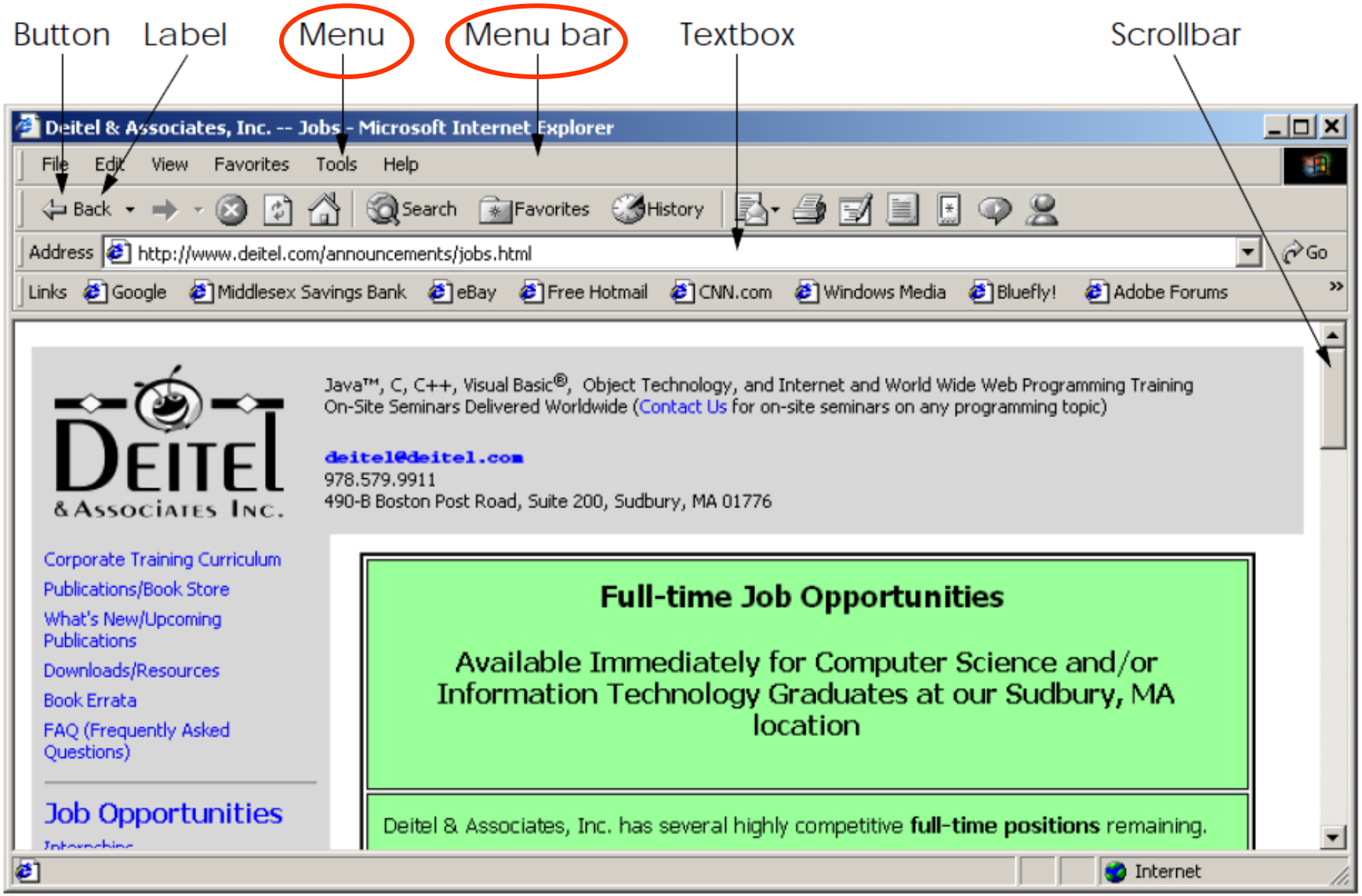
# Chapter 12 - Graphical User Interface Concepts: Part 1

# Introduction



# Introduction

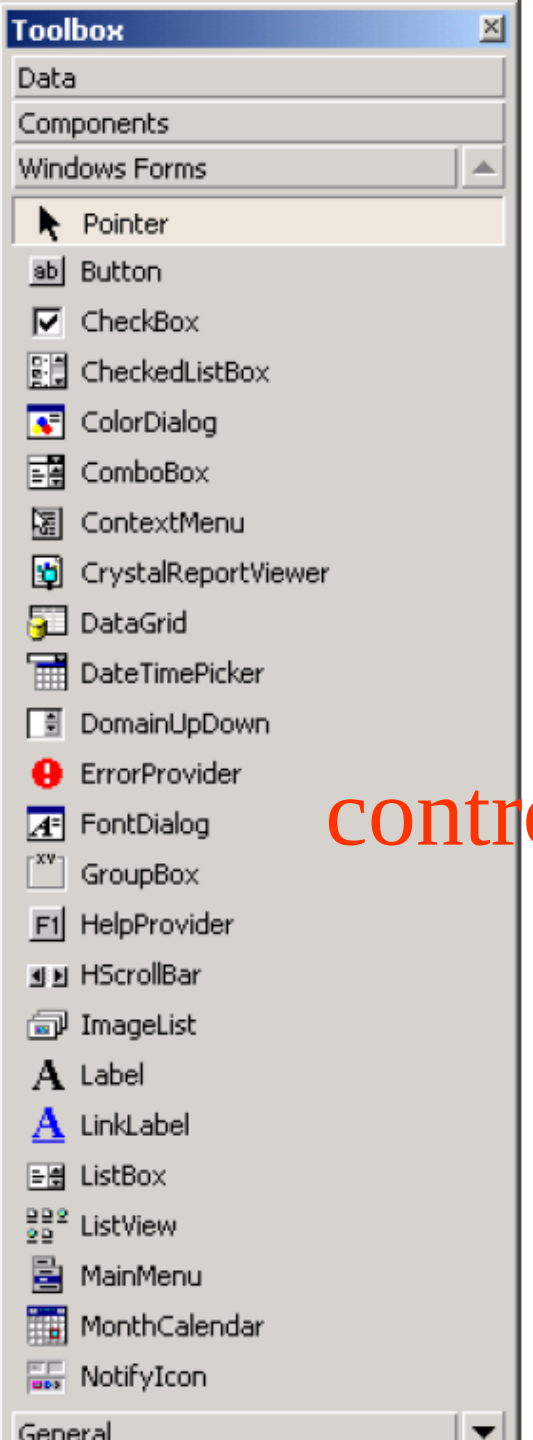
| Control | Description |
|---|---|
| `Label` | An area in which icons or uneditable text can be displayed. |
| `TextBox` | An area in which the user inputs data from the keyboard. The area also can display information. |
| `Button` | An area that triggers an event when clicked. |
| `CheckBox` | A GUI 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 .

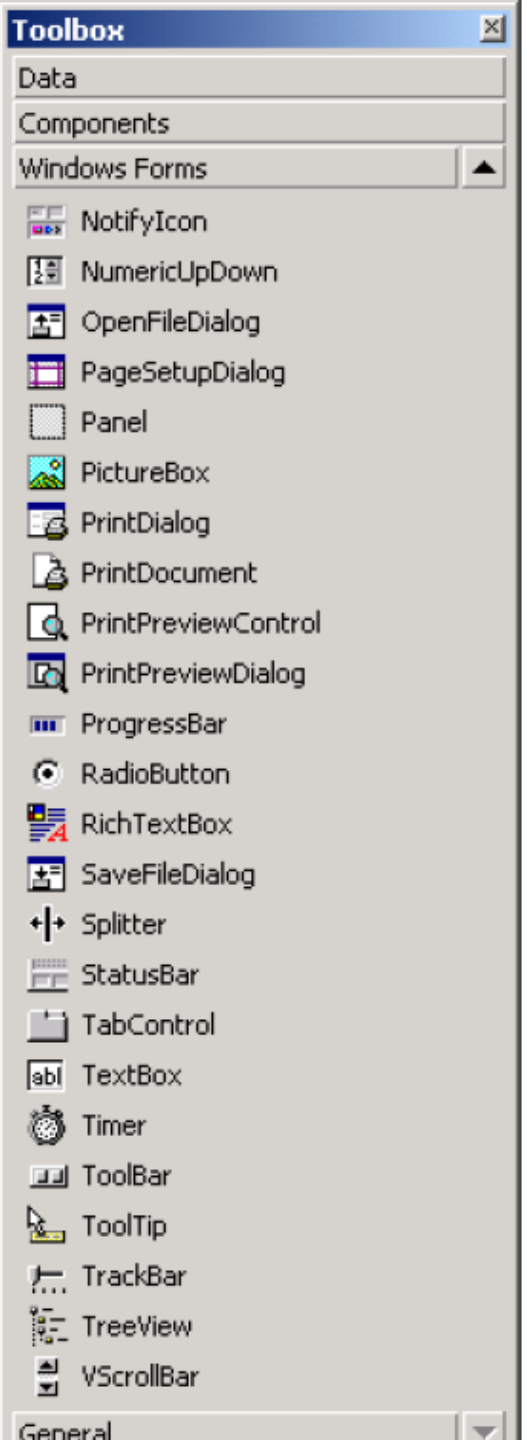
# 12.2 Windows Forms

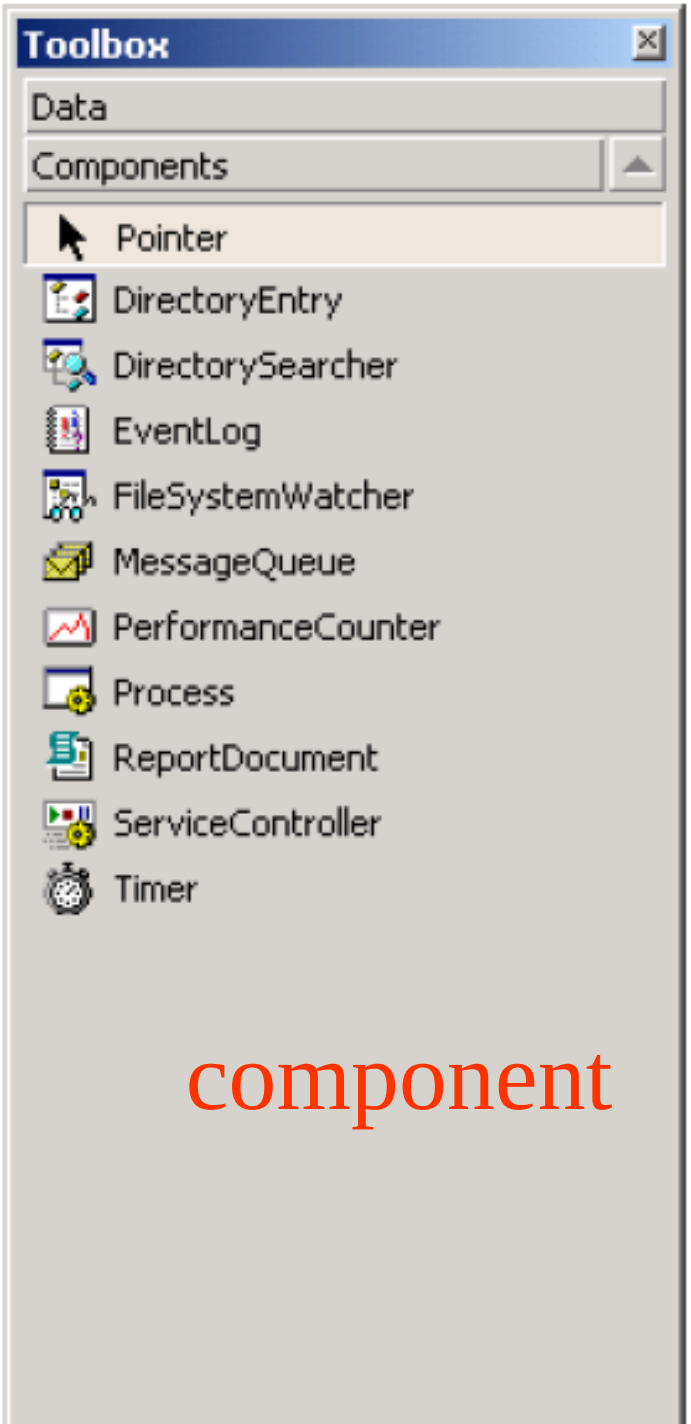


- Component
  - the term "component" is generally used for an entity that is reusable
  - It is a class with the property that it implements IComponent interface
    - Such as printReporter, Socket(networking) component,
  - Generally, it lacks visual parts
- Control
  - (It is also a class) Component with graphical part
    - Such as button, textbox or label
  - A control is a component that provides user-interface (UI) capabilities.




control




component

# Component & control from MSDN



- Component:
  - In the .NET Framework, a component is a class that
    - implements System.ComponentModel.Icomponent interface
    - Or derives directly or indirectly from a class that implements IComponent.
- Control:
  - .NET Framework provides two base classes for controls:
    - System.Windows.Forms.Control for client-side Windows Forms controls
      - It derives from Component and itself provides UI capabilities.
    - System.Web.UI.Control for ASP.NET server controls
      - it implements IComponent and provides the infrastructure on which it is easy to add UI functionality.
  - All controls derive directly or indirectly from these two classes.

