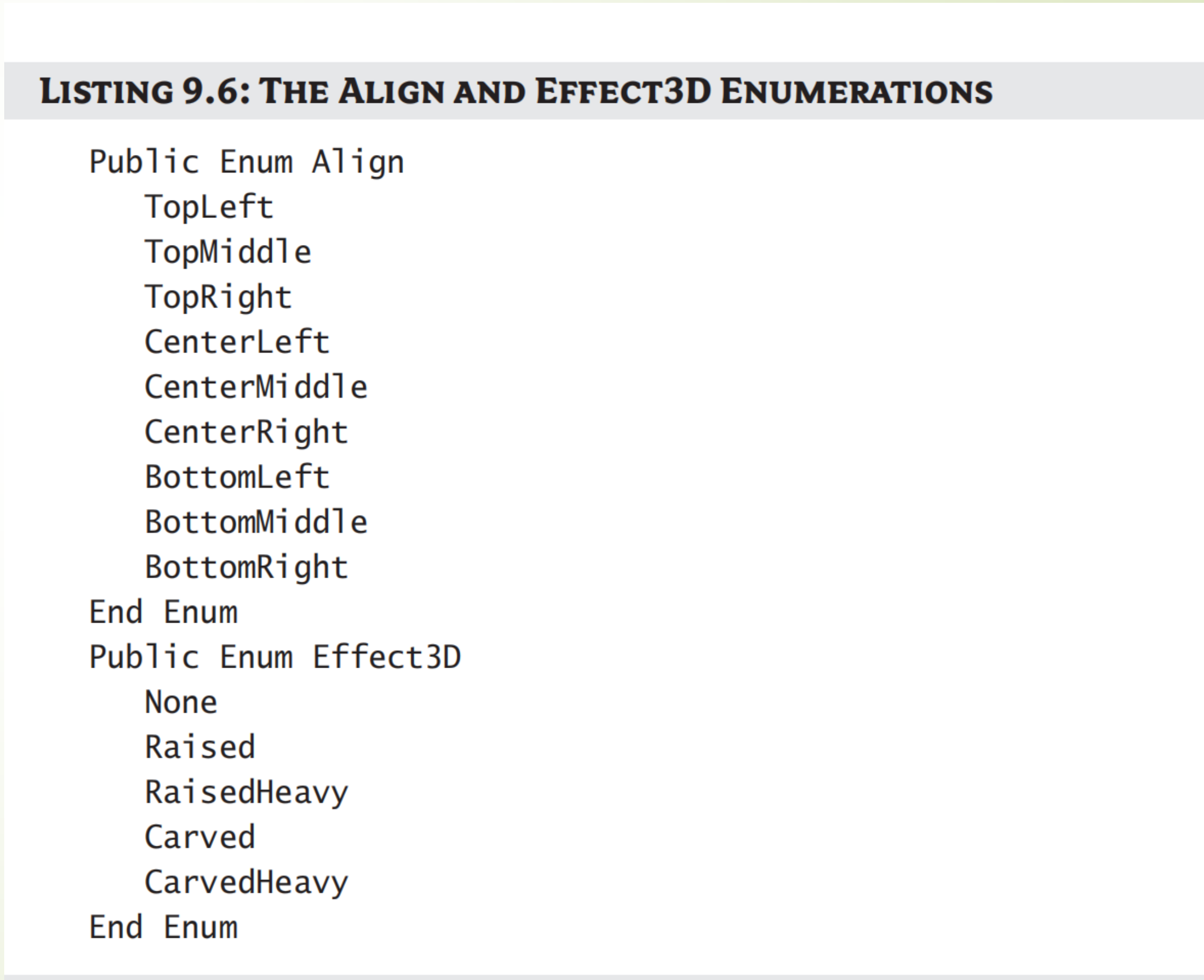
```
Public Class Label3D
Inherits System.Windows.Forms.UserControl
```



NOTE *Every time you place a Windows control on a form, it's named according to the UserControl object's name and a sequence digit. The first instance of the custom control you place on a form will be named Label3D1, the next one will be named Label3D2, and so on. Obviously, it's important to choose a meaningful name for your UserControl object.*

