Chapter 14
Graphical User Interfaces with Windows Forms: Part 1

Visual C# 2012 How to Program

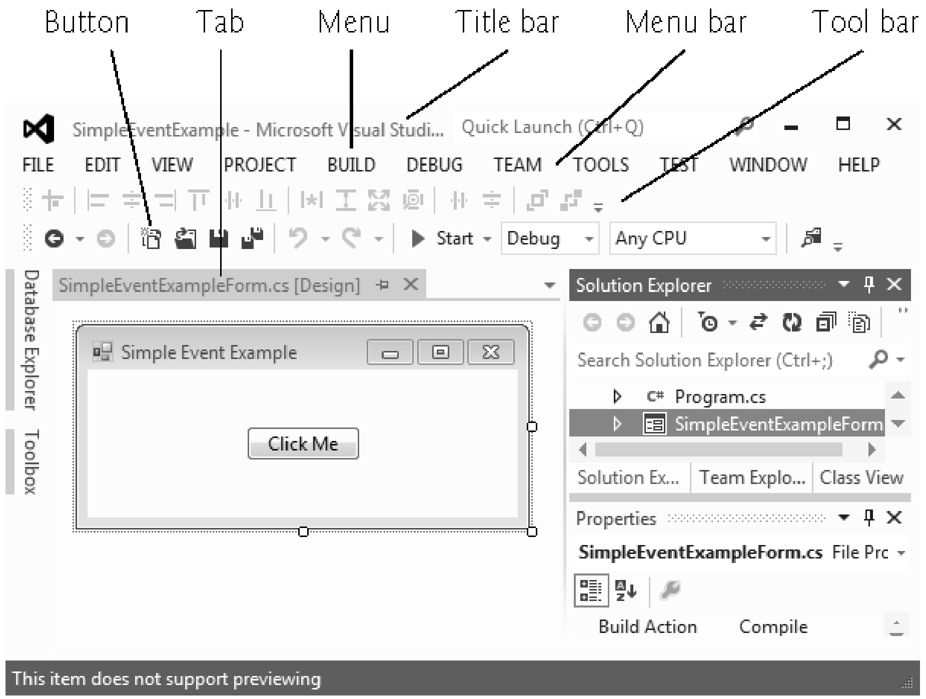
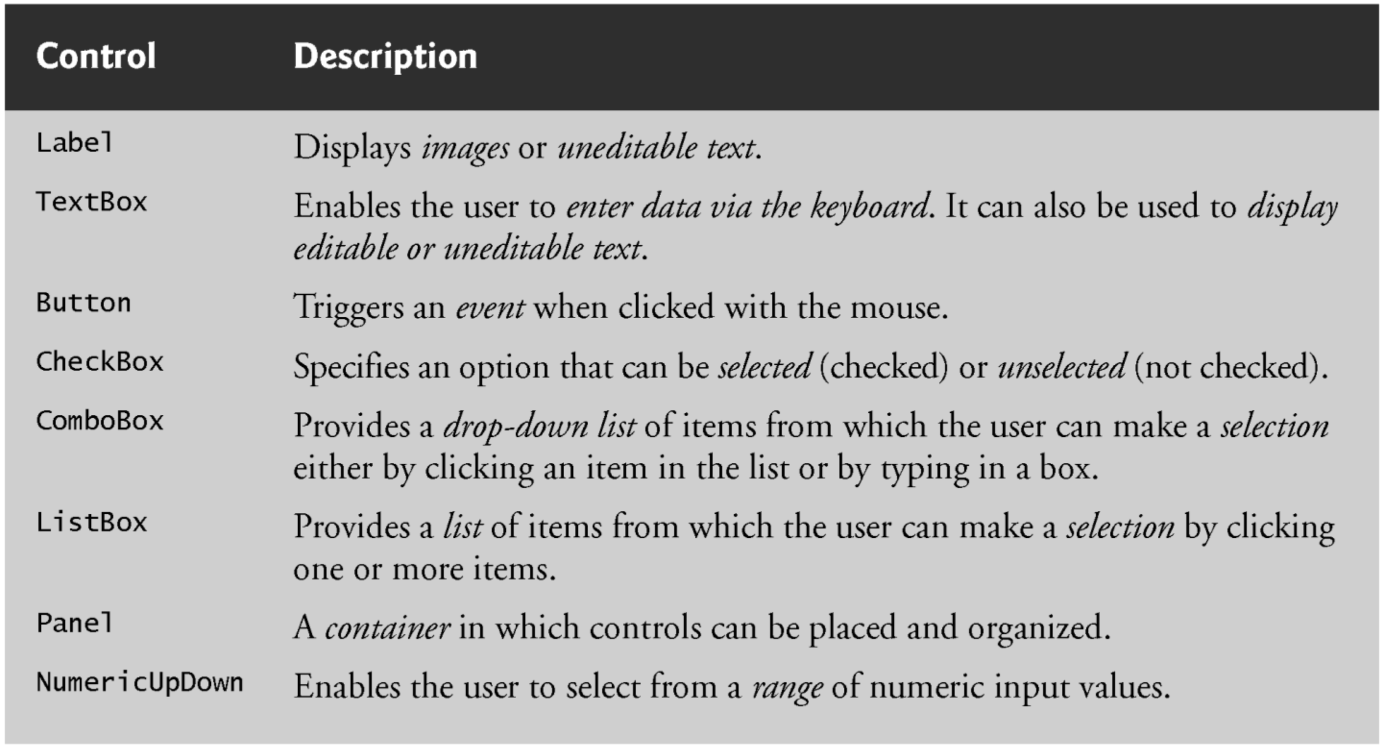