檔案(F) 編輯(E) 檢視(V) 專案(P) 建置(B) 偵錯(D)

開始 Debug

Form1.cs [設計] Program.cs

Form1

方案總管

搜尋 方案總管 (Ctrl+;)

- 方案 'WindowsFormsApplication1' (1 專案)
  - WindowsFormsApplication1
    - Properties
    - 參考
    - App.config
    - Form1.cs
      - Form1.Designer.cs
      - Form1.resx
      - Form1
        - Form1()
        - components : IContainer
        - Dispose(bool) : void
        - InitializeComponent() : void
    - Program.cs

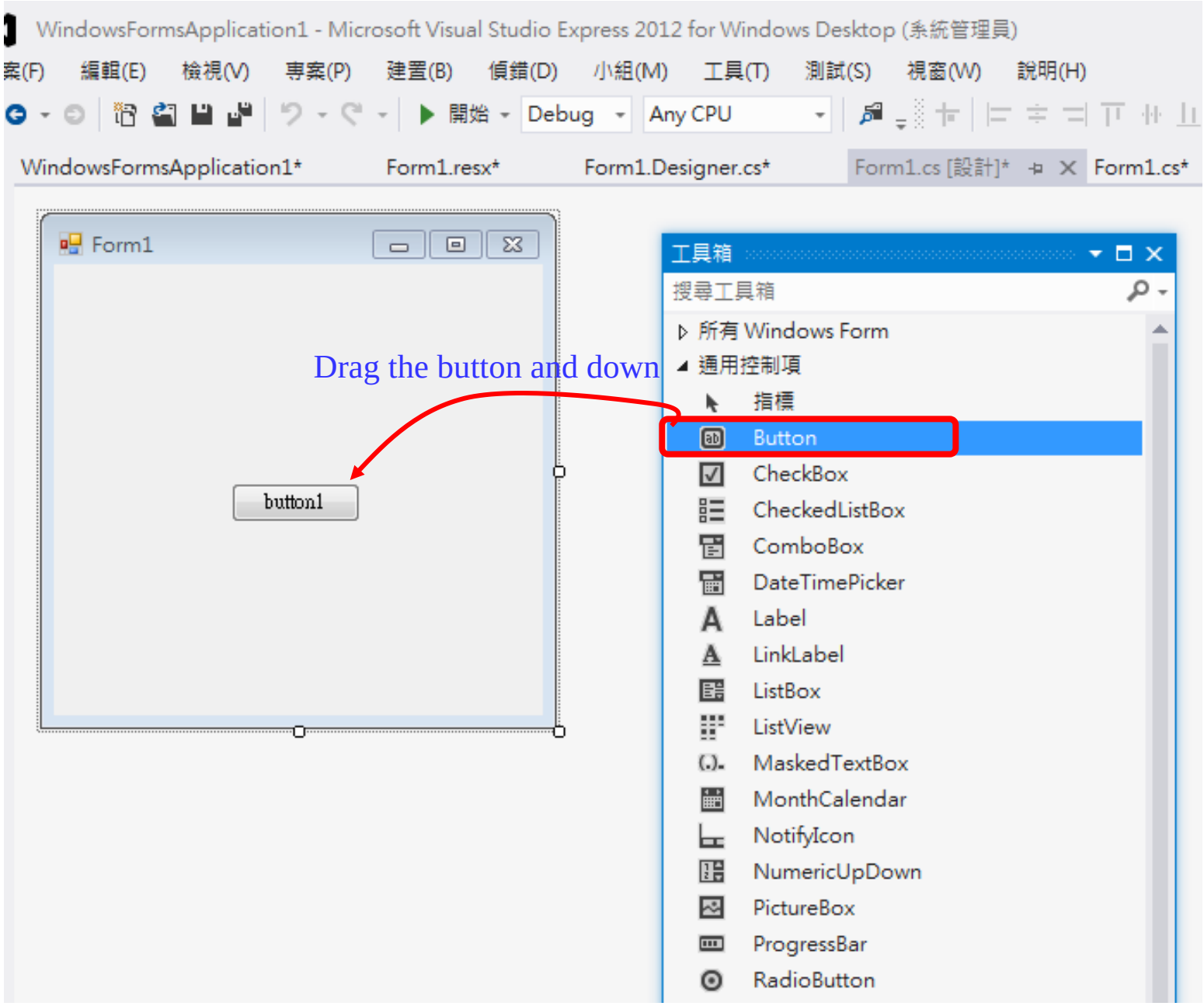
屬性

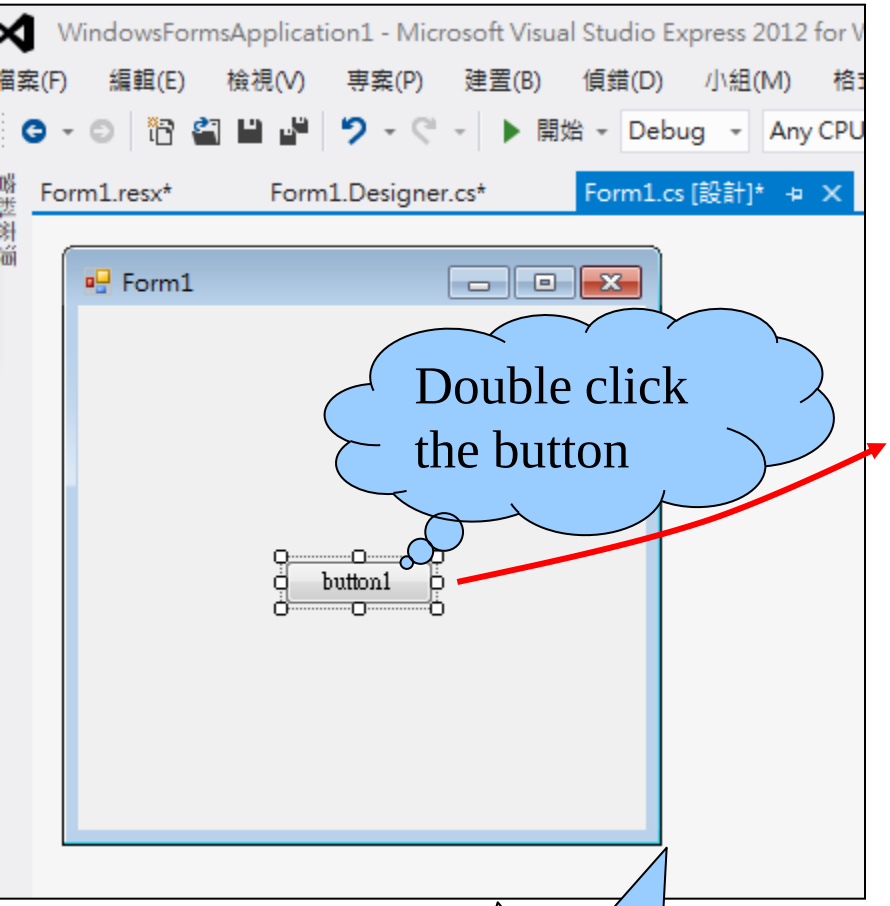
Form1 System.Windows.Forms.Form

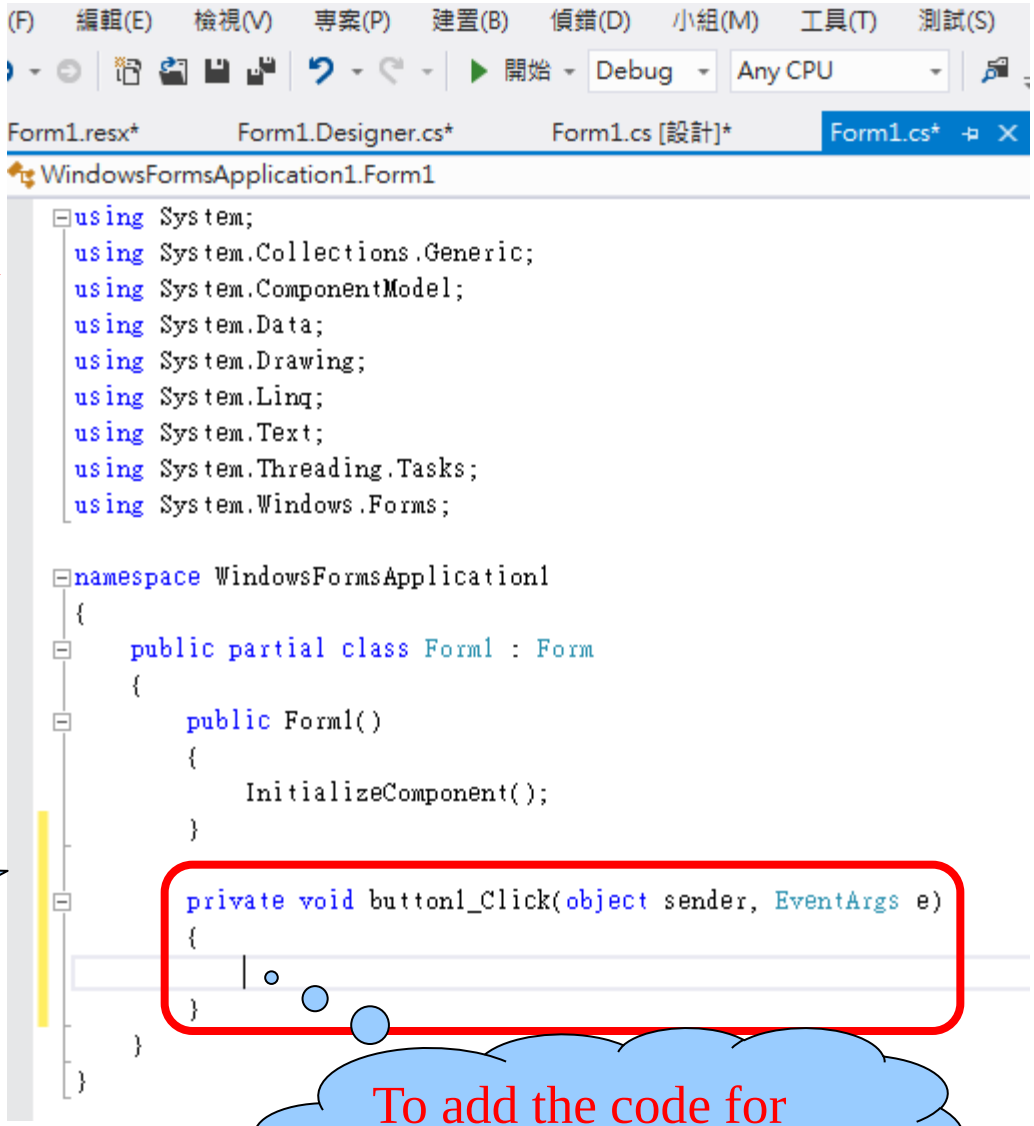
外觀

| | |
|---|---|
| BackColor | Control |
| BackgroundImage | (無) |
| BackgroundImageLayout | Tile |
| Cursor | Default |
| Font | 新細明體, 9pt |
| ForeColor | ControlText |
| FormBorderStyle | Sizable |
| RightToLeft | No |

| Form Properties and Events | Description / Delegate and Event Arguments |
|---|---|
| *Common Properties* | |
| **AcceptButton** | Which button will be clicked when *Enter* is pressed. |
| **AutoScroll** | Whether scrollbars appear when needed (if data fill more than one screen). |
| **CancelButton** | Button that is clicked when the *Escape* 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. |
| *Common Methods* | |
| **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. |
| *Common Events* | *(Delegate* ***EventHandler****, event arguments* ***EventArgs****)* |
| **Load** | Occurs before a form is shown. Visual Studio .NET generates a default event handler when the programmer double clicks on the form in the designer. |






Codes for button click add to Form1. cs

| Class `Control` Properties and Methods | Description |
|---|---|
| *Common Properties* | |
| **`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. |

| Label Properties | Description / Delegate and Event Arguments |
|---|---|
| *Common Properties* | |
| **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**). |

| TextBox Properties and Events | Description / Delegate and Event Arguments |
|---|---|
| *Common Properties* | |
| **AcceptsReturn** | If **true**, pressing *Enter* creates a new line if textbox spans multiple lines. If **false**, pressing *Enter* 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. |

# 12.4 Control Properties and Layout

- Common properties of control
  - Text property
    - Specifies the text appearing on a control
  - Focus method
    - Transfers the focus to a control, becoming active control
  - TabIndex property
    - Order in which controls are given focus when pressed tab
    - Automatically set by Visual Studio .NET if not specified
  - Enable property
    - Indicate a control's accessibility

# 12.4 Control Properties and Layout

- Common properties of control
  - Visibility control
    - Hide control from user, using method Hide
  - Anchor property
    - Anchoring control to specific location, like top margin
      - Constant distance from specified location
      - Unanchored control will move
    - Docking allows control to spread itself along and entire side
    - Both options refer to the parent container
  - Size structure
    - Allow for specifying size range
      - Can use MinimumSize and MaximumSize property

# 12.4 Control Properties and Layout

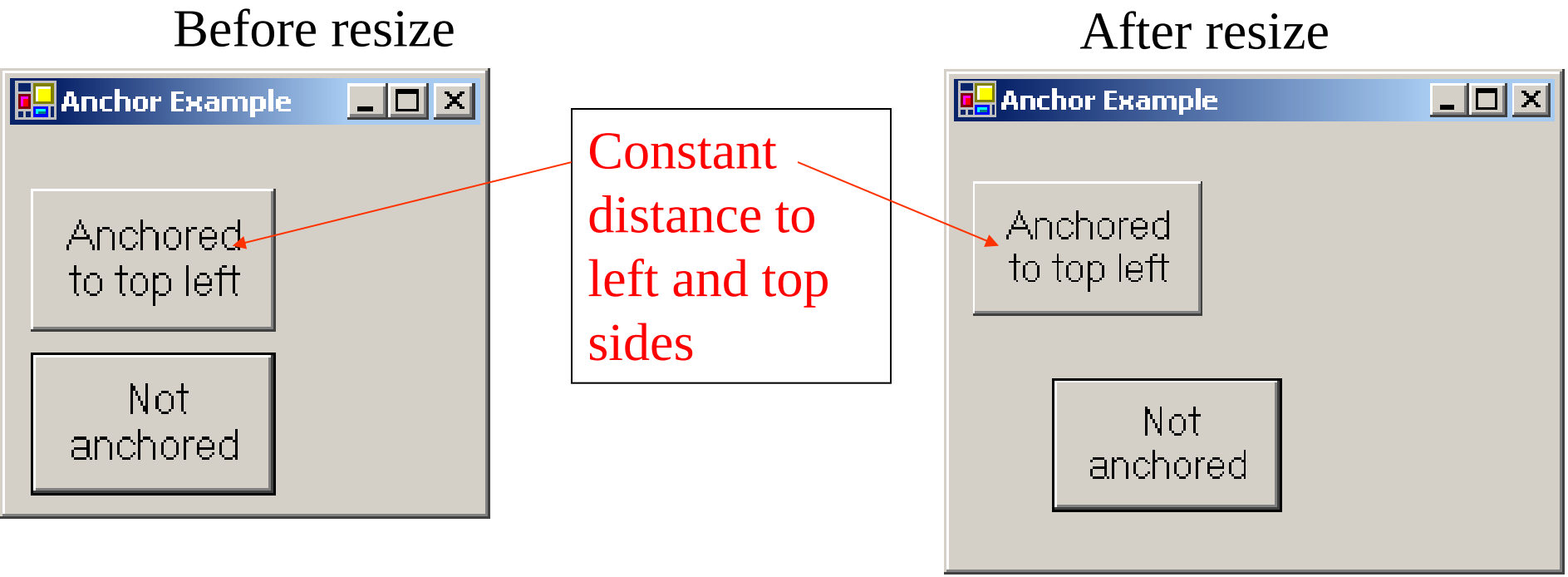
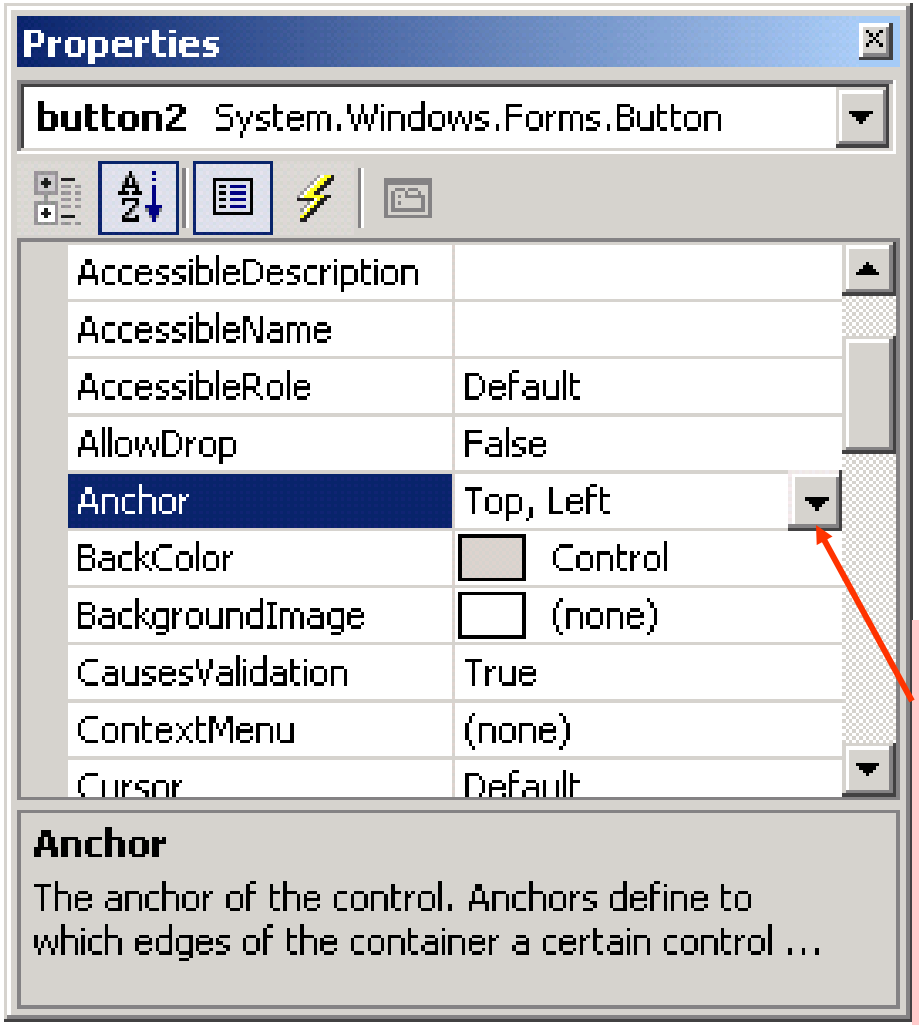
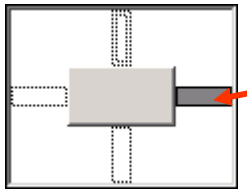