Designing the Custom Control

- The Enum statements for the two enumerations

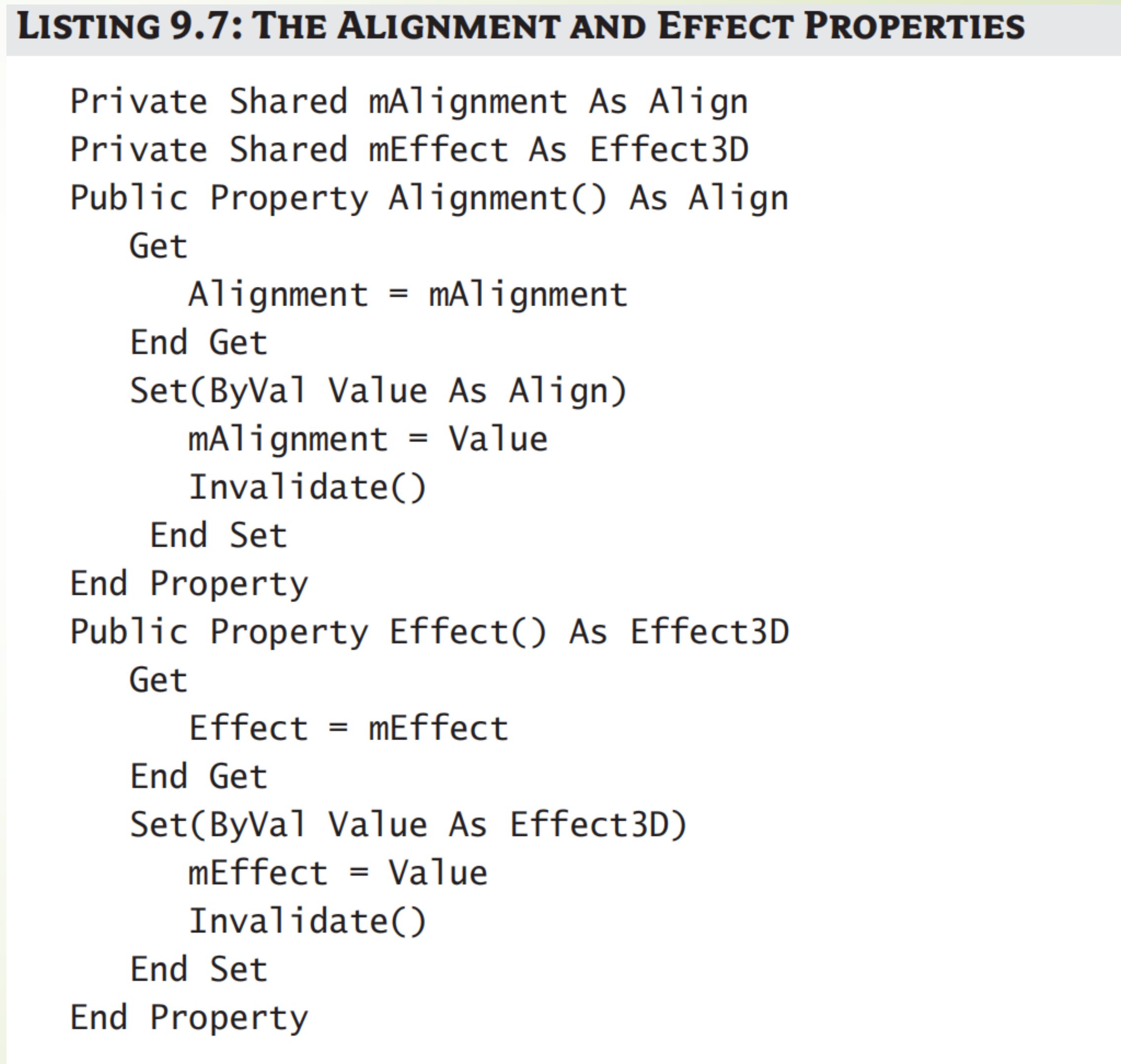


LISTING 9.6: THE ALIGN AND EFFECT3D ENUMERATIONS

```
Public Enum Align
   TopLeft
   TopMiddle
   TopRight
   CenterLeft
   CenterMiddle
   CenterRight
   BottomLeft
   BottomMiddle
   BottomRight
End Enum
Public Enum Effect3D
   None
   Raised
   RaisedHeavy
   Carved
   CarvedHeavy
End Enum
```

