

Chapter 12 - Graphical User Interface Concepts: Part 1

Outline

- 12.1 Introduction
- 12.2 Windows Forms
- 12.3 Event-Handling Model
 - 12.3.1 Basic Event Handling
- 12.4 Control Properties and Layout
- 12.5 Labels, TextBoxes and Buttons
- 12.6 GroupBoxes and Panels
- 12.7 CheckBoxes and RadioButtons
- 12.8 PictureBoxes
- 12.9 Mouse Event Handling
- 12.10 Keyboard Event Handling



12.1 Introduction

- Graphical user interface
 - Allow interaction with program visually
 - Give program distinct look and feel
 - Built from window gadgets
 - Is an object, accessed via keyboard or mouse



12.1 Introduction

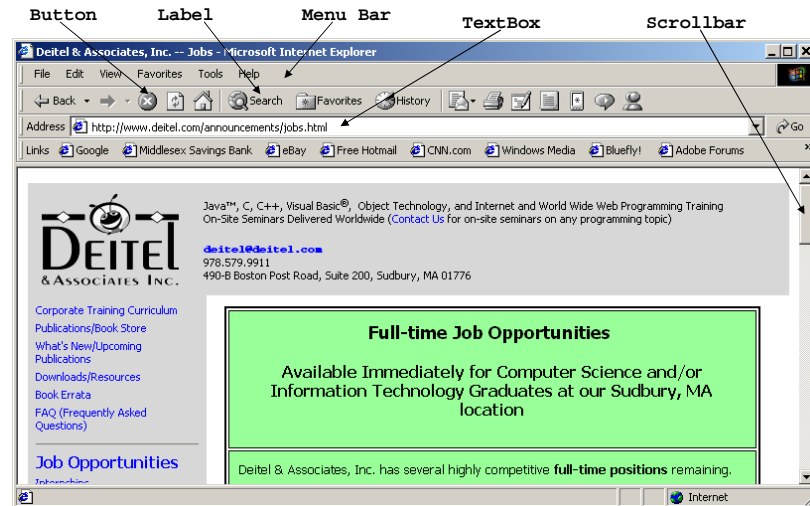


Fig. 12.1 Sample Internet Explorer window with GUI components.



12.1 Introduction

Control	Description
Label	An area in which icons or uneditable text can be displayed.
TextBox	An area in which the user inputs data from the keyboard. The area also can display information.
Button	An area that triggers an event when clicked.
CheckBox	A GUI control that is either selected or not selected.
ComboBox	A drop-down list of items from which the user can make a selection, by clicking an item in the list or by typing into the box, if permitted.
ListBox	An area in which a list of items is displayed from which the user can make a selection by clicking once on any element. Multiple elements can be selected.
Panel	A container in which components can be placed.
ScrollBar	Allows the user to access a range of values that cannot normally fit in its container.

Fig. 12.2 Some basic GUI components.



12.2 Windows Forms

- WinForms
 - Create GUIs for programs
 - Element on the desktop
 - Represented by:
 - Dialog
 - Window
 - MDI window



12.2 Windows Forms

- Component
 - Class that implements IComponent interface
 - Lacks visual parts
- Control
 - Component with graphical part
 - Such as button or label
 - Are visible
- Event
 - Generated by movement from mouse or keyboard
 - Event handlers performs action
 - Specifics written by programmer



12.2 Windows Forms

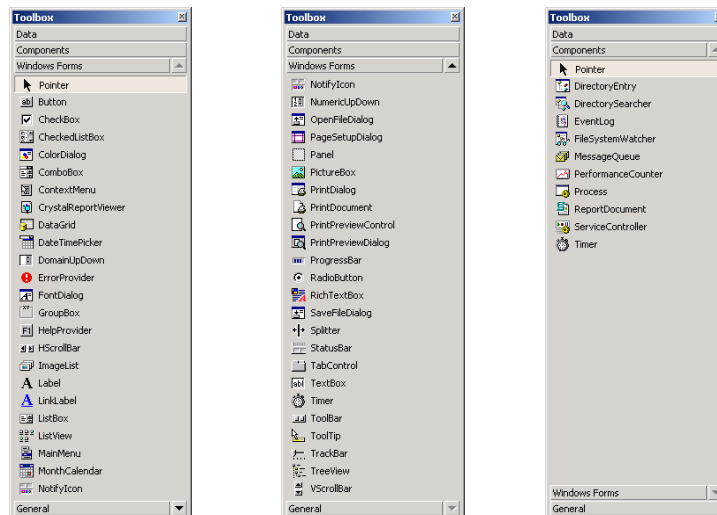


Fig. 12.3 Components and controls for Windows Forms.

© 2002 Prentice Hall. All rights reserved.



12.2 Windows Forms

Form Properties and Events	Description / Delegate and Event Arguments
<i>Common Properties</i>	
AcceptButton	Which button will be clicked when <i>Enter</i> is pressed.
AutoScroll	Whether scrollbars appear when needed (if data fills more than one screen).
CancelButton	Button that is clicked when the <i>Escape</i> key is pressed.
FormBorderStyle	Border of the form (e.g., none , single , 3D , sizable).
Font	Font of text displayed on the form, as well as the default font of controls added to the form.
Text	Text in the form's title bar.
<i>Common Methods</i>	
Close	Closes form and releases all resources. A closed form cannot be reopened.
Hide	Hides form (does not release resources).
Show	Displays a hidden form.
<i>Common Events</i>	<i>(Delegate EventHandler, event arguments EventArgs)</i>
Load	Occurs before a form is shown. This event is the default when the form is double-clicked in the Visual Studio .NET designer.

Fig. 12.4 Common Form properties and events.

© 2002 Prentice Hall. All rights reserved.



12.3 Event-Handling Model

- GUIs are event driven
- Event handlers
 - Methods that process events and perform tasks.
- Associated delegate
 - Objects that reference methods
 - Contain lists of method references
 - Must have same signature
 - Intermediaries for objects and methods
 - Signature for control's event handler



12.3 Event-Handling Model

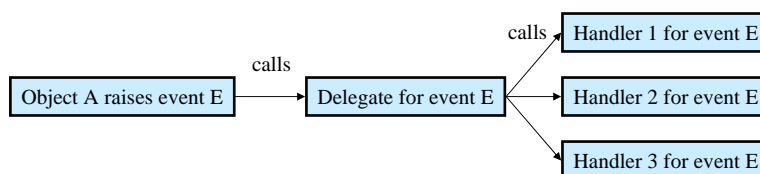


Fig. 12.5 Event-handling model using delegates.



