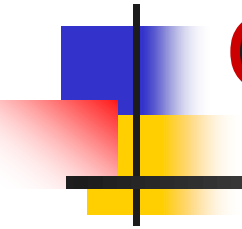


Chapter 2. Graphical User Interface

A reference of MSDN Library for Visual Studio 2017

IT Faculty, TDM University

Contents

- Introduction
- Windows Forms
- Event-Handling Model
- Control Properties and Layout
- Some basic GUI components
- Mouse Event Handling
- Keyboard Event Handling

Introduction

- Graphical User Interface (GUI)
  - Allow interaction with program visually
  - Is an object, accessed via keyboard or mouse
- Some basic GUI components
  - **`Label`**: Used to display text or images that cannot be edited by the user.
  - **`TextBox`**: An area in which the user inputs data from the keyboard. The area also can display information.

Introduction

- Some basic GUI components
  - **`Button`**: allows the user to click it to perform an action.
  - **`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.
  - **`Panel`**: A container in which components can be placed.

Windows Forms

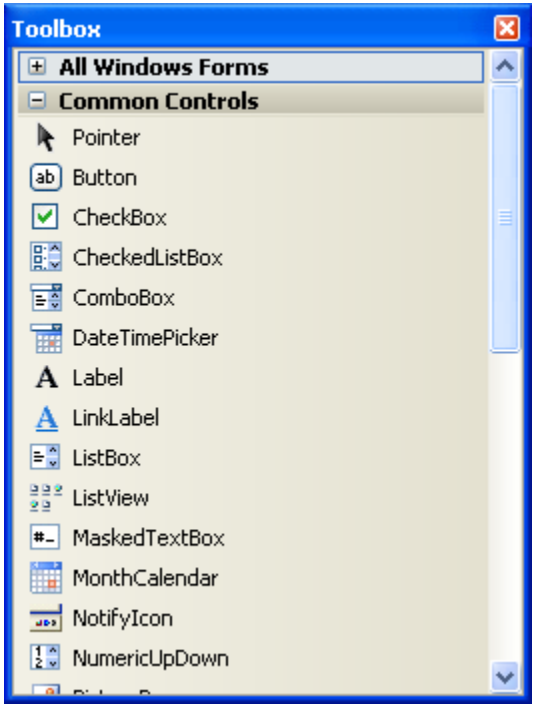
- WinForms
  - Create GUIs for programs
  - Element on the desktop
  - Represented by:
    - Dialog
    - Window
    - MDI window

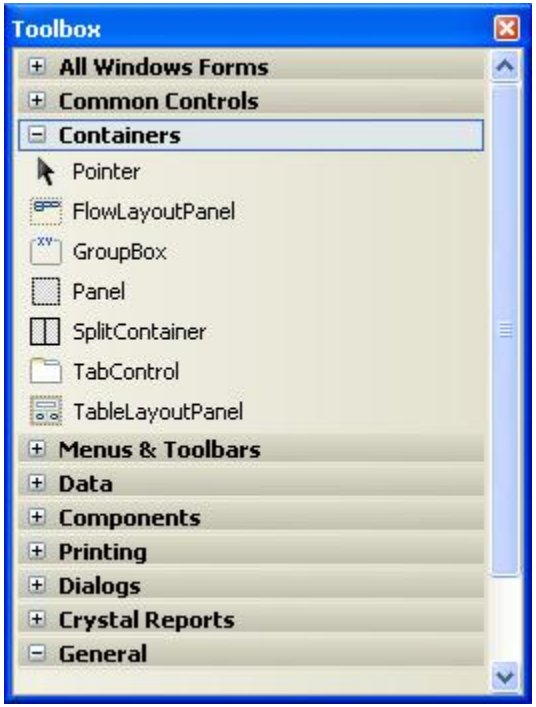
Windows Forms

- 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

Windows Forms

- Controls for Windows Forms





Windows Forms

- Common Properties
  - **`AcceptButton`**: Which button will be clicked when *Enter* is pressed.
  - **`AutoScroll`**: Whether scrollbars appear when needed (if data fills 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`**).

Windows Forms

- Common Properties
  - **`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.