Designing the Custom Control

- To implement the Alignment and Effect properties

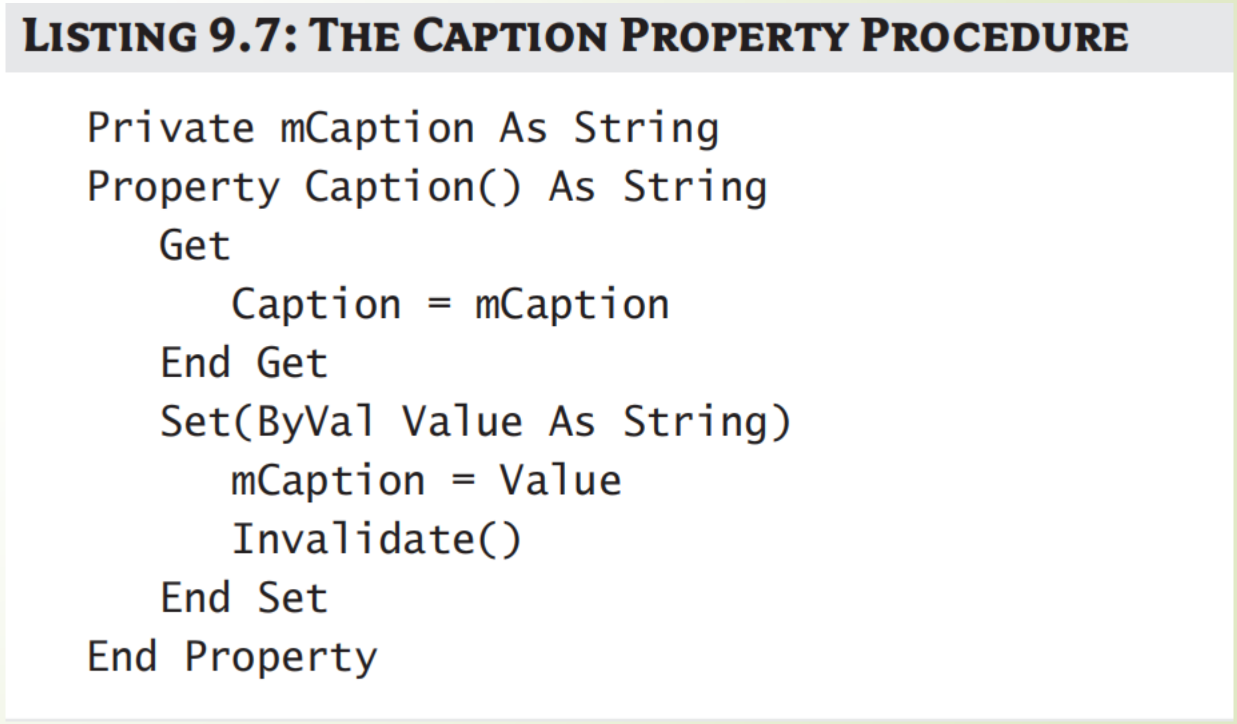


LISTING 9.7: THE ALIGNMENT AND EFFECT PROPERTIES

```
Private Shared mAlignment As Align
Private Shared mEffect As Effect3D
Public Property Alignment() As Align
   Get
      Alignment = mAlignment
   End Get
   Set(ByVal Value As Align)
      mAlignment = Value
      Invalidate()
    End Set
End Property
Public Property Effect() As Effect3D
   Get
      Effect = mEffect
   End Get
   Set(ByVal Value As Effect3D)
      mEffect = Value
      Invalidate()
   End Set
End Property
```