12.3.1 Basic Event Handling

- Event handler
 - Must have same signature as corresponding delegate
 - Two object reference are passed in
 - ControlName_EventName
 - Must be registered with delegate object
 - Add event handlers to the delegate's invocation list
 - New delegate object for each event handler
- Event multicasting
 - Have multiple handlers for one event
 - Order called for event handlers is indeterminate

© 2002 Prentice Hall. All rights reserved.



12.3.1 Basic Event Handling

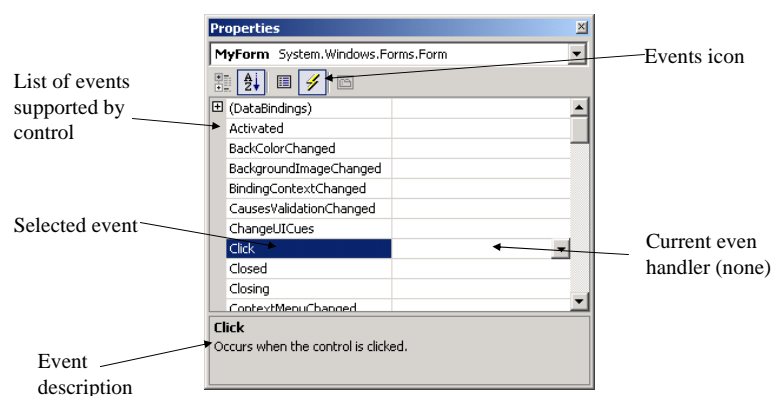


Fig. 12.6 Events section of the **Properties** window.

© 2002 Prentice Hall. All rights reserved.



13
Outline
SimpleEventExample.cs

```

1 // Fig. 12.7: SimpleEventExample.cs
2 // Using Visual Studio .NET to create event handlers.
3
4 using System;
5 using System.Drawing;
6 using System.Collections;
7 using System.ComponentModel;
8 using System.Windows.Forms;
9 using System.Data;
10
11 // program that shows a simple event handler
12 public class MyForm : System.Windows.Forms.Form
13 {
14     private System.ComponentModel.Container components = null;
15
16     // Visual Studio .NET generated code
17
18     [STAThread]
19     static void Main()
20     {
21         Create an event handler new MyForm() );
22     }
23
24     // Visual Studio .NET creates an empty handler,
25     // we write definition: show message box when form clicked
26     private void MyForm_Click( object sender, System.EventArgs e )
27     {
28         Reference to raised the ev Reference to an event
29         MessageBox.Show( "Form was pressed" );
30     }
31 } // end class MyForm

```

Create an event handler

Class EventArgs is base class for objects with event information

Reference to raised the ev

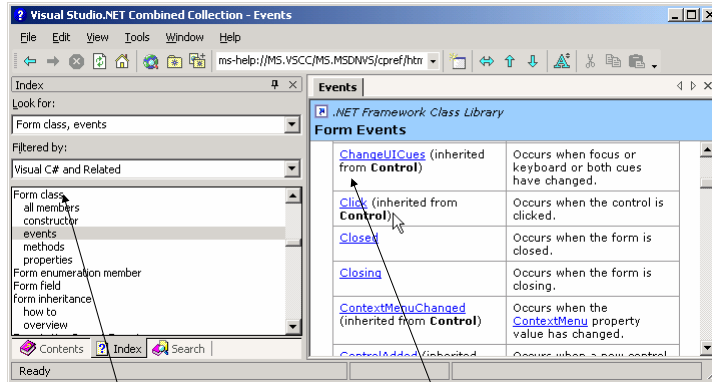
Reference to an event arguments object (e)

© 2002 Prentice Hall.
 All rights reserved.

14
Outline
SimpleEventExample.cs
Program Output

© 2002 Prentice Hall.
 All rights reserved.

12.3.1 Basic Event Handling



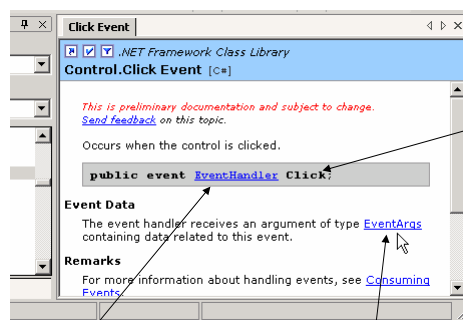
Class name

List of events

Fig. 12.8 List of Form events.



12.3.1 Basic Event Handling



Event name

Event delegate

Event argument class

Fig. 12.9 Details of Click event.



12.4 Control Properties and Layout

- Common properties
 - Derive from class Control
 - Text property
 - Specifies the text that appears on a control
 - Focus method
 - Transfers the focus to a control
 - Becomes active control
 - TabIndex property
 - Order in which controls are given focus
 - Automatically set by Visual Studio .NET
 - Enable property
 - Indicate a control's accessibility



12.4 Control Properties and Layout

- Visibility control
 - Hide control from user
 - Or use method Hide
- Anchor property
 - Anchoring control to specific location
 - Constant distance from specified location
 - Unanchored control moves relative to the position
 - Docking allows control to spread itself along an entire side
 - Both options refer to the parent container
- Size structure
 - Allow for specifying size range
 - MinimumSize and MaximumSize property



12.4 Control Properties and Layout

Class Control Properties and Methods	Description
<i>Common Properties</i>	
BackColor	Background color of the control.
BackgroundImage	Background image of the control.
Enabled	Whether the control is enabled (i.e., if the user can interact with it). A disabled control will still be displayed, but "grayed-out"—portions of the control will become gray.
Focused	Whether a control has focus. (The control that is currently being used in some way.)
Font	Font used to display control's Text .
ForeColor	Foreground color of the control. This is usually the color used to display the control's Text property.
TabIndex	Tab order of the control. When the Tab key is pressed, the focus is moved to controls in increasing tab order. This order can be set by the programmer.
TabStop	If true , user can use the Tab key to select the control.
Text	Text associated with the control. The location and appearance varies with the type of control.
TextAlign	The alignment of the text on the control. One of three horizontal positions (left, center or right) and one of three vertical positions (top, middle or bottom).
Visible	Whether the control is visible.
<i>Common Methods</i>	
Focus	Transfers the focus to the control.
Hide	Hides the control (sets Visible to false).
Show	Shows the control (sets Visible to true).

Fig. 12.10 Class **Control** properties and methods.

© 2002 Prentice Hall. All rights reserved.