Windows Forms

- Common Events
  - **`Load`**: Occurs before a form is shown. This event is the default when the form is double-clicked in the Visual Studio .NET designer.
  - **`FormClosing`**: Occurs whenever the user closes the form.
- Example
  - Manipulate the **`Form`** properties

Event-Handling Model

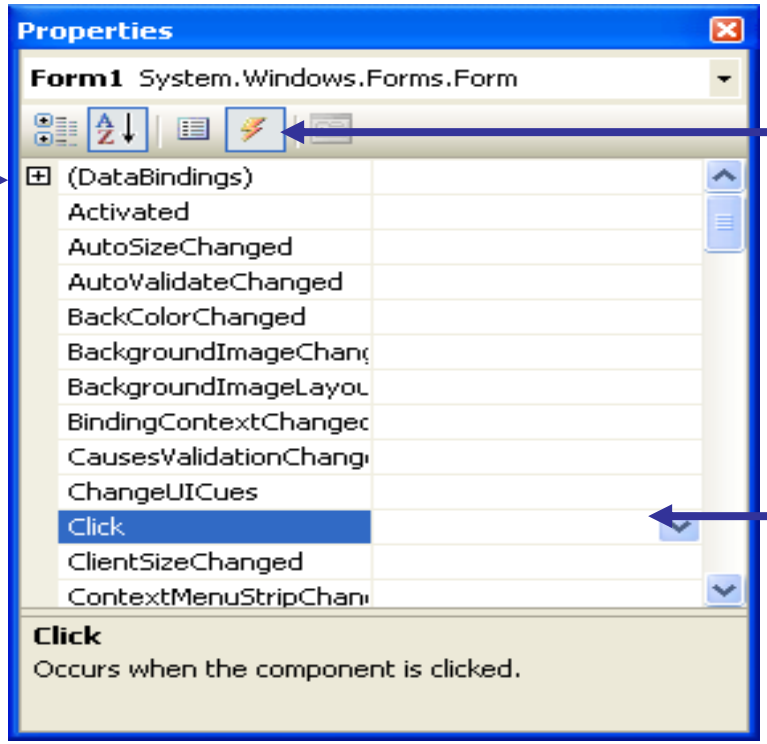
- GUIs are event driven
- Event handlers
  - Methods that process events and perform tasks.
- Associated delegate
  - Objects that reference methods
  - Contain lists of method references
    - Must have same signature
  - Intermediaries for objects and methods
  - Signature for control's event handler

Event-Handling Model

Basic Event Handling

- Event handler
  - Must have same signature as corresponding delegate
  - 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

Basic Event Handling



List of events supported by control

Selected event

Event description

Events icon

Current even handler (none)

Control Properties and Layout

- 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)

Control Properties and Layout

- Common Properties
  - **`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.

Control Properties and Layout

- Common Properties
  - **`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).

Control Properties and Layout

- Common Properties
  - **`Visible`**: Whether the control is visible or hidden.
  - **`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.

