15

Graphical User Interfaces with Windows Forms: Part 2

15.2 Menus

- **Menus** provide groups of related commands (Fig. 15.1).
- Menus organize commands without “cluttering” the GUI.

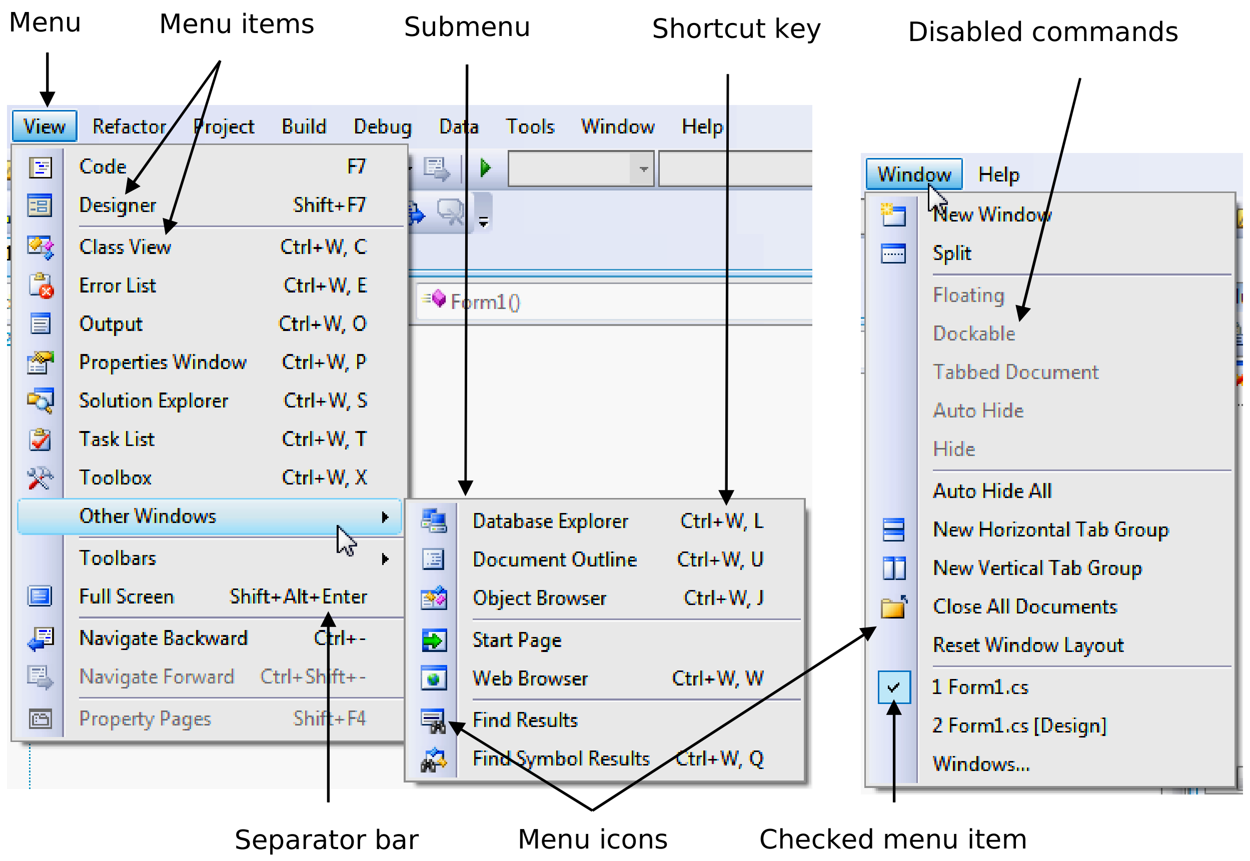


Fig. 15.1 | Menus, submenus and menu items.

15.2 Menus (Cont.)

- To create a menu, open the **Toolbox** and drag a `MenuStrip` control onto the `Form`.
- To add menu items to the menu, click the **Type Here** `TextBox` (Fig. 15.2) and type the menu item's name.

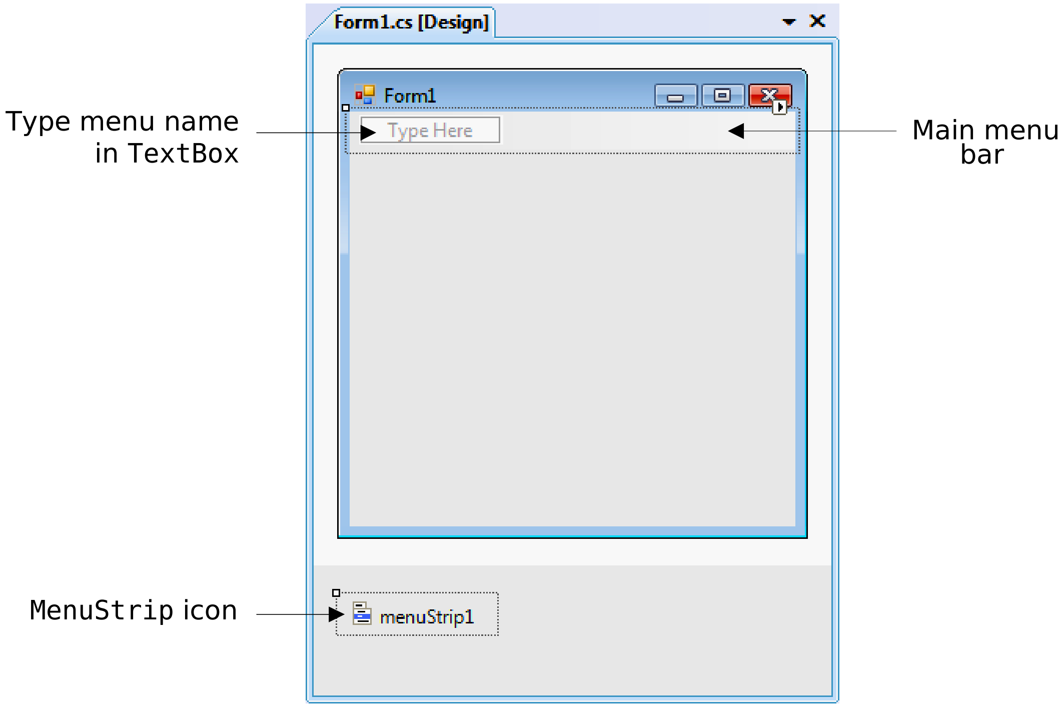
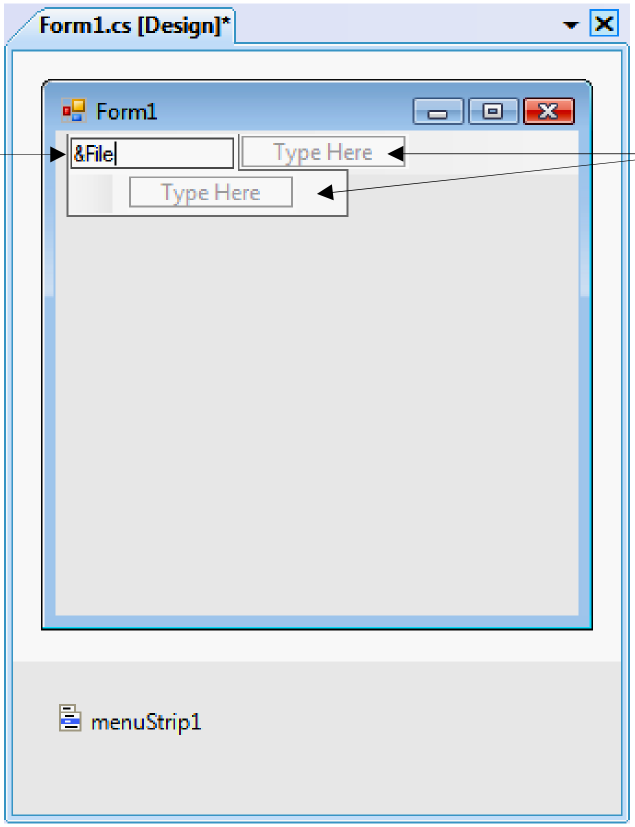


Fig. 15.2 | Editing menus in Visual Studio.

15.2 Menus (Cont.)

- After you press the *Enter* key, the menu item is added.
- More **Type Here** `TextBox`es allow you to add more items (Fig. 15.3).

Place & character before a letter to underline it in the menu, so the character can be used as an access shortcut



`TextBox`es for adding items to the menu

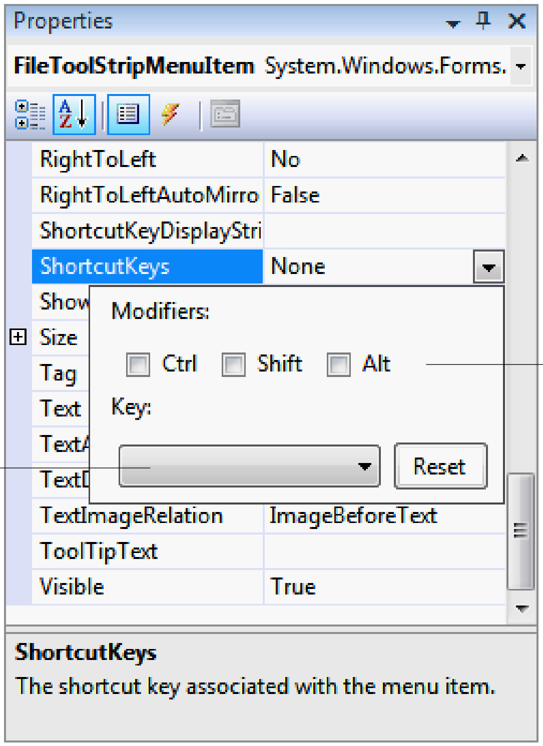
Fig. 15.3 | Adding `ToolStripMenuItem`s to a `MenuStrip`.

15.2 Menus (Cont.)

- Menus can have *Alt* key shortcuts which are accessed by pressing *Alt* and the underlined letter.
- To make the **File** menu item have a key shortcut, type `&File`.
- The letter **F** is underlined to indicate that it is a shortcut.

15.2 Menus (Cont.)

- Menu items can have shortcut keys as well (*Ctrl*, *Shift*, *Alt, F1*, *F2*, letter keys, and so on).
- To add other shortcut keys, set the **ShortcutKeys** property (Fig. 15.4).


Properties
FileToolStripMenuItem System.Windows.Forms.
Modifiers:
Ctrl
Shift
Alt
Key:
Reset
Setting modifier keys
Select key (modifier and key combination specifies the shortcut key for the menu item)

RightToLeft	No
RightToLeftAutoMirro	False
ShortcutKeyDisplayStri	
ShortcutKeys	None
Show	
Size	
Tag	
Text	
TextA	
TextD	
TextImageRelation	ImageBeforeText
ToolTipText	
Visible	True

ShortcutKeys
The shortcut key associated with the menu item.

Fig. 15.4 | Setting a menu item's shortcut keys.

15.2 Menus (Cont.)

Look-and-Feel Observation 15.1

Buttons can have access shortcuts. Place the & symbol immediately before the desired character in the Button's text. To press the button, the user presses *Alt* and the underlined character.

- You can remove a menu item by selecting it with the mouse and pressing the *Delete* key.
- Menu items can be grouped by **separator bars**, which are inserted by right clicking and selecting **Insert Separator** or by typing "-" for the text of a menu item.

15.2 Menus (Cont.)

- Visual Studio allows you to add `TextBoxes` and `ComboBoxes` as menu items.
- Before you enter text for a menu item, you are provided with a drop-down list.
  - Clicking the down arrow allows you to select the type of item to add (Fig. 15.5).

15.2 Menus (Cont.)

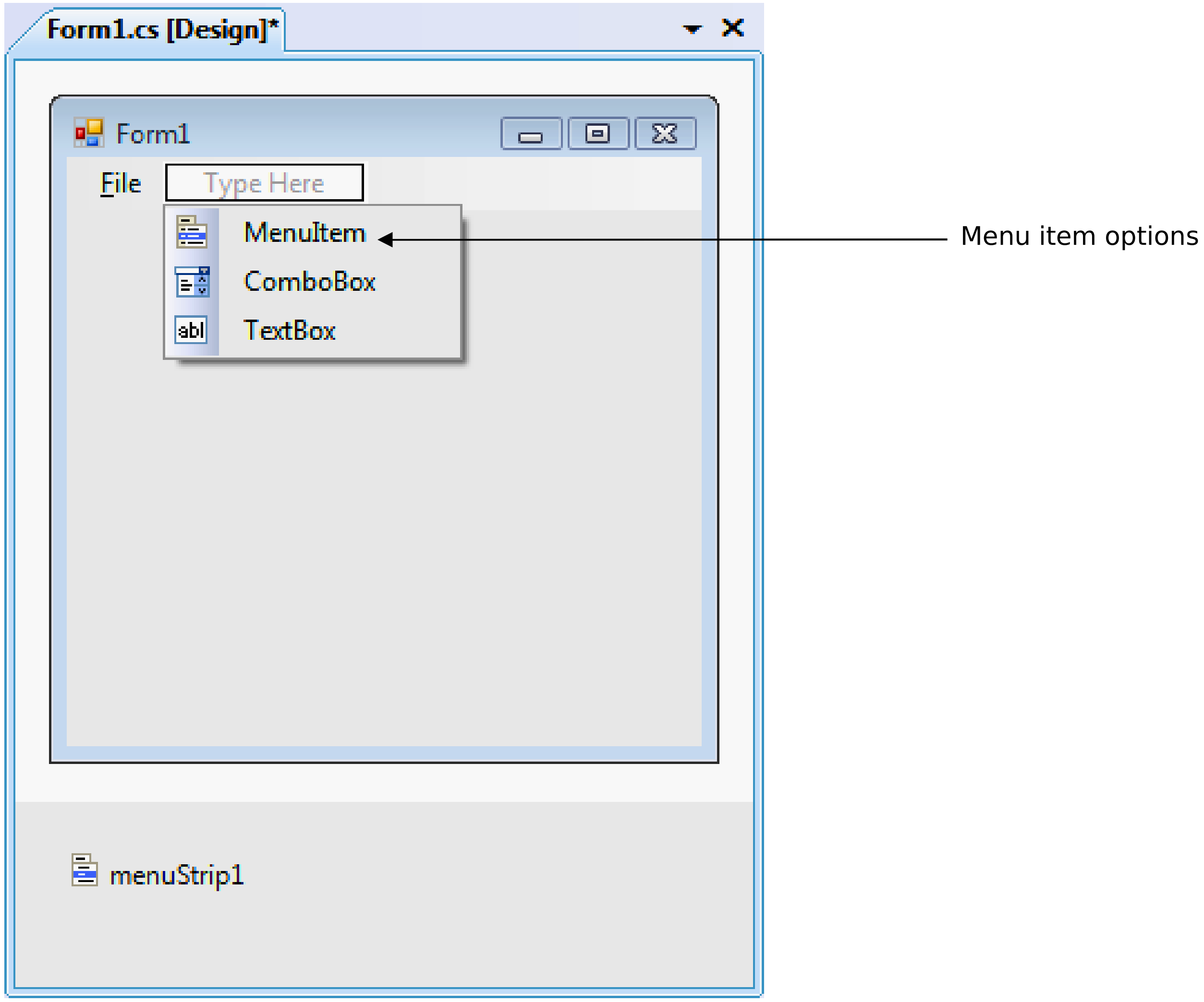


Fig. 15.5 | Menu-item options.

15.2 Menus (Cont.)

MenuStrip and ToolStripMenuItem properties and an event	Description
MenuStrip Properties	
MenuItems	Contains the top-level menu items for this MenuStrip.
HasChildren	Indicates whether MenuStrip has any child menu items.
RightToLeft	Causes text to display from right to left.
ToolStripMenuItem Properties	
Checked	Indicates whether a menu item is checked.
CheckOnClick	Indicates that a menu item should appear checked or unchecked as it is clicked.