12.4 Control Properties and Layout

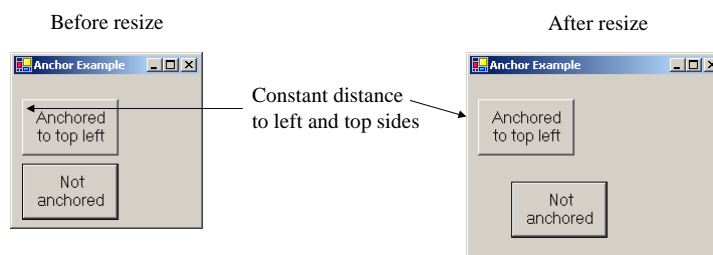


Fig. 12.11 Anchoring demonstration.

© 2002 Prentice Hall. All rights reserved.



12.4 Control Properties and Layout

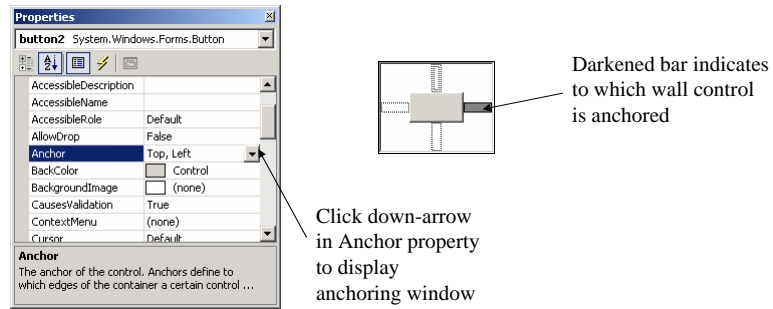


Fig. 12.12 Manipulating the **Anchor** property of a control.

© 2002 Prentice Hall. All rights reserved.



12.4 Control Properties and Layout

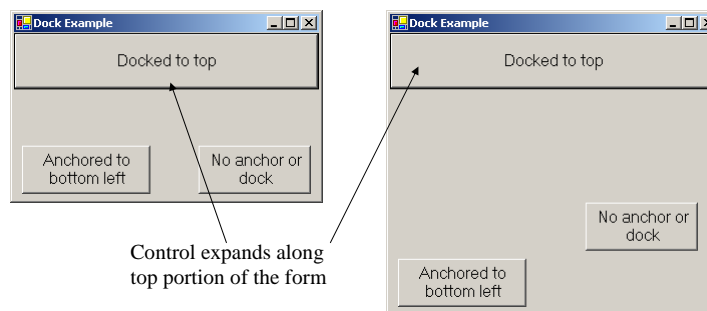


Fig. 12.13 Docking demonstration.

© 2002 Prentice Hall. All rights reserved.



12.4 Control Properties and Layout

Common Layout Properties	Description
<i>Common Properties</i>	
Anchor	Side of parent container at which to anchor control—values can be combined, such as Top , Left .
Dock	Side of parent container to dock control—values cannot be combined.
DockPadding (for containers)	Sets the dock spacing for controls inside the container. Default is zero, so controls appear flush against the side of the container.
Location	Location of the upper-left corner of the control, relative to it's container.
Size	Size of the control. Takes a Size structure, which has properties Height and Width .
MinimumSize , MaximumSize (for Windows Forms)	The minimum and maximum size of the form.

Fig. 12.14 Class `Control` layout properties



12.5 Labels, TextBoxes and Buttons

- Labels
 - Provide text instruction
 - Read only text
 - Defined with class `Label1`
 - Derived from class `Control`
- Textbox
 - Class `TextBox`
 - Area for text input
 - Password textbox



12.5 Labels, TextBoxes and Buttons

- Button
 - Control to trigger a specific action
 - Checkboxes or radio buttons
 - Derived from ButtonBase



12.5 Labels, TextBoxes and Buttons

Label Properties	Description / Delegate and Event Arguments
<i>Common Properties</i>	
Font	The font used by the text on the Label .
Text	The text to appear on the Label .
TextAlign	The alignment of the Label 's text on the control. One of three horizontal positions (left , center or right) and one of three vertical positions (top , middle or bottom).

Fig. 12.15 Label properties.



12.5 Labels TextBoxes and Buttons

TextBox Properties and Events	Description / Delegate and Event Arguments
<i>Common Properties</i>	
AcceptsReturn	If true , pressing <i>Enter</i> creates a new line if textbox spans multiple lines. If false , pressing <i>Enter</i> clicks the default button of the form.
Multiline	If true , textbox can span multiple lines. Default is false .
PasswordChar	Single character to display instead of typed text, making the TextBox a password box. If no character is specified, Textbox displays the typed text.
ReadOnly	If true , TextBox has a gray background and its text cannot be edited. Default is false .
ScrollBars	For multiline textboxes, indicates which scrollbars appear (none , horizontal , vertical or both).
Text	The text to be displayed in the text box.
<i>Common Events</i>	(<i>Delegate EventHandler, event arguments EventArgs</i>)
TextChanged	Raised when text changes in TextBox (the user added or deleted characters). Default event when this control is double clicked in the designer.

Fig. 12.16 TextBox properties and events.





12.5 Labels TextBoxes and Buttons

Button properties and events	Description / Delegate and Event Arguments
<i>Common Properties</i>	
Text	Text displayed on the Button face.
<i>Common Events</i>	(<i>Delegate EventHandler, event arguments EventArgs</i>)
Click	Raised when user clicks the control. Default event when this control is double clicked in the designer.

Fig. 12.17 Button properties and events.



29

 **Outline**

LabelTextBoxButtonTest.cs

```

1 // Fig. 12.18: LabelTextBoxButtonTest.cs
2 // Using a Textbox, Label and Button to display
3 // the hidden text in a password box.
4
5 using System;
6 using System.Drawing;
7 using System.Collections;
8 using System.ComponentModel;
9 using System.Windows.Forms;
10 using System.Data;
11
12 // namespace contains our form to display hidden text
13 namespace LabelTextBoxButtonTest
14 {
15     /// <summary>
16     /// form that creates a password textbox and
17     /// a label to display textbox contents
18     /// </summary>
19     public class LabelTextBoxButtonTest :
20         System.Windows.Forms.Form
21     {
22         private System.Windows.Forms.Button displayPasswordButton;
23         private System.Windows.Forms.Label displayPasswordLabel;
24         private System.Windows.Forms.TextBox inputPasswordTextBox;
25
26         /// <summary>
27         /// Required designer variable.
28         /// </summary>
29         private System.ComponentModel.IContainer components = null;
30
31         // default constructor
32         public LabelTextBoxButtonTest()
33         {
34             InitializeComponent();
35         }

```

Visual Studio .NET adds comments to our code

The IDE manages these declarations for the control we added to the form



Declare reference

Reference is null (array)

Method **InitializeComponent** creates components and controls in the form and sets their properties

© 2002 Prentice Hall. All rights reserved.