**Fig. 12.11** Anchoring demonstration.

# 12.4 Control Properties and Layout



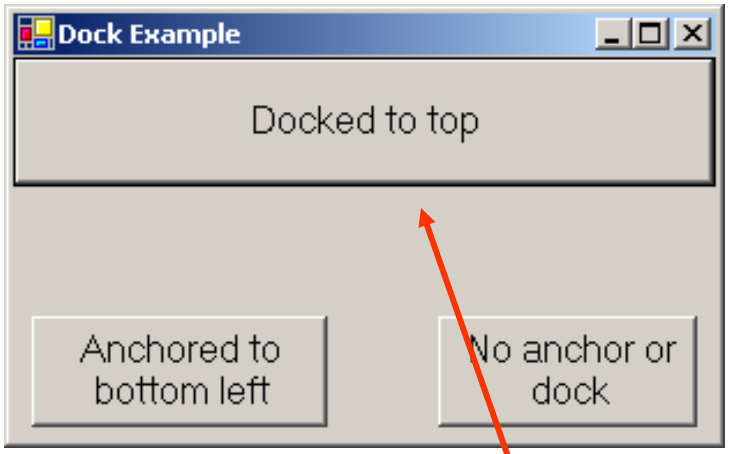
Darkened bar indicates to which wall control is anchored

Click down-arrow in Anchor property to display anchoring window

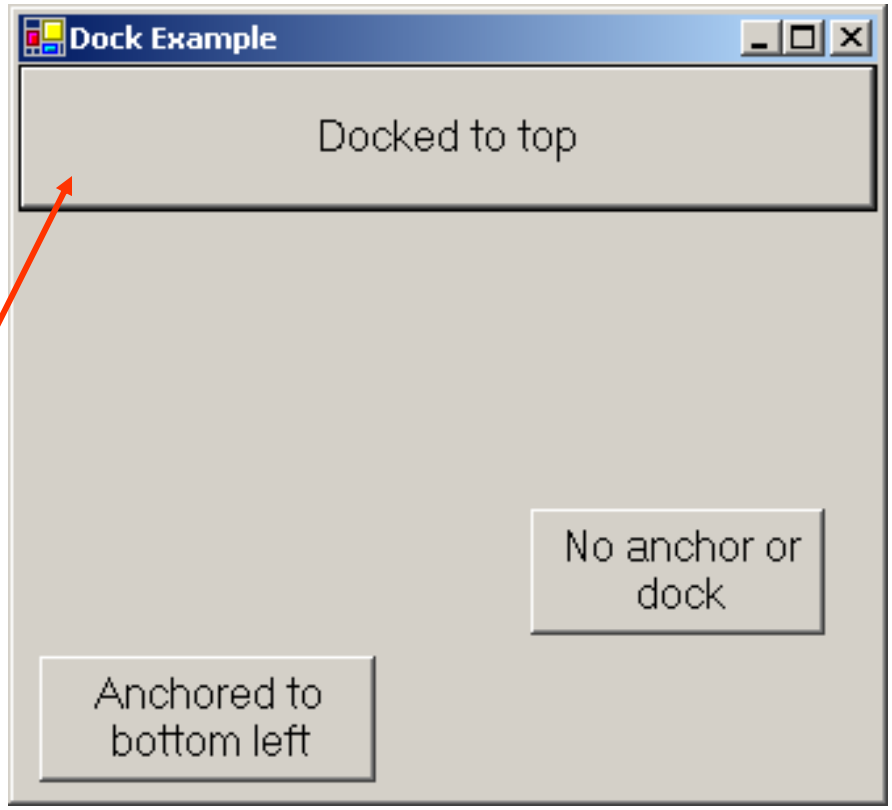
**Fig. 12.12** Manipulating the **Anchor** property of a control.

# 12.4 Control Properties and Layout

Before resize




After resize




Control expands along top portion of the form

# Basic Event Handling

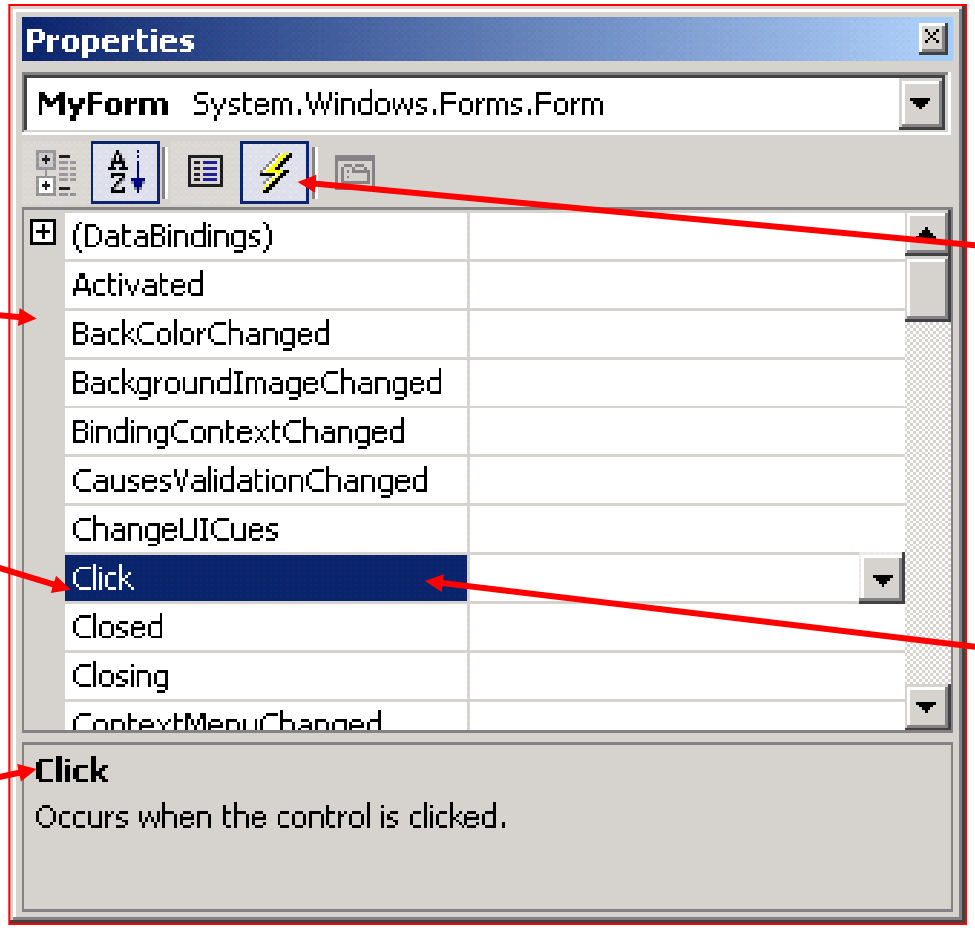



**Fig. 12.6** Events section of the **Properties** window.

# 13.4 Event Handling

- Event
  - Generally, it is generated by users or outer system,
    - like movement from mouse or keyboard, package arriving from some web site that you are querying
  - Event handlers performs action (codes) response to the events
    - codes written by programmer or system default

# 13.4 Event Handling

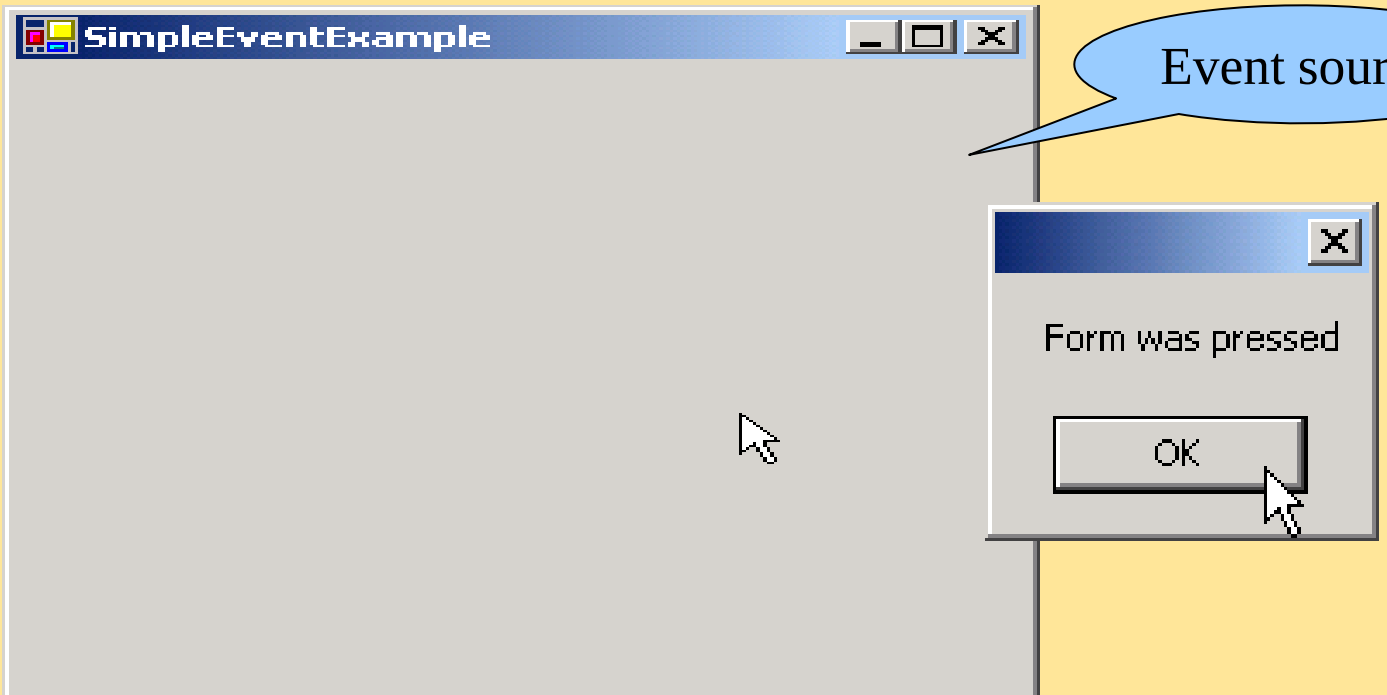
- Event-handling model
  - Three parts (programmer caring part) (not including OS and User)
    1. Event source (like button)
       - is a GUI component with which user can interact
    2. Event object (like mouse_Move, button_Click)
       - Encapsulates information about event that occurred
    3. Event listener
       - Receives event object when notified by OS, then the listener responds to the event
  - Programmer must perform two tasks
    1. Register event listener for event source (C# can be done automatically)
    2. Implement event-handling method (event handler)

```
3   Using System;
4   using System.Drawing;
5   using System.Collections;
6   using System.ComponentModel;
7   using System.Windows.Forms;
8   using System.Data;
10  public class MyForm : System.Windows.Forms.Form {
12      private System.ComponentModel.Container components = null;
14      [STAThread]
15      static void Main() {
17          Application.Run( new MyForm() );
18      }
21      private void MyForm_Click( object sender, System.EventArgs e ) {
23          MessageBox.Show( "Form was pressed" );
24      }
25  }
```

偽Event listener

Event object

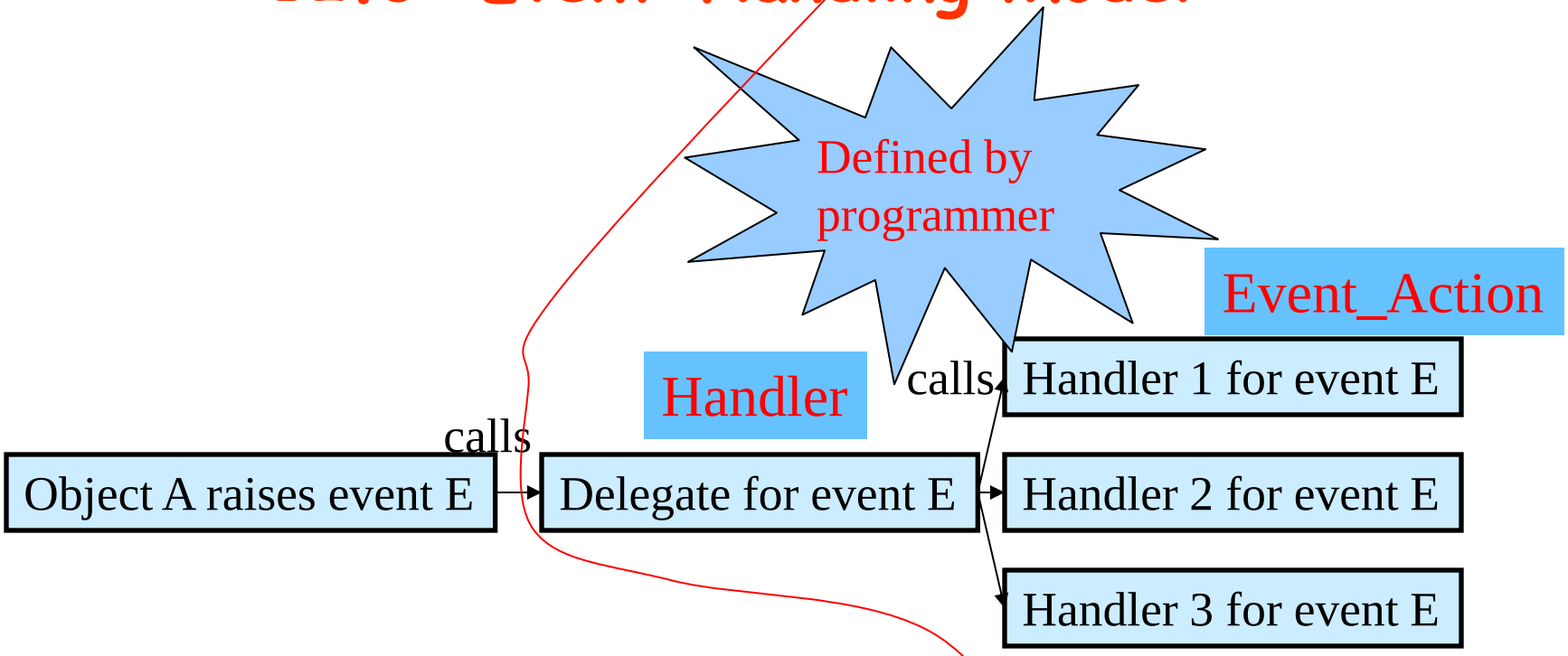
Event source



# 12.3 Event-Handling Model

- (Associated with event) delegate
  - Contain lists of method references
    - The referred Method and delegatge's parameter must have same signature (note: delegate 接收 methodname 當作參數)
  - Delegate is intermediaries for objects and methods
  - Two object reference (sender and event) are passed into, through
    - ControlName_EventName
- Steps for delegate used in event handling for GUI control (automatically in C#)
  1. Declare a delegate
  2. Create a delegate
  3. Add event handler to delegate