Fig. 15.6 | MenuStrip and ToolStripMenuItem properties and an event. (Part 1 of 2.)

15.2 Menus (Cont.)

`MenuStrip` and `ToolStripMenuItem` properties and an event	Description
`Index`	Specifies an item's position in its parent menu.
`MenuItems`	Lists the submenu items for a particular menu item.
`ShortcutKeyDisplayString`	Specifies text that should appear beside a menu item for a shortcut key.
`ShortcutKeys`	Specifies the shortcut key for the menu item.
`ShowShortcutKeys`	Indicates whether a shortcut key is shown beside menu item text.
`Text`	Specifies the menu item's text.
Common `ToolStripMenuItem` *Event*	
`Click`	Generated when an item is clicked or a shortcut key is used.

Fig. 15.6 | `MenuStrip` and `ToolStripMenuItem` properties and an event. (Part 2 of 2.)

```
// Fig15.7: MenuTestForm.cs
// Using Menus to change font colors and styles.
using System;
using System.Drawing;
using System.Windows.Forms;

namespace MenuTest
{
   // our Form contains a Menu that changes the font color
   // and style of the text displayed in Label
   public partial class MenuTestForm : Form
   {
      // constructor
      public MenuTestForm()
      {
         InitializeComponent();
      } // end constructor
```

Fig. 15.7 | Menus for changing text font and color. (Part 1 of 9.)

Outline

```

      // display MessageBox when About ToolStripMenuItem is selected
   private void aboutToolStripMenuItem_Click(
      object sender, EventArgs e )
   {
      MessageBox.Show( "This is an example\nof using menus", "About",
         MessageBoxButtons.OK, MessageBoxIcon.Information );
   } // end method aboutToolStripMenuItem_Click

      // exit program when Exit ToolStripMenuItem is selected
   private void exitToolStripMenuItem_Click(
      object sender, EventArgs e )
   {
      Application.Exit();
   } // end method exitToolStripMenuItem_Click

      // reset checkmarks for Color ToolStripMenuItems
   private void ClearColor()
   {
         // clear all checkmarks
      blackToolStripMenuItem.Checked = false;
      blueToolStripMenuItem.Checked = false;
      redToolStripMenuItem.Checked = false;
      greenToolStripMenuItem.Checked = false;
   } // end method ClearColor
```

MenuTestForm.cs

(3 of 10)

The **About** menu item displays a `MessageBox` when clicked.

The **Exit** menu item closes the application through method **`Exit`** of class **`Application`**.

Fig. 15.7 | Menus for changing text font and color. (Part 2 of 9.)

Outline

MenuTestForm.cs

(4 of 10)

```
    // update Menu state and color display black
    private void blackToolStripMenuItem_Click(
       object sender, EventArgs e )
    {
       // reset checkmarks for Color ToolStripMenuItems
       ClearColor();

       // set Color to Black
       displayLabel.ForeColor = Color.Black;
       blackToolStripMenuItem.Checked = true;
    } // end method blackToolStripMenuItem_Click

    // update Menu state and color display blue
    private void blueToolStripMenuItem_Click(
       object sender, EventArgs e )
    {
       // reset checkmarks for Color ToolStripMenuItems
       ClearColor();

```

Each **Color** menu item calls `ClearColor` before setting its `Checked` property (making the checks mutually exclusive).

Fig. 15.7 | Menus for changing text font and color. (Part 3 of 9.)

Outline

```
         // set Color to Blue
         displayLabel.ForeColor = Color.Blue;
      blueToolStripMenuItem.Checked = true;
   } // end method blueToolStripMenuItem_Click

     // update Menu state and color display red
   private void redToolStripMenuItem_Click(
      object sender, EventArgs e )
   {
         // reset checkmarks for Color ToolStripMenuItems
      ClearColor();

         // set Color to Red
      displayLabel.ForeColor = Color.Red;
      redToolStripMenuItem.Checked = true;
   } // end method redToolStripMenuItem_Click

    // update Menu state and color display green
   private void greenToolStripMenuItem_Click(
      object sender, EventArgs e )
   {
         // reset checkmarks for Color ToolStripMenuItems
      ClearColor();
```

MenuTestForm.cs

(5 of 10)

Each **Color** menu item calls `ClearColor` before setting its `Checked` property (making the checks mutually exclusive).

Fig. 15.7 | Menus for changing text font and color. (Part 4 of 9.)

Outline

`MenuTestForm.cs`

(6 of 10)

```
         // set Color to Green
         displayLabel.ForeColor = Color.Green;
         greenToolStripMenuItem.Checked = true;
      } // end method greenToolStripMenuItem_Click

      // reset checkmarks for Font ToolStripMenuItems
      private void ClearFont()
      {
         // clear all checkmarks
         timesToolStripMenuItem.Checked = false;
         courierToolStripMenuItem.Checked = false;
         comicToolStripMenuItem.Checked = false;
      } // end method ClearFont

      // update Menu state and set Font to Times New Roman
      private void timesToolStripMenuItem_Click(
         object sender, EventArgs e )
      {
         // reset checkmarks for Font ToolStripMenuItems
         ClearFont();
```

Each **Color** menu item calls `ClearColor` before setting its `Checked` property (making the checks mutually exclusive).

Each **Font** menu item calls `ClearFont` before setting its `Checked` property (making the checks mutually exclusive).

Fig. 15.7 | Menus for changing text font and color. (Part 5 of 9.)

Outline

```

      // set Times New Roman font
      timesToolStripMenuItem.Checked = true;
      displayLabel.Font = new Font( "Times New Roman", 14,
         displayLabel.Font.Style );
   } // end method timesToolStripMenuItem_Click

   // update Menu state and set Font to Courier
   private void courierToolStripMenuItem_Click(
      object sender, EventArgs e )
   {
      // reset checkmarks for Font ToolStripMenuItems
      ClearFont();

      // set Courier font
      courierToolStripMenuItem.Checked = true;
      displayLabel.Font = new Font( "Courier", 14,
         displayLabel.Font.Style );
   } // end method courierToolStripMenuItem_Click

   // update Menu state and set Font to Comic Sans MS
   private void comicToolStripMenuItem_Click(
     object sender, EventArgs e )
   {
     // reset checkmarks for Font ToolStripMenuItems
```

MenuTestForm.cs

(7 of 10)

Each **Font** menu item calls `ClearFont` before setting its `Checked` property (making the checks mutually exclusive).

Fig. 15.7 | Menus for changing text font and color. (Part 6 of 9.)

Outline

MenuTestForm.cs

(8 of 10)

```
            ClearFont();

            // set Comic Sans font
        comicToolStripMenuItem.Checked = true;
        displayLabel.Font = new Font( "Comic Sans MS", 14,
           displayLabel.Font.Style );
     } // end method comicToolStripMenuItem_Click

      // toggle checkmark and toggle bold style
     private void boldToolStripMenuItem_Click(
        object sender, EventArgs e )
     {
            // toggle checkmark
        boldToolStripMenuItem.Checked = !boldToolStripMenuItem.Checked;

            // use Xor to toggle italic, keep all other styles
        displayLabel.Font = new Font( displayLabel.Font
           displayLabel.Font.Style ^ FontStyle.Bold );
     } // end method boldToolStripMenuItem_Click

```

Each **Font** menu item calls `ClearFont` before setting its `Checked` property (making the checks mutually exclusive).

The **Bold** and **Italic** menu items use the bitwise logical exclusive OR operator to combine font styles.

Fig. 15.7 | Menus for changing text font and color. (Part 7 of 9.)

Outline

MenuTestForm.cs

(9 of 10)

```
      // toggle checkmark and toggle italic style
      private void italicToolStripMenuItem_Click(
         object sender, EventArgs e )
      {
         // toggle checkmark
         italicToolStripMenuItem.Checked =
            !italicToolStripMenuItem.Checked;

         // use Xor to toggle italic, keep all other styles
         displayLabel.Font = new Font( displayLabel.Font,
            displayLabel.Font.Style ^ FontStyle.Italic );
      } // end method italicToolStripMenuItem_Click
   } // end class MenuTestForm
} // end namespace MenuTest
```

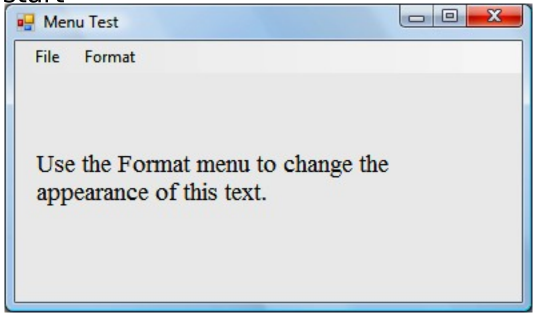
The **Bold** and **Italic** menu items use the bitwise logical exclusive OR operator to combine font styles.

Fig. 15.7 | Menus for changing text font and color. (Part 8 of 9.)

MenuTestForm.cs

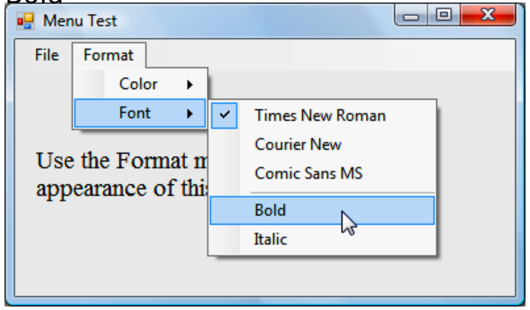
(10 of 10)

a) Application at start



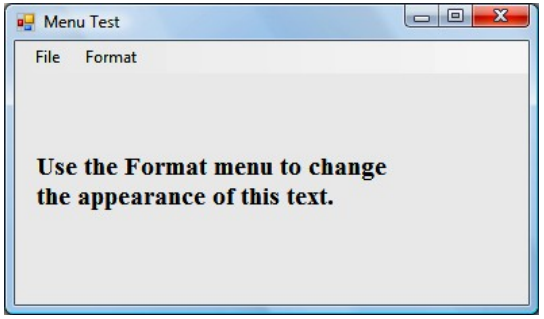
Use the Format menu to change the appearance of this text.

b) Changing font to Bold



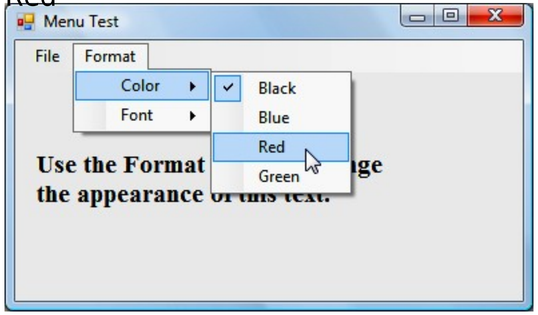
Use the Format m appearance of thi

c) Application with bold font



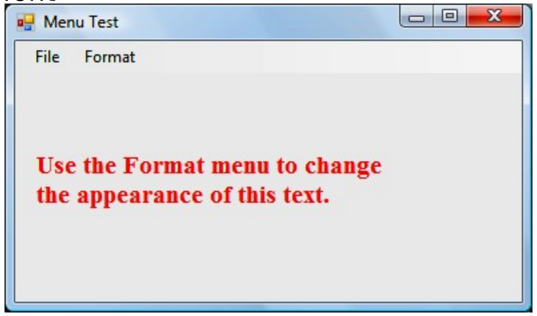
Use the Format menu to change the appearance of this text.

d) Changing font to Red



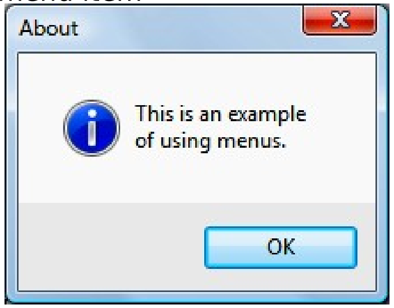
Use the Format ige the appearance

e) Application with Red font



Use the Format menu to change the appearance of this text.

f) Message from About menu item



This is an example of using menus.

Fig. 15.7 | Menus for changing text font and color. (Part 9 of 9.)

15.3 MonthCalendar Control

- The **MonthCalendar** control (Fig. 15.8) displays a monthly calendar on the Form.
- Multiple dates can be selected by clicking dates on the calendar while holding down the *Shift* key.

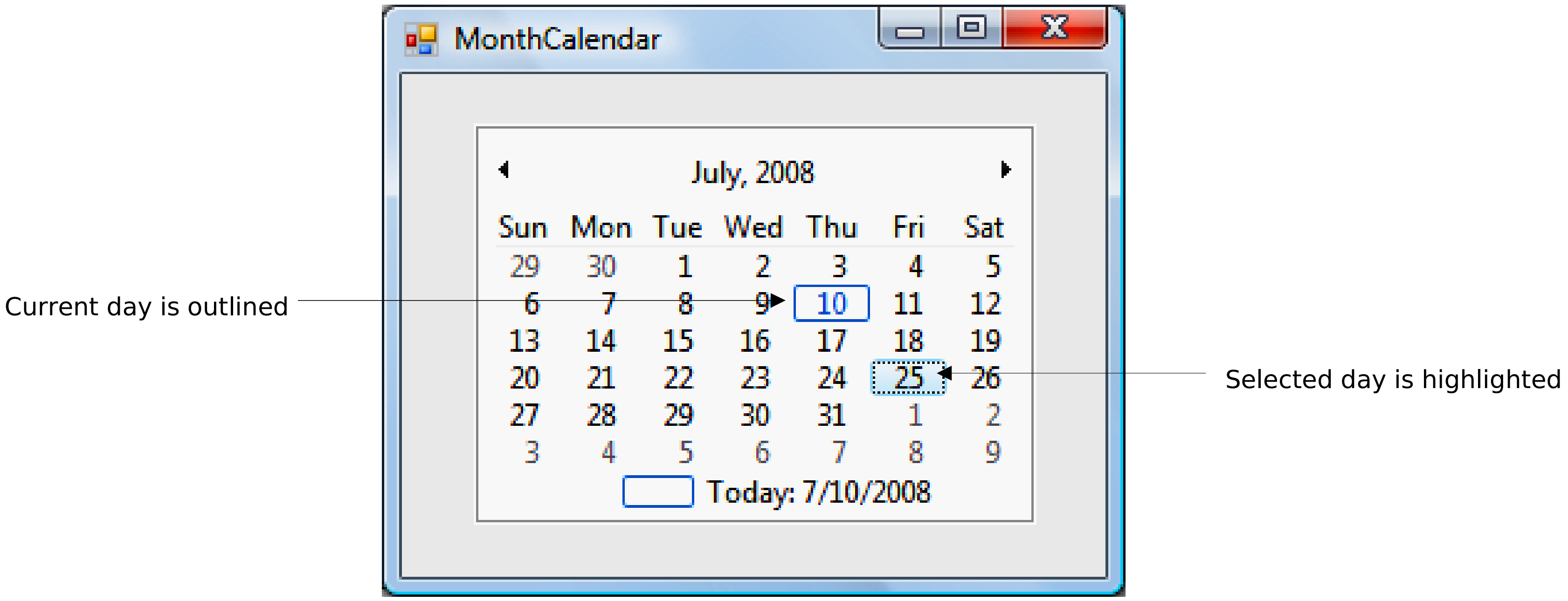


Fig. 15.8 | MonthCalendar control.