30

 **Outline**

LabelTextBoxButtonTest.cs

```

36
37     /// <summary>
38     /// Clean up any resources being used.
39     /// </summary>
40     protected override void Dispose( bool disposing )
41     {
42         if ( disposing )
43         {
44             if ( components != null )
45             {
46                 components.Dispose();
47             }
48         }
49         base.Dispose( disposing );
50     }
51
52     #region Windows Form Designer generated code
53     /// <summary>
54     /// Required method for Designer support - do not modify
55     /// the contents of this method with the code editor.
56     /// </summary>
57     private void InitializeComponent()
58     {
59         {
60             this.displayPasswordButton =
61                 new System.Windows.Forms.Button();
62             this.inputPasswordTextBox =
63                 new System.Windows.Forms.TextBox();
64             this.displayPasswordLabel =
65                 new System.Windows.Forms.Label();
66             this.SuspendLayout();
67         }

```



Method **Dispose** cleans up allocated resources

#region preprocessor directives allow collapsible code in Visual Studio .NET

Create new objects for the control we added

© 2002 Prentice Hall. All rights reserved.

31

 **Outline**
 **LabelTextBoxButtonTest.cs**

```

68         //
69         // displayPasswordButton
70         //
71         this.displayPasswordButton.Location =
72             new System.Drawing.Point( 96, 96 );
73         this.displayPasswordButton.Name =
74             "displayPasswordButton";
75         this.displayPasswordButton.TabIndex = 1;
76         this.displayPasswordButton.Text = "Show Me";
77         this.displayPasswordButton.Click +=
78             new System.EventHandler(
79                 this.displayPasswordButton_Click );
80
81         //
82         // inputPasswordTextBox
83         //
84         this.inputPasswordTextBox.Location =
85             new System.Drawing.Point( 16, 16 );
86         this.inputPasswordTextBox.Name =
87             "inputPasswordTextBox";
88         this.inputPasswordTextBox.PasswordChar = '*';
89         this.inputPasswordTextBox.Size =
90             new System.Drawing.Size( 264, 20 );
91         this.inputPasswordTextBox.TabIndex = 0;
92         this.inputPasswordTextBox.Text = "";
93
94         //
95         // displayPasswordLabel
96         //
97         this.displayPasswordLabel.BorderStyle =
98             System.Windows.Forms.BorderStyle.Fixed3D;
99         this.displayPasswordLabel.Location =
100            new System.Drawing.Point( 16, 48 );
101         this.displayPasswordLabel.Name =
102            "displayPasswordLabel";



```

Visual Studio .NET register
the event handler for us

Set the Name, PasswordChar
and Text properties for
inputPasswordTextBox

© 2002 Prentice Hall.
All rights reserved.

32

 **Outline**
 **LabelTextBoxButtonTest.cs**

```

103         this.displayPasswordLabel.Size =
104             new System.Drawing.Size( 264, 23 );
105         this.displayPasswordLabel.TabIndex = 2;
106
107         //
108         // LabelTextBoxButtonTest
109         //
110         this.AutoScaleBaseSize =
111             new System.Drawing.Size( 5, 13 );
112         this.ClientSize =
113             new System.Drawing.Size( 292, 133 );
114         this.Controls.AddRange(
115             new System.Windows.Forms.Control[] {
116                 this.displayPasswordLabel,
117                 this.inputPasswordTextBox,
118                 this.displayPasswordButton});
119         this.Name = "LabelTextBoxButtonTest";
120         this.Text = "LabelTextBoxButtonTest";
121         this.ResumeLayout( false );
122
123     } // end method InitializeComponent
124
125     // end collapsible region started on line 124
126     #endregion
127
128     /// <summary>
129     /// The main entry point for the application.
130     /// </summary>
131     [STAThread]
132     static void Main()
133     {
134         Application.Run( new LabelTextBoxButtonTest() );
135     }
136

```

#endregion signal the end
of the collapsible region

© 2002 Prentice Hall.
All rights reserved.

Outline 33

```

137 // display user input on label
138 protected void displayPasswordButton_Click(
139     object sender, System.EventArgs e )
140 {
141     // text has not changed
142     displayPasswordLabel.Text =
143         inputPasswordTextBox.Text;
144 }
145 } // end class LabelTextBoxButtonTest
146 } // end namespace LabelTextBoxButtonTest

```

To show the text set
User must program this line manually
inputPasswordTextBox's Text
Create an empty event handler named
displayPasswordButton_Click

Program Output

© 2002 Prentice Hall. All rights reserved.

34

12.6 GroupBoxes and Panels

- Arrange components on a GUI
 - **GroupBoxes** can display a caption
 - Text property determines its caption
 - **Panels** can have scrollbar
 - View additional controls inside the Panel

© 2002 Prentice Hall. All rights reserved.

12.6 GroupBoxes and Panels

GroupBox Properties	Description
<i>Common Properties</i>	
Controls	The controls that the GroupBox contains.
Text	Text displayed on the top portion of the GroupBox (its caption).
Fig. 12.19 GroupBox properties	



12.6 GroupBoxes and Panels

Panel Properties	Description
<i>Common Properties</i>	
AutoScroll	Whether scrollbars appear when the Panel is too small to hold its controls. Default is false .
BorderStyle	Border of the Panel (default None ; other options are Fixed3D and FixedSingle).
Controls	The controls that the Panel contains.
Fig. 12.20 Panel properties	



12.6 GroupBoxes and Panels

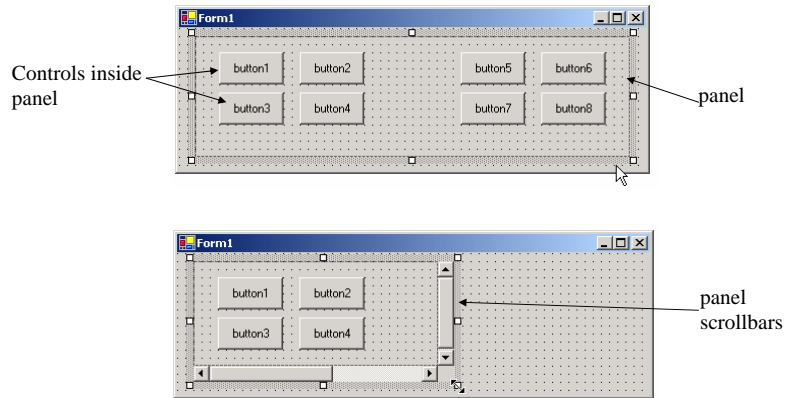


Fig. 12.21 Creating a Panel with scrollbars.