# 12.3 Event-Handling Model

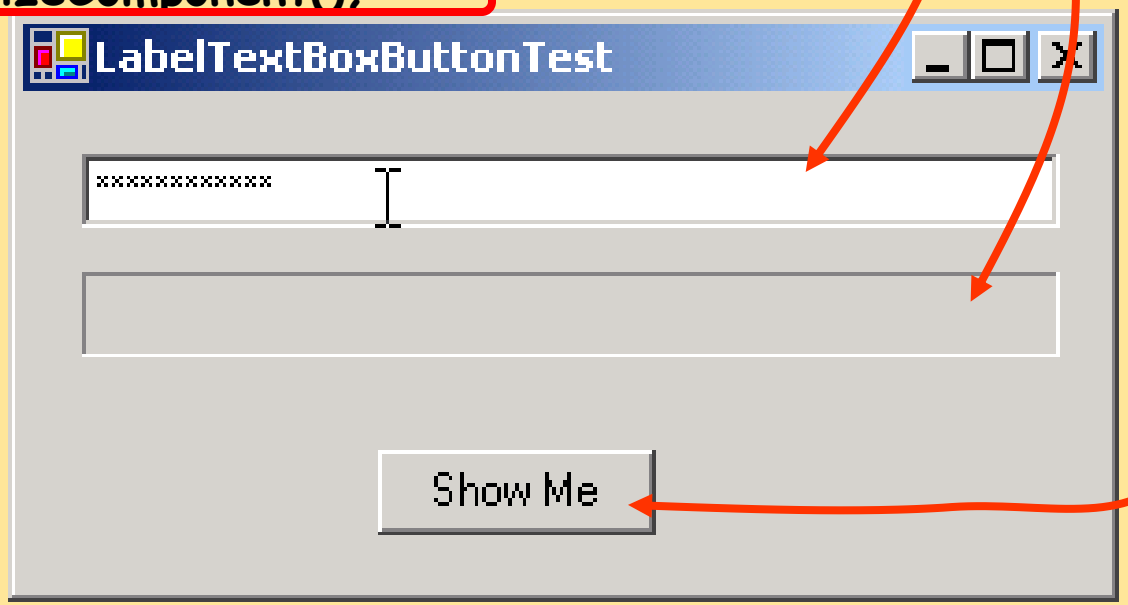



Event multicasting

Have multiple handlers for one event

**Fig. 12.5 Event-handling model using delegates.**

```
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
11 namespace LabelTextBoxButtonTest {
17  public class LabelTextBoxButtonTest : System.Windows.Forms.Form{
20     private System.Windows.Forms.Button displayPasswordButton;
21     private System.Windows.Forms.Label displayPasswordLabel;
22     private System.Windows.Forms.TextBox inputPasswordTextBox;
26     private System.ComponentModel.Container components = null;
28         public LabelTextBoxButtonTest() {
30            InitializeComponent();
31         }
```

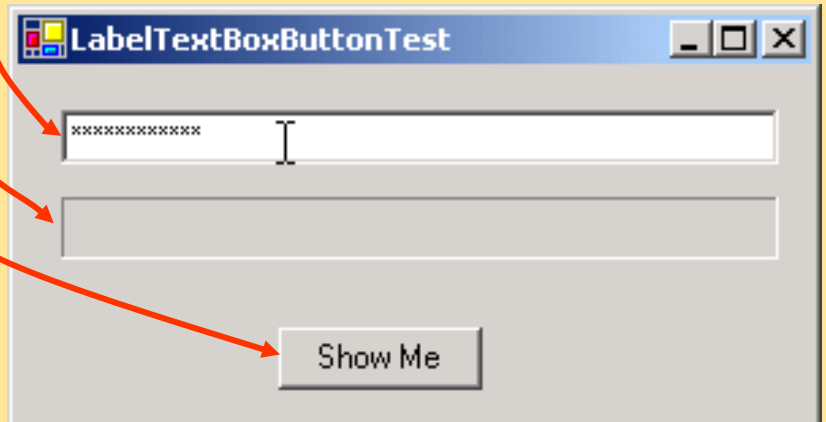


```
35      protected override void Dispose( bool disposing ) {
37         if ( disposing ) {
39            if ( components != null ) {
41               components.Dispose();
42            }
43         }
44         base.Dispose( disposing );
45      }
46   #region Windows Form Designer generated code
51      private void InitializeComponent() {
53         this.displayPasswordButton =
54            new System.Windows.Forms.Button();
55         this.inputPasswordTextBox =
56            new System.Windows.Forms.TextBox();
57         this.displayPasswordLabel =
58            new System.Windows.Forms.Label();
59         this.SuspendLayout();
```

Code in region can be expand or collapse by vs.net editor

"this" is the form

Call form class's method



```
63          this.displayPasswordButton.Location =
64             new System.Drawing.Point( 96, 96 );
65          this.displayPasswordButton.Name =
66             "displayPasswordButton";
67          this.displayPasswordButton.TabIndex = 1;
68          this.displayPasswordButton.Text = "Show Me";
69          this.displayPasswordButton.Click +=
70             new System.EventHandler(
71             this.displayPasswordButton_Click );
75          this.inputPasswordTextBox.Location =
76             new System.Drawing.Point( 16, 16 );
77          this.inputPasswordTextBox.Name =
78             "inputPasswordTextBox";
79          this.inputPasswordTextBox.PasswordChar='*';
80          this.inputPasswordTextBox.Size =
81             new System.Drawing.Size( 264, 20 );
82          this.inputPasswordTextBox.TabIndex = 0;
83          this.inputPasswordTextBox.Text = "";
87          this.displayPasswordLabel.BorderStyle =
88             System.Windows.Forms.BorderStyle.Fixed3D;
89          this.displayPasswordLabel.Location =
90             new System.Drawing.Point( 16, 48 );
91          this.displayPasswordLabel.Name =
92             "displayPasswordLabel";
```

EventHandler is a delegate

displayxx_Click are user's methods

displayxx.Click are system delegate

```
93          this.displayPasswordLabel.Size =
94             new System.Drawing.Size( 264, 23 );
95          this.displayPasswordLabel.TabIndex = 2;
99          this.AutoScaleBaseSize =
100            new System.Drawing.Size( 5, 13 );
101         this.ClientSize =
102            new System.Drawing.Size( 292, 133 );
103         this.Controls.AddRange(
104            new System.Windows.Forms.Control[] {
105               this.displayPasswordLabel,
106               this.inputPasswordTextBox,
107               this.displayPasswordButton  } );
108         this.Name = "LabelTextBoxButtonTest";
109         this.Text = "LabelTextBoxButtonTest";
110         this.ResumeLayout( false );
111      }
113      #endregion
```

Adds an array of control objects to a collection.

New an array of control objects

controls in the form

Control.Name is for identified (ID) in codes;

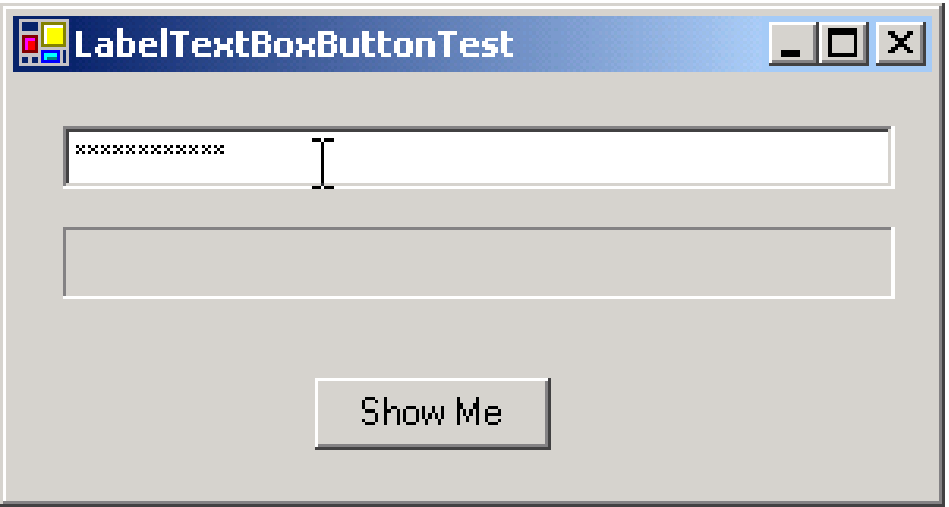
Control.Text is for shown in screen

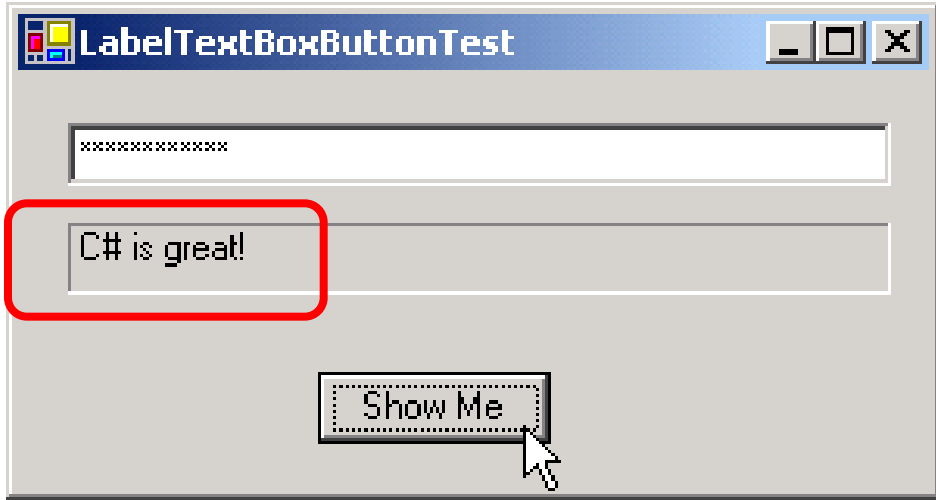
```
117      [STAThread]
118      static void Main() {
120         Application.Run( new LabelTextBoxButtonTest() );
121      }
```

```
123        protected void displayPasswordButton_Click(
124            object sender, System.EventArgs e ) {
127            displayPasswordLabel.Text =
                   inputPasswordTextBox.Text;
129        }
130    }
131 }
```

the method will pass to multi-delegate displayxx.Click()

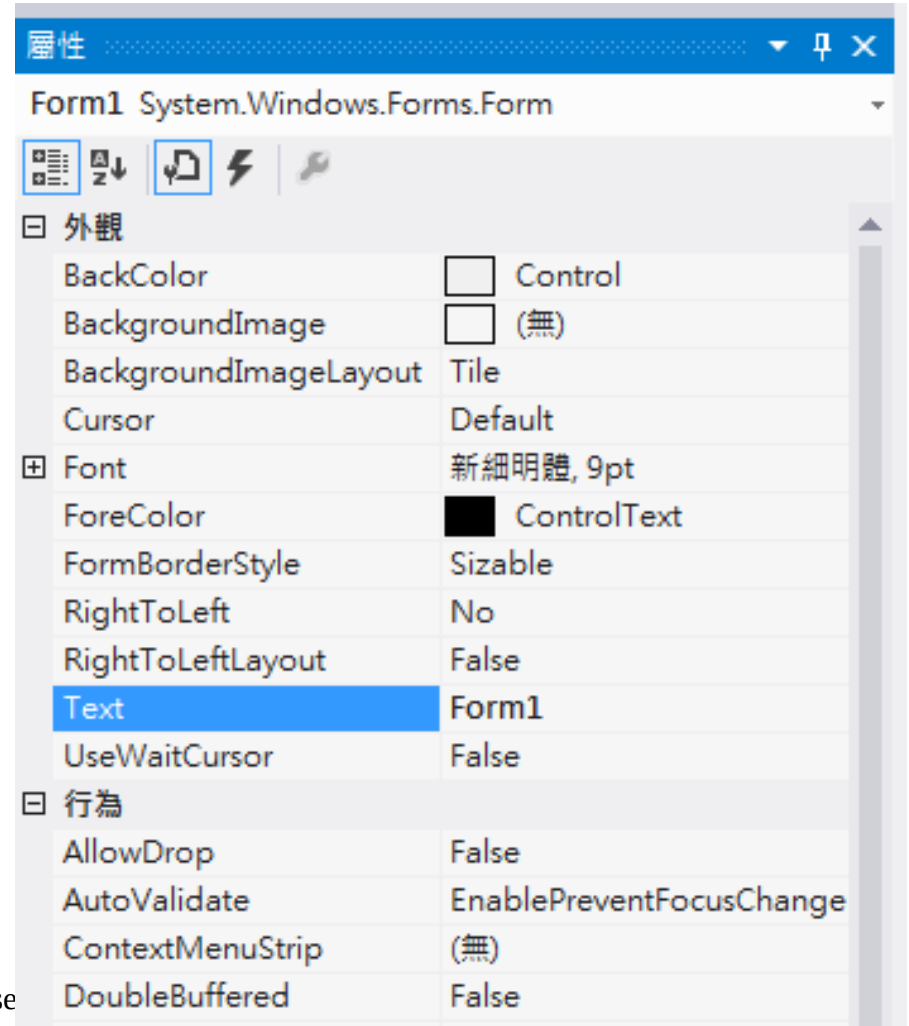
displayxx_Click are user's methods with signature void (object, EventArgs).