15.3 MonthCalendar Control

MonthCalendar properties and an event	Description
MonthCalendar Properties	
FirstDayOfWeek	Sets which day of the week is the first displayed for each week in the calendar.
MaxDate	The last date that can be selected.
MaxSelectionCount	The maximum number of dates that can be selected at once.
MinDate	The first date that can be selected.
MonthlyBoldedDates	An array of dates that will displayed in bold in the calendar.
SelectionEnd	The last of the dates selected by the user.
SelectionRange	The dates selected by the user.
SelectionStart	The first of the dates selected by the user.
Common MonthCalendar Event	
DateChanged	Generated when a date is selected in the calendar.

Fig. 15.9 | MonthCalendar properties and an event.

15.4 DateTimePicker Control

- The **DateTimePicker** control displays a calendar when a down arrow is selected.
- The DateTimePicker can be used to retrieve date and time information from the user.

15.4 DateTimePicker Control

DateTimePicker properties and an event	**Description**
DateTimePicker Properties	
CalendarForeColor	Sets the text color for the calendar.
CalendarMonth Background	Sets the calendar's background color.
CustomFormat	Sets the custom format string for the user's options.
Date	The date.
Format	Sets the format of the date and/or time used for the user's options.
MaxDate	The maximum date and time that can be selected.

Fig. 15.10 | DateTimePicker properties and an event. (Part 1 of 2.)

15.4 DateTimePicker Control

DateTimePicker properties and an event	Description
MinDate	The minimum date and time that can be selected.
ShowCheckBox	Indicates if a CheckBox should be displayed to the left.
ShowUpDown	Indicates whether the control displays up and down Buttons.
TimeOfDay	The time.
Value	The data selected by the user.
Common DateTimePicker *Event*	
ValueChanged	Generated when the Value property changes.

Fig. 15.10 | DateTimePicker properties and an event. (Part 2 of 2.)

- Figure 15.11 demonstrates using the `DateTimePicker` control to select an item's drop-off time.
- The `DateTimePicker` has its `Format` property set to `Long`.
- In this application, the arrival date is always two days after drop-off, or three days if a Sunday is reached.

`DateTimePickerForm.cs`

(1 of 5)

Outline

DateTimePickerForm.cs

(2 of 5)

```
// Fig15.11: DateTimePickerForm.cs
// Using a DateTimePicker to select a drop-off time.
using System;
using System.Windows.Forms;

namespace DateTimePickerTest
{
   // Form lets user select a drop-off date using a DateTimePicker
   // and displays an estimated delivery date
   public partial class DateTimePickerForm : Form
  {
      // constructor
     public DateTimePickerForm()
     {
       InitializeComponent();
     } // end constructor
```

Fig. 15.11 | Demonstrating `DateTimePicker`. (Part 1 of 4.)

Outline

DateTimePickerForm.cs

(3 of 5)

```

      private void dateTimePickerDropOff_ValueChanged(
         object sender, EventArgs e )
      {
         DateTime dropOffDate = dateTimePickerDropOff.Value;

         //  add extra time when items are dropped off  around Sunday
         if ( dropOffDate.DayOfWeek == DayOfWeek.Friday ||
            dropOffDate.DayOfWeek == DayOfWeek.Saturday ||
            dropOffDate.DayOfWeek == DayOfWeek.Sunday  )

            //estimate three days for delivery
            outputLabel.Text =
               dropOffDate.AddDays( 3 ).ToLongDateString();
         else
```

Retrieving the selected date from the **Value** property.

The DateTime structure's **DayOfWeek** property determines the day of the week on which the selected date falls.

Using DateTime's **AddDays** method to increase the date by two days or three days.

Fig. 15.11 | Demonstrating DateTimePicker. (Part 2 of 4.)

Outline

DateTimePicker Form.cs

(4 of 5)

```
            // otherwise estimate only two days for delivery
            outputLabel.Text =
               dropOffDate.AddDays( 2 ).ToLongDateString();
      } // end method dateTimePickerDropOff_ValueChanged

      private void DateTimePickerForm_Load( object sender, EventArgs e )
      {
         // user cannot select days before today
         dateTimePickerDropOff.MinDate = DateTime.Today;

         // user can only select days of this year
         dateTimePickerDropOff.MaxDate = DateTime.Today.AddYears( 1 );
      } // end method DateTimePickerForm_Load
   } // end class DateTimePickerForm
```

Setting the `MinDate` and `MaxDate` properties to keep drop-off sometime in the next year.

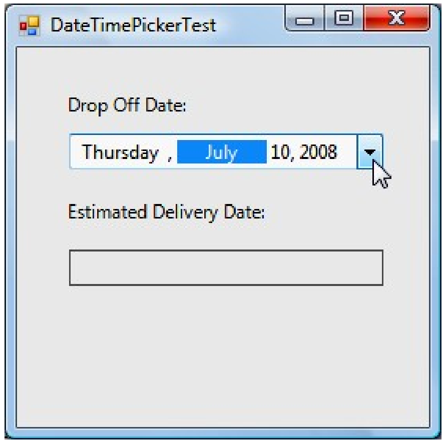
Fig. 15.11 | Demonstrating `DateTimePicker`. (Part 3 of 4.)

Outline

`DateTimePicker Form.cs`

(5 of 5)

a) Clicking the down arrow

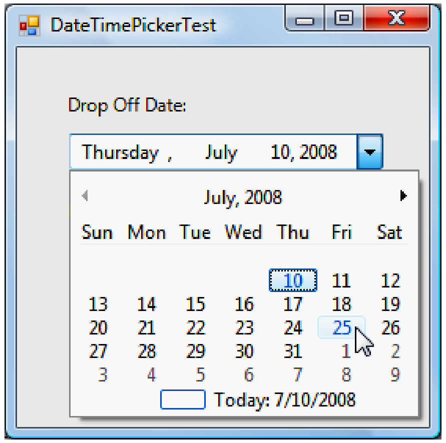


Drop Off Date:

Thursday , July 10, 2008

Estimated Delivery Date:

b) Selecting a day from the calendar

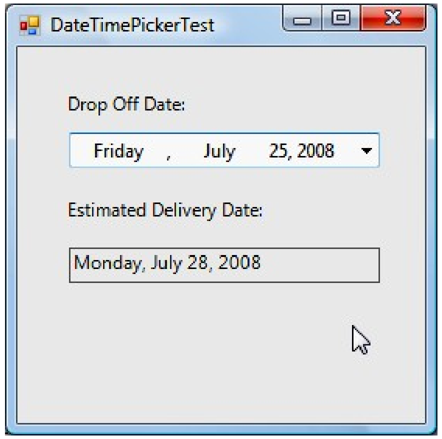


Drop Off Date:

Thursday , July 10, 2008

```
             July, 2008
Sun  Mon  Tue  Wed  Thu  Fri  Sat
                     10   11   12
 13   14   15   16   17   18   19
 20   21   22   23   24   25   26
 27   28   29   30   31    1    2
  3    4    5    6    7    8    9
       Today: 7/10/2008
```

c) The Label updates



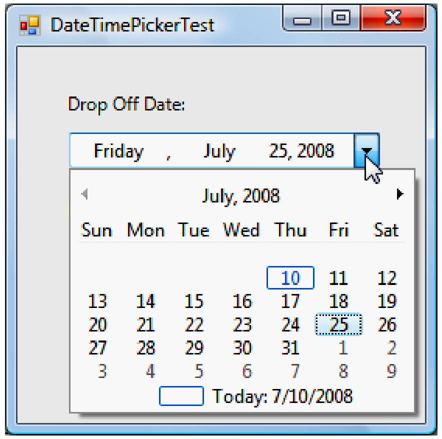
Drop Off Date:

Friday , July 25, 2008

Estimated Delivery Date:

Monday, July 28, 2008

d) Selecting another day



Drop Off Date:

Friday , July 25, 2008

```
             July, 2008
Sun  Mon  Tue  Wed  Thu  Fri  Sat
                     10   11   12
 13   14   15   16   17   18   19
 20   21   22   23   24   25   26
 27   28   29   30   31    1    2
  3    4    5    6    7    8    9
       Today: 7/10/2008
```

Fig. 15.11 | Demonstrating `DateTimePicker`. (Part 4 of 4.)

15.5 LinkLabel Control

- The **LinkLabel** control displays links to other resources, such as files or web pages (Fig. 15.12).

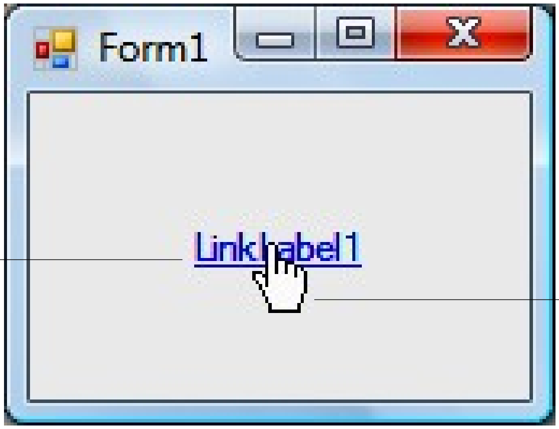


Fig. 15.12 | LinkLabel control in running program.

Look-and-Feel Observation 15.3

A LinkLabel is the preferred control for indicating that the user can click a link to jump to a resource such as a web page, though other controls can perform similar tasks.

15.5 LinkLabel Control (Cont.)

- When clicked, the `LinkLabel` generates a **`LinkClicked`** event (Fig. 15.13).
- Class `LinkLabel` is derived from class `Label` and therefore inherits all of class `Label`'s functionality.

LinkLabel properties and an event	**Description**
Common Properties	
`ActiveLinkColor`	Specifies the color of the active link when clicked.
`LinkArea`	Specifies which portion of text in the `LinkLabel` is part of the link.
`LinkBehavior`	Specifies the link's behavior.
`LinkColor`	Specifies the color of all links before they have been visited.

Fig. 15.13 | `LinkLabel` properties and an event. (Part 1 of 2.)

15.5 LinkLabel Control (Cont.)

LinkLabel properties and an event	Description
LinkVisited	If true, the link appears as though it has been visited.
Text	Specifies the control's text.
UseMnemonic	Makes the & character in the Text property act as a shortcut.
VisitedLinkColor	Specifies the color of visited links.
Common Event	*(Event arguments* LinkLabelLinkClickedEventArgs*)*
LinkClicked	Generated when the link is clicked.

Fig. 15.13 | LinkLabel properties and an event. (Part 2 of 2.)

<u>Outline</u>

LinkLabelTestForm.cs

(1 of 6)

- Class LinkLabelTestForm (Fig. 15.14) uses three LinkLabels.
- Method **Start** of class **Process** allows you to execute other programs, or load documents or web sites from an application.

```
// Fig15.14: LinkLabelTestForm.cs
// Using LinkLabels to create hyperlinks.
using System;
using System.Windows.Forms;

namespace LinkLabelTest
{
   // Form using LinkLabels to browse the C:\ drive,
   // load a web page and run Notepad
   public partial class LinkLabelTestForm : Form
  {
      // constructor
     public LinkLabelTestForm()
     {
       InitializeComponent();
     } // end constructor
```

Fig. 15.14 | LinkLabels used to link to a drive, a web page and an application. (Part 1 of 6.)

Outline

LinkLabelTest Form.cs

(2 of 6)

```

      // browse C:\ drive
    private void cDriveLinkLabel_LinkClicked( object sender,
      LinkLabelLinkClickedEventArgs e )
    {
      // change LinkColor after it has been clicked
      driveLinkLabel.LinkVisited = true;

      System.Diagnostics.Process.Start( @"C:\" );
    } // end method driveLinkLabel_LinkClicked

      // load www.deitel.com in web browser
    private void deitelLinkLabel_LinkClicked( object sender,
      LinkLabelLinkClickedEventArgs e )
    {
          // change LinkColor after it has been clicked
      deitelLinkLabel.LinkVisited = true;

      System.Diagnostics.Process.Start( "http://www.deitel.cc );"
    } // end method deitelLinkLabel_LinkClicked

```

Setting the `LinkVisited` property to `true`, changing the link's color to purple.

Opening a **Windows Explorer** window. (the `@` symbol indicates that characters in the `string` should be interpreted literally).

Opening a web page in the user's default web browser.

Fig. 15.14 | `LinkLabels` used to link to a drive, a web page and an application. (Part 2 of 6.)

Outline

LinkLabelTestForm.cs

(3of 6)

```
      // run application Notepad
   private void notepadLinkLabel_LinkClicked( object sender,
      LinkLabelLinkClickedEventArgs e )
   {
      // change LinkColor after it has been clicked
      notepadLinkLabel.LinkVisited = true;

      // program called as if in run
      // menu and full path not needed
      System.Diagnostics.Process.Start( "notepad" );
   } // end method driveLinkLabel_LinkClicked
} // end class LinkLabelTestForm
```

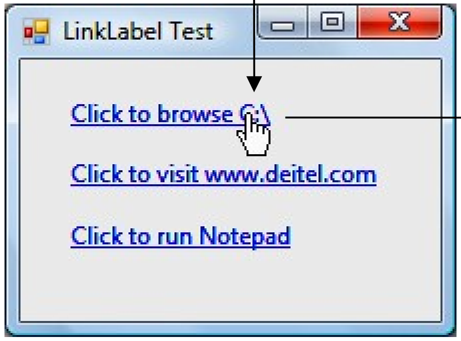
Opening an application. Windows recognizes the argument without a directory or file extension.

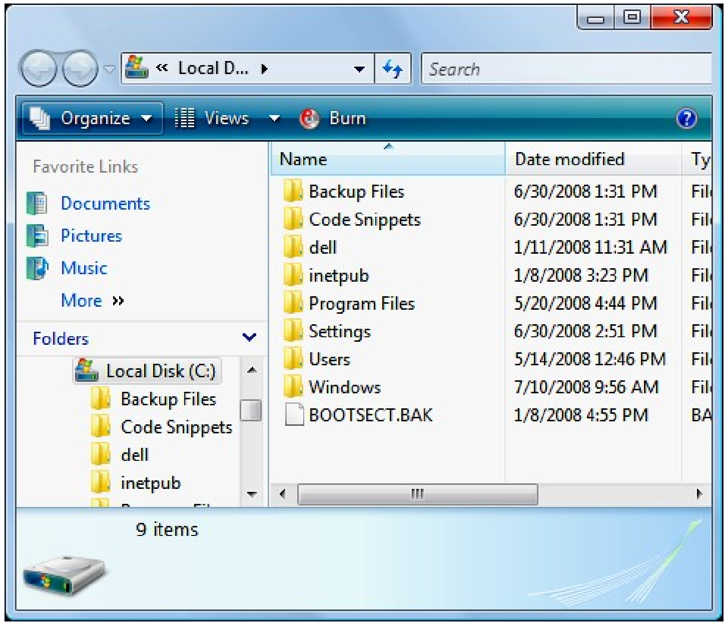
Fig. 15.14 | `LinkLabels` used to link to a drive, a web page and an application. (Part 3 of 6.)

LinkLabelTest Form.cs

(4 of 6)

Click first `LinkLabel` to
look at contents of
C: drive





Name	Date modified	Ty
Backup Files	6/30/2008 1:31 PM	Fil
Code Snippets	6/30/2008 1:31 PM	Fil
dell	1/11/2008 11:31 AM	Fil
inetpub	1/8/2008 3:23 PM	Fil
Program Files	5/20/2008 4:44 PM	Fil
Settings	6/30/2008 2:51 PM	Fil
Users	5/14/2008 12:46 PM	Fil
Windows	7/10/2008 9:56 AM	Fil
BOOTSECT.BAK	1/8/2008 4:55 PM	BA

Fig. 15.14 | `LinkLabels` used to link to a drive, a web page and an application. (Part 4 of 6.)