© 2002 Prentice Hall. All rights reserved.



```

1  // Fig. 12.22: GroupBoxPanelExample.cs
2  // Using GroupBoxes and Panels to hold buttons.
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11  /// form to display a groupbox versus a panel
12  public class GroupBoxPanelExample : System.Windows.Forms.Form
13  {
14      private System.Windows.Forms.Button hiButton;
15      private System.Windows.Forms.Button byeButton;
16      private System.Windows.Forms.Button leftButton;
17      private System.Windows.Forms.Button rightButton;
18
19      private System.Windows.Forms.GroupBox mainGroupBox;
20      private System.Windows.Forms.Label messageLabel;
21      private System.Windows.Forms.Panel mainPanel;
22
23      private System.ComponentModel.Container components = null;
24
25      // Visual Studio .NET-generated Dispose method
26
27      [STAThread]
28      static void Main()
29      {
30          Application.Run( new GroupBoxPanelExample() );
31      }
32

```

GroupBox (name mainGroupBox)

Panel (name mainPanel)

Control AutoScroll

property set to TRUE

messageLabel is initially blank



Outline

GroupBoxPanelExample.cs

© 2002 Prentice Hall.
All rights reserved.

39

Outline

GroupBoxPanelExample.cs

```

33 // event handlers to change messageLabel
34
35 // event handler for hi button
36 private void hiButton_Click(
37     object sender, System.EventArgs e )
38 {
39     messageLabel.Text= "Hi pressed";
40 }
41
42 // event handler for bye button
43 private void byeButton_Click(
44     object sender, System.EventArgs e )
45 {
46     messageLabel.Text = "Bye pressed";
47 }
48
49 // event handler for far left button
50 private void leftButton_Click(
51     object sender, System.EventArgs e )
52 {
53     messageLabel.Text = "Far left pressed";
54 }
55
56 // event handler for far right button
57 private void rightButton_Click(
58     object sender, System.EventArgs e )
59 {
60     messageLabel.Text = "Far right pressed";
61 }
62
63 } // end class GroupBoxPanelExample

```

hiButton and byeButton belong to GroupBox

Represent event handlers

Line messageLabel added to customize the text

Panel has two buttons, leftButton and rightButton

© 2002 Prentice Hall.
All rights reserved.

40

Outline

GroupBoxPanelExample.cs
Program Output

hiButton_Click

leftButton_Click

rightButton_Click

© 2002 Prentice Hall.
All rights reserved.

12.7 Checkboxes and RadioButtons

- State buttons
 - On/off or true/false state
 - Derived from class ButtonBase
 - CheckBox
 - No restriction on usage
 - RadioButton
 - Grouped together
 - Only one can be true
 - Mutually exclusive options



12.7 CheckBoxes and RadioButtons

CheckBox events and properties	Description / Delegate and Event Arguments
<i>Common Properties</i>	
Checked	Whether or not the CheckBox has been checked.
CheckState	Whether the Checkbox is checked (contains a black checkmark) or unchecked (blank). An enumeration with values Checked , Unchecked or Indeterminate .
Text	Text displayed to the right of the CheckBox (called the label).
<i>Common Events</i>	<i>(Delegate EventHandler, event arguments EventArgs)</i>
CheckedChanged	Raised every time the Checkbox is either checked or unchecked. Default event when this control is double clicked in the designer.
CheckStateChanged	Raised when the CheckState property changes.

Fig. 12.23 **CheckBox** properties and events.



43

Outline

CheckBoxTest.cs

```

1 // Fig. 12.24: CheckBoxTest.cs
2 // Using CheckBoxes to toggle italic and bold styles.
3
4 using System;
5 using System.Drawing;
6 using System.Collections;
7 using System.ComponentModel;
8 using System.Windows.Forms;
9 using System.Data;
10
11 /// form contains checkboxes to allow
12 /// the user to modify sample text
13 public class CheckBoxTest : System.Windows.Forms.Form
14 {
15     private System.Windows.Forms.CheckBox boldCheckBox;
16     private System.Windows.Forms.CheckBox italicCheckBox;
17
18     private System.Windows.Forms.Label outputLabel;
19
20     private System.ComponentModel.Container components = null;
21
22     // Visual Studio .NET-generated Dispose method
23
24     /// The main entry point for the application.
25     [STAThread]
26     static void Main()
27     {
28         Application.Run( new CheckBoxTest() );
29     }
30

```

When program start, both
Checkbox is unchecked
Text property set to Bold

The Label OutputLabel is labeled
Watch the font style change

© 2002 Prentice Hall.
All rights reserved.

44

Outline

CheckBoxTest.cs

```

31 // make text bold if not bold,
32 // if already bold make not bold
33 private void boldCheckBox_CheckedChanged(
34     object sender, System.EventArgs e )
35 {
36     outputLabel.Font =
37         new Font( outputLabel.Font.Name,
38                 outputLabel.Font.Size,
39                 outputLabel.Font.Style ^ FontStyle.Bold );
40 }
41
42 // Style property is read
43 // Style can use bitwise operators
44 // Style is a member of the
45 // Style enumeration
46 // set when object is created
47 private void italicCheckBox_CheckedChanged(
48     object sender, System.EventArgs e )
49 {
50     outputLabel.Font =
51         new Font( outputLabel.Font.Name,
52                 outputLabel.Font.Size,
53                 outputLabel.Font.Style ^ FontStyle.Italic );
54 }
55 } // end class CheckBoxTest

```

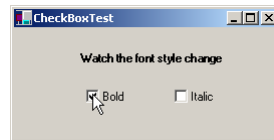
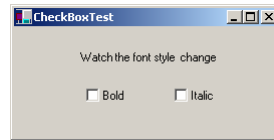
Font constructor takes in the
font name, size, and style

Style is a member of the
Style enumeration

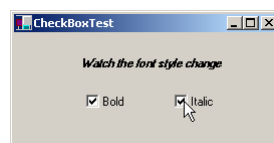
Style can use bitwise operators
set when object is created

Style property is read
Style is read

© 2002 Prentice Hall.
All rights reserved.