# Setting a property on your form

- The property window is a powerful tool that you can use to change all visual and functional properties for the form and the control in the form



Form1.Designer.cs | Form1.cs [設計] | Form1.cs

WindowsFormsApplication1.Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

方案總管

搜尋 方案總管 (Ctrl+;)

- 方案 'WindowsFormsApplication1' (1 專案)
  - WindowsFormsApplication1
    - Properties
    - 參考
    - App.config
    - Form1.cs
      - Form1.Designer.cs
      - Form1.resx
      - Form1
        - Form1()
        - components : IContainer
        - Dispose(bool) : void
        - InitializeComponent() : void
    - Program.cs

Partial class means a class can be defined in two files (but with the same class name)

Form1.Designer.cs | Form1.cs [設計] | Form1.cs | Program.cs

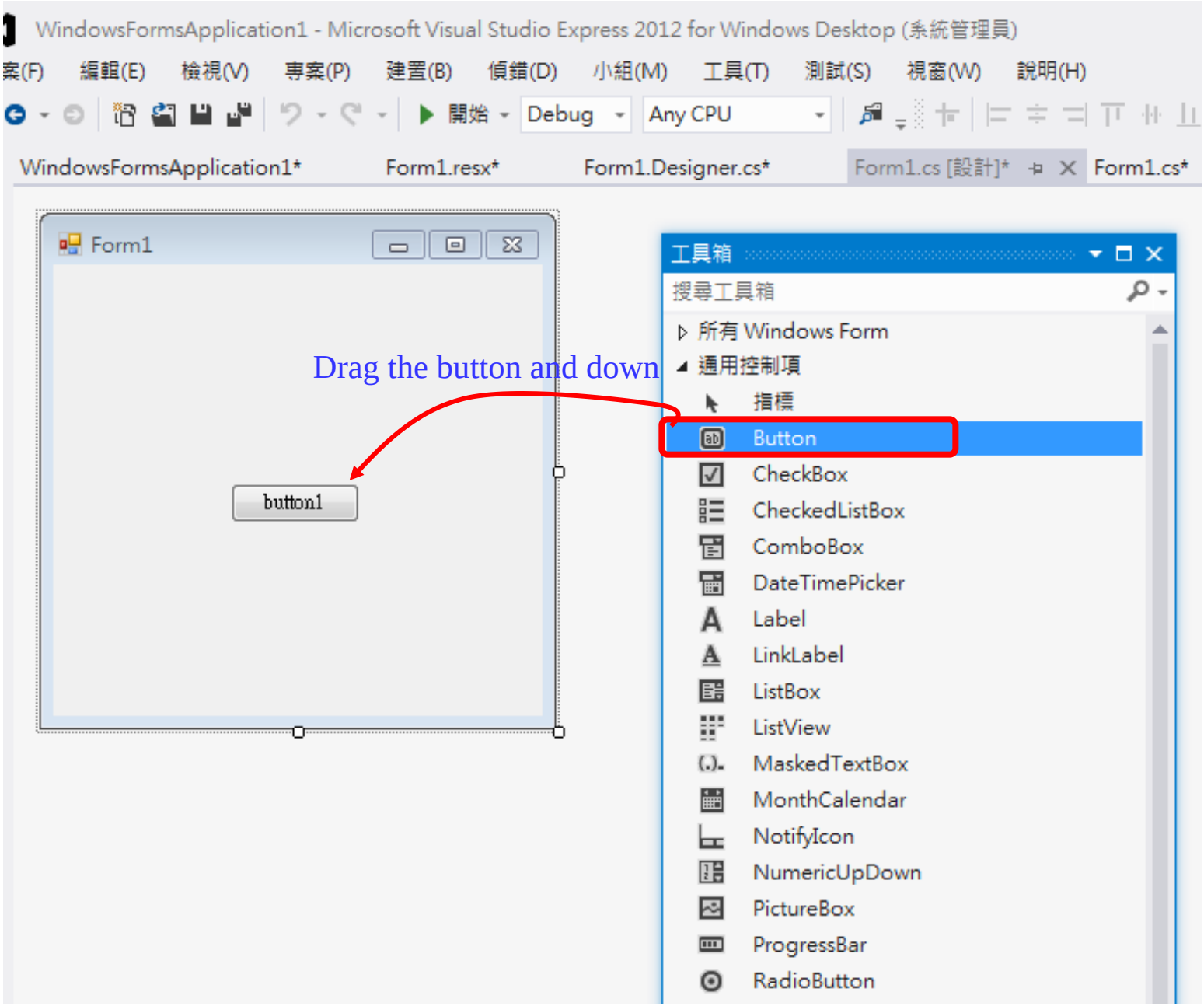
WindowsFormsApplication1.Form1

```
namespace WindowsFormsApplication1
{
    partial class Form1
    {
        /// <summary>
        /// 設計工具所需的變數。
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// 清除任何使用中的資源。
        /// </summary>
        /// <param name="disposing">如果應該處置 Managed 資源則為 true，否則為 false。</
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form 設計工具產生的程式碼

        /// <summary>
        /// 此為設計工具支援所需的方法 - 請勿使用程式碼編輯器
        /// 修改這個方法的內容。
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.Text = "Form1";
        }
```

方案總管

搜尋 方案總管 (Ctrl+;)

- 方案 'WindowsFormsApplication1' (1 專案)
  - WindowsFormsApplication1
    - Properties
    - 參考
    - App.config
    - Form1.cs
      - Form1.Designer.cs
      - Form1.resx
      - Form1
        - Form1()
        - components : IContainer
        - Dispose(bool) : void
        - InitializeComponent() : void
    - Program.cs



WindowsFormsApplication1* Form1.resx* Form1.Designer.cs* Form1.cs [設計]

WindowsFormsApplication1.Form1

```
namespace WindowsFormsApplication1
{
    partial class Form1
    {
        /// <summary> ...
        private System.ComponentModel.IContainer components = null;

        /// <summary> ...
        protected override void Dispose(bool disposing)...

        #region Windows Form 設計工具產生的程式碼

        /// <summary>
        /// 此為設計工具支援所需的方法 - 請勿使用程式碼編輯器
        /// 修改這個方法的內容。
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(103, 127);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 0;
            this.button1.Text = "button1";
            this.button1.UseVisualStyleBackColor = true;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 12F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(284, 262);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
```

C# IDE automatically reflect the adding component in Form1.Desginer.cs

# SuspendLayout()

C# | C++ | VB

```
private void AddButtons()
{
    // Suspend the form layout and add two buttons.
    this.SuspendLayout();
    Button buttonOK = new Button();
    buttonOK.Location = new Point(10, 10);
    buttonOK.Size = new Size(75, 25);
    buttonOK.Text = "OK";

    Button buttonCancel = new Button();
    buttonCancel.Location = new Point(90, 10);
    buttonCancel.Size = new Size(75, 25);
    buttonCancel.Text = "Cancel";

    this.Controls.AddRange(new Control[]{buttonOK, buttonCancel});
    this.ResumeLayout();
}
```

# Control.SuspendLayout 方法 ()

.NET Framework (current version) | 其他版本

暫停控制項的配置邏輯。

**命名空間:** System.Windows.Forms
**組件:** System.Windows.Forms (於 System.Windows.Forms.dll)

控制項的配置邏輯會暫停直到 ResumeLayout 方法呼叫。

SuspendLayout 和 ResumeLayout 方法一起用來隱藏多個 Layout 事件而調整控制項的多個屬性。 例如，您通常會呼叫 SuspendLayout 方法，然後設定 Size, ，Location, ，Anchor, ，或 Dock 的控制項，然後呼叫屬性 ResumeLayout 方法，才能讓變更生效。

# Control.ControlCollection.AddRange 方法 (Control[])

.NET Framework (current version) | 其他版本

Adds an array of control objects to the collection.

**命名空間:** System.Windows.Forms
**組件:** System.Windows.Forms (於 System.Windows.Forms.dll)

## 註解

Control 物件中包含 *controls* 陣列會附加至集合結尾。

您可以使用 AddRange 方法來快速新增一群 Control 而非手動新增每個集合的物件 Control 集合 Add 方法。

若要移除 Control 您先前加入，請使用 Remove, ，RemoveAt, ，或 Clear 方法。

# AddRange()

C# | C++ | VB

```
// Create two RadioButtons to add to the Panel.
private RadioButton radioAddButton = new RadioButton();
private RadioButton radioRemoveButton = new RadioButton();

// Add controls to the Panel using the AddRange method.
private void addRangeButton_Click(object sender, System.EventArgs e)
{
    // Set the Text the RadioButtons will display.
    radioAddButton.Text = "radioAddButton";
    radioRemoveButton.Text = "radioRemoveButton";

    // Set the appropriate location of radioRemoveButton.
    radioRemoveButton.Location = new System.Drawing.Point(
      radioAddButton.Location.X,
      radioAddButton.Location.Y + radioAddButton.Height);

    //Add the controls to the Panel.
    panel1.Controls.AddRange(new Control[]{radioAddButton, radioRemoveButton});
}
```

# Collections in C#

- For many applications, you want to create and manage groups of related objects.
- two ways to group objects:
  1. creating arrays of objects
     - Simple but fixed number of strongly-typed objects
  2. creating collections of objects.
     - flexible way to work with groups of objects
     - the group of objects can grow and shrink dynamically
- A collection is a class
  - you must declare an instance of the class before adding elements to that collection.
  - It can be a general type

# Collection example

C#

```
// Create a list of strings by using a
// collection initializer.
var salmons = new List<string> { "chinook", "coho", "pink", "sockeye" };

// Iterate through the list.
foreach (var salmon in salmons)
{
    Console.Write(salmon + " ");
}
// Output: chinook coho pink sockeye
```

Syntax of initialization of collection, like a set

# collection

C#

```
// Create a list of strings.
var salmons = new List<string>();
salmons.Add("chinook");
salmons.Add("coho");
salmons.Add("pink");
salmons.Add("sockeye");

// Iterate through the list.
foreach (var salmon in salmons)
{
    Console.Write(salmon + " ");
}
// Output: chinook coho pink sockeye
```

Syntax of dynamically add control to collection

# Delegate in JAVA (for reference)

```
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
7   public class TextFieldTest extends JFrame {
8   private JTextField textField1, textField2, textField3;
9   private JPasswordField passwordField;
12   public TextFieldTest()  {
14        super( "Testing JTextField and JPasswordField" );
16        Container container = getContentPane();
17        container.setLayout( new FlowLayout() );
20        textField1 = new JTextField( 10 );
21        container.add( textField1 );
24        textField2 = new JTextField( "Enter text here" );
25        container.add( textField2 );
```

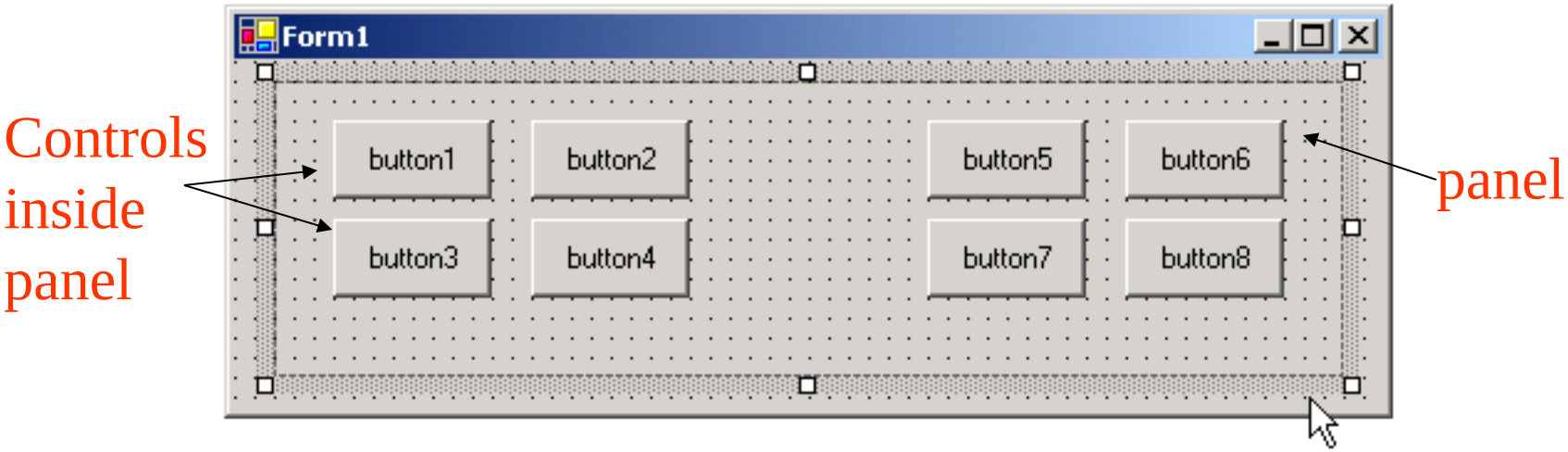
```
29  textField3 = new JTextField( "Uneditable text
    field",20);
30  textField3.setEditable( false );
31  container.add( textField3 );
34  passwordField = new JPasswordField( "Hidden text" );
35  container.add( passwordField );
38  TextFieldHandler handler = new TextFieldHandler();
39  textField1.addActionListener( handler );
40  textField2.addActionListener( handler );
41  textField3.addActionListener( handler );
42  passwordField.addActionListener( handler );
44  setSize( 325, 100 );
45  setVisible( true );
47  }
```

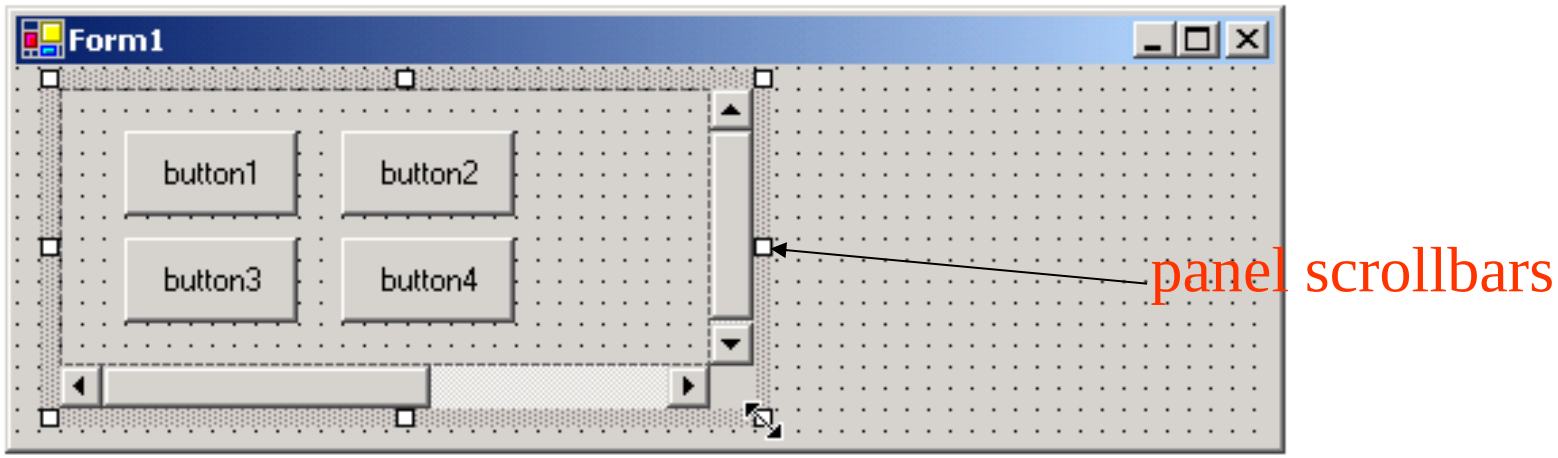

```
49  public static void main( String args[] ) {
51  TextFieldTest application = new TextFieldTest();
52  application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
53  }
```

# Chapter 12 - Graphical User Interface Concepts: Part 2

# 12.6 GroupBoxes and Panels

- Arrange components on a GUI
  - **Panels** can have scrollbar
    - View additional controls inside the Panel