Control Properties and Layout

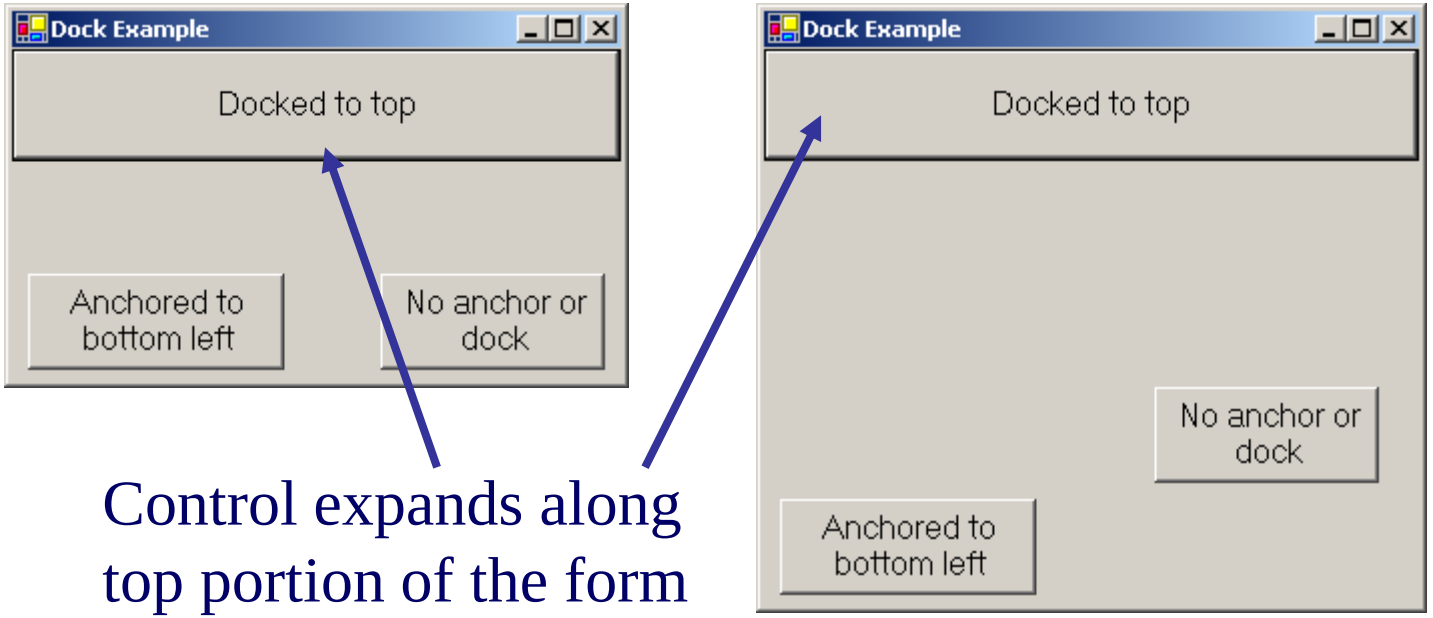
- Common Properties
  - **`DockPadding`** (for containers): Sets the dock spacing for controls inside the container. Default is zero, so controls appear flush against the side of the container.
  - **`Location`**: Size of the control. Takes a **`Size`** structure, which has properties **`Height`** and **`Width`**.
  - **`MinimumSize`**, **`MaximumSize`** (for Windows Forms): The minimum and maximum size of the form.

Control Properties and Layout

- Common Methods
  - **`Focus`**: Transfers the focus to the control.
  - **`Hide`**: Hides the control (sets **`Visible`** to **`false`**).
  - **`Show`**: Shows the control (sets **`Visible`** to **`true`**).

Control Properties and Layout

- Example
  - Manipulating the **`Anchor`** property of control



Labels

- Used to display text or images that cannot be edited by the user
- Common Properties
  - **`Fonts`**: 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.

TextBoxes

- An area in which the user inputs data from the keyboard. The area also can display information.
- 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.

TextBoxes

- Common Properties
  - **`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`**.

TextBoxes

- Common Properties
  - **`ScrollBars`**: For multiline textboxes, indicates which scrollbars appear (**`none`**, **`horizontal`**, **`vertical`** or **`both`**).
  - **`Text`**: The text to be displayed in the text box.
- Common Events
  - **`TextChanged`**: Occurs when text changes in **`TextBox`** (the user added or deleted characters). Default event when this control is double clicked in the designer.

Buttons

- Button control allows the user to click it to perform an action. The Button control can display both text and images.
- Common Property
  - **`Text`**: Text displayed on the **`Button`** face.
- Common Events
  - **`Click`**: Occurs when the user clicks the control. Default event when this control is double clicked in the designer.

Examples

- Example 1: ax + b=0
- Example 2:
  - input: month, year
  - Output: number of date.
- Example 3: s=1+1/2+1/3+..+1/n (n>0).