Designing the Custom Control

- To implement the Caption property



LISTING 9.7: THE CAPTION PROPERTY PROCEDURE

```
Private mCaption As String
Property Caption() As String
   Get
      Caption = mCaption
   End Get
   Set(ByVal Value As String)
      mCaption = Value
      Invalidate()
   End Set
End Property
```

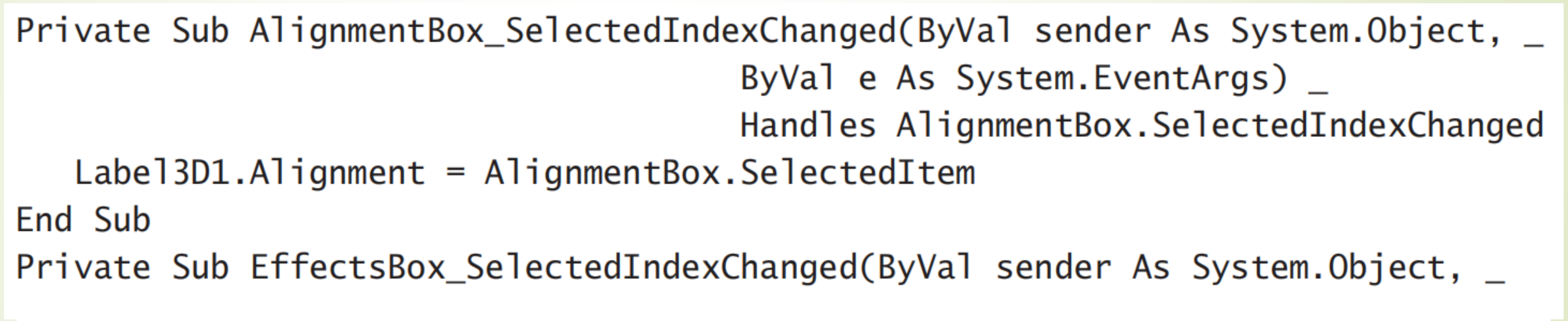
Test your New Control

- first add it to the Toolbox
- Add the TestProject to the current solution
- And Rename its Form to TestForm

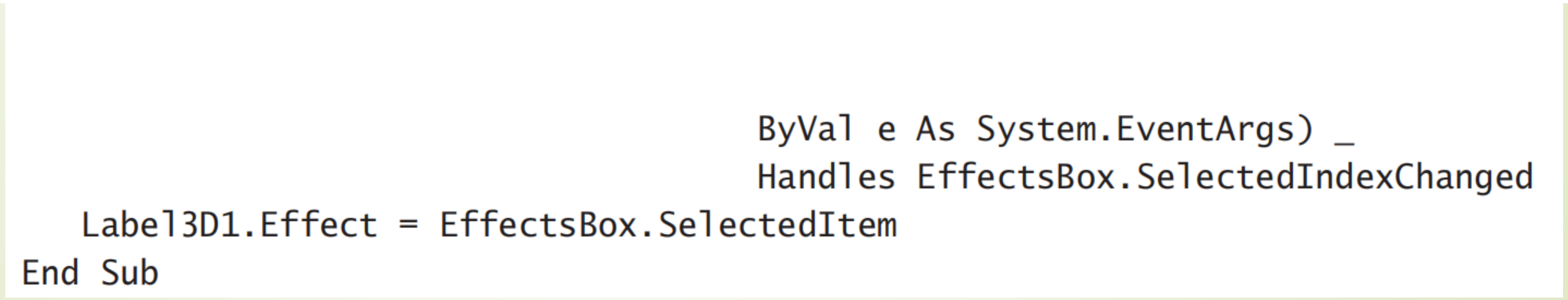
```
Private Sub Label3D1_Click(ByVal sender As Object, _
               ByVal e As System.EventArgs) Handles Label3D1.Click
    MsgBox("My properties are " & vbCrLf & _
           "Caption = " & Label3D1.Caption & vbCrLf & _
           "Alignment = " & Label3D1.Alignment & vbCrLf & _
           "Effect = " & Label3D1.Effect)
End Sub
```