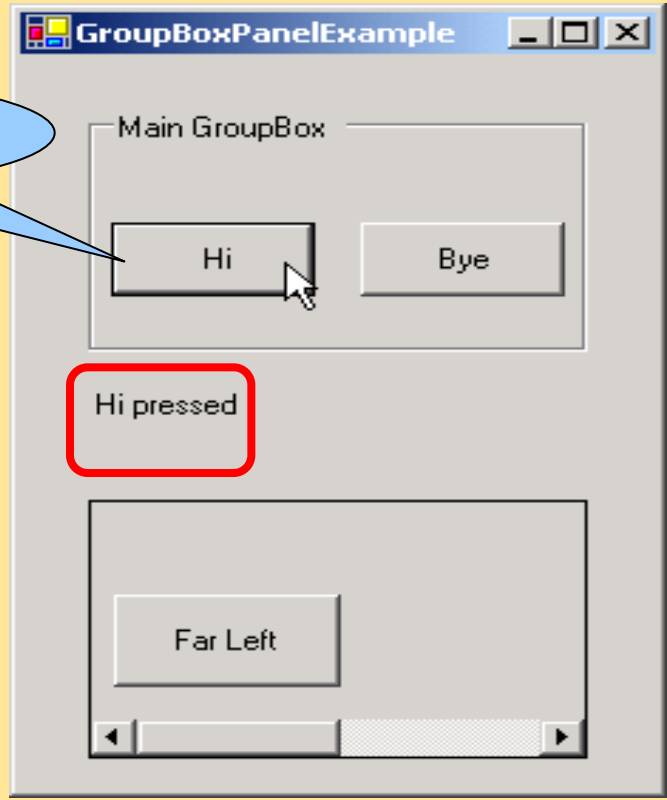
```
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
12  public class GroupBoxPanelExample:System.Windows.Forms.Form {
14      private System.Windows.Forms.Button hiButton;
15      private System.Windows.Forms.Button byeButton;
16      private System.Windows.Forms.Button leftButton;
17      private System.Windows.Forms.Button rightButton;
19      private System.Windows.Forms.GroupBox mainGroupBox;
20      private System.Windows.Forms.Label messageLabel;
21      private System.Windows.Forms.Panel mainPanel;
23      private System.ComponentModel.Container components = null;
27      [STAThread]
28      static void Main() {
30          Application.Run( new GroupBoxPanelExample() );
31      }
32
```

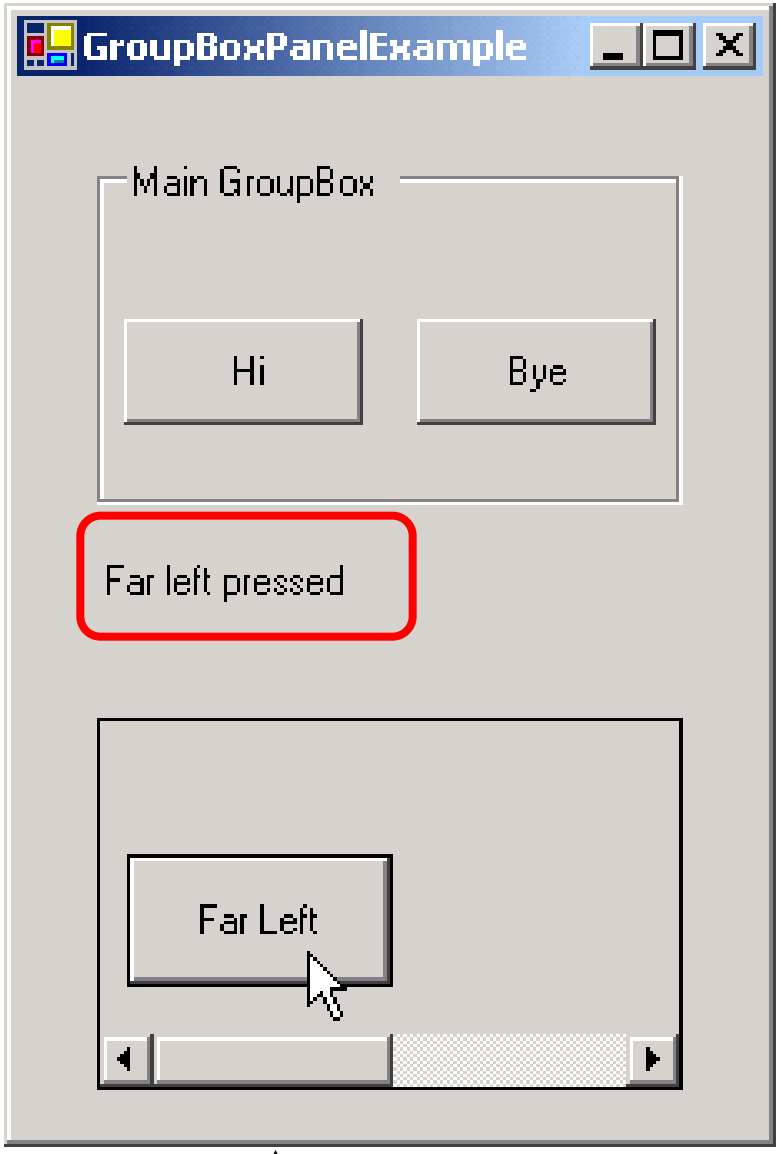
```
36      private void hiButton_Click(object sender, System.EventArgs e ) {
39         messageLabel.Text= "Hi pressed";
40      }
43      private void byeButton_Click(object sender, System.EventArgs e ) {
46         messageLabel.Text = "Bye pressed";
47      }
50      private void leftButton_Click(object sender, System.EventArgs e ) {
53         messageLabel.Text = "Far left pressed";
54      }
57      private void rightButton_Click(object sender,System.EventArgs e){
60         messageLabel.Text =
            "Far right pressed";
61      }
63   }
```

hiButton_Click

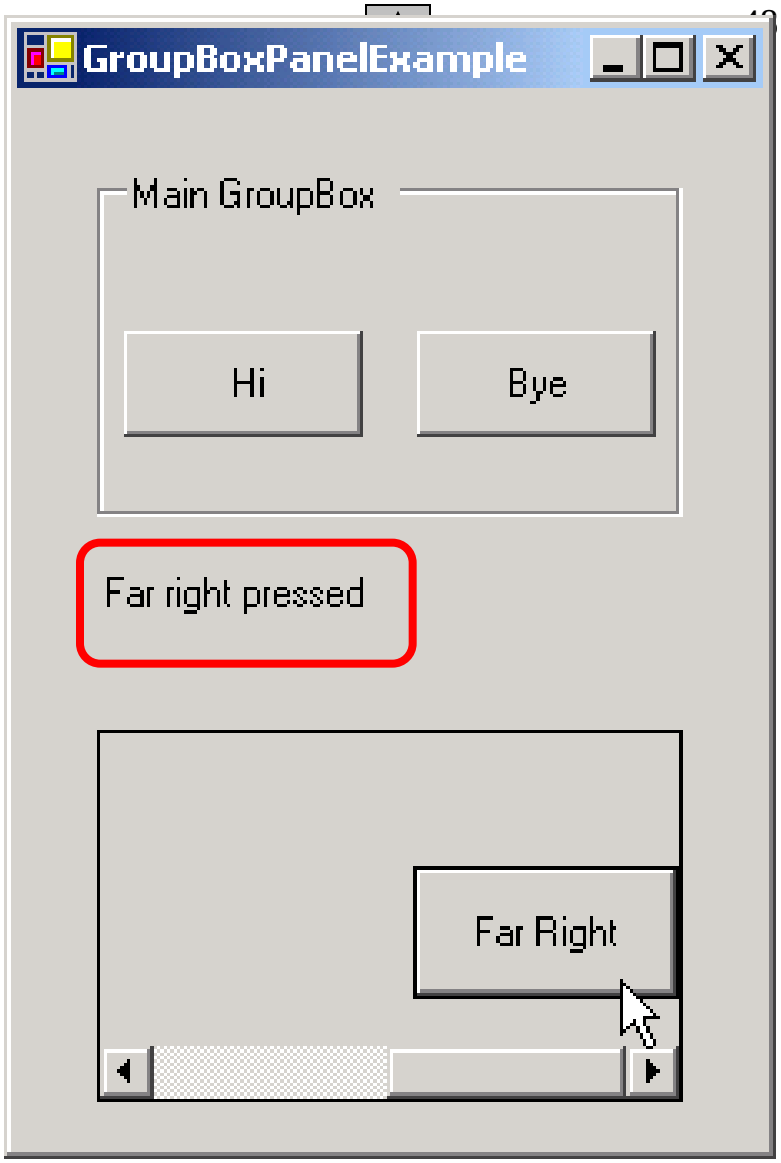
Where is setting of delegate for xx_click()

In the initialization of the win form






**leftButton_Click**




**rightButton_**Click

# 12.7 Checkboxes and RadioButtons

- State buttons
  - On/off or true/false state
  - Two buttons derived from class ButtonBase
    - CheckBox: usually for multiple choice
    - RadioButton: usually for single choice
- A font is a class with three attributes
  - i.e., name, size, style
- A style have 5 attributes:
  - i.e., bold, italic, strikeout, regular, underline
  - **Each attribute is 0 or 1, indicating true or false**
- FontStyle.Bold and FontStyle.Italic, defined beforehand, are constant (i.e., = 1)
- Note: ^ is an XOR operation, i.e.,

  1 ^ 1 = 0; 0 ^ 0 = 0;

  0 ^ 1 = 1; 1 ^ 0 = 1;

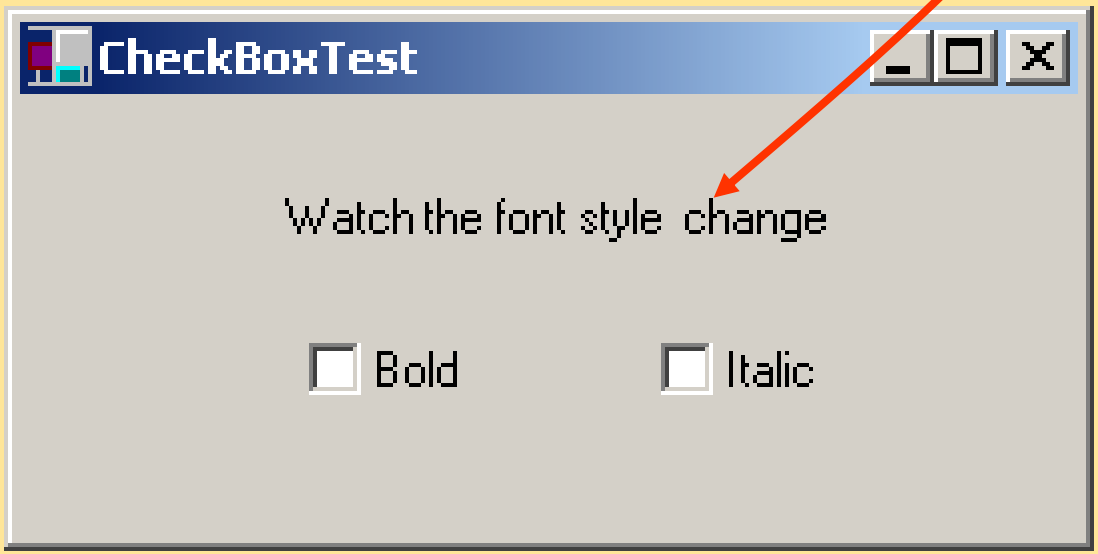
# (in Java) `Font`-related methods and constants

| Method or constant | Description |
|---|---|
| *`Font` constants, constructors and methods for drawing polygons* | |
| `public final static int PLAIN` | |
| | A constant representing a plain font style. |
| `public final static int BOLD` | |
| | A constant representing a bold font style. |
| `public final static int ITALIC` | |
| | A constant representing an italic font style. |
| `public Font( String name, int style, int size )` | |
| | Creates a `Font` object with the specified font, style and size. |
| `public int getStyle()` | |
| | Returns an integer value indicating the current font style. |
| `public int getSize()` | |
| | Returns an integer value indicating the current font size. |
| `public String getName()` | |
| | Returns the current font name as a string. |
| `public String getFamily()` | |
| | Returns the font's family name as a string. |
| `public boolean isPlain()` | |
| | Tests a font for a plain font style. Returns `true` if the font is plain. |
| `public boolean isBold()` | |
| | Tests a font for a bold font style. Returns `true` if the font is bold. |
| `public boolean isItalic()` | |
| | Tests a font for an italic font style. Returns `true` if the font is italic. |

**Style is integer**

font **is a class, check MSDN**

```
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
13  public class CheckBoxTest : System.Windows.Forms.Form {
15      private System.Windows.Forms.CheckBox boldCheckBox;
16      private System.Windows.Forms.CheckBox italicCheckBox;
18      private System.Windows.Forms.Label outputLabel;
20      private System.ComponentModel.Container components = null;
25      [STAThread]
26      static void Main() {
28          Application.Run( new CheckBoxTest() );
29      }
30
```

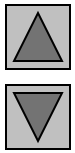


```
33      private void boldCheckBox_CheckedChanged(
34         object sender, System.EventArgs e ) {
36         outputLabel.Font =
37            new Font( outputLabel.Font.Name, outputLabel.Font.Size,
                  outputLabel.Font.Style ^ FontStyle.Bold );
40      }
44      private void italicCheckBox_CheckedChanged(
45         object sender, System.EventArgs e ) {
47         outputLabel.Font =
48            new Font( outputLabel.Font.Name, outputLabel.Font.Size,
                  outputLabel.Font.Style ^ FontStyle.Italic );
51      }
53   }
```

Old Font style

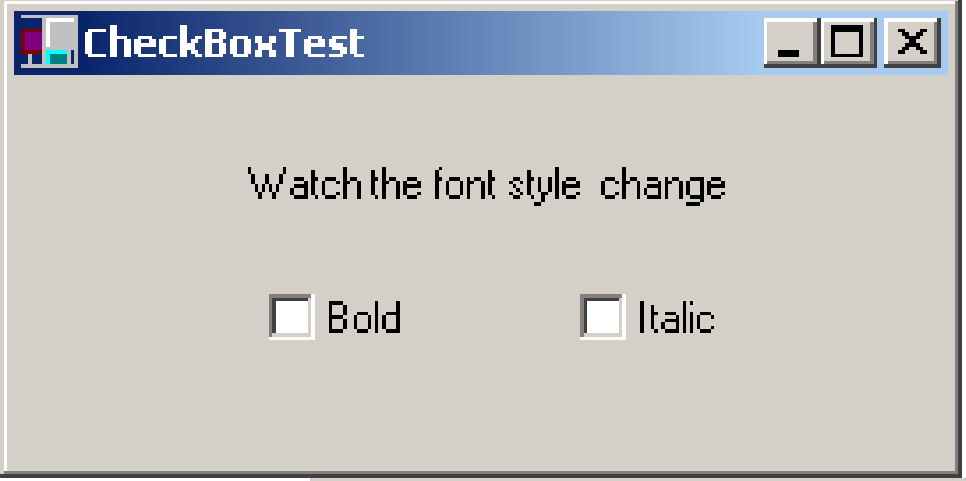
On ◂▸Off

System constant

Outline

**CheckBoxTest.cs**
**Program Output**





# FontStyle
# .NET Framework (current version)

C# | C++ | F# | VB

```
[FlagsAttribute]
public enum FontStyle
```

## 成員

| | 成員名稱 | 描述 |
|---|---|---|
| | Bold | 粗體的文字。 |
| | Italic | 斜體文字。 |
| | Regular | 一般文字。 |
| | Strikeout | 通過中間的線條的文字。 |
| | Underline | 加底線的文字。 |