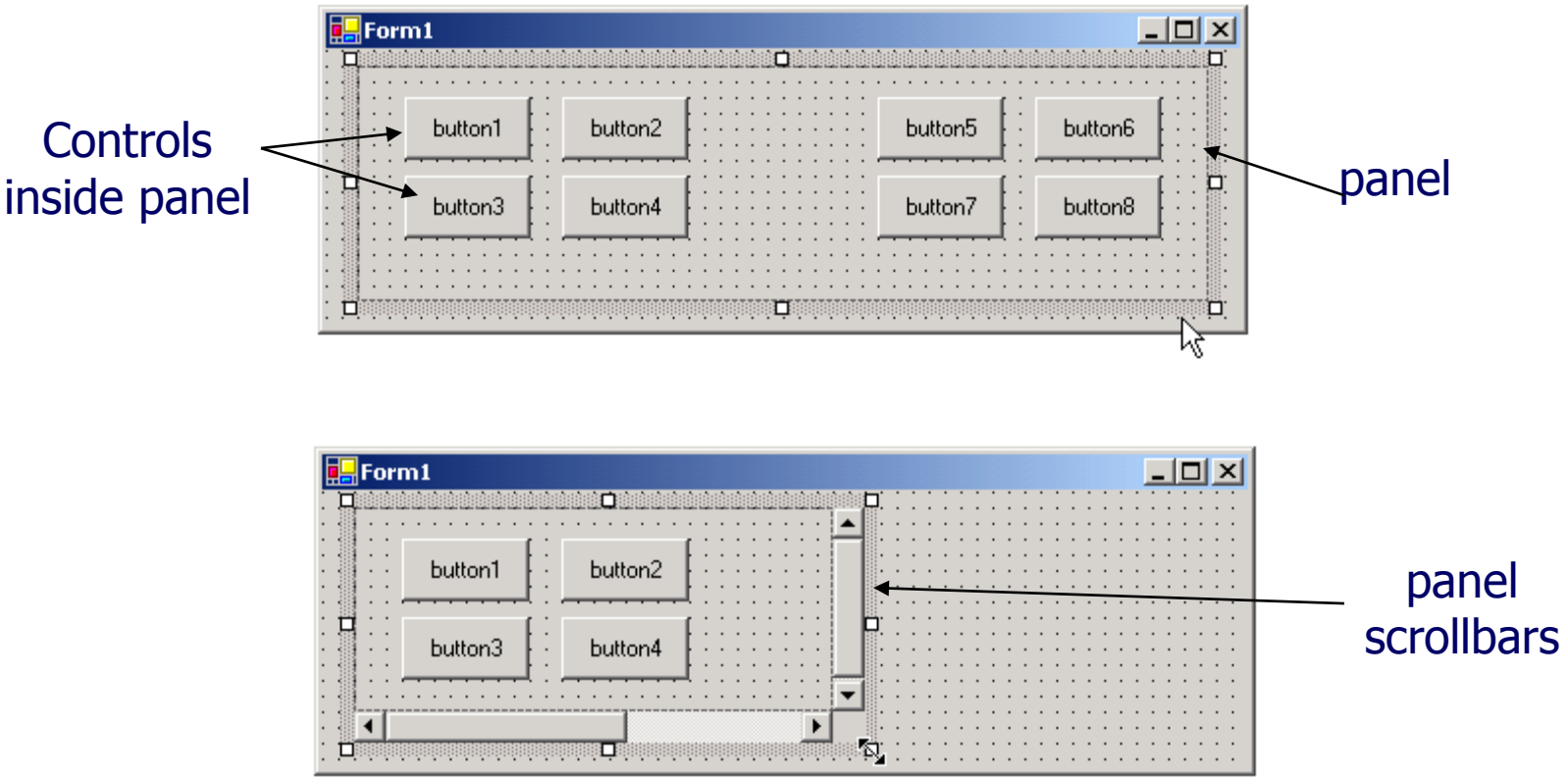
GroupBoxes

- **`GroupBoxes`** are used to group other controls. The **`GroupBox`** control is similar to the Panel control; however, only the **`GroupBox`** control displays a caption, and only the **`Panel`** control can have scroll bars.
- Common Properties
  - **`Controls`**: The controls that the **`GroupBox`** contains.
  - **`Text`**: Text displayed on the top portion of the **`GroupBox`** (its caption).

Panels

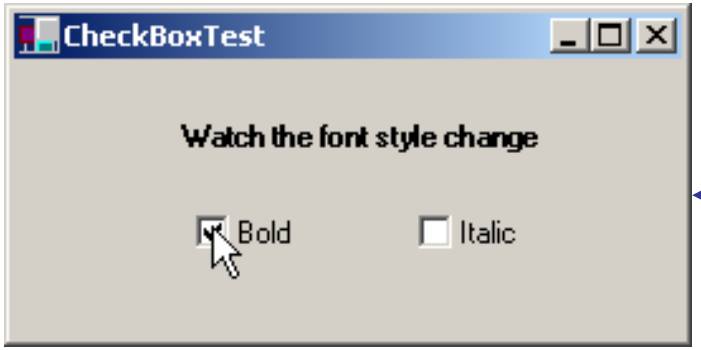
- Panels are used to group other controls. They can have scrollbars.
- Common Properties
  - **`AutoScroll`**: Whether scrollbars appear when the **`Panel`** is too small to hold its controls. Default is **`false`**.
  - **`BorderStyle`**: Border of the **`Panel`**. Default is **`None`**.
  - **`Controls`**: The controls that the **`Panel`** contains.