Button
Tab
Menu
Title bar
Menu bar
Tool bar
SimpleEventExample - Microsoft Visual Studi...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Start
Debug
Any CPU
SimpleEventExampleForm.cs [Design]
Database Explorer
Toolbox
Simple Event Example
Click Me
Solution Explorer
Search Solution Explorer (Ctrl+;)
Program.cs
SimpleEventExampleForm
Solution Ex...
Team Explo...
Class View
Properties
SimpleEventExampleForm.cs File Prc
Build Action
Compile
This item does not support previewing

Fig. 14.1 | GUI controls in the Visual Studio Express 2012 for Windows Desktop window.



Control	Description
`Label`	Displays *images* or *uneditable text*.
`TextBox`	Enables the user to *enter data via the keyboard*. It can also be used to *display editable or uneditable text*.
`Button`	Triggers an *event* when clicked with the mouse.
`CheckBox`	Specifies an option that can be *selected* (checked) or *unselected* (not checked).
`ComboBox`	Provides a *drop-down list* of items from which the user can make a *selection* either by clicking an item in the list or by typing in a box.
`ListBox`	Provides a *list* of items from which the user can make a *selection* by clicking one or more items.
`Panel`	A *container* in which controls can be placed and organized.
`NumericUpDown`	Enables the user to select from a *range* of numeric input values.

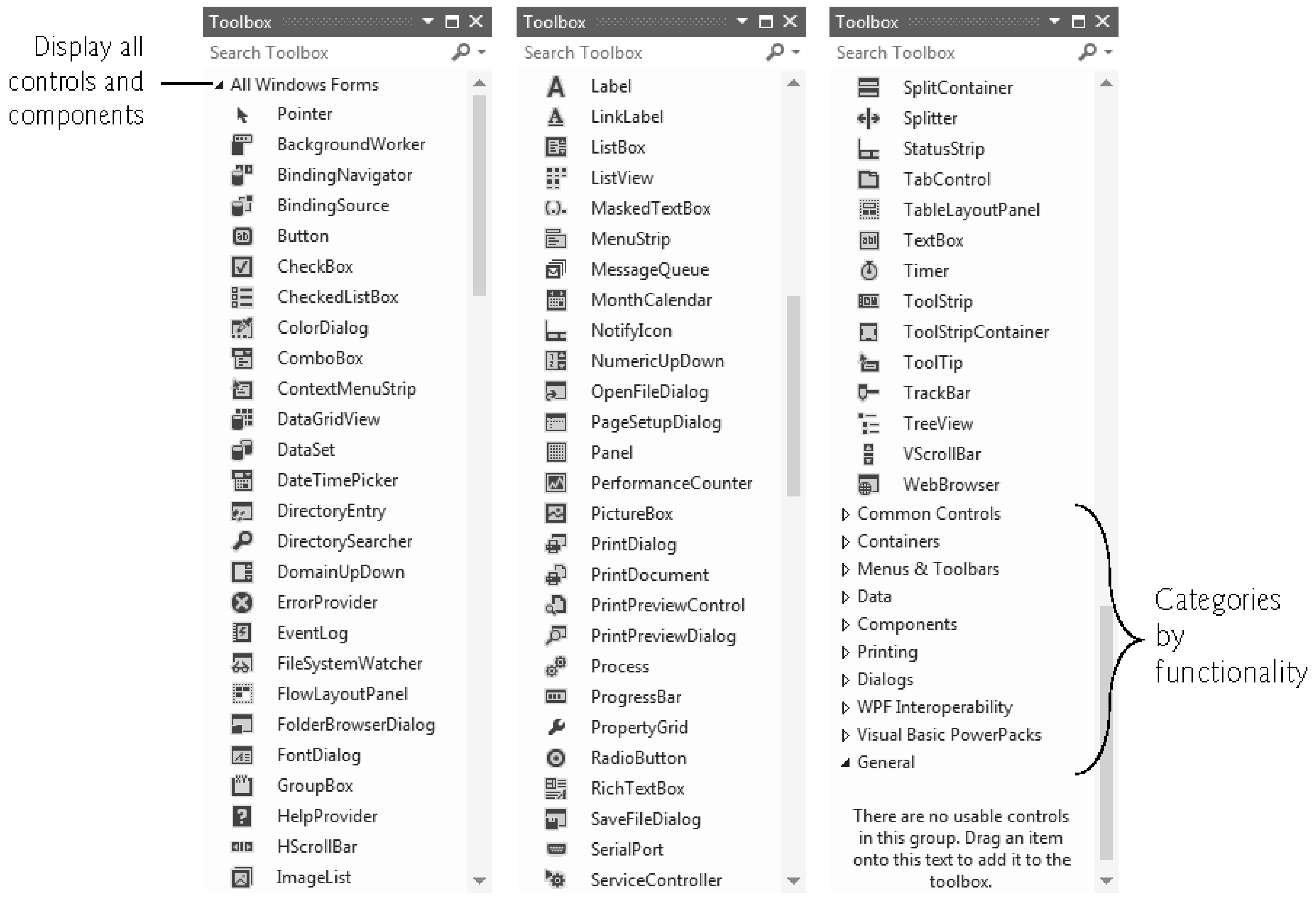
Fig. 14.2 | Some basic GUI controls.