Test your New Control

- set the corresponding property to the selected value



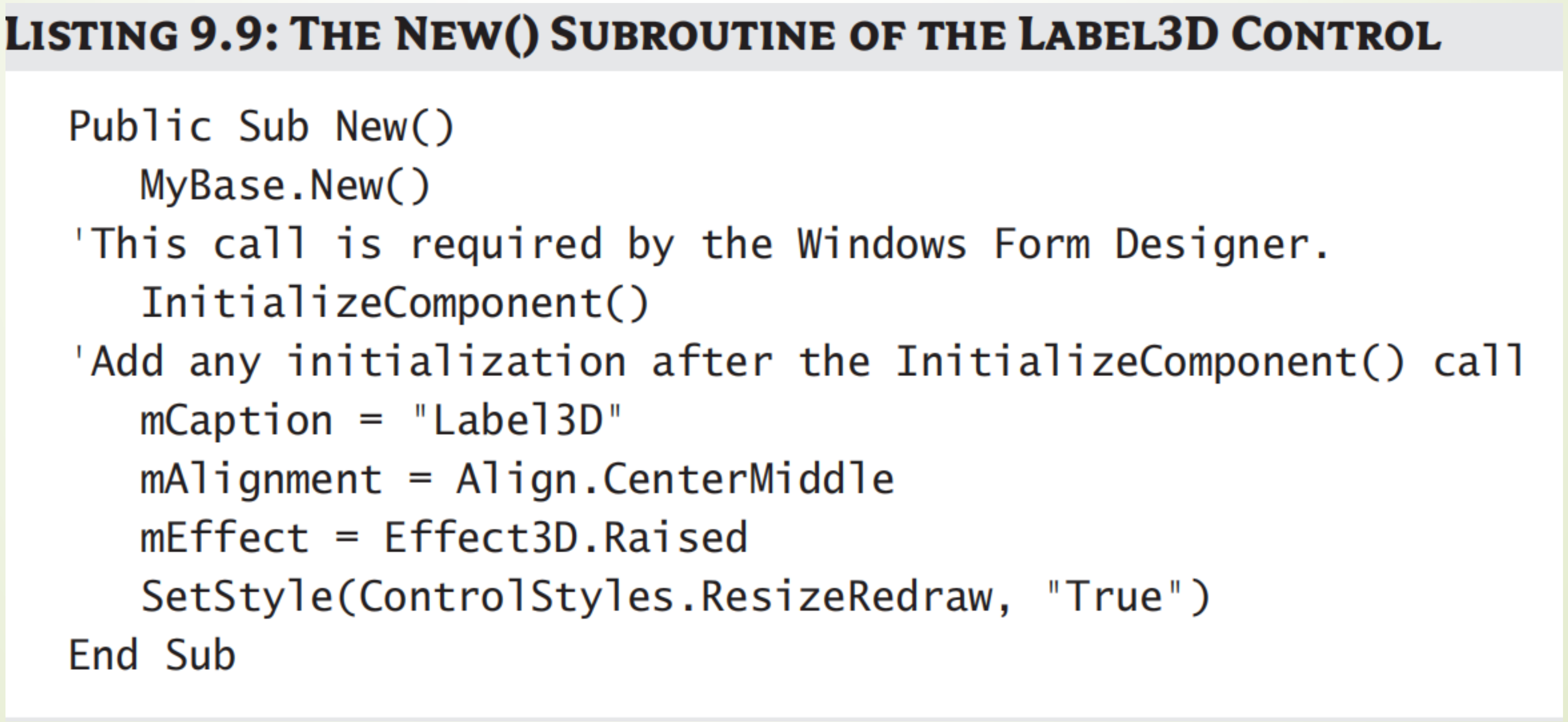
```
Private Sub AlignmentBox_SelectedIndexChanged(ByVal sender As System.Object, _
                                     ByVal e As System.EventArgs) _
                                     Handles AlignmentBox.SelectedIndexChanged
   Label3D1.Alignment = AlignmentBox.SelectedItem
End Sub
Private Sub EffectsBox_SelectedIndexChanged(ByVal sender As System.Object, _
```



```
                                     ByVal e As System.EventArgs) _
                                     Handles EffectsBox.SelectedIndexChanged
   Label3D1.Effect = EffectsBox.SelectedItem
End Sub
```