Panels


Controls
inside panel
Form1
button1
button2
button5
button6
button3
button4
button7
button8
panel
Form1
button1
button2
button3
button4
panel
scrollbars

Checkboxes

- A control that is either selected or not selected
  - True or false state
  - No restriction on usage



Result when bold is selected

Checkboxes

- Common Properties
  - **`Checked`**: Whether or not the **`CheckBox`** has been checked.
  - **`Text`**: Text displayed to the right of the **`CheckBox`** (called the label).
- Common Events
  - **`CheckedChanged`**: Occurs whenever the **`Check`** property is changed. Default event when this control is double clicked in the designer.

CheckBoxes

- Common Events
  - **`CheckStateChanged`**: Occurs whenever the **`CheckState`** property is changed.
- Example

Result when bold is selected

Result when both styles are selected

CheckBoxes

- Code pattern

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
   {
    label1.Font = new Font(label1.Font.Name,
                           label1.Font.Size,
                           label1.Font.Style ^ FontStyle.Bold);
   }
private void checkBox2_CheckedChanged(object sender, EventArgs e)
   {
    label1.Font = new Font(label1.Font.Name,
                           label1.Font.Size,
                           label1.Font.Style ^ FontStyle.Italic);
   }
```

RadioButtons

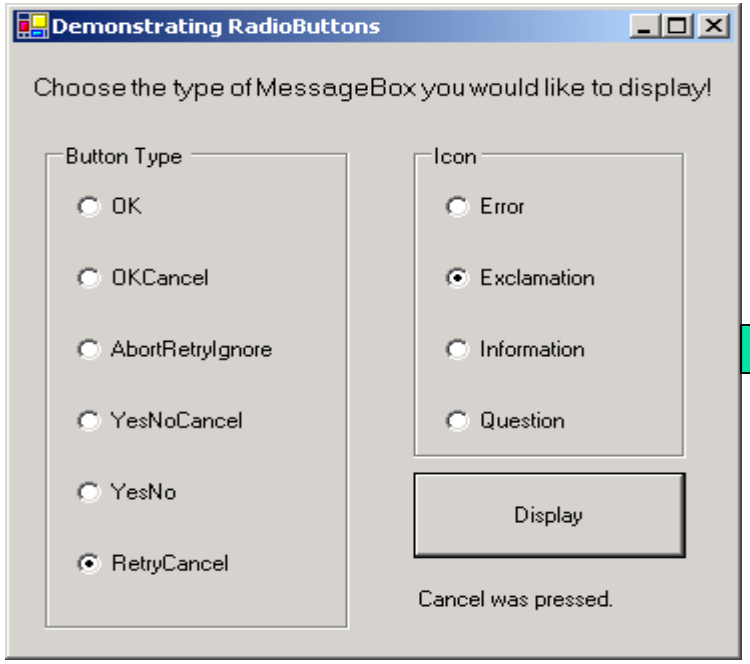
- RadioButton
  - Grouped together
  - Only one can be true
  - Mutually exclusive options

RadioButtons

- Common Properties
  - **`Checked`**: Whether the **`RadioButton`** is checked.
  - **`Text`**: Text displayed to the right of the **`RadioButton`** (called the label).
- Common Events
  - **`Click`**: Occurs when the control is clicked.
  - **`CheckedChanged`**:
    - Occurs whenever the **`Check`** property changes value.
    - Default event when this control is double clicked in the designer.

RadioButtons

- Example



Radio button style allow user to select one per column

Exclamation icon type

OKCancel button type

Error icon type

OK button type

RadioButtons

- Code pattern