Five different elements in FontStyle, which are 0 or 1

# enum

列舉型別 (也稱為列舉) 提供有效的方式，讓您定義一組可以指派給變數的具名整數常數。 例如，假設您必須定義的變數的值是代表星期幾。 則只有 7 個有意義的值是該變數所要儲存的。 若要定義這些值，您可以使用列舉型別，這是藉由使用 enum 關鍵字來宣告的。

C#

```
enum Days { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
enum Months : byte { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
```

根據預設，列舉中每個項目的基礎型別是 int。 您可以使用冒號指定其他整數數字型別 (Numeric Type)，如前述範例所示。 如需可能型別的完整清單，請參閱 enum (C# 參考)。

在您沒有為列舉程式清單的項目指定值時，會以 1 自動累加這些值。 在前述範例中，Days.Sunday 的值為 0，而 Days.Monday 的值為 1，以此類推。 如果在建立新 Days 物件時沒有明確指派值，則會使用預設值 Days.Sunday (0)。 所以在您建立列舉時，請選取邏輯上最適合的預設值，並讓其值為零。 這樣在建立列舉時即使沒有明確指派值，所有的列舉都會使用該預設值。

# Enum

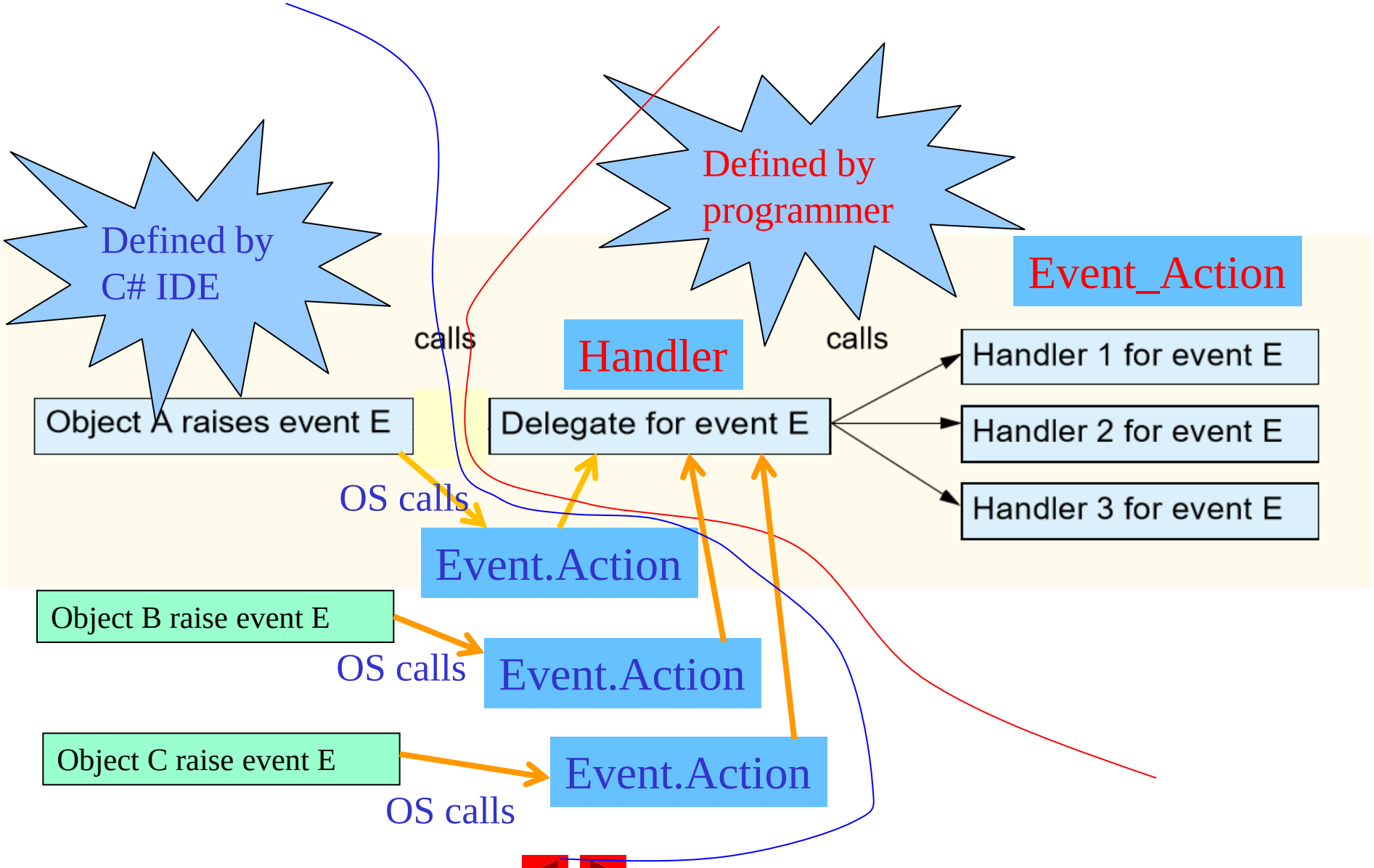
下列範例會定義另一個版本的 Days 列舉，名為 Days2。 Days2 具有 Flags 屬性，且每個值會指派為 2 的乘冪。 這樣可以讓所建立 Days2 變數的值為 Days2.Tuesday 和 Days2.Thursday。

C#

```
[Flags]
enum Days2
{
    None = 0x0,        → 0000 0000
    Sunday = 0x1,      → 0000 0001
    Monday = 0x2,      → 0000 0010
    Tuesday = 0x4,     → 0000 0100
    Wednesday = 0x8,   → 0000 1000
    Thursday = 0x10,   → 0001 0000
    Friday = 0x20,     → 0010 0000
    Saturday = 0x40    → 0100 0000
}
class MyClass
{
        0001 0100      0000 0100      0001 0100
    Days2 meetingDays = Days2.Tuesday | Days2.Thursday;
}
```

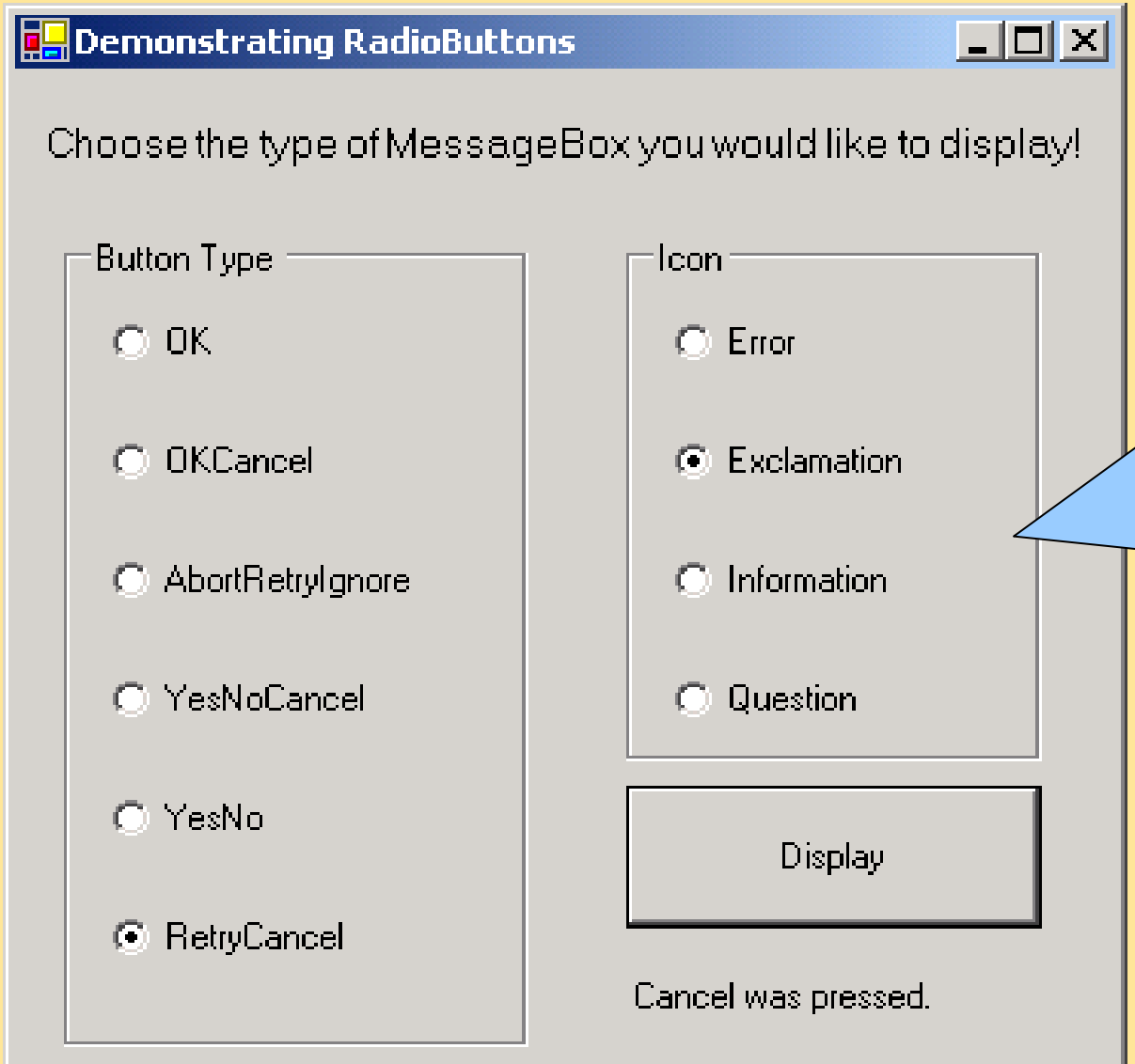
| **`RadioButton`** properties and events | Description / Delegate and Event Arguments |
|---|---|
| *Common Properties* | |
| **`Checked`** | Whether the **`RadioButton`** is checked. |
| **`Text`** | Text displayed to the right of the **`RadioButton`** (called the label). |
| *Common Events* | *(Delegate **`EventHandler`**, event arguments **`EventArgs`**)* |
| **`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. |

# C# Event's Framework



```
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
11   public class RadioButtonsTest : System.Windows.Forms.Form {
13       private System.Windows.Forms.Label promptLabel;
14       private System.Windows.Forms.Label displayLabel;
15       private System.Windows.Forms.Button displayButton;
16       private System.Windows.Forms.RadioButton questionButton;
17       private System.Windows.Forms.RadioButton informationButton;
18       private System.Windows.Forms.RadioButton exclamationButton;
19       private System.Windows.Forms.RadioButton errorButton;
20       private System.Windows.Forms.RadioButton retryCancelButton;
21       private System.Windows.Forms.RadioButton yesNoButton;
22       private System.Windows.Forms.RadioButton yesNoCancelButton;
23       private System.Windows.Forms.RadioButton okCancelButton;
24       private System.Windows.Forms.RadioButton okButton;
```

```
25      private System.Windows.Forms.RadioButton abortRetryIgnoreButton;
27      private System.Windows.Forms.GroupBox groupBox2;
28      private System.Windows.Forms.GroupBox groupBox1;
29      private MessageBoxIcon iconType = MessageBoxIcon.Error;
30      private MessageBoxButtons buttonType = MessageBoxButtons.OK;
```




Radio buttons are put in a GroupBox by visual setting (setting codes in initializeComponent)

```
33      [STAThread]
34      static void Main() {
          Application.Run( new RadioButtonsTest() );
37      }
39      private void buttonType_CheckedChanged(
        object sender, System.EventArgs e ) {
42        if ( sender == okButton )
43            buttonType = MessageBoxButtons.OK;
45        else if ( sender == okCancelButton )
46            buttonType = MessageBoxButtons.OKCancel;
48        else if ( sender == abortRetryIgnoreButton )
49            buttonType = MessageBoxButtons.AbortRetryIgnore;
51        else if ( sender == yesNoCancelButton )
52            buttonType = MessageBoxButtons.YesNoCancel;
54        else if ( sender == yesNoButton )
55            buttonType = MessageBoxButtons.YesNo;
58        else
59            buttonType = MessageBoxButtons.RetryCancel;
60      }
```

Visually new this event handler and set checkedChange for all buttons to this new handler (setting codes in initializeComponent)

```
62      private void iconType_CheckedChanged(
          object sender, System.EventArgs e ) {
65        if ( sender == errorButton )
66            iconType = MessageBoxIcon.Error;
68        else if ( sender == exclamationButton )
69            iconType = MessageBoxIcon.Exclamation
71        else if ( sender == informationButton )
72            iconType = MessageBoxIcon.Information;
73        else // only one option left--display question mark
74            iconType = MessageBoxIcon.Question;
75      }
77      protected void displayButton_Click(
         object sender, System.EventArgs e ) {
80        DialogResult result = MessageBox.Show( "This is Your Custom
            MessageBox.", "Custom MessageBox",buttonType, iconType, 0, 0 );
```

Visually new this event handler and set checkedChange for all buttons to this new handler

**Exclamation** icon type


**OKCancel** button type

**Error** icon type


**OK** button type

```
84          switch ( result ) {
86             case DialogResult.OK:
87                displayLabel.Text = "OK was pressed.";
88                break;
89             case DialogResult.Cancel:
90                displayLabel.Text = "Cancel was pressed.";
91                break;
92             case DialogResult.Abort:
93                displayLabel.Text = "Abort was pressed.";
94                break;
95             case DialogResult.Retry:
96                displayLabel.Text = "Retry was pressed.";
97                break;
98             case DialogResult.Ignore:
99                displayLabel.Text = "Ignore was pressed.";
100               break;
101            case DialogResult.Yes:
102               displayLabel.Text = "Yes was pressed.";
103               break;
104            case DialogResult.No:
105               displayLabel.Text = "No was pressed.";
106               break;
107         }
108      }
109   }
```