Result when bold is selected



Result when both styles are selected

12.7 CheckBoxes and RadioButtons

RadioButton properties and events	Description / Delegate and Event Arguments
<i>Common Properties</i>	
Checked	Whether the RadioButton is checked.
Text	Text displayed to the right of the RadioButton (called the label).
<i>Common Events</i>	<i>(Delegate EventHandler, event arguments EventArgs)</i>
Click	Raised when user clicks the control.
CheckedChanged	Raised every time the RadioButton is checked or unchecked. Default event when this control is double clicked in the designer.

Fig. 12.25 [RadioButton properties and events](#)



47

Outline

RadioButtonsTest
.cs

```

1 // Fig. 12.26: RadioButtonsTest.cs
2 // Using RadioButtons to set message window options.
3
4 using System;
5 using System.Drawing;
6 using System.Collections;
7 using System.ComponentModel;
8 using System.Windows.Forms;
9 using System.Data;
10
11 /// form contains several radio buttons--user chooses one
12 /// from each group to create a custom MessageBox
13 public class RadioButtonsTest : System.Windows.Forms.Form
14 {
15     private System.Windows.Forms.Label promptLabel;
16     private System.Windows.Forms.Label displayLabel;
17     private System.Windows.Forms.Button displayButton;
18
19     private System.Windows.Forms.RadioButton questionButton;
20     private System.Windows.Forms.RadioButton informationButton;
21     private System.Windows.Forms.RadioButton exclamationButton;
22     private System.Windows.Forms.RadioButton errorButton;
23     private System.Windows.Forms.RadioButton retryCancelButton;
24     private System.Windows.Forms.RadioButton yesNoButton;
25     private System.Windows.Forms.RadioButton yesNoCancelButton;
26     private System.Windows.Forms.RadioButton okCancelButton;
27     private System.Windows.Forms.RadioButton abortRetryIgnoreButton;
28     private System.Windows.Forms.GroupBox groupBox2;
29     private System.Windows.Forms.GroupBox groupBox1;
30
31     private System.Windows.Forms.GroupBox groupBox2;
32     private System.Windows.Forms.GroupBox groupBox1;
33
34     private MessageBoxIcon iconType = MessageBoxIcon.Error;

```

Label is used to prompt user

Label is used to display
which button was pressed

Display the text Display

RadioButtons are
created for the
enumeration options

To store user's choice of
options iconType is created

One event handling exists for
all the radio buttons in
groupBox1 and groupBox2
return button to display

© 2002 Prentice Hall.
All rights reserved.

48

Outline

RadioButtonsTest
.cs

```

35 private MessageBoxButtons buttonType =
36     MessageBoxButtons.OK;
37
38 /// The main entry point for the application
39 [STAThread]
40 static void Main()
41 {
42     Application.Run( new RadioButtonsTest() );
43 }
44
45 // change button based on option chosen by sender
46 private void buttonType_CheckedChanged(
47     object sender, System.EventArgs e )
48 {
49     if ( sender == okButton ) // display OK button
50         buttonType = MessageBoxButtons.OK;
51
52     // display OK and Cancel buttons
53     else if ( sender == okCancelButton )
54         buttonType = MessageBoxButtons.OKCancel;
55
56     // display Abort, Retry and Ignore buttons
57     else if ( sender == abortRetryIgnoreButton )
58         buttonType = MessageBoxButtons.AbortRetryIgnore;
59
60     // display Yes, No and Cancel buttons
61     else if ( sender == yesNoCancelButton )
62         buttonType = MessageBoxButtons.YesNoCancel;
63
64     // display Yes and No buttons
65     else if ( sender == yesNoButton )
66         buttonType = MessageBoxButtons.YesNo;
67
68     // only one option left--display
69     // Retry and Cancel buttons

```

The enumeration
name indicate which
button to display

buttonType is a
button enumeration

Handlers compare the sender object
with every radio button to determine
which button was selected

Each radio button generates
a CheckedChanged when
clicked

© 2002 Prentice Hall.
All rights reserved.

49

Outline

RadioButtonsTest .CS

```

70     else
71         buttonType = MessageBoxB
72     } // end method buttonType_Che
73
74
75     // change icon based on option chosen by sender
76     private void iconType_CheckedChanged(
77         object sender, System.EventArgs e )
78     {
79         if ( sender == errorButton ) // display error icon
80             iconType = MessageBoxIcon.Error;
81
82         // display exclamation point
83         else if ( sender == exclamationButton )
84             iconType = MessageBoxIcon.Exclamation;
85
86         // display information icon
87         else if ( sender == informationButton )
88             iconType = MessageBoxIcon.Information;
89
90         else // only one option left--display question mark
91             iconType = MessageBoxIcon.Question;
92     } // end method iconType_CheckedChanged
93
94
95     // display MessageBox and button user pressed
96     protected void displayButton_Click(
97         object sender, System.EventArgs e )
98     {
99
100         Custom MessageBox.",
101         iconType, 0, 0 );
102
103         // check for dialog result and display it in label
104         switch ( result )

```

Handlers compare the sender object with every radio button to determine which button was selected

Result of message box is a DialogResult enumeration

Button

Switch statement tests for the result and set displayLabel.Text appropriately

© 2002 Prentice Hall. All rights reserved.

50

Outline

RadioButtonsTest .CS

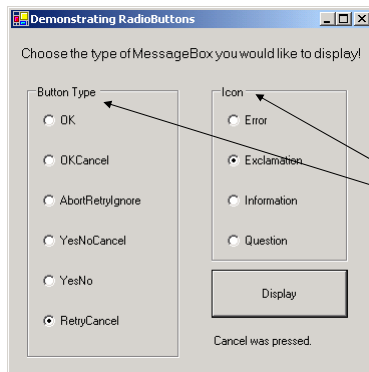
```

105     {
106         case DialogResult.OK:
107             displayLabel.Text = "OK was pressed.";
108             break;
109
110         case DialogResult.Cancel:
111             displayLabel.Text = "Cancel was pressed.";
112             break;
113
114         case DialogResult.Abort:
115             displayLabel.Text = "Abort was pressed.";
116             break;
117
118         case DialogResult.Retry:
119             displayLabel.Text = "Retry was pressed.";
120             break;
121
122         case DialogResult.Ignore:
123             displayLabel.Text = "Ignore was pressed.";
124             break;
125
126         case DialogResult.Yes:
127             displayLabel.Text = "Yes was pressed.";
128             break;
129
130         case DialogResult.No:
131             displayLabel.Text = "No was pressed.";
132             break;
133     } // end switch
134 } // end method displayButton_Click
137 } // end class RadioButtonsTest

```

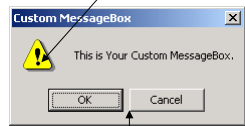
The result input will help determine which text to display among the cases

© 2002 Prentice Hall. All rights reserved.