14.2 Windows Forms

- A `Form` is a graphical element that appears on your computer's desktop; it can be a dialog, a window or an **MDI window**.
- A *component* is an instance of a class that implements the **`IComponent` interface**, which defines the behaviors that components must implement, such as how the component is loaded.
- A *control*, such as a `Button` or `Label`, has a graphical representation at runtime.

14.2 Windows Forms (Cont.)

- Figure 14.3 displays the Windows Forms controls and components from the C# **Toolbox**.
- To add a control or component, select it from the **Toolbox** and drag it onto the `Form`.


Display all controls and components
Categories by functionality

Toolbox

Search Toolbox

All Windows Forms
- Pointer
- BackgroundWorker
- BindingNavigator
- BindingSource
- Button
- CheckBox
- CheckedListBox
- ColorDialog
- ComboBox
- ContextMenuStrip
- DataGridView
- DataSet
- DateTimePicker
- DirectoryEntry
- DirectorySearcher
- DomainUpDown
- ErrorProvider
- EventLog
- FileSystemWatcher
- FlowLayoutPanel
- FolderBrowserDialog
- FontDialog
- GroupBox
- HelpProvider
- HScrollBar
- ImageList
- Label
- LinkLabel
- ListBox
- ListView
- MaskedTextBox
- MenuStrip
- MessageQueue
- MonthCalendar
- NotifyIcon
- NumericUpDown
- OpenFileDialog
- PageSetupDialog
- Panel
- PerformanceCounter
- PictureBox
- PrintDialog
- PrintDocument
- PrintPreviewControl
- PrintPreviewDialog
- Process
- ProgressBar
- PropertyGrid
- RadioButton
- RichTextBox
- SaveFileDialog
- SerialPort
- ServiceController
- SplitContainer
- Splitter
- StatusStrip
- TabControl
- TableLayoutPanel
- TextBox
- Timer
- ToolStrip
- ToolStripContainer
- ToolTip
- TrackBar
- TreeView
- VScrollBar
- WebBrowser

Common Controls
Containers
Menus & Toolbars
Data
Components
Printing
Dialogs
WPF Interoperability
Visual Basic PowerPacks
General

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Fig. 14.3 | Components and controls for Windows Forms.

`Form` **properties, methods and an event**	**Description**
Common Methods	
`Close`	Closes a `Form` and *releases all resources*, such as the memory used for the `Form`'s contents. A closed `Form` cannot be reopened.
`Hide`	Hides a `Form`, but does *not* destroy the `Form` or release its resources.
`Show`	Displays a *hidden* `Form`.
Common Event	
`Load`	Occurs before a `Form` is displayed to the user. You'll learn about events and event-handling in the next section.

Fig. 14.4 | Common `Form` properties, methods and an event. (Part 2 of 2.)

14.3 Event Handling

- GUIs are **event driven**.
- When the user interacts with a GUI component, the **event** drives the program to perform a task.
- A method that performs a task in response to an event is called an **event handler**.

14.3 Event Handling (Cont.)

Event Handler Parameters

- Each event handler receives two parameters when it's called:
  - This first—an `object` reference typically named `sender`—is a reference to *the object that generated the event*.
  - The second is a reference to an `EventArgs` object (or an object of an `EventArgs` derived class) which contains additional information about the event.

14.6 GroupBoxes and Panels

- `GroupBoxes` and `Panels` arrange related controls on a GUI.
- All of the controls in a `GroupBox` or `Panel` move together when the `GroupBox` or `Panel` is moved.
- The primary difference is that `GroupBoxes` can display a caption and do not include scrollbars, whereas `Panels` can include scrollbars and do not include a caption.

14.6 GroupBoxes and Panels (Cont.)

- To create a `GroupBox` or `Panel`, drag its icon from the **Toolbox** onto a `Form`.
- Then, drag new controls from the **Toolbox** directly into the `GroupBox` or `Panel`.
- To enable the scrollbars, set the `Panel`’s `AutoScroll` property to `true`.
- If the `Panel` cannot display all of its controls, scrollbars appear (Fig. 14.23).