LinkLabelTest Form.cs

(5 of 6)

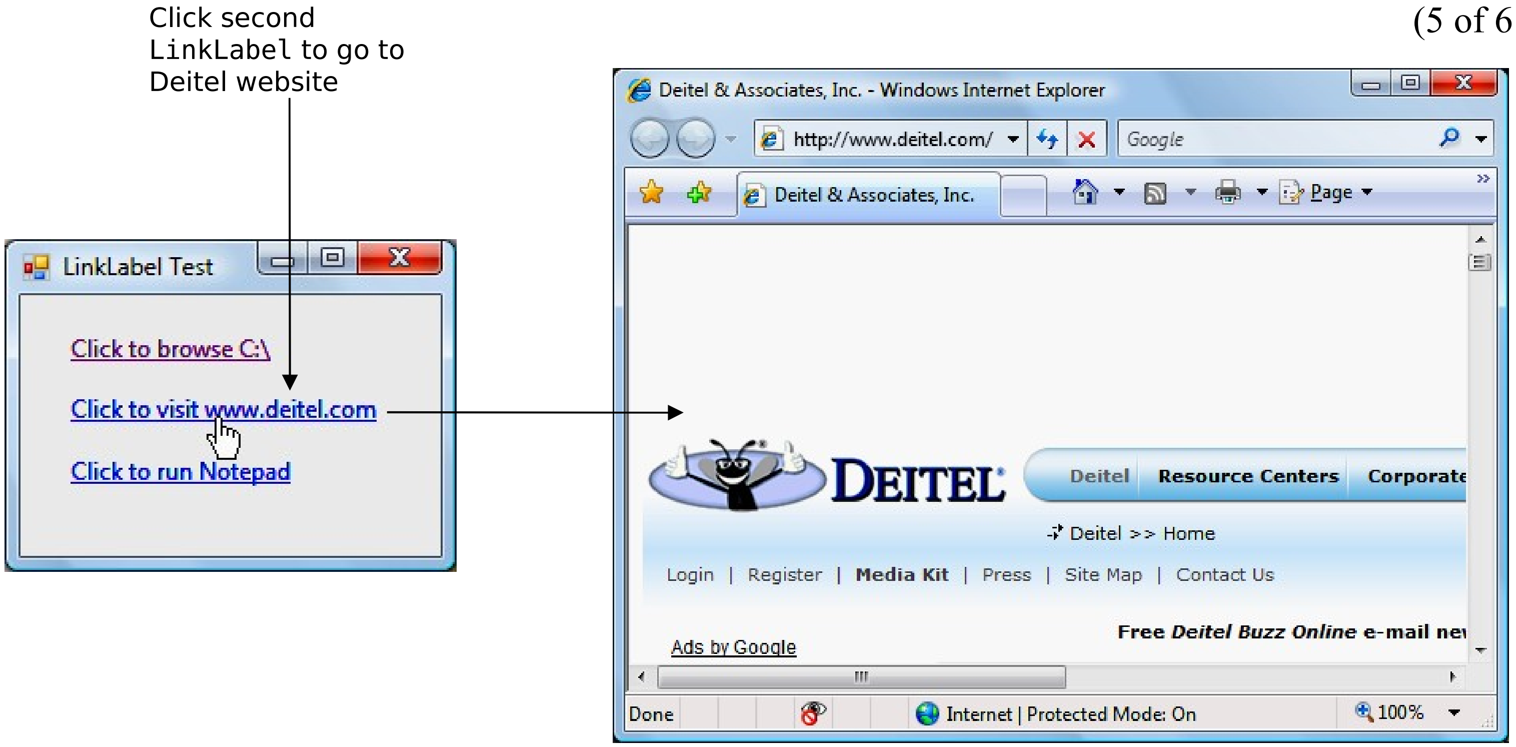

Click second LinkLabel to go to Deitel website
LinkLabel Test
Click to browse C:\
Click to visit www.deitel.com
Click to run Notepad
Deitel & Associates, Inc. - Windows Internet Explorer
http://www.deitel.com/
Google
Deitel & Associates, Inc.
Page
DEITEL
Deitel
Resource Centers
Corporate
Deitel >> Home
Login | Register | Media Kit | Press | Site Map | Contact Us
Free Deitel Buzz Online e-mail nev
Ads by Google
Done
Internet | Protected Mode: On
100%

Fig. 15.14 | LinkLabels used to link to a drive, a web page and an application. (Part 5 of 6.)

Outline

`LinkLabelTest Form.cs`

(6 of 6)

Click on third `LinkLabel` to open Notepad

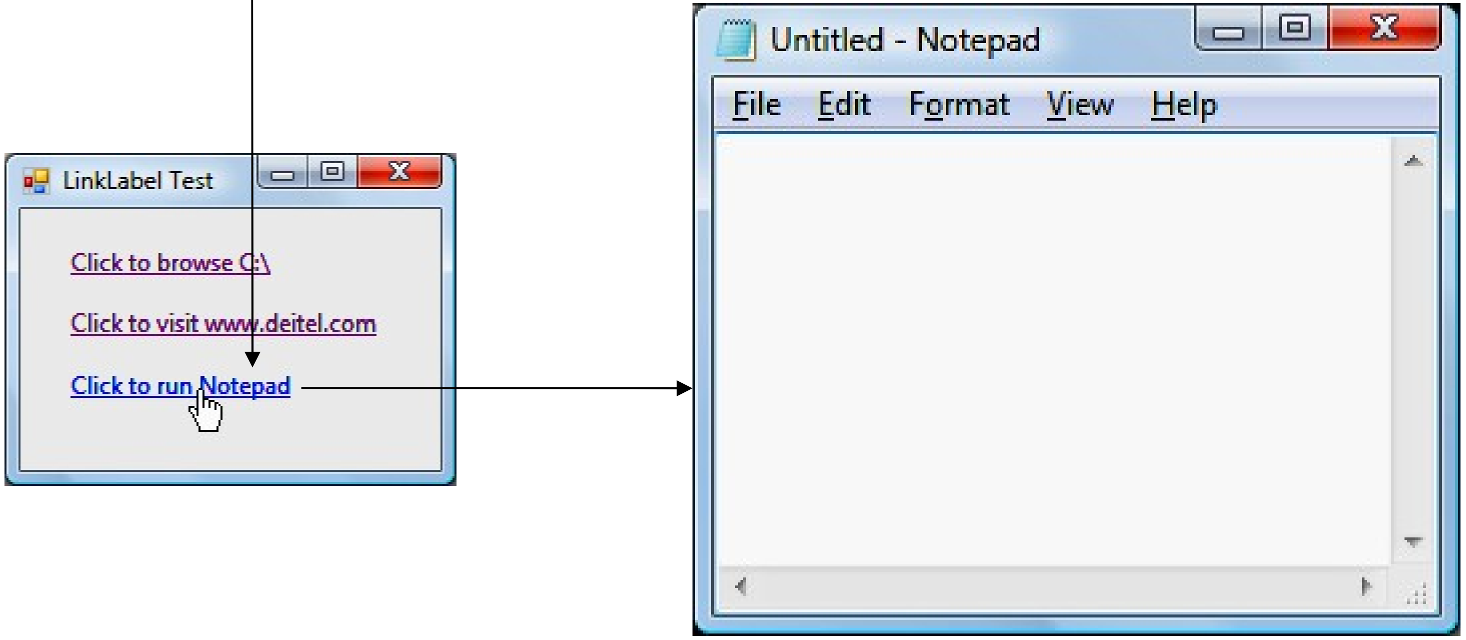


Fig. 15.14 | `LinkLabel`s used to link to a drive, a web page and an application. (Part 6 of 6.)

15.6 `ListBox` Control

- The **`ListBox`** control allows the user to view and select from multiple items in a list.
- The **`CheckedListBox`** control extends a `ListBox` by including `CheckBox`es next to each item in the list.
- Figure 15.15 displays a `ListBox` and a `CheckedListBox`.

15.6 ListBox Control (Cont.)

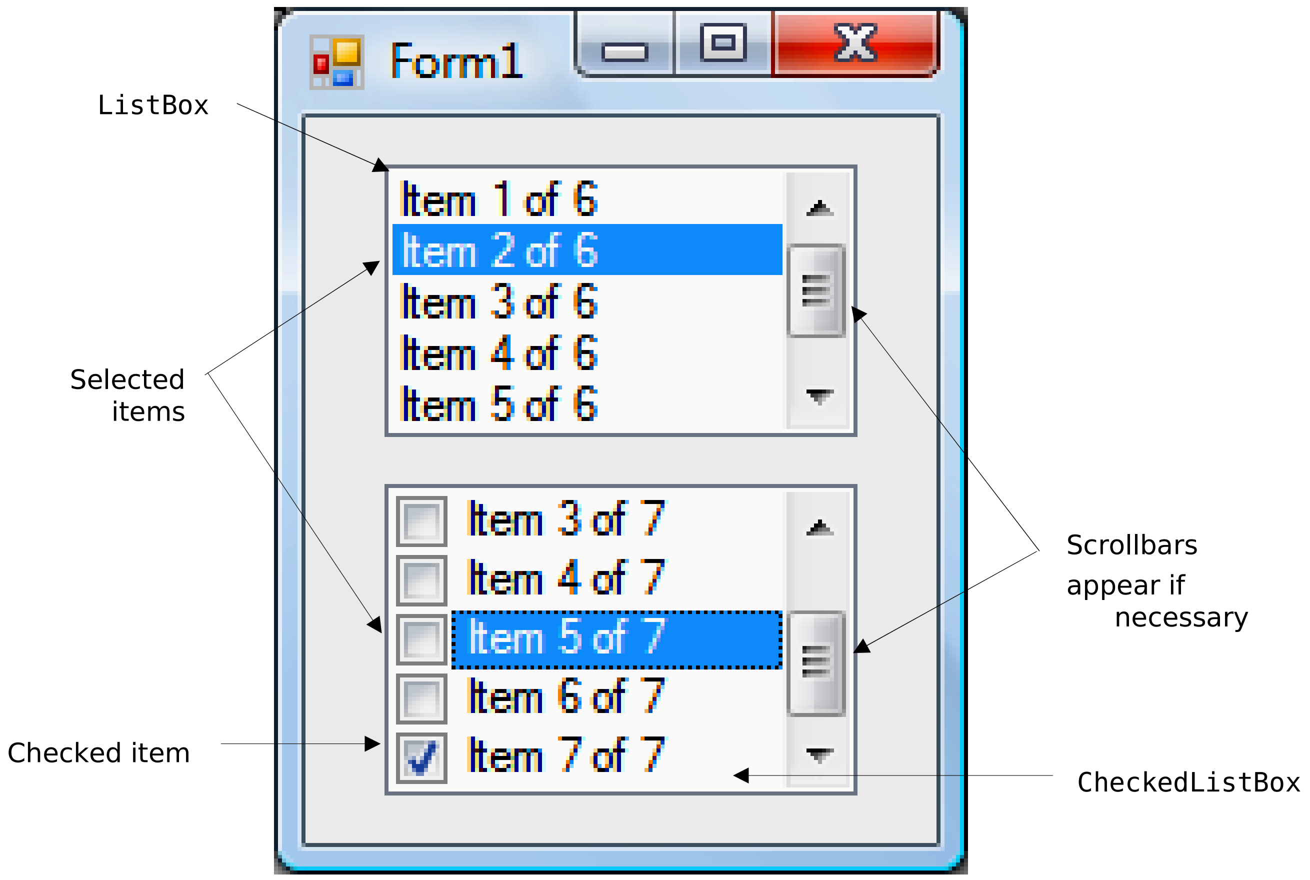


Fig. 15.15 | ListBox and CheckedListBox on a Form.

15.6 ListBox Control (Cont.)

ListBox properties, methods and an event	Description
Common Properties	
Items	The collection of items in the ListBox
MultiColumn	Indicates whether the ListBox can display multiple columns.
SelectedIndex	Returns the index of the selected item.
SelectedIndices	Returns a collection containing the indices for all selected items.
SelectedItem	Returns a reference to the selected item.

Fig. 15.16 | ListBox properties, methods and an event. (Part 1 of 2.)

15.6 ListBox Control (Cont.)

ListBox properties, methods and an event	Description
SelectedItems	Returns a collection of the selected item(s).
SelectionMode	Determines the number of items that can be selected and the means through which multiple items can be selected.
Sorted	Indicates whether items are sorted alphabetically.
Common Methods	
ClearSelected	Deselects every item.
GetSelected	Takes an index as an argument and returns `true`if the corresponding item is selected.
Common Event	
SelectedIndexChanged	Generated when the selected index changes.

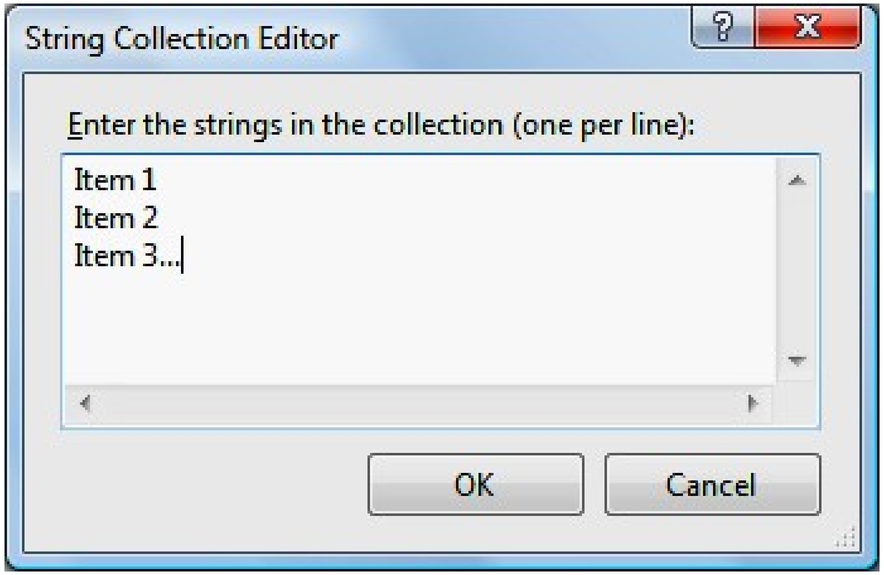
Fig. 15.16 | ListBox properties, methods and an event.(Part 2 of 2.)

15.6 `ListBox` Control (Cont.)

- To add items to a `ListBox` or to a `CheckedListBox`, we must add objects to its `Items` collection.

myListBox`.Items.Add(` *myListItem* `);`

- You can add items to `ListBox`es and `CheckedListBox`es visually by examining the `Items` property in the **Properties** window (Fig. 15.17).



Enter the strings in the collection (one per line):

Item 1
Item 2
Item 3...

Fig. 15.17 | String Collection Editor.

- Figure 15.18 uses class `ListBoxTestForm` to add, remove and clear items from `ListBox` `displayListBox`.\

ListBoxTestForm.cs

(1 of 5)

```
// Fig15.18: ListBoxTestForm.cs
// Program to add, remove and clear ListBox items
using System;
using System.Windows.Forms;

namespace ListBoxTest
{
   // Form uses a TextBox and Buttons to add,
   // remove, and clear ListBox items
   public partial class ListBoxTestForm : Form
  {
     // constructor
     public ListBoxTestForm()
     {
       InitializeComponent();
     } // end constructor
```

Fig. 15.18 | Program that adds, removes and clears `ListBox` items. (Part 1 of 5.)

Outline

**ListBoxTest
Form.cs**

(2 of 5)

```

      // add new item to ListBox (text from input TextBox)
      // and clear input TextBox
      private void addButton_Click( object sender, EventArgs e )
   {
      displayListBox.Items.Add( inputTextBox.Text );
      inputTextBox.Clear();
   } // end method addButton_Click

      // remove item if one is selected
   private void removeButton_Click( object sender, EventArgs e )
   {
      // check whether item is selected, remove if selected
      if ( displayListBox.SelectedIndex != -1 )
         displayListBox.Items.RemoveAt(
            displayListBox.SelectedIndex );
   } // end method removeButton_Click
```

Adding `strings` using method `Add` of the `Items` collection.