```
if (btnOkCancel.Checked  == true){
  if (btnExclamation.Checked == true)
   {
    DialogResult dlg;
    dlg = MessageBox.Show("OkCancel and Exclamation were pressed,
                          "Exclamation...",
                          MessageBoxButtons.OKCancel,
                          MessageBoxIcon.Exclamation);
    if (dlg == DialogResult.OK)
     {
       MessageBox.Show("Ok button was Pressed");
     }
   }
}
```

PictureBoxes

- Used to display graphics in bitmap, GIF, JPEG, metafile, or icon format.
- Common Properties
  - **`Image`**: Image to display in the **`PictureBox`**.
  - **`SizeMode`**: Enumeration that controls image sizing and positioning.
    - **`Normal`** (default) puts image in top-left corner of **`PictureBox`** and **`CenterImage`** puts image in middle.
    - **`StretchImage`** resizes image to fit in **`PictureBox`**.
    - **`AutoSize`** resizes **`PictureBox`** to hold image.

PictureBoxes

- Common Events
  - **`Click`**:
    - Occurs when the user clicks the control.
    - Default event when this control is double clicked in the designer.

Mouse Event Handling

- Mouse Events
  - **`MouseEnter`**: Occurs when the mouse cursor enters the area of the control.
  - **`MouseLeave`**: Occurs when the mouse cursor leaves the area of the control.
  - **`MouseDown`**: Occurs when the mouse button is pressed while its cursor is over the area of the control.

Mouse Event Handling

- Mouse Events
  - **`MouseHover`**: Occurs when the mouse cursor hovers over the area of the control.
  - **`MouseMove`**: Occurs when the mouse cursor is moved while in the area of the control.
  - **`MouseUp`**: Occurs when the mouse button is released when the cursor is over the area of the control.

Mouse Event Handling

- **`MouseEventArg`:** Provides data for the **`MouseUp`**, **`MouseDown`**, and **`MouseMove`** events
  - **`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.

Keyboard Event Handling

- Common Events
  - **`KeyDown`**: Occurs when key is initially pushed down.
  - **`KeyUp`**: Occurs when key is released.
    - **`KeyEventArg`** provides data for the **`KeyDown`** or **`KeyUp`** event.
  - **`KeyPress`**: Occurs when the control has focus and the user presses and releases a key.
    - **`KeyPressEventArg`** provides data for the **`KeyPress`** event

Keyboard Event Handling

- KeyPressEventArgs Properties
  - Provides data for the **`KeyPress`** event
  - **`KeyChar`**: Returns the ASCII character for the key pressed.
  - **`Alt`**: Indicates whether the *Alt* key was pressed.
  - **`Control`**: Indicates whether the *Control* key was pressed.
  - **`Handled`**: Whether or not the **`KeyPress`** event was handled.

Keyboard Event Handling

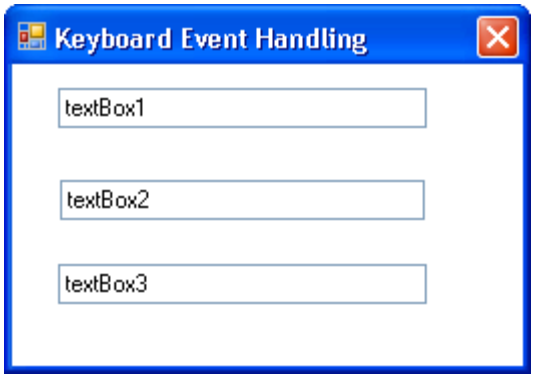
- KeyEventArgs Properties
  - Provides data for the **`KeyDown`** or **`KeyUp`** event.
  - **`Shift`**: Indicates whether the *Shift* key was pressed.
  - **`Handled`**: Whether the event was handled.
  - **`KeyCode`**: Returns the key code for the key, as a **`Keys`** enumeration. This does not include modifier key information. Used to test for a specific key.

Keyboard Event Handling

- KeyEventArgs Properties
  - **`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.

Keyboard Event Handling

- Example
  - Presses Enter, up arrow, down arrow to move curror.
  - Code pattern for textBox2




```
private void textBox2_KeyDown(object sender, KeyEventArgs e)
{
 if ((e.KeyValue == 13) || (e.KeyValue == 40)) textBox3.Focus();
 if (e.KeyValue == 38) textBox1.Focus();
}
```