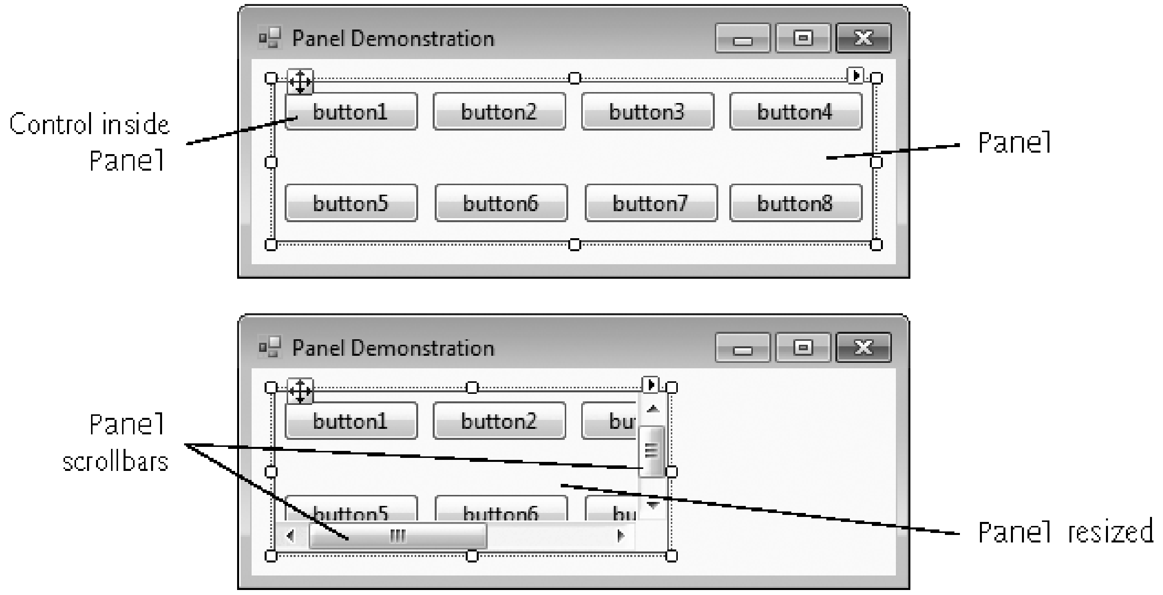


Fig. 14.23 | Creating a Panel with scrollbars.

CheckBox **properties and events**	**Description**
Common Properties	
`Appearance`	By default, this property is set to `Normal`, and the `CheckBox` displays as a traditional checkbox. If it's set to `Button`, the `CheckBox` displays as a `Button` that looks pressed when the `CheckBox` is checked.
`Checked`	Indicates whether the `CheckBox` is *checked* (contains a check mark) or unchecked (blank). This property returns a `bool` value. The default is `false` (*unchecked*).
`CheckState`	Indicates whether the `CheckBox` is *checked* or *unchecked* with a value from the `CheckState` enumeration (`Checked`, `Unchecked` or `Indeterminate`). `Indeterminate` is used when it's unclear whether the state should be `Checked` or `Unchecked`. When `CheckState` is set to `Indeterminate`, the `CheckBox` is usually shaded.
`Text`	Specifies the text displayed to the right of the `CheckBox`.

Fig. 14.25 | CheckBox properties and events. (Part 1 of 2.)

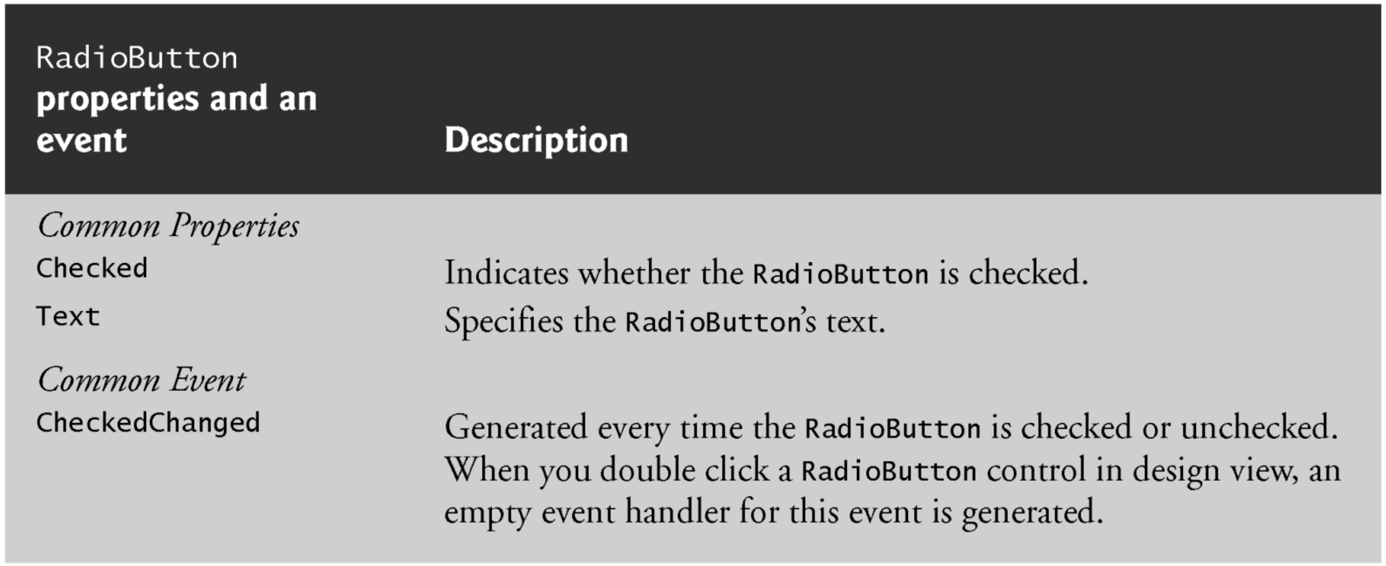
`CheckBox` **properties and events**	**Description**
`ThreeState`	When this property is `true`, the `CheckBox` has three states—*checked*, *unchecked* and *indeterminate*. By default, this property is `false` and the `CheckBox` has only two states—*checked* and *unchecked*.
Common Events	
`CheckedChanged`	Generated when the `Checked` or `CheckState` property changes. This is a `CheckBox`'s default event. When a user double clicks the `CheckBox` control in design view, an empty event handler for this event is generated.
`CheckStateChanged`	Generated when the `Checked` or `CheckState` property changes.