Using method `RemoveAt` to remove the item at the selected index.

Fig. 15.18 | Program that adds, removes and clears `ListBox` items. (Part 2 of 5.)

Outline

ListBoxTest Form.cs

(3 of 5)

```

      // clear all items in ListBox
      private void clearButton_Click( object sender, EventArgs e )
      {
         displayListBox.Items.Clear();
      } // end method clearButton_Click

      // exit application
      private void exitButton_Click( object sender, EventArgs e )
      {
         Application.Exit();
      } // end method exitButton_Click
   } // end class ListBoxTestForm
} // end namespace ListBoxTest
```

Using method `Clear` of the `Items` collection to remove all the entries.

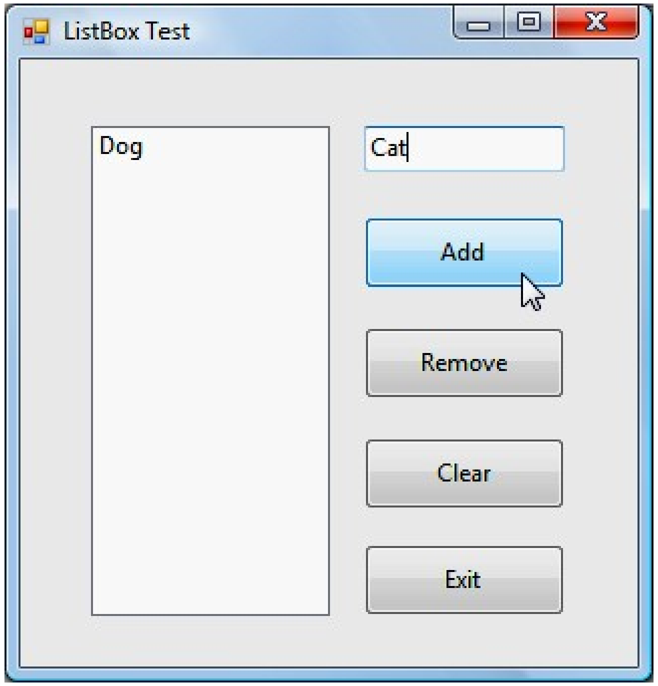
Fig. 15.18 | Program that adds, removes and clears `ListBox` items. (Part 3 of 5.)

Outline

`ListBoxTest Form.cs`

(4 of 5)

a) Adding an item



b) Adding more items

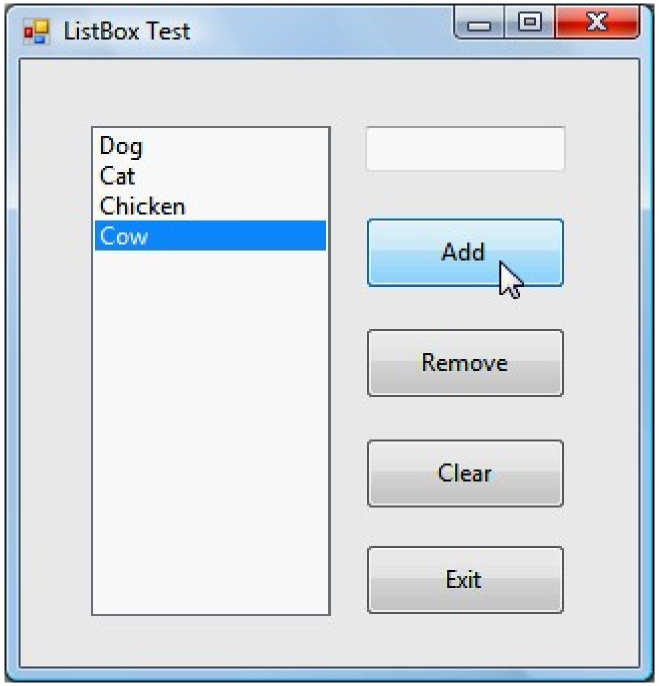


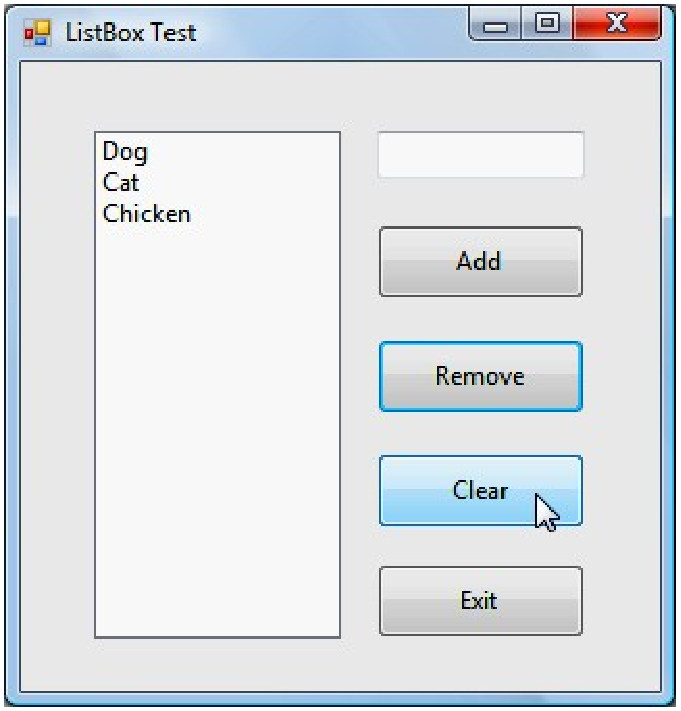
Fig. 15.18 | Program that adds, removes and clears `ListBox` items. (Part 4 of 5.)

Outline

`ListBoxTestForm.cs`

(5 of 5)

c) An item has been removed



d) Clearing the list

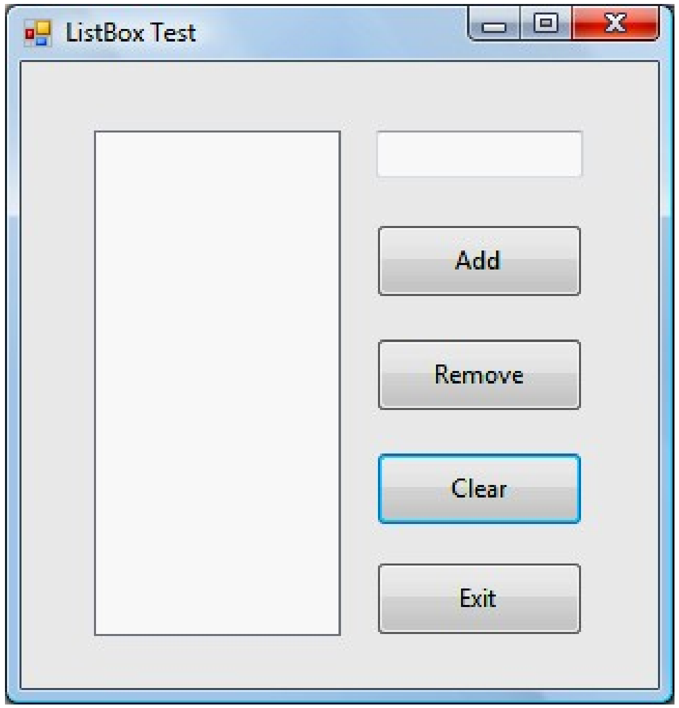


Fig. 15.18 | Program that adds, removes and clears `ListBox` items. (Part 5 of 5.)

15.7 CheckedListBox Control

- The `CheckedListBox` control derives from `ListBox` and displays a `CheckBox` with each item (Fig. 15.19).

CheckedListBox properties, methods and events	Description
Common Properties	*(All the* `ListBox` *properties, methods and events are inherited by* `CheckedListBox`*.)*
`CheckedItems`	Contains the collection of items that are checked.
`CheckedIndices`	Returns indices for all checked items.
`CheckOnClick`	When `true` and the user clicks an item, the item is both selected and checked or unchecked.
`SelectionMode`	Determines whether items can be checked. The possible values are `One` (allows multiple checks to be placed) or `None` (does not allow any checks to be placed).

Fig. 15.19 | `CheckedListBox` properties, methods and events. (Part 1 of 2.)

15.7 CheckedListBox Control (Cont.)

CheckedListBox properties, methods and events	**Description**
Common Method	
`GetItemChecked`	Takes an index and returns `true` if the corresponding item is checked.
Common Event (Event arguments `ItemCheckEventArgs`)	
`ItemCheck`	Generated when an item is checked or unchecked.
`ItemCheckEventArgs` Properties	
`CurrentValue`	Indicates whether the current item is checked or unchecked.
`Index`	Returns the zero-based index of the item that changed.
`NewValue`	Specifies the new state of the item.

Fig. 15.19 | `CheckedListBox` properties, methods and events. (Part 2 of 2.)

Outline

- Class `CheckedListBoxTestForm` uses a `CheckedListBox` and a `ListBox` to display a user's selection of books (Fig. 15.20).

`CheckedListBox TestForm.cs`

(1 of 4)

```
// Fig15.20: CheckedListBoxTestForm.cs
// Using the checked ListBox to add items to a display ListBox
using System;
using System.Windows.Forms;

namespace CheckedListBoxTest
{
   // Form uses a checked ListBox to add items to a display ListBox
   public partial class CheckedListBoxTestForm : Form
  {
      // constructor
      public CheckedListBoxTestForm()
      {
        InitializeComponent();
      } // end constructor
```

Fig. 15.20 | `CheckedListBox` and `ListBox` used in a program to display a user selection. (Part 1 of 4.)

Outline

`CheckedListBox`
`TestForm.cs`

(2 of 4)

```
      // item about to change
      // add or remove from display ListBox
      private void itemCheckedListBox_ItemCheck(
         object sender, ItemCheckEventArgs e )
      {
         // obtain reference of selected item
         string item = itemCheckedListBox.SelectedItem.ToString();

         // if item checked, add to ListBox
         // otherwise remove from ListBox
         if ( e.NewValue == CheckState.Checked )
            displayListBox.Items.Add( item );
         else
            displayListBox.Items.Remove( item );
      } // end method inputCheckedListBox_ItemCheck
   } // end class CheckedListBoxTestForm
} // end namespace CheckedListBoxTest
```

This event handler maintains a list of checked items in the `ListBox`.

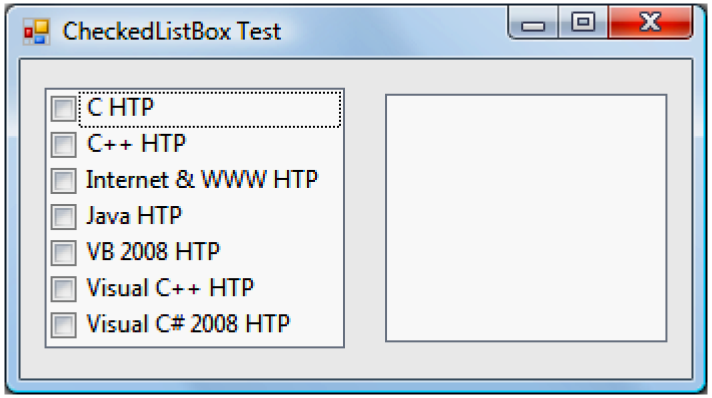
Determining whether the user checked or unchecked the item.

Fig. 15.20 | `CheckedListBox` and `ListBox` used in a program to display a user selection. (Part 2 of 4.)

CheckedListBox TestForm.cs

(3 of 4)

a) Application at start



b) Selecting and checking items

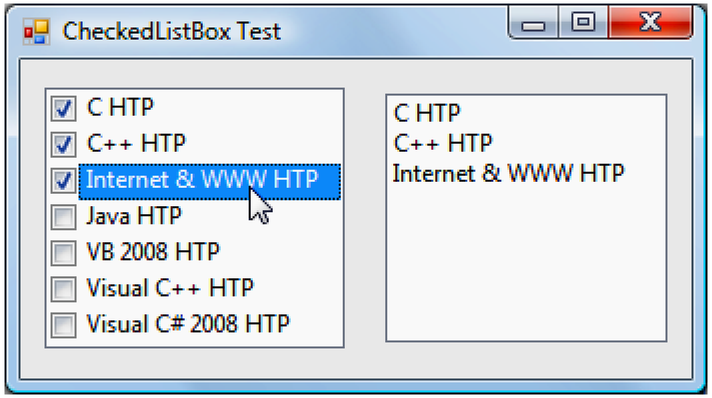


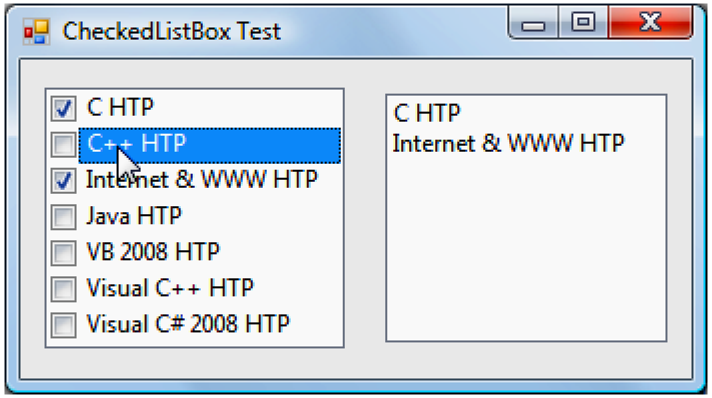
Fig. 15.20 | `CheckedListBox` and `ListBox` used in a program to display a user selection. (Part 3 of 4.)

Outline

`CheckedListBox TestForm.cs`

(4 of 4)

c) Unchecking selected items



d) Checking items

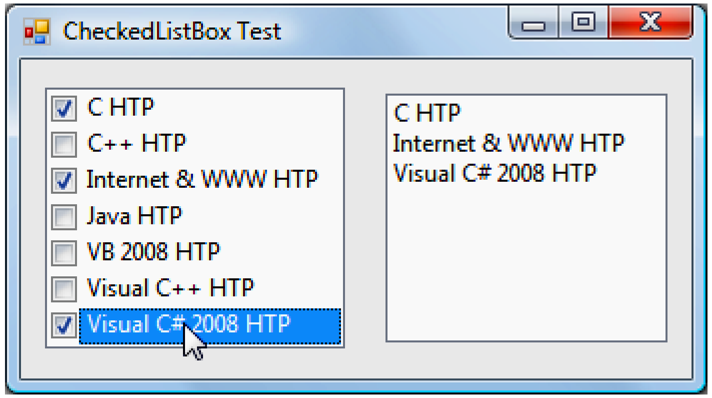


Fig. 15.20 | `CheckedListBox` and `ListBox` used in a program to display a user selection. (Part 4 of 4.)

15.8 ComboBox Control

- The **ComboBox** control combines `TextBox` features with a **drop-down list**.
- Figure 15.21 shows a sample `ComboBox` in three different states.