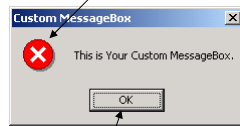
Radio button style allow user to select one per column

Exclamation icon type



OKCancel button type

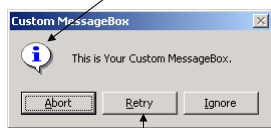
Error icon type



OK button type

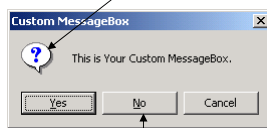


Information icon type

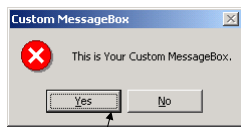


AbortRetryIgnore button type

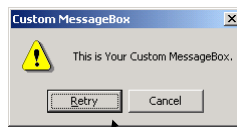
Question icon type



YesNoCancel button type



YesNo button type



RetryCancel button type

12.8 PictureBoxes

- Class `PictureBox`
 - Displays an image
 - Image set by object of class `Image`.
 - The `Image` property sets the `Image` object to use
 - `SizeMode` property sets how the image is displayed



12.8 PictureBoxes

PictureBox properties and events	Description / Delegate and Event Arguments
<i>Common Properties</i>	
Image	Image to display in the <code>PictureBox</code> .
SizeMode	Enumeration that controls image sizing and positioning. Values Normal (default), StretchImage , AutoSize and CenterImage . Normal puts image in top-left corner of <code>PictureBox</code> and CenterImage puts image in middle (both cut off image if too large). StretchImage resizes image to fit in <code>PictureBox</code> . AutoSize resizes <code>PictureBox</code> to hold image.
<i>Common Events</i>	<i>(Delegate EventHandler, event arguments EventArgs)</i>
Click	Raised when user clicks the control. Default event when this control is double clicked in the designer.

Fig. 12.27 `PictureBox` properties and events



55

Outline

PictureBoxTest.cs

```

1 // Fig. 12.28: PictureBoxTest.cs
2 // Using a PictureBox to display images.
3
4 using System;
5 using System.Drawing;
6 using System.Collections;
7 using System.ComponentModel;
8 using System.Windows.Forms;
9 using System.Data;
10 using System.IO;
11
12 /// form to display different images when clicked
13 public class PictureBoxTest : System.Windows.Forms.Form
14 {
15     private System.Windows.Forms.PictureBox imagePictureBox;
16     private System.Windows.Forms.Label promptLabel;
17
18     private int imageNum = -1;
19
20     /// The main entry point for the application.
21     [STAThread]
22     static void Main()
23     {
24         Application.Run(new PictureBoxTest());
25     }
26
27     // change image whenever PictureBox clicked
28     private void imagePictureBox_Click(
29         object sender, System.EventArgs e)
30     {
31         imageNum = ( imageNum + 1 ) % 3; // imageNum from 0 to 2
32     }
33 }

```

PictureBox imagePictureBox use to display one of three bitmap images

Includes instructions Click On Picture Box to View Images

Store the image we want display

Modulus calculation insures that number is between 0 and 2

© 2002 Prentice Hall.
All rights reserved.

56

Outline

PictureBoxTest.cs

```

33 // create Image object from file, display on PictureBox
34 imagePictureBox.Image = Image.FromFile(
35     Directory.GetCurrentDirectory() + "\\images\\image" +
36     imageNum + ".bmp" );
37 }
38
39

```

Set imageNum to append the correct number

GetCurrentDirectory of Class Directory returns current directory of file as a string

Image.FromFile takes a string and creates an Image object

Program Output

© 2002 Prentice Hall.
All rights reserved.

12.9 Mouse Event Handling

- Class `MouseEventArgs`
 - Contain coordinates of the mouse pointer
 - The mouse pressed
 - Number of clicks
 - Number of notches the wheel turned
 - Passing mouse event
 - Mouse event-handling methods take an object and `MouseEventArgs` object as argument
 - The Click event uses delegate `EventHandler` and event arguments `EventArgs`



12.9 Mouse Event Handling

Mouse Events, Delegates and Event Arguments	
<i>Mouse Events (Delegate EventHandler, event arguments EventArgs)</i>	
MouseEnter	Raised if the mouse cursor enters the area of the control.
MouseLeave	Raised if the mouse cursor leaves the area of the control.
<i>Mouse Events (Delegate MouseEventHandler, event arguments MouseEventArgs)</i>	
MouseDown	Raised if the mouse button is pressed while its cursor is over the area of the control.
MouseHover	Raised if the mouse cursor hovers over the area of the control.
MouseMove	Raised if the mouse cursor is moved while in the area of the control.
MouseUp	Raised if the mouse button is released when the cursor is over the area of the control.
<i>Class MouseEventArgs Properties</i>	
Button	Mouse button that was pressed (left , right , middle or none).
Clicks	The number of times the mouse button was clicked.
X	The x-coordinate of the event, relative to the component.
Y	The y-coordinate of the event, relative to the component.

Fig. 12.29 Mouse events, delegates and event arguments.



59

Outline

Painter.cs

```

1 // Fig 12.30: Painter.cs
2 // Using the mouse to draw on a form.
3
4 using System;
5 using System.Drawing;
6 using System.Collections;
7 using System.ComponentModel;
8 using System.Windows.Forms;
9 using System.Data;
10
11 // creates a form as a drawing surface
12 public class Painter : System.Windows.Forms.Form
13 {
14     bool shouldPaint = false; // whether to paint
15
16     // The main entry point for the application.
17     [STAThread]
18     static void Main()
19     {
20         Application.Run( new Painter() );
21     }
22
23     // should paint after mouse button has been pressed
24     private void Painter_MouseDown(
25         object sender, System.Windows.Forms.MouseEventArgs
26         e)
27     {
28         shouldPaint = true;
29     }
30
31     // stop painting when mouse button released
32     private void Painter_MouseUp(
33         object sender, System.Windows.Forms.MouseEventArgs
34         e)
35     {
36         shouldPaint = false;
37     }
38 }

```

Creates variable **shouldPaint** to determine whether to draw on the form

The event handler for event **MouseDown**

Mouse cursor will set to true draw occurs

The event handler of **MouseUp**

shouldPaint set to false, mouse cursor will not draw

© 2002 Prentice Hall. All rights reserved.