Fig. 14.25 | `CheckBox` properties and events. (Part 2 of 2.)

14.7 CheckBoxes and RadioButtons (Cont.)

`RadioButtons`

- Radio buttons are similar to `CheckBox`es in that they also have two states—**selected** and **not selected**.
- `RadioButton`s normally appear as a **group**, in which only one `RadioButton` can be selected at a time.
- All `RadioButton`s added to a container become part of the same group.



`RadioButton` **properties and an event**	**Description**
Common Properties	
`Checked`	Indicates whether the `RadioButton` is checked.
`Text`	Specifies the `RadioButton`'s text.
Common Event	
`CheckedChanged`	Generated every time the `RadioButton` is checked or unchecked. When you double click a `RadioButton` control in design view, an empty event handler for this event is generated.

Fig. 14.27 | `RadioButton` properties and an event.

14.8 PictureBoxes

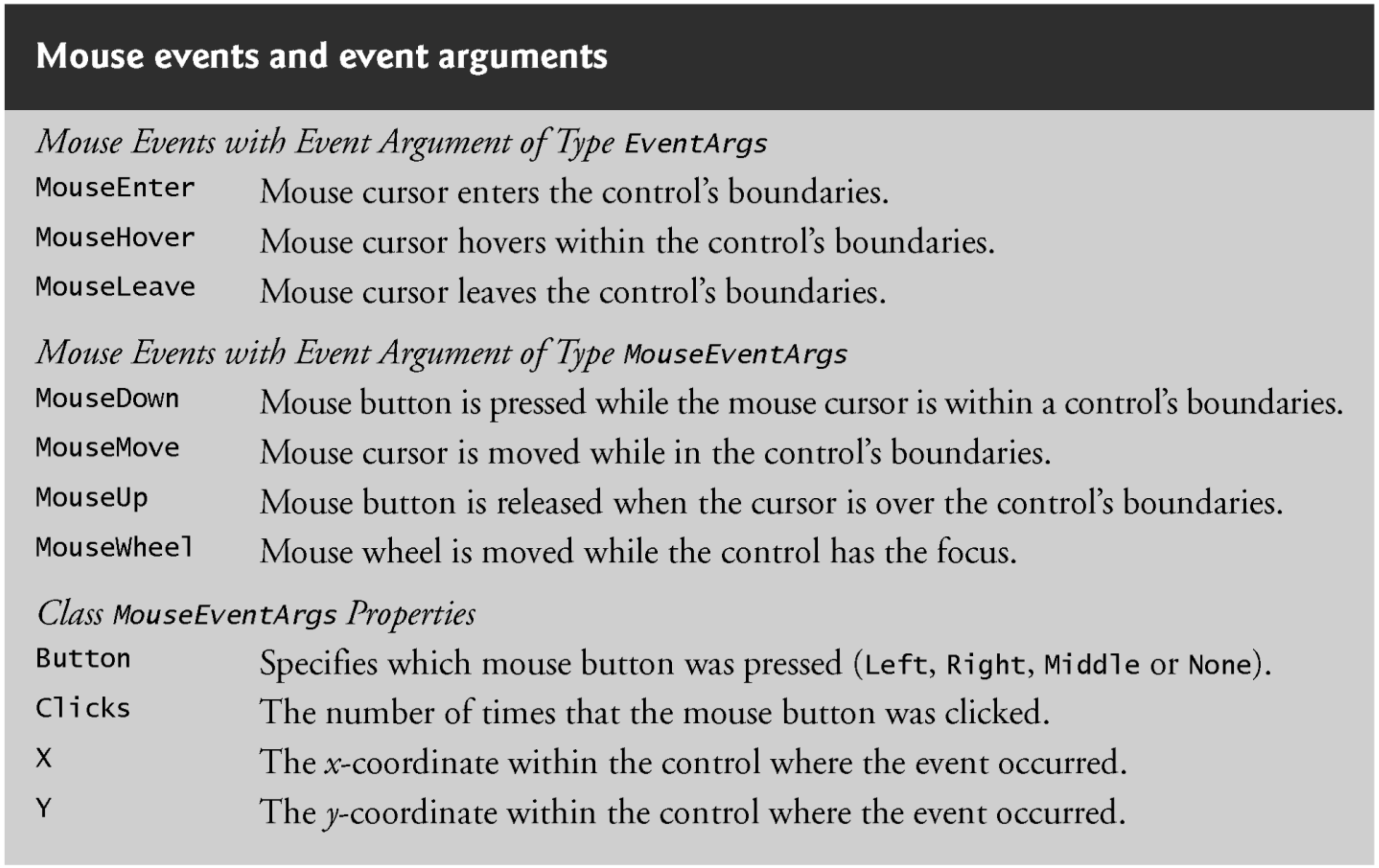
- A PictureBox displays an image.
- Figure 14.29 describes common PictureBox properties and a common event.