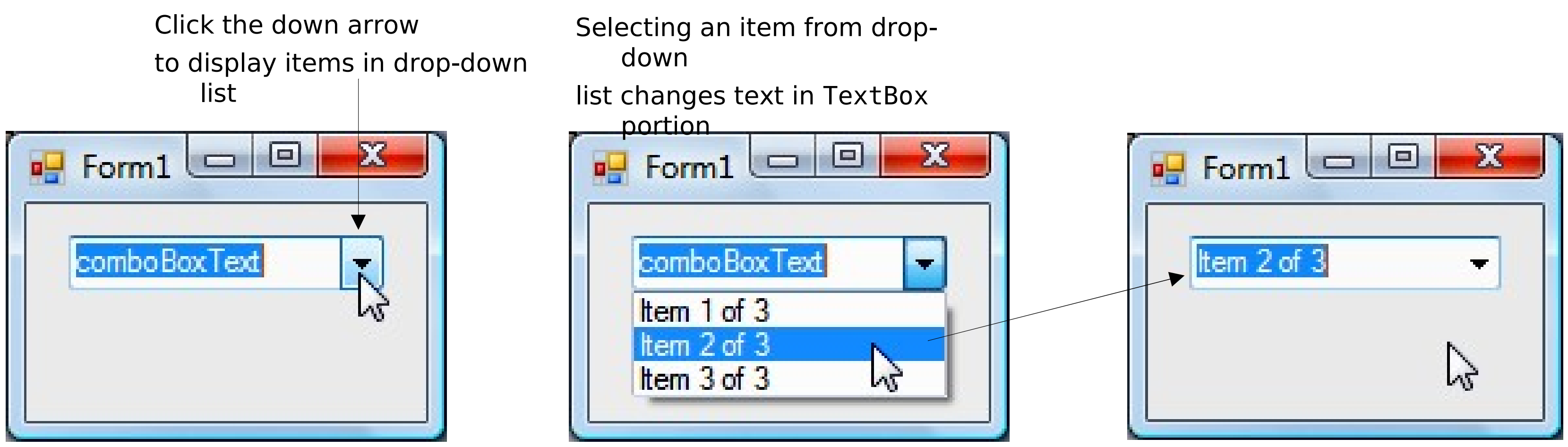


Fig. 15.21 | ComboBox demonstration.

15.8 ComboBox Control (Cont.)

ComboBox properties and an event	Description
Common Properties	
DropDownStyle	Determines the type of ComboBox .
Items	The collection of items in the ComboBox control.
MaxDropDownItems	Specifies the maximum number of items that the drop-down list can display. If the number of items exceeds the maximum number of items, a scrollbar appears.
SelectedIndex	Returns the index of the selected item.
SelectedItem	Returns a reference to the selected item.
Sorted	Indicates whether items are sorted alphabetically.
Common Event	
SelectedIndexChanged	Generated when the selected index changes.

Fig. 15.22 | ComboBox properties and an event.

Outline

- Class `ComboBoxTestForm` (Fig. 15.23) allows users to select a shape to draw by using a `ComboBox`.
- Set the `ComboBox`'s `DropDownStyle` to `DropDownList` to restrict users to preset options.

ComboBoxTestForm.cs

(1 of 5)

```
// Fig. 15.23: ComboBoxTestForm.cs
// Using ComboBox to select a shape to draw.
using System;
using System.Drawing;
using System.Windows.Forms;

namespace ComboBoxTest
{
  // Form uses a ComboBox to select different shapes to draw
  public partial class ComboBoxTestForm : Form
  {
    // constructor
     public ComboBoxTestForm()
     {
       InitializeComponent();
     } // end constructor
```

Fig. 15.23 | ComboBox used to draw a selected shape. (Part 1 of 5.)

Outline

ComboBoxTestForm.cs

(2 of 5)

```

      // get index of selected shape, draw shape
      private void imageComboBox_SelectedIndexChanged(
         object sender, EventArgs e )
      {
         // create graphics object, Pen and SolidBrush
         Graphics myGraphics = base.CreateGraphics();

         // create Pen using color DarkRed
         Pen myPen = new Pen( Color.DarkRed );

         // create SolidBrush using color DarkRed
         SolidBrush mySolidBrush = new SolidBrush( Color.DarkRed );

         // clear drawing area, setting it to color white
         myGraphics.Clear( Color.White );

         // find index, draw proper shape
```

The Graphics object allows a pen or brush to draw on a component

The Pen object is used to draw the outlines of shapes.

The SolidBrush object is used to fill solid shapes.

Fig. 15.23 | ComboBox used to draw a selected shape. (Part 2 of 5.)

Outline

ComboBoxTest Form.cs

(3 of 5)

```
         switch ( imageComboBox.SelectedIndex )
            myGraphics.DrawEllipse( myPen, 50, 85, 150, 115 );
            break;
         case 3: // case Pie is selected
            myGraphics.DrawPie( myPen, 50, 50, 150, 150, 0, 45 );
            break;
         case 4: // case Filled Circle is selected
            myGraphics.FillEllipse( mySolidBrush, 50, 50, 150, 150 );
            break;
```

`SelectedIndex` determines which item the user selected.

`DrawEllipse` takes a `Pen`, the coordinates of the center and the dimensions of the ellipse.

`DrawRectangle` takes a `Pen`, the coordinates of the upper-left corner and the dimensions of the rectangle.

`DrawPie` takes a `Pen`, the coordinates of the upper-left corner, its width and height, the start angle and the sweep angle of the pie.

Fig. 15.23 | `ComboBox` used to draw a selected shape. (Part 3 of 5.)

Outline

ComboBoxTestForm.cs

(4 of 5)

```
         case 5: // case Filled Rectangle is selected
            myGraphics.FillRectangle( mySolidBrush, 50, 50, 150,
               150 );
            break;
         case 6: // case Filled Ellipse is selected
            myGraphics.FillEllipse( mySolidBrush, 50, 85, 150, 115 );
            break;
         case 7: // case Filled Pie is selected
            myGraphics.FillPie( mySolidBrush, 50, 50, 150, 150, 0,
               45 );
            break;
      } // end switch

      myGraphics.Dispose(); // release the Graphics object
   } // end method imageComboBox_SelectedIndexChanged
  } // end class ComboBoxTestForm
} // end namespace ComboBoxTest
```

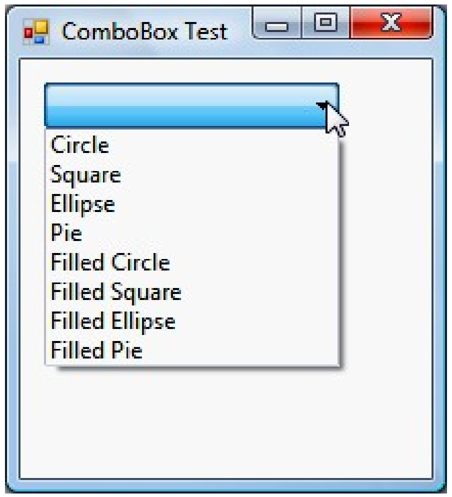
Fig. 15.23 | ComboBox used to draw a selected shape. (Part 4 of 5.)

Outline

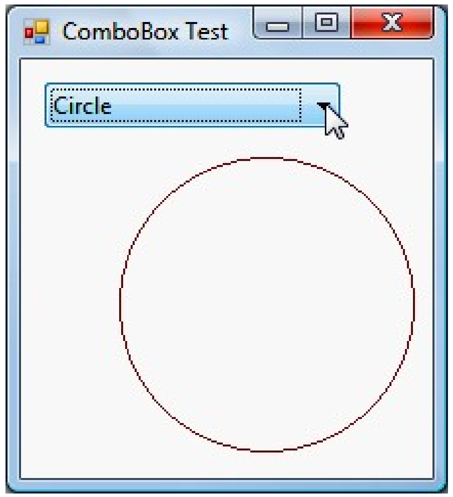
`ComboBoxTestForm.cs`

(5 of 5)

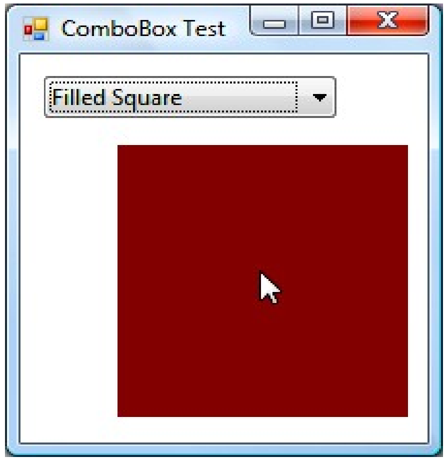
a)



b)



c)



d)

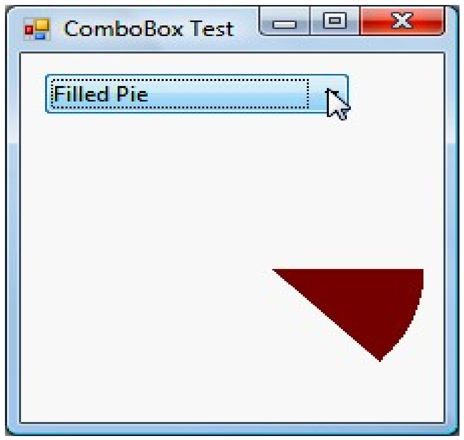


Fig. 15.23 | `ComboBox` used to draw a selected shape. (Part 5 of 5.)

15.9 TreeView Control

- The **`TreeView`** control displays **nodes** hierarchically in a **tree**.
- A **parent node** contains **child nodes**, and the child nodes can be parents to other nodes.
- Two child nodes that have the same parent node are considered **sibling nodes**.
- The first parent node of a tree is the **root** node (a `TreeView` can have multiple roots).

15.9 TreeView Control (Cont.)

- The nodes in a `TreeView` (Fig. 15.24) are instances of class **`TreeNode`**.
- Each `TreeNode` has a **`Nodes` collection** (type **`TreeNodeCollection`**), which contains a list of its children.

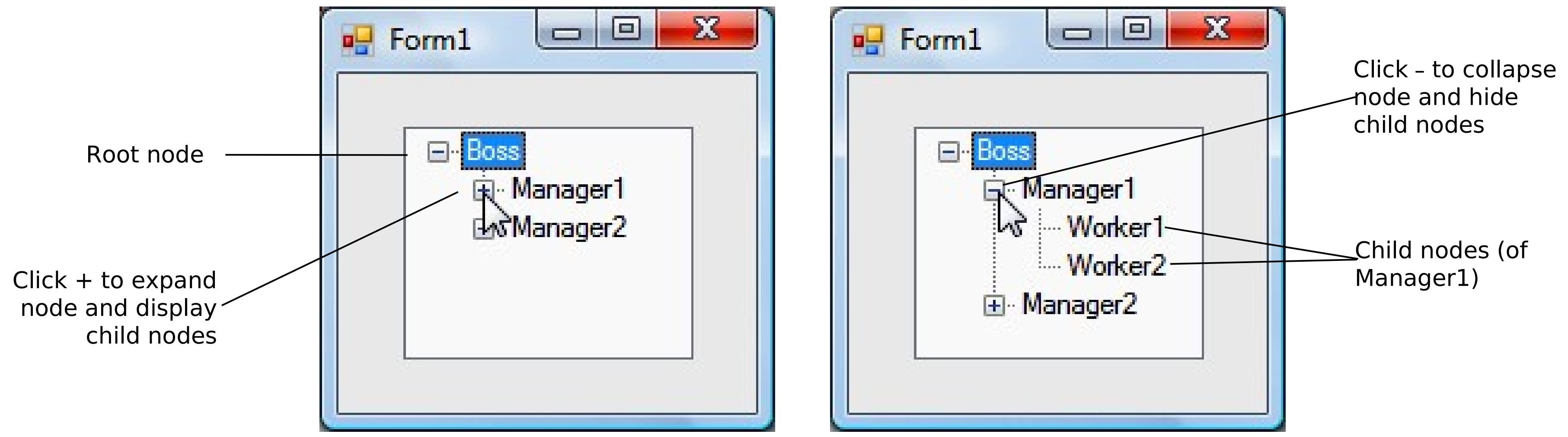

Form1
Boss
Manager1
Manager2
Form1
Boss
Manager1
Worker1
Worker2
Manager2
Root node
Click + to expand node and display child nodes
Click – to collapse node and hide child nodes
Child nodes (of Manager1)

Fig. 15.24 | `TreeView` displaying a sample tree.

15.9 TreeView Control (Cont.)

TreeView properties and an event	Description
Common Properties	
CheckBoxes	Indicates whether CheckBoxes appear next to nodes.
ImageList	Specifies an ImageList object containing the node icons.
Nodes	Lists the collection of TreeNodes in the control.
SelectedNode	The selected node.
Common Event (Event arguments ***TreeViewEventArgs*** *)*	
AfterSelect	Generated after selected node changes.

Fig. 15.25 | TreeView properties and an event.

15.9 TreeView Control (Cont.)

TreeNode properties and methods	Description
Common Properties	
Checked	Indicates whether the TreeNode is checked.
FirstNode	Specifies the first node in the Nodes collection.
FullPath	Indicates the path of the node, starting at the root of the tree.
ImageIndex	Specifies the index of the image shown when the node is deselected.
LastNode	Specifies the last node in the Nodes collection.
NextNode	Next sibling node.
Nodes	Collection of TreeNodes contained in the current node.

Fig. 15.26 | TreeNode properties and methods. (Part 1 of 2.)

15.9 TreeView Control (Cont.)

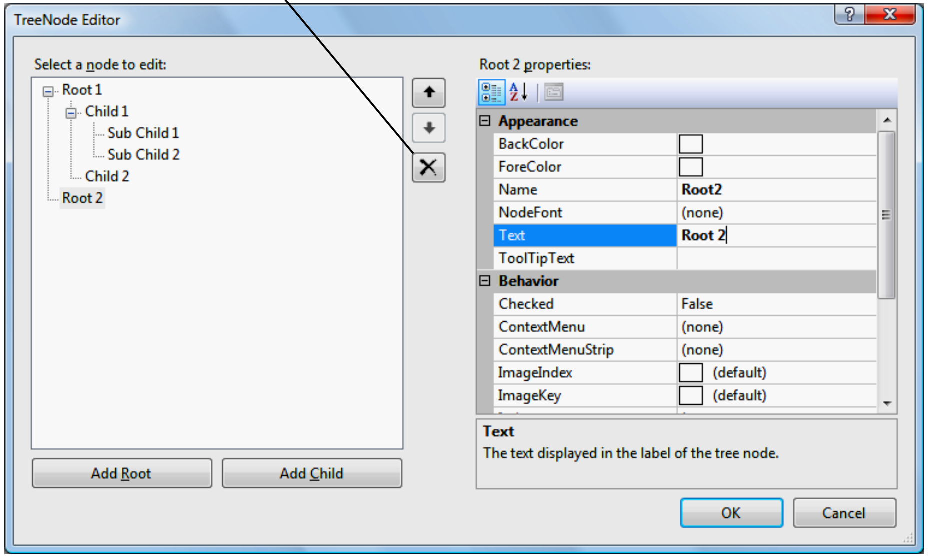
TreeNode properties and methods	Description
PrevNode	Previous sibling node.
SelectedImageIndex	Specifies the index of the image to use when the node is selected.
Text	Specifies the TreeNode's text.
Common Methods	
Collapse	Collapses a node.
Expand	Expands a node.
ExpandAll	Expands all the children of a node.
GetNodeCount	Returns the number of child nodes.

Fig. 15.26 | TreeNode properties and methods. (Part 2 of 2)

15.9 TreeView Control (Cont.)

- To add nodes to the `TreeView` visually, click the ellipsis next to the `Nodes` property in the **Properties** window.
- This opens the **TreeNode Editor** (Fig. 15.27).

Delete current node



Appearance	
BackColor	
ForeColor	
Name	Root2
NodeFont	(none)
Text	Root 2
ToolTipText	
Behavior	
Checked	False
ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	(default)
ImageKey	(default)

Text
The text displayed in the label of the tree node.

Fig. 15.27 | **TreeNode Editor**.