60

Outline

Painter.cs

```

36
37 // draw circle whenever mouse button
38 // moves (and mouse is down)
39 protected void Painter_MouseMove(
40     object sender, System.Windows.Forms.MouseEventArgs e)
41 {
42     if ( shouldPaint )
43     {
44         Graphics graphics = CreateGraphics();
45         graphics.FillEllipse(
46             new SolidBrush( Color.BlueViolet ),
47             e.X, e.Y, 40, 40);
48     }
49 } // end Paint
50
51 } // end class Painter
52

```

Creates the graphic object for the form **only if true**

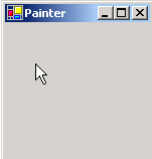
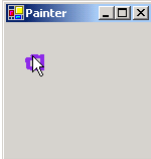

Method **FillEllipse** draws a circle at every point that **should**

Create new **SolidBrush** object by passing the constructor a **Color**

Coordinates of x and y and the pixels height and width are supplied to the parameter list

Structure **Color** contains predefined color constants

Program Output

© 2002 Prentice Hall. All rights reserved.

12.10 Keyboard Event Handling

- Key events
 - Control that inherits from System.Windows.Forms.Control
 - Delegate KeyPressEventHandler
 - Event argument KeyPressEventArgs
 - KeyPress
 - ASCII character pressed
 - No modifier keys
 - Delegate KeyEventHandler
 - Event argument KeyEventArgs
 - KeyUp or KeyDown
 - Special modifier keys
 - Key enumeration value



12.10 Keyboard Event Handling

Keyboard Events, Delegates and Event Arguments	
<i>Key Events (Delegate KeyEventHandler, event arguments KeyEventArgs)</i>	
KeyDown	Raised when key is initially pushed down.
KeyUp	Raised when key is released.
<i>Key Events (Delegate KeyPressEventHandler, event arguments KeyPressEventArgs)</i>	
KeyPress	Raised when key is pressed. Occurs repeatedly while key is held down, at a rate specified by the operating system.
<i>Class KeyPressEventArgs Properties</i>	
KeyChar	Returns the ASCII character for the key pressed.
Handled	Whether or not the KeyPress event was handled.
<i>Class KeyEventArgs Properties</i>	
Alt	Indicates whether the <i>Alt</i> key was pressed.
Control	Indicates whether the <i>Control</i> key was pressed.
Shift	Indicates whether the <i>Shift</i> key was pressed.
Handled	Whether the event was handled.
KeyCode	Returns the key code for the key, as a Keys enumeration. This does not include modifier key information. Used to test for a specific key.
KeyData	Returns the key code as a Keys enumeration, combined with modifier information. Used to determine all information about the key pressed.
KeyValue	Returns the key code as an int , rather than as a Keys enumeration. Used to obtain a numeric representation of the key pressed.
Modifiers	Returns a Keys enumeration for any modifier keys pressed (<i>Alt</i> , <i>Control</i> and <i>Shift</i>). Used to determine modifier key information only.

Fig. 12.31 Keyboard events, delegates and event arguments.



63

Outline

KeyDemo.cs

```

1 // Fig. 12.32: KeyDemo.cs
2 // Displaying information about the key the user pressed.
3
4 using System;
5 using System.Drawing;
6 using System.Collections;
7 using System.ComponentModel;
8 using System.Windows.Forms;
9 using System.Data;
10
11 // form to display key press
12 // information--contains two labels
13 public class KeyDemo : System.Windows.Forms.Form
14 {
15     private System.Windows.Forms.Label charLabel;
16     private System.Windows.Forms.Label keyInfoLabel;
17
18     private System.ComponentModel.Container components = null;
19
20     /// The main entry point for the application.
21     [STAThread]
22     static void Main()
23     {
24         Application.Run( new KeyDemo() );
25     }
26
27     // display the character pressed using keyChar property
28     protected void KeyDemo_KeyPress(
29         object sender, System.Windows.Forms.KeyPressEventArgs e )
30     {
31         charLabel.Text = "Key pressed: " + e.KeyChar;
32     }
33

```

Labels

Initially empty Labels

Label for modifier information

If key pressed is not ASCII, charLabel remains empty

Key pressed as a char

Char property EventArgs object

Display the key pressed

© 2002 Prentice Hall. All rights reserved.

64

Outline

```

34 // display modifier keys, key code, key data and key value
35 private void KeyDemo_KeyDown(
36     object sender, System.Windows.Forms.KeyEventArgs e )
37 {
38     keyInfoLabel.Text =
39         "Alt: " + ( e.Alt ? "Yes" : "No" ) + " " +
40         "Shift: " + ( e.Shift ? "Yes" : "No" ) + " " +
41         "Ctrl: " + ( e.Control ? "Yes" : "No" ) + " " +
42         "KeyCode: " + e.KeyCode + " " +
43         "KeyData: " + e.KeyData + " " +
44         "KeyValue: " + e.KeyValue;
45 }
46
47 // clear labels when key released
48 private void KeyDemo_KeyUp(
49     object sender, System.Windows.Forms.KeyEventArgs e )
50 {
51     keyInfoLabel.Text = "";
52     charLabel.Text = "";
53 }

```

Uses Alt, Shift, and Control properties

EventArgs object

KeyCode returns the key pressed on a keyboard if matched

KeyData property returns a Keys enumeration with data about modifier keys

Integer value is the Windows virtual key code

KeyUp event handler clears both labels

© 2002 Prentice Hall. All rights reserved.



```
KeyDemo
Key pressed: H
Alt: No
Shift: Yes
Ctrl: No
KeyCode: H
KeyData: H, Shift
KeyValue: 72
```

Keys enumeration can test for specific keys by comparing key pressed to **KeyCode**

```
Alt: No
Shift: No
Ctrl: No
KeyCode: Return
KeyData: Return
KeyValue: 13
```

KeyDown event still raised so **keyInfoLabel** displays information

```
KeyDemo
Alt: No
Shift: Yes
Ctrl: No
KeyCode: ShiftKey
KeyData: ShiftKey, Shift
KeyValue: 16
```

```
KeyDemo
Key pressed: '
Alt: No
Shift: No
Ctrl: No
KeyCode: OemQuotes
KeyData: OemQuotes
KeyValue: 222
```