PictureBox properties and an event	Description
Common Properties	
`Image`	Sets the image to display in the `PictureBox`.
`SizeMode`	Enumeration that controls image sizing and positioning. Values are `Normal` (default), `StretchImage`, `AutoSize`, `CenterImage`, and `Zoom`. `Normal` places the image in the `PictureBox`'s top-left corner, and `CenterImage` puts the image in the middle. These two options truncate the image if it's too large. `StretchImage` resizes the image to fit in the `PictureBox`. `AutoSize` resizes the `PictureBox` to hold the image. `Zoom` resizes the image to to fit the `PictureBox` but maintains the original aspect ratio.
Common Event	
`Click`	Occurs when the user clicks a control. When you double click this control in the designer, an event handler is generated for this event.

Fig. 14.29 | `PictureBox` properties and an event.

14.11 Mouse-Event Handling

- **Mouse events** are generated when the user interacts with a control via the mouse.
- Information about the event is passed through a `MouseEventArgs` object, and the delegate type is `MouseEventHandler`.
- `MouseEventArgs` *x*- and *y*-coordinates are relative to the control that generated the event.
- Several common mouse events and event arguments are described in Figure 14.37.



Mouse events and event arguments	
Mouse Events with Event Argument of Type `EventArgs`	
`MouseEnter`	Mouse cursor enters the control's boundaries.
`MouseHover`	Mouse cursor hovers within the control's boundaries.
`MouseLeave`	Mouse cursor leaves the control's boundaries.
Mouse Events with Event Argument of Type `MouseEventArgs`	
`MouseDown`	Mouse button is pressed while the mouse cursor is within a control's boundaries.
`MouseMove`	Mouse cursor is moved while in the control's boundaries.
`MouseUp`	Mouse button is released when the cursor is over the control's boundaries.
`MouseWheel`	Mouse wheel is moved while the control has the focus.
Class `MouseEventArgs` *Properties*	
`Button`	Specifies which mouse button was pressed (`Left`, `Right`, `Middle` or `None`).
`Clicks`	The number of times that the mouse button was clicked.
`X`	The *x*-coordinate within the control where the event occurred.
`Y`	The *y*-coordinate within the control where the event occurred.

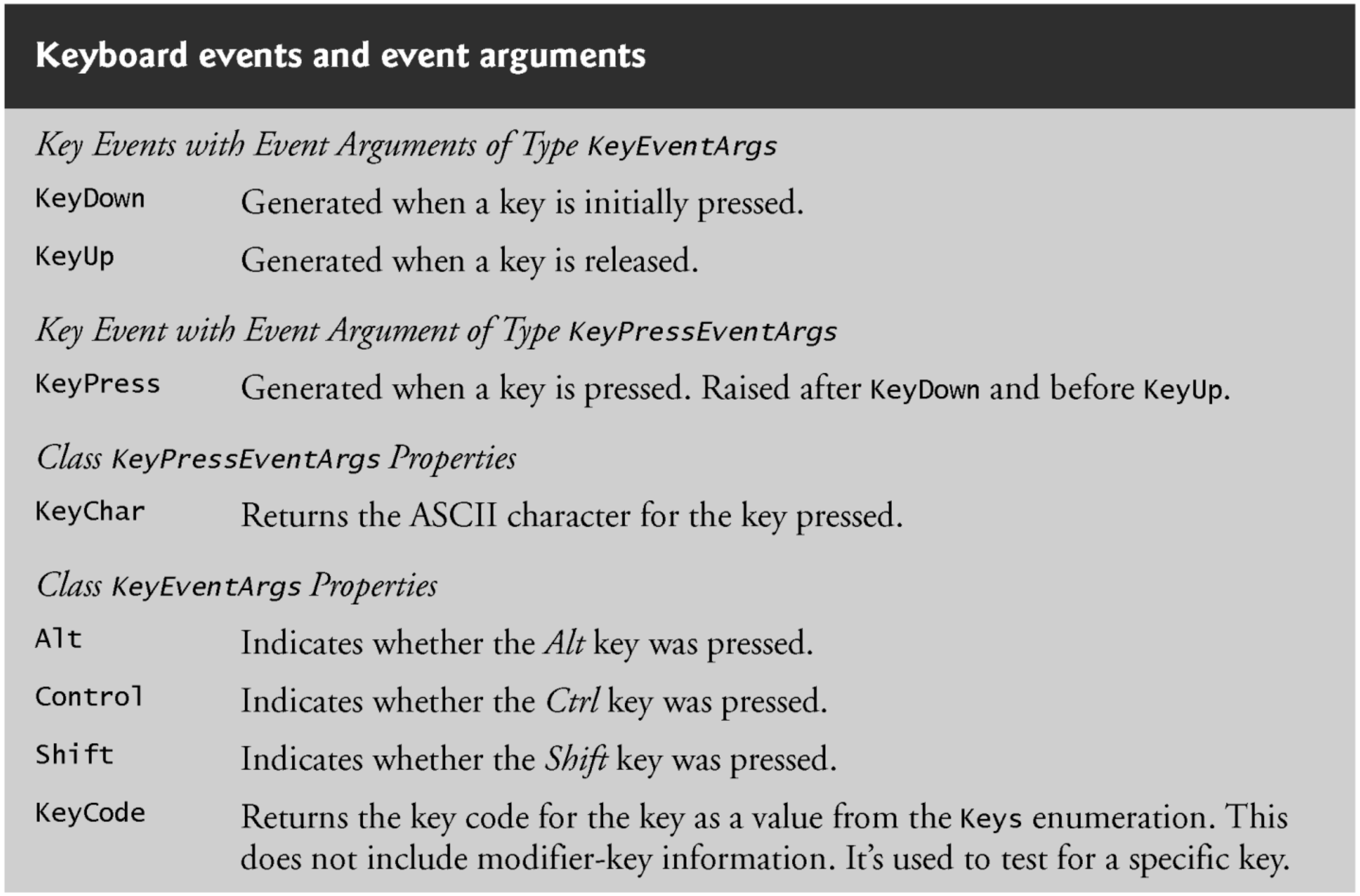
Fig. 14.37 | Mouse events and event arguments.