15.9 TreeView Control (Cont.)

- To add nodes programmatically, first create a root node.

```
myTreeView.Nodes.Add( new TreeNode( rootLabel ) );
```

- To add children to a root node first select the appropriate root node:

```
myTreeView.Nodes[ myIndex ]
```

- To add a child to the root node at index *myIndex*, write:

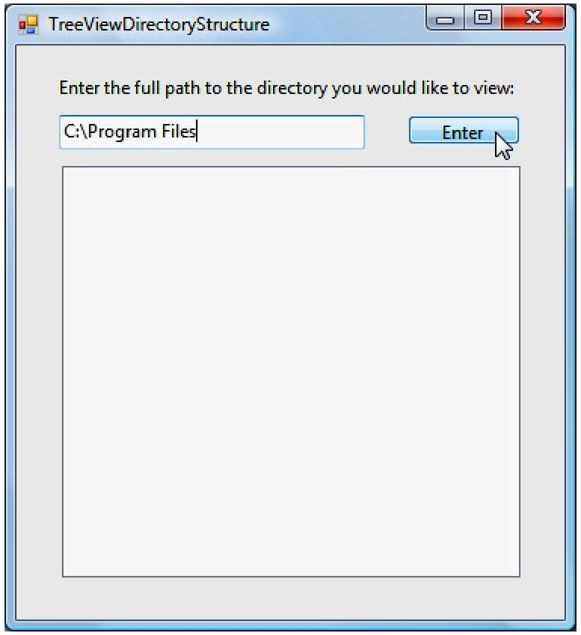
```
myTreeView.Nodes[ myIndex ].Nodes.Add( new
  TreeNode( ChildLabel ) );
```

Outline

`TreeViewDirectoryStructureForm.cs`

(6 of 6)

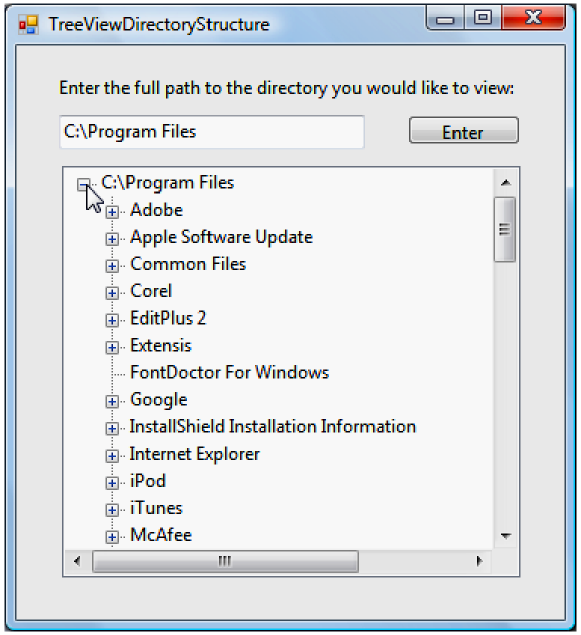
a)



Enter the full path to the directory you would like to view:

C:\Program Files

b)



Enter the full path to the directory you would like to view:

C:\Program Files

```
C:\Program Files
    Adobe
    Apple Software Update
    Common Files
    Corel
    EditPlus 2
    Extensis
    FontDoctor For Windows
    Google
    InstallShield Installation Information
    Internet Explorer
    iPod
    iTunes
    McAfee
```

Fig. 15.28 | `TreeView` used to display directories. (Part 6 of 6.)

- Class `TreeViewDirectoryStructureForm` (Fig. 15.28) uses a `TreeView` to display the contents of a directory.

`TreeViewDirectory`
`StructureForm`
`.Cs`

(1 of 6)

```
// Fig15.28: TreeViewDirectoryStructureForm.cs
// Using TreeView to display directory structure.
using System;
using System.Windows.Forms;
using System.IO;

namespace TreeViewDirectoryStructure
{
   // Form uses TreeView to display directory structure
   public partial class TreeViewDirectoryStructureForm : Form
  {
    string substringDirectory; // store last part of full path name

    // constructor
    public TreeViewDirectoryStructureForm()
    {
```

Fig. 15.28 | `TreeView` used to display directories. (Part 1 of 6.)

Outline

TreeViewDirectoryStructureForm.Cs

(2 of 6)

```
            InitializeComponent();
        } // end constructor

      // populate current node with subdirectories
      public void PopulateTreeView(
         string directoryValue, TreeNode parentNode )
      {
         // array stores all subdirectories in the directory
         string[] directoryArray =
            Directory.GetDirectories( directoryValue );

         // populate current node with subdirectories
         try
         {
            // check to see if any subdirectories are present
            if ( directoryArray.Length != 0 )
            {
               // for every subdirectory, create new TreeNode,
               // add as a child of current node and recursively
               // populate child nodes with subdirectories
```

Method `GetDirectories` takes the given directory and returns an array of `string`s (the subdirectories).

Fig. 15.28 | `TreeView` used to display directories. (Part 2 of 6.)

Outline

TreeViewDirectoryStructureForm.cs

(3 of 6)

```
            foreach ( string directory in directoryArray )
            // create TreeNode for current directory
             TreeNode myNode = new TreeNode( substringDirectory );

            // add current directory node to parent node
             parentNode.Nodes.Add( myNode );

            // recursively populate every subdirectory
             PopulateTreeView( directory, myNode );
           } // end foreach
         } // end if
       } //end try

```

The full path name is reduced to just the directory name.

Each subdirectory's name is created as a node.

Using method Add to add the subdirectories as child nodes.

Calling PopulateTreeView recursively on every subdirectory populates the TreeView with the entire directory structure.

Fig. 15.28 | TreeView used to display directories. (Part 3 of 6.)

Outline

`TreeViewDirectoryStructureForm.cs`

(4 of 6)

```
// catch exception
// check if the directory entered by user exists
// if it does, then fill in the TreeView,
// if not, display error MessageBox
 if ( Directory.Exists( inputTextBox.Text ) )
 {
   // add full path name to directoryTreeView
    directoryTreeView.Nodes.Add( inputTextBox.Text );
```

When access is denied, a node is added containing “`Access Denied`”.

Clearing all the nodes in `directoryTreeView`.

Adding the specified directory as the root node.

Fig. 15.28 | `TreeView` used to display directories. (Part 4 of 6.)

Outline

TreeViewDirectoryStructureForm.cs

(5 of 6)

```

            // insert subfolders
               PopulateTreeView(
                  inputTextBox.Text, directoryTreeView.Nodes[ 0 ] );
         }
         // display error MessageBox if directory not found
         else
            MessageBox.Show( inputTextBox.Text + " could not be found.",
               "Directory Not Found", MessageBoxButtons.OK,
               MessageBoxIcon.Error );
      } // end method enterButton_Click
   } // end class TreeViewDirectoryStructureForm
} // end namespace TreeViewDirectoryStructure
```

Fig. 15.28 | `TreeView` used to display directories. (Part 5 of 6.)

15.10 `ListView` Control (Cont.)

- The **`ListView`** control is more versatile than a `ListBox` and can display items in different formats.
- For example, a `ListView` can display icons next to the list items and show the details of items in columns.

15.10 ListView Control (Cont.)

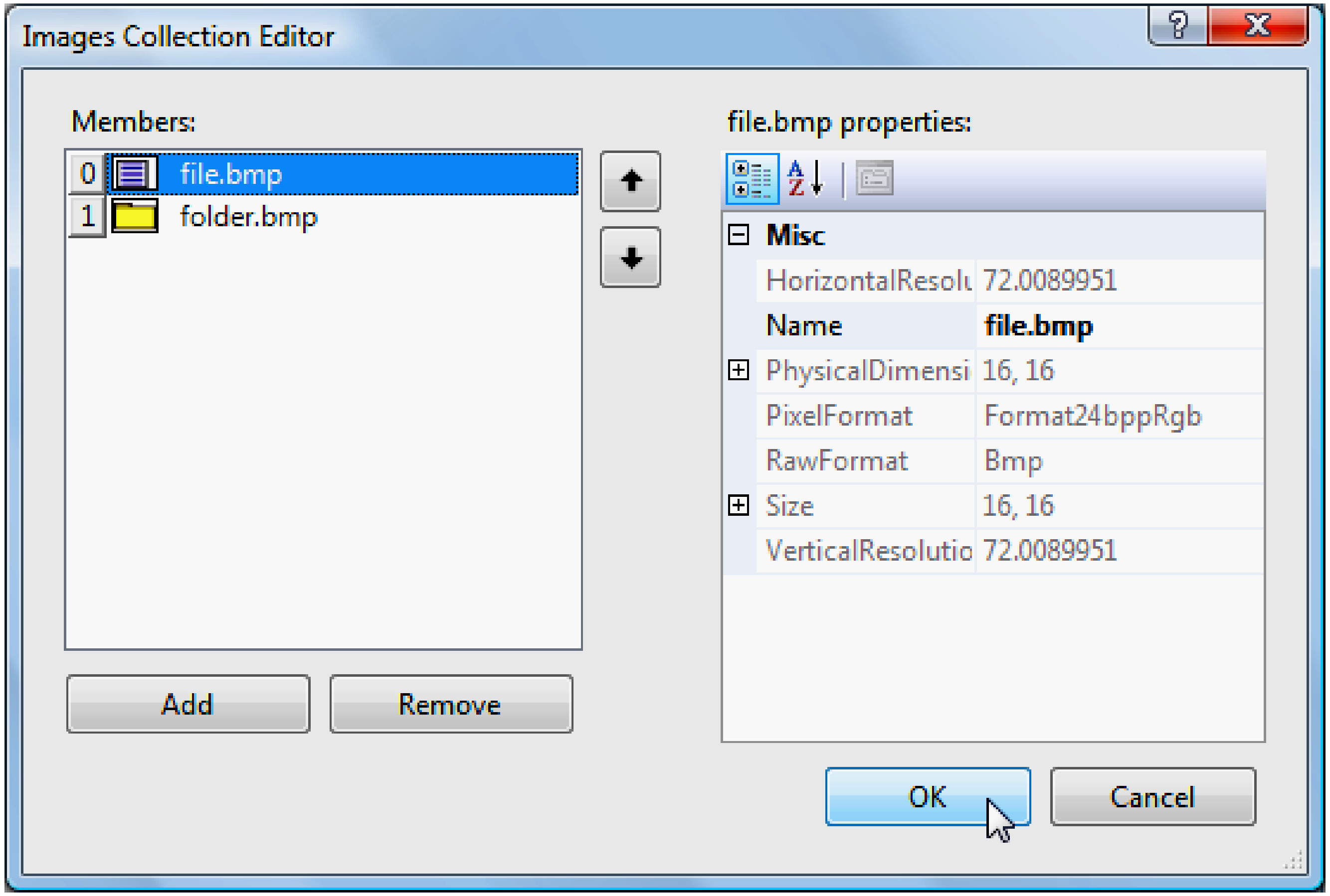
ListView properties and an event	Description
Common Properties	
Activation	Determines how the user activates an item.
CheckBoxes	Indicates whether items appear with CheckBoxes.
LargeImageList	Specifies the ImageList containing large icons for display.
Items	Returns the collection of ListViewItems in the control.
MultiSelect	Determines whether multiple selection is allowed.
SelectedItems	Gets the collection of selected items.
SmallImageList	Specifies the ImageList containing small icons for display.
View	Determines appearance of ListViewItems.
Common Event	
ItemActivate	Generated when an item in the ListView is activated.

Fig. 15.29 | ListView properties and an event.

15.10 ListView Control (Cont.)

- `ListView` allows you to define the images used as icons for items.
- Create a `ListView`, then select the **`Images`** property in the **Properties** window to display the **Image Collection Editor** (Fig. 15.30).
- Adding images this way embeds them into the application.

15.10 `ListView` Control (Cont.)



Members:

0	file.bmp
1	folder.bmp

file.bmp properties:

Misc	
HorizontalResolu	72.0089951
Name	file.bmp
PhysicalDimensi	16, 16
PixelFormat	Format24bppRgb
RawFormat	Bmp
Size	16, 16
VerticalResolutio	72.0089951

Fig. 15.30 | **Image Collection Editor** window for an `ImageList` component.

15.10 `ListView` Control (Cont.)

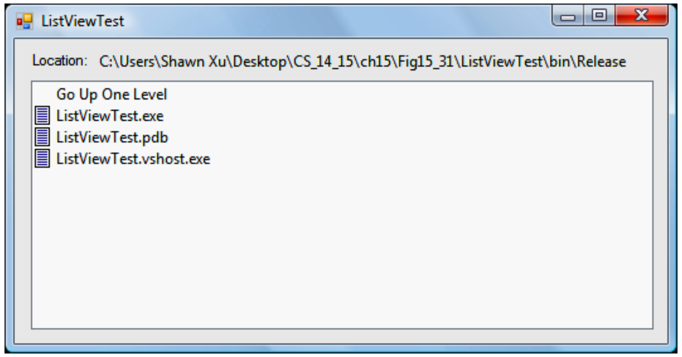
- Set property `SmallImageList` of the `ListView` to the new `ImageList` object.
- Property **`SmallImageList`** specifies the image list for the small icons.
- Property **`LargeImageList`** sets the `ImageList` for large icons.

Outline

`ListViewTest Form.cs`

(8 of 8)

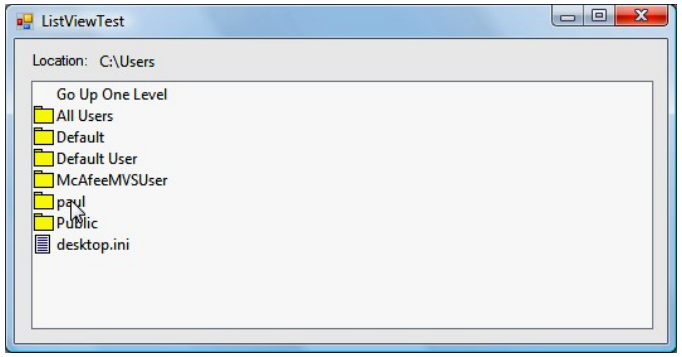
a)



Location: C:\Users\Shawn Xu\Desktop\CS_14_15\ch15\Fig15_31\ListViewTest\bin\Release

Go Up One Level
ListViewTest.exe
ListViewTest.pdb
ListViewTest.vshost.exe

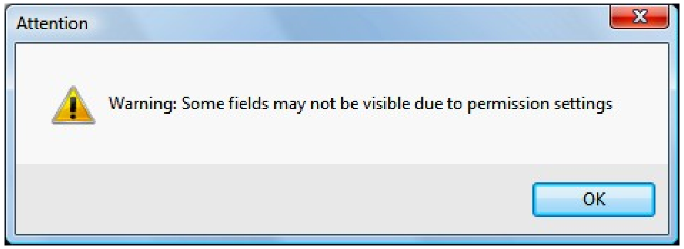
b)



Location: C:\Users

Go Up One Level
All Users
Default
Default User
McAfeeMVSUser
paul
Public
desktop.ini

c)



Warning: Some fields may not be visible due to permission settings

Fig. 15.31 | `ListView` displaying files and folders. (Part 8 of 8.)

Outline

- Class `ListViewTestForm` (Fig. 15.31) displays files and folders in a `ListView`.
- If a file or folder is inaccessible because of permission settings, a `MessageBox` appears.

`ListViewTestForm.cs`

(1 of 8)

```
// Fig15.31: ListViewTestForm.cs
// Displaying directories and their contents in ListView.
using System;
using System.Windows.Forms;
using System.IO;

namespace ListViewTest
{
   // Form contains a ListView which displays
   // folders and files in a directory
   public partial class ListViewTestForm : Form
   {
      // store current directory
      string currentDirectory = Directory.GetCurrentDirectory();

```

Fig. 15.31 | `ListView` displaying files and folders. (Part 1 of 8.)