**Exclamation** icon type


**OKCancel** button type

**Error** icon type

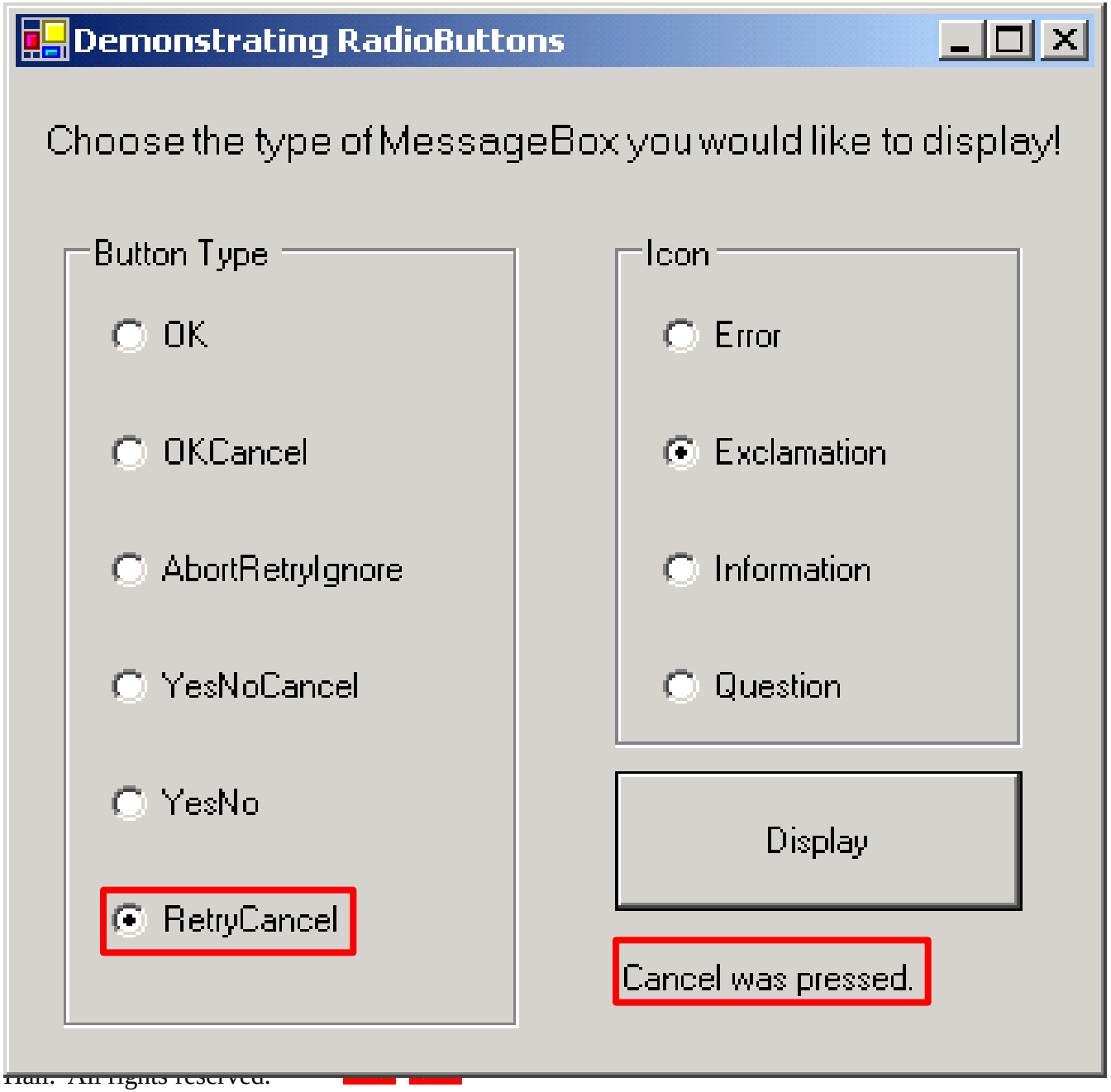

**OK** button type

**Information** icon type


**AbortRetryIgnore** button type

**Question** icon type


**YesNoCancel** button type



# 12.8 PictureBoxes

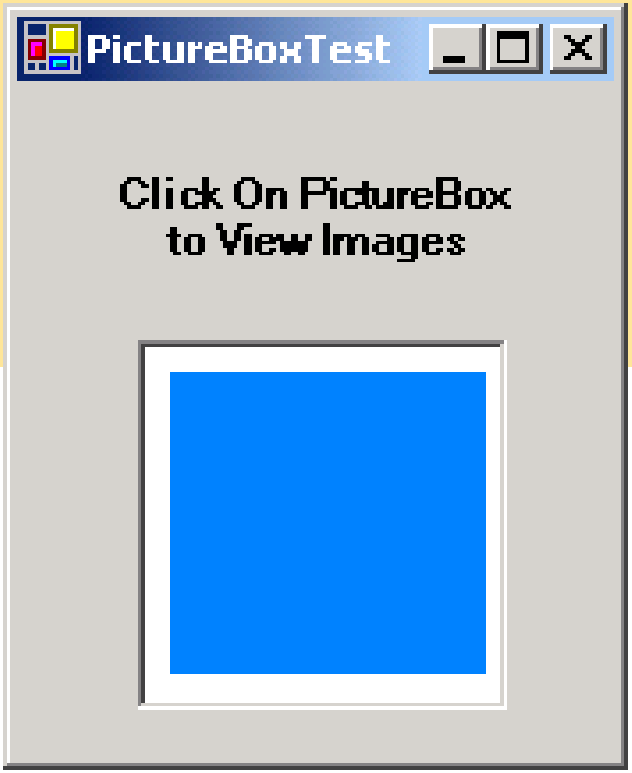
- Class PictureBox
  - Displays an image
    - Image set by object of class Image.
      - The Image property sets the Image object to use
      - SizeMode property sets how the image is displayed

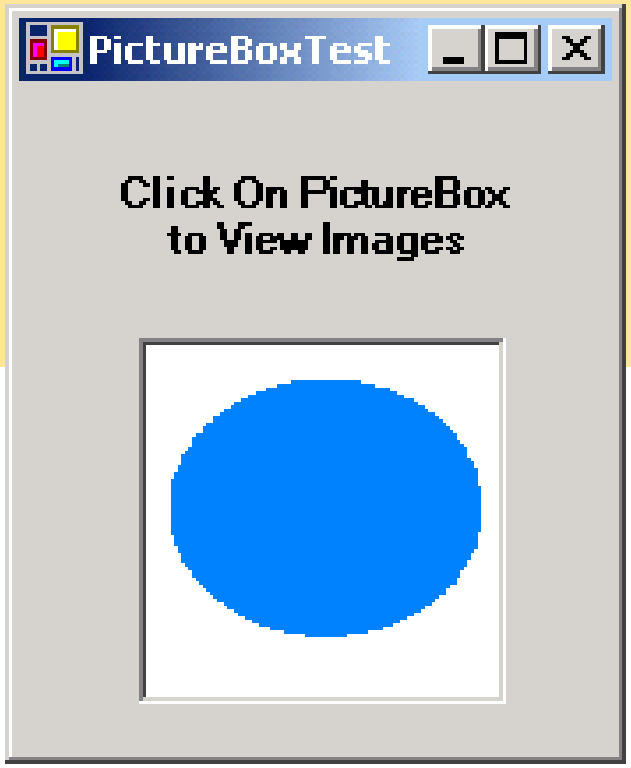
| **PictureBox** properties and events | Description / Delegate and Event Arguments |
|---|---|
| *Common Properties* | |
| **Image** | Image to display in the **PictureBox**. |
| **SizeMode** | Enumeration that controls image sizing and positioning. Values **Normal** (default), **StretchImage**, **AutoSize**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

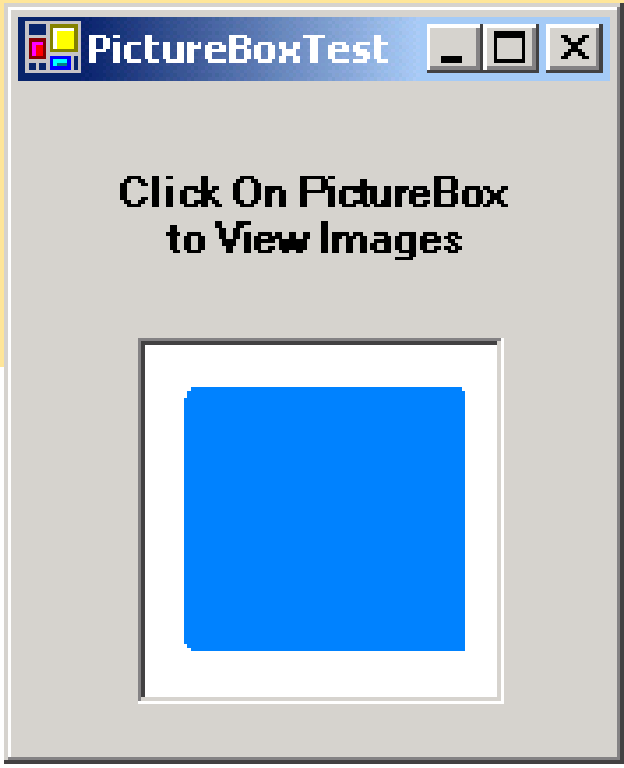
```
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
9    using System.IO;
11   public class PictureBoxTest : System.Windows.Forms.Form {
13      private System.Windows.Forms.PictureBox imagePictureBox;
14      private System.Windows.Forms.Label promptLabel;
15      private int imageNum = -1;
17      [STAThread]
18      static void Main() {
20         Application.Run( new PictureBoxTest() );
21      }
23      private void imagePictureBox_Click(
24         object sender, System.EventArgs e ) {
26         imageNum = ( imageNum + 1 ) % 3;
```

imageNum
in {0, 1, 2}

```
28          imagePictureBox.Image = Image.FromFile(
             Directory.GetCurrentDirectory()+ "\\images\\image" +
             imageNum + ".bmp" );
31       }
32   }
```



## Mouse Events, Delegates and Event Arguments

*Mouse Events (Delegate* **`EventHandler`**, *event arguments* **`EventArgs`***)*

| | |
|---|---|
| **`MouseEnter`** | Raised if the mouse cursor enters the area of the control. |
| **`MouseLeave`** | Raised if the mouse cursor leaves the area of the control. |

*Mouse Events (Delegate* **`MouseEventHandler`**, *event arguments* **`MouseEventArgs`***)*

| | |
|---|---|
| **`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. |

*Class* **`MouseEventArgs`** *Properties*

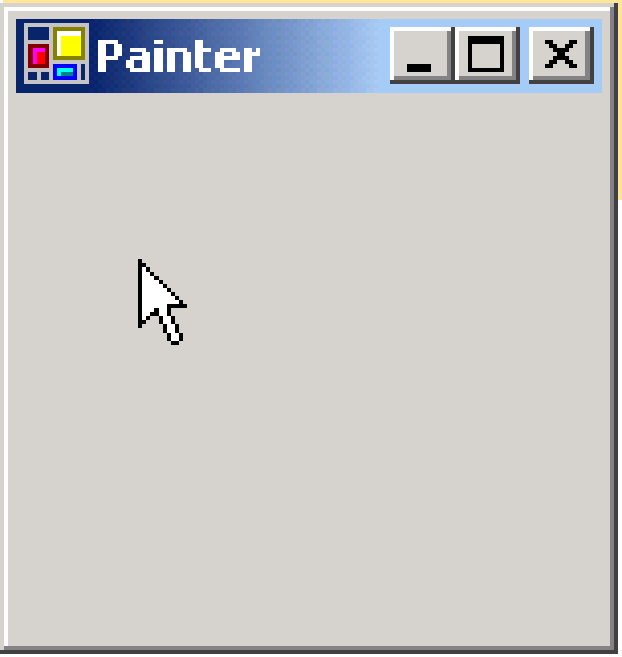
| | |
|---|---|
| **`Button`** | Mouse button that was pressed (**`left`**, **`right`**, **`middle`** or **`none`**). |
| **`Clicks`** | The number of times the mouse button was clicked. |
| **`X`** | The *x*-coordinate of the event, relative to the control. |
| **`Y`** | The *y*-coordinate of the event, relative to the control. |

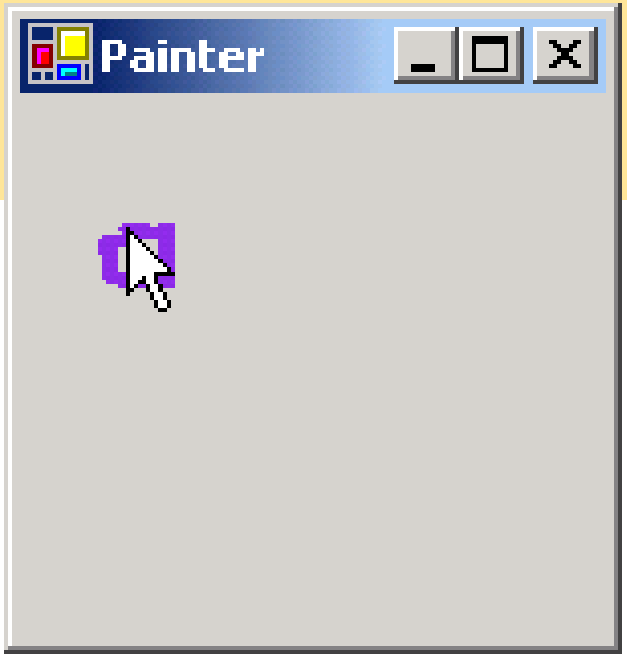
# 12.9 Mouse Event Handling

- 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

```
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
12  public class Painter : System.Windows.Forms.Form {
14      bool shouldPaint = false;
17      [STAThread]
18      static void Main() {
20          Application.Run( new Painter() );
21      }
24      private void Painter_MouseDown(
25          object sender, System.Windows.Forms.MouseEventArgs e ) {
27          shouldPaint = true;
28      }
31      private void Painter_MouseUp(
32          object sender, System.Windows.Forms.MouseEventArgs e ) {
34          shouldPaint = false;
35      }
```

```
39   protected void Painter_MouseMove(
40         object sender, System.Windows.Forms.MouseEventArgs e ) {
42      if ( shouldPaint ) {
44         Graphics graphics = CreateGraphics();
45         graphics.FillEllipse(new SolidBrush(Color.BlueViolet ), e.X, e.Y, 4, 4);
48       }
50    }
52   }
```




## Keyboard Events, Delegates and Event Arguments

| | |
|---|---|
| *Key Events (Delegate* **`KeyEventHandler`**, *event arguments* **`KeyEventArgs`***)* | |
| **`KeyDown`** | Raised when key is initially pushed down. |
| **`KeyUp`** | Raised when key is released. |
| *Key Events (Delegate* **`KeyPressEventHandler`**, *event arguments* **`KeyPressEventArgs`***)* | |
| **`KeyPress`** | Raised when key is pressed. Occurs repeatedly while key is held down, at a rate specified by the operating system. |
| *Class* **`KeyPressEventArgs`** *Properties* | |
| **`KeyChar`** | Returns the ASCII character for the key pressed. |
| **`Handled`** | Whether the **`KeyPress`** event was handled. |
| *Class* **`KeyEventArgs`** *Properties* | |
| **`Alt`** | Indicates whether the *Alt* key was pressed. |
| **`Control`** | Indicates whether the *Control* key was pressed. |
| **`Shift`** | Indicates whether the *Shift* 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 (*Alt*, *Control* and *Shift*). Used to determine modifier key information only. |

# 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

```
3    using System;
4    using System.Drawing;
5    using System.Collections;
6    using System.ComponentModel;
7    using System.Windows.Forms;
8    using System.Data;
11   public class KeyDemo : System.Windows.Forms.Form {
13      private System.Windows.Forms.Label charLabel;
14      private System.Windows.Forms.Label keyInfoLabel;
15      private System.ComponentModel.Container components = null;
17      [STAThread]
18      static void Main() {
20         Application.Run( new KeyDemo() );
21      }
23      protected void KeyDemo_KeyPress(
        object sender, System.Windows.Forms.KeyPressEventArgs e) {
26         charLabel.Text = "Key pressed: " + e.KeyChar;
27      }
```

```
29      private void KeyDemo_KeyDown(
         object sender, System.Windows.Forms.KeyEventArgs e ) {
32          keyInfoLabel.Text =
33              "Alt: " + ( e.Alt ? "Yes" : "No") + '\n' +
34              "Shift: " + ( e.Shift ? "Yes" : "No" ) + '\n' +
35              "Ctrl: " + ( e.Control ? "Yes" : "No" ) + '\n' +
36              "KeyCode: " + e.KeyCode + '\n' +
37              "KeyData: " + e.KeyData + '\n' +
38              "KeyValue: " + e.KeyValue;
39      }
41      private void KeyDemo_KeyUp(
         object sender, System.Windows.Forms.KeyEventArgs e ){
44          keyInfoLabel.Text = "";
45          charLabel.Text = "";
46      }
```