14.11 Mouse-Event Handling (Cont.)

- Recall from Chapter 13 that the `using` statement automatically calls `Dispose` on the object that was created in the parentheses following keyword `using`.
- This is important because `Graphics` objects are a limited resource.
- Calling `Dispose` on a `Graphics` object ensures that its resources are returned to the system for reuse.

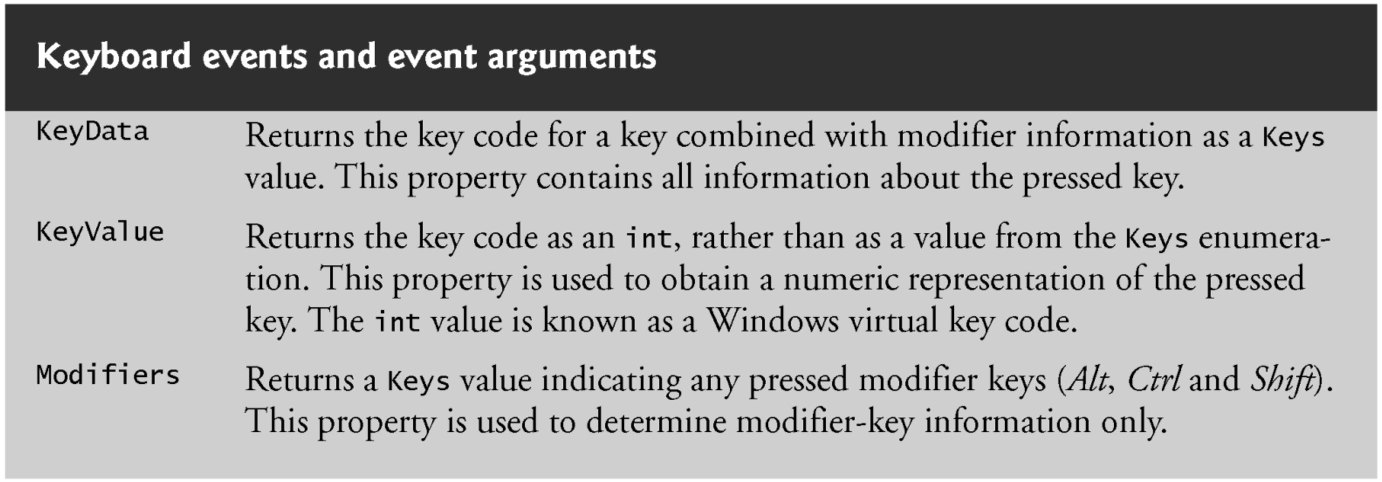
14.12 Keyboard-Event Handling

- There are three key events:
  - The `KeyPress` event occurs when the user presses a key that represents an ASCII character.
  - The `KeyPress` event does not indicate whether **modifier keys** (e.g., *Shift*, *Alt* and *Ctrl*) were pressed; if this information is important, the `KeyUp` or `KeyDown` events can be used.



Keyboard events and event arguments	
Key Events with Event Arguments of Type `KeyEventArgs`	
`KeyDown`	Generated when a key is initially pressed.
`KeyUp`	Generated when a key is released.
Key Event with Event Argument of Type `KeyPressEventArgs`	
`KeyPress`	Generated when a key is pressed. Raised after `KeyDown` and before `KeyUp`.
Class `KeyPressEventArgs` *Properties*	
`KeyChar`	Returns the ASCII character for the key pressed.
Class `KeyEventArgs` *Properties*	
`Alt`	Indicates whether the *Alt* key was pressed.
`Control`	Indicates whether the *Ctrl* key was pressed.
`Shift`	Indicates whether the *Shift* key was pressed.
`KeyCode`	Returns the key code for the key as a value from the `Keys` enumeration. This does not include modifier-key information. It's used to test for a specific key.