Initializing a Custom Control

- insert the appropriate code in the New() subroutine

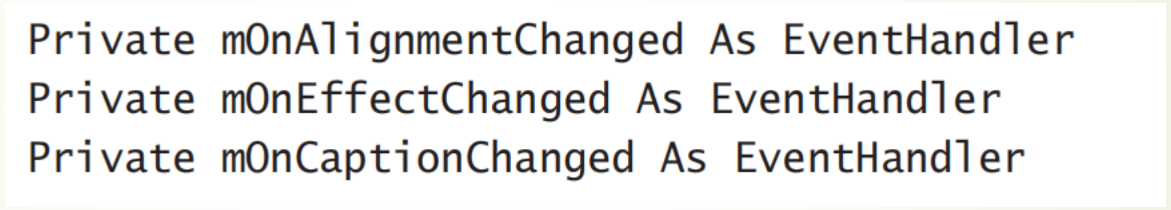


LISTING 9.9: THE NEW() SUBROUTINE OF THE LABEL3D CONTROL

```
Public Sub New()
   MyBase.New()
'This call is required by the Windows Form Designer.
   InitializeComponent()
'Add any initialization after the InitializeComponent() call
   mCaption = "Label3D"
   mAlignment = Align.CenterMiddle
   mEffect = Effect3D.Raised
   SetStyle(ControlStyles.ResizeRedraw, "True")
End Sub
```

The Changed Events

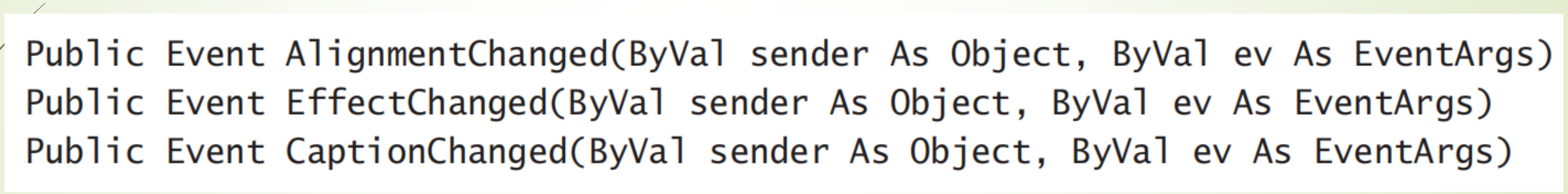
- Declare an event handler for each of the Changed events



```
Private mOnAlignmentChanged As EventHandler
Private mOnEffectChanged As EventHandler
Private mOnCaptionChanged As EventHandler
```

The Changed Events

- Then declare the actual events and their handlers



```
Public Event AlignmentChanged(ByVal sender As Object, ByVal ev As EventArgs)
Public Event EffectChanged(ByVal sender As Object, ByVal ev As EventArgs)
Public Event CaptionChanged(ByVal sender As Object, ByVal ev As EventArgs)
```

The Changed Events

- finally invoke the event handlers from within the appropriate On*EventName* method

```
Protected Overridable Sub OnAlignmentChanged(ByVal E As EventArgs)
   Invalidate()
   If Not (mOnAlignmentChanged Is Nothing) Then mOnAlignmentChanged.Invoke(Me, E)
End Sub
Protected Overridable Sub OnEffectChanged(ByVal E As EventArgs)
   Invalidate()
   If Not (mOnEffectChanged Is Nothing) Then mOnEffectChanged.Invoke(Me, E)
End Sub
Protected Overridable Sub OnCaptionChanged(ByVal E As EventArgs)
   Invalidate()
   If Not (mOnCaptionChanged Is Nothing) Then mOnCaptionChanged.Invoke(Me, E)
End Sub
```

Exercise

- (1) What does “TopLeft” mean in the setting of the Effect property?
- (2) What does “Carved” mean in the setting of the Effect property?