Fig. 14.39 | Keyboard events and event arguments. (Part 1 of 2.)



Keyboard events and event arguments	
`KeyData`	Returns the key code for a key combined with modifier information as a `Keys` value. This property contains all information about the pressed key.
`KeyValue`	Returns the key code as an `int`, rather than as a value from the `Keys` enumeration. This property is used to obtain a numeric representation of the pressed key. The `int` value is known as a Windows virtual key code.
`Modifiers`	Returns a `Keys` value indicating any pressed modifier keys (*Alt*, *Ctrl* and *Shift*). This property is used to determine modifier-key information only.

Fig. 14.39 | Keyboard events and event arguments. (Part 2 of 2.)

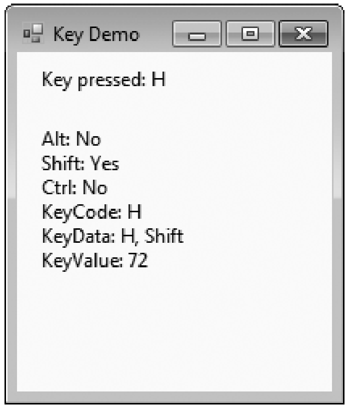
```
// Fig. 14.40: KeyDemo.cs
// Displaying information about the key the user pressed.
using System;
using System.Windows.Forms;

namespace KeyDemo
{
   // Form to display key information when key is pressed
   public partial class KeyDemo : Form
   {
      // default constructor
      public KeyDemo()
      {
         InitializeComponent();
      } // end constructor

      // display the character pressed using KeyChar
      private void KeyDemo_KeyPress(
         object sender, KeyPressEventArgs e )
      {
         charLabel.Text = "Key pressed: " + e.KeyChar;
      } // end method KeyDemo_KeyPress

```

Fig. 14.40 | Demonstrating keyboard events. (Part 1 of 3.)

```
      // display modifier keys, key code, key data and key value
      private void KeyDemo_KeyDown( object sender, KeyEventArgs e )
      {
         keyInfoLabel.Text =
            "Alt: " + ( e.Alt ? "Yes" : "No" ) + '\n' +
            "Shift: " + ( e.Shift ? "Yes" : "No" ) + '\n' +
            "Ctrl: " + ( e.Control ? "Yes" : "No" ) + '\n' +
            "KeyCode: " + e.KeyCode + '\n' +
            "KeyData: " + e.KeyData + '\n' +
            "KeyValue: " + e.KeyValue;
      } // end method KeyDemo_KeyDown

      // clear Labels when key released
      private void KeyDemo_KeyUp( object sender, KeyEventArgs e )
      {
         charLabel.Text = "";
         keyInfoLabel.Text = "";
      } // end method KeyDemo_KeyUp
   } // end class KeyDemo
} // end namespace KeyDemo
```

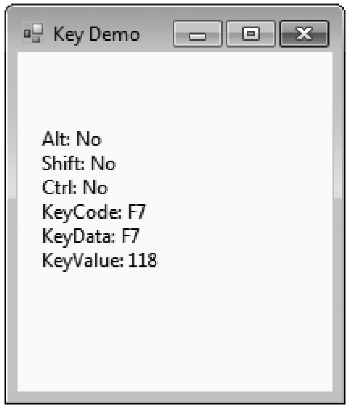
Fig. 14.40 | Demonstrating keyboard events. (Part 2 of 3.)

a) *H* pressed



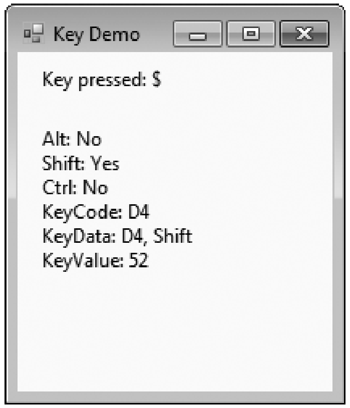
```
Key pressed: H

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

b) *F7* pressed



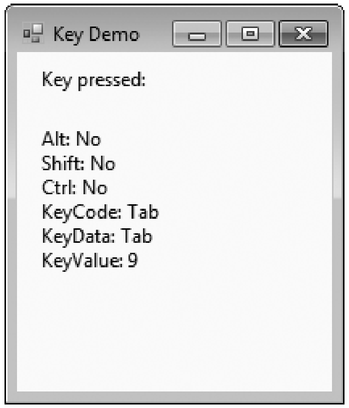
```
Alt: No
Shift: No
Ctrl: No
KeyCode: F7
KeyData: F7
KeyValue: 118
```

c) *$* pressed



```
Key pressed: $

Alt: No
Shift: Yes
Ctrl: No
KeyCode: D4
KeyData: D4, Shift
KeyValue: 52
```

d) *Tab* pressed



```
Key pressed:

Alt: No
Shift: No
Ctrl: No
KeyCode: Tab
KeyData: Tab
KeyValue: 9
```

Fig. 14.40 | Demonstrating keyboard events. (Part 3 of 3.)