Outline

ListViewTest Form.cs

(2 of 8)

```
      // constructor
      public ListViewTestForm()
      {
         InitializeComponent();
      } // end constructor

      // browse directory user clicked or go up one level
      private void browserListView_Click( object sender, EventArgs e )
      {
         // ensure an item is selected
         if ( browserListView.SelectedItems.Count != 0 )
         {
            // if first item selected, go up one level
            if ( browserListView.Items[ 0 ].Selected )
            {
               // create DirectoryInfo object for directory
               DirectoryInfo directoryObject =
                  new DirectoryInfo( currentDirectory );

               // if directory has parent, load it
```

Checking whether anything is selected in `browserListView`.

Determining whether the user chose the first item (**Go up one level**).

Fig. 15.31 | `ListView` displaying files and folders. (Part 2 of 8.)

Outline

ListViewTest Form.cs

(3 of 8)

```
         // directory or file chosen
         if ( directoryObject.Parent != null )
         string chosen = browserListView.SelectedItems[ 0 ].Text;

         // if item selected is directory, load selected directory
          if ( Directory.Exists(
             Path.Combine( currentDirectory, chosen ) ) )
          {
           LoadFilesInDirectory(
              Path.Combine( currentDirectory, chosen ) );
          } // end if
```

Using property **Parent** to return the parent directory.

Fig. 15.31 | `ListView` displaying files and folders. (Part 3 of 8.)

Outline

ListViewTest Form.cs

(4 of 8)

```
         } // end else
        // update current directory
        currentDirectory = currentDirectoryValue;
        DirectoryInfo newCurrentDirectory =
```

Fig. 15.31 | `ListView` displaying files and folders. (Part 4 of 8.)

Outline

```
                new DirectoryInfo( currentDirectory );
```

ListViewTest Form.cs

(5 of 8)

Method **GetDirectories** returns an array of `DirectoryInfo` objects representing the subdirectories.

Method **GetFiles** returns an array of class **FileInfo** objects.

Iterating through subdirectories and adding them to `browserListView`.

Iterating through subdirectories and adding them to `browserListView`.

Fig. 15.31 | `ListView` displaying files and folders. (Part 5 of 8.)

Outline

ListViewTest Form.cs

(6 of 8)

```
            // add file to ListView
      // access denied
      catch ( UnauthorizedAccessException )
      {
         MessageBox.Show( "Warning: Some fields may not be " +
            "visible due to permission settings",
            "Attention", 0, MessageBoxIcon.Warning );
      } // end catch
   } // end method LoadFilesInDirectory

   // handle load event when Form displayed for first time
   private void ListViewTestForm_Load( object sender, EventArgs e )
   {
```

Iterating through subdirectories and adding them to `browserListView`.

Fig. 15.31 | `ListView` displaying files and folders. (Part 6 of 8.)

Outline

ListViewTestForm.cs

(7 of 8)

```
         // add icon images to ImageList
         fileFolderImageList.Images.Add( Properties.Resources.folder );
         fileFolderImageList.Images.Add( Properties.Resources.file );

         // load current directory into browserListView
         LoadFilesInDirectory( currentDirectory );
         displayLabel.Text = currentDirectory;
      } // end method ListViewTestForm_Load
   } // end class ListViewTestForm
} // end namespace ListViewTest
```

Folder and file icon images are added to the `Images` collection of `fileFolderImageList`.

The application loads its home directory into the `ListView` when it first loads.

Fig. 15.31 | `ListView` displaying files and folders. (Part 7 of 8.)

15.11 TabControl Control

- The **TabControl** creates tabbed windows, such as those we have seen in Visual Studio (Fig. 15.32)

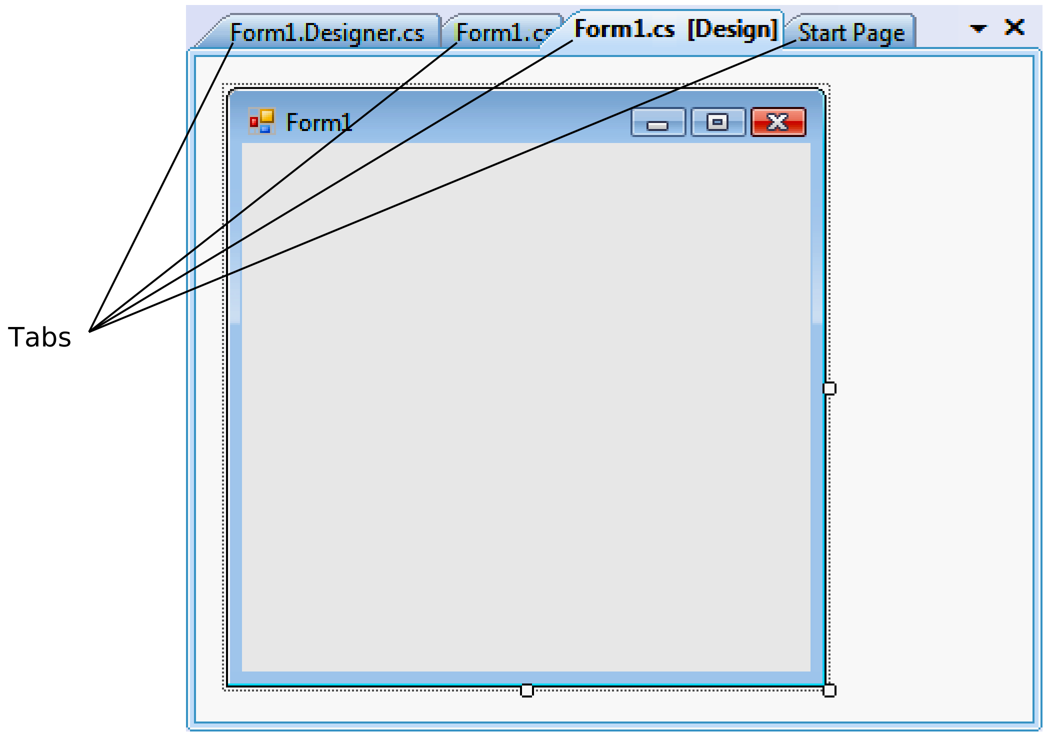


Fig. 15.32 | Tabbed windows in Visual Studio.

15.11 TabControl Control (Cont.)

- TabControls contain **TabPage** objects, which are similar to Panels.
- First add controls to the TabPage objects, then add the TabPages to the TabControl.

```
myTabPage.Controls.Add( myControl );
myTabControl.TabPages.Add( myTabPage );
```

- We can use method AddRange to add an array of TabPages or controls to a TabControl or TabPage.

15.11 TabControl Control (Cont.)

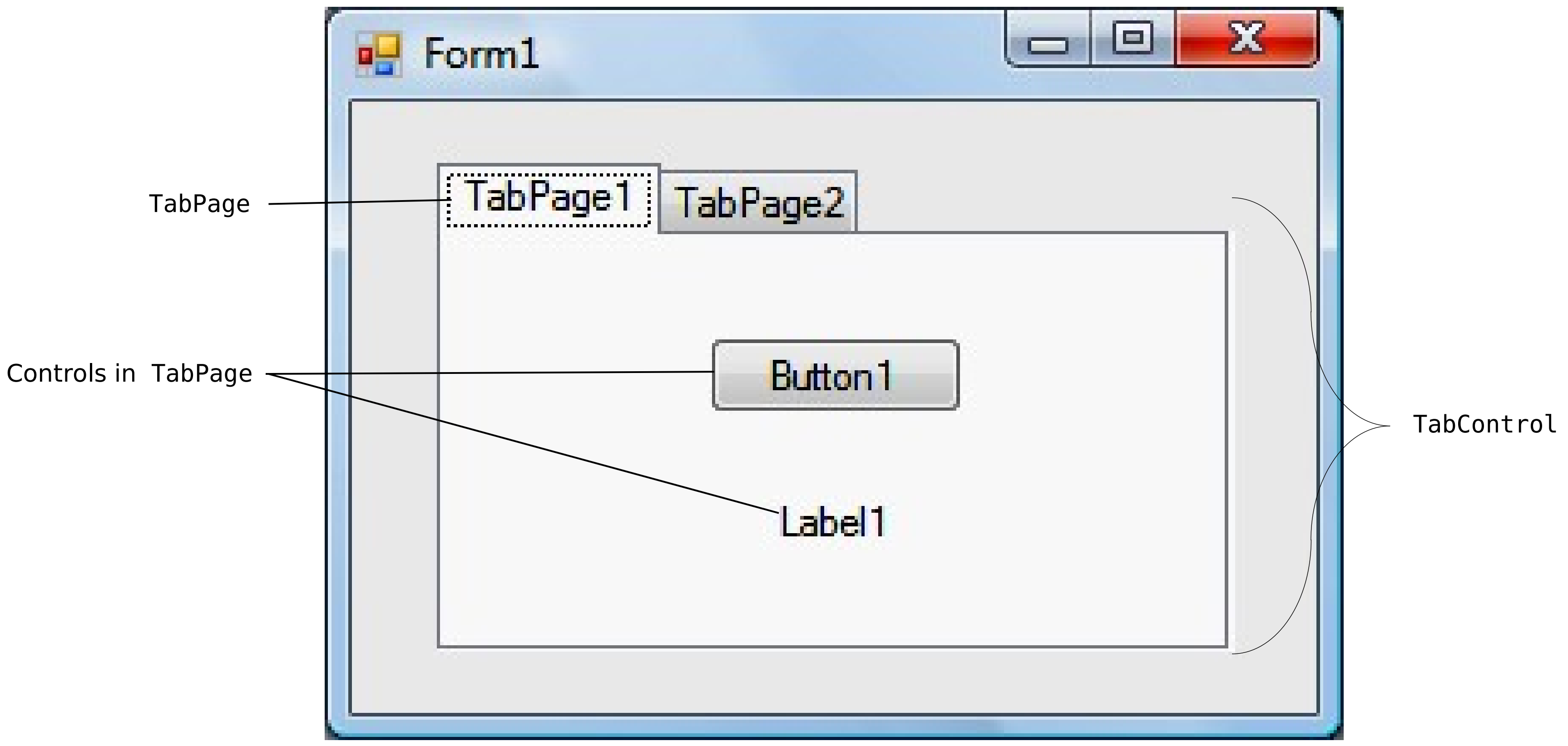
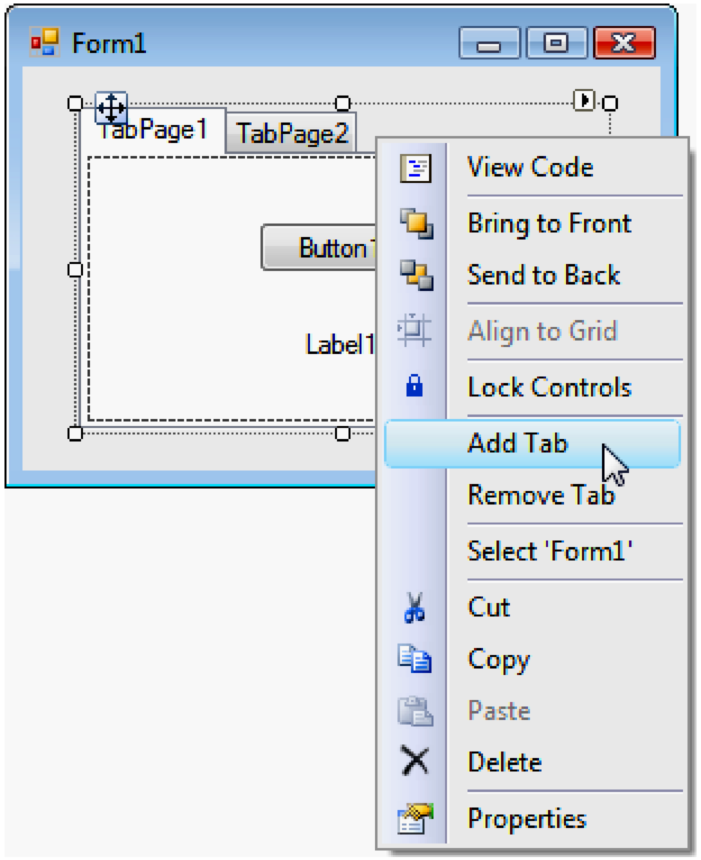


Fig. 15.33 | TabControl with TabPages example.

15.11 TabControl Control (Cont.)

- Add `TabControls` visually by dragging and dropping them onto a `Form` in **Design** mode.
- To add `TabPages` in **Design** mode, right click the `TabControl` and select **Add Tab** (Fig. 15.34).
- To select a `TabPage`, click the control area underneath the tabs.

15.11 TabControl Control (Cont.)



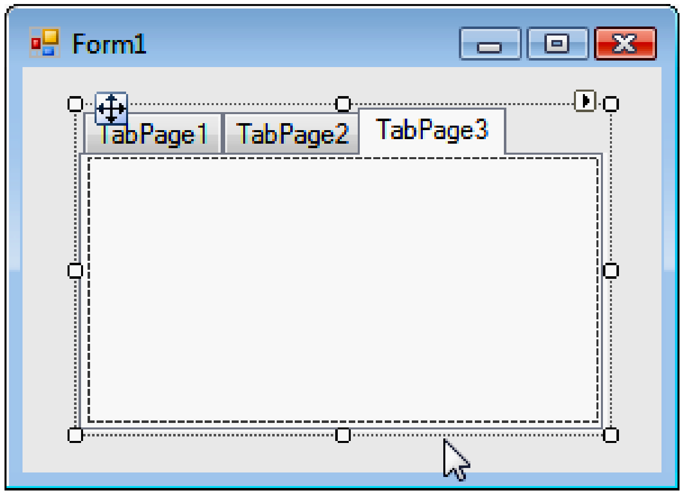


Fig. 15.34 | TabPages added to a TabControl.

15.11 TabControl Control (Cont.)

TabControl properties and an event	Description
Common Properties	
ImageList	Specifies images to be displayed on tabs.
ItemSize	Specifies the tab size.
Multiline	Indicates whether multiple rows of tabs can be displayed.
SelectedIndex	Index of the selected TabPage.
SelectedTab	The selected TabPage.
TabCount	Returns the number of tab pages.
TabPages	Collection of TabPages within the TabControl.
Common Event	
SelectedIndexChanged	Generated when SelectedIndex changes (i.e., another TabPage is selected).

Fig. 15.35 | TabControl properties and an event.

Outline

UsingTabsForm.cs

(1 of 6)

- Class `UsingTabsForm` (Fig. 15.36) uses a `TabControl` to display various options relating to the text on a label (**Color**, **Size** and **Message**).

```
// Fig15.36: UsingTabsForm.cs
// Using TabControl to display various font settings.
using System;
using System.Drawing;
using System.Windows.Forms;

namespace UsingTabs
{
   // Form uses Tabs and RadioButtons to display various font settings
   public partial class UsingTabsForm : Form
   {
      // constructor
      public UsingTabsForm()
      {
         InitializeComponent();
      } // end constructor
```

Fig. 15.36 | `TabControl` used to display various font settings. (Part 1 of 6.)

Outline

UsingTabsForm.cs

(2 of 6)

```
   //  event  handler  for  Black  RadioButton
   private void blackRadioButton_CheckedChanged(
      object sender, EventArgs e )
   {
      displayLabel.ForeColor = Color.Black; // change color to black
   } // end method blackRadioButton_CheckedChanged

   // event handler for Red RadioButton
   private void redRadioButton_CheckedChanged(
      object sender, EventArgs e )
   {
      displayLabel.ForeColor = Color.Red; // change color to red
   } // end method redRadioButton_CheckedChanged

   // event handler for Green RadioButton
   private void greenRadioButton_CheckedChanged(
      object sender, EventArgs e )
   {
      displayLabel.ForeColor = Color.Green; // change color to green
   } // end method greenRadioButton_CheckedChanged
```

Fig. 15.36 | `TabControl` used to display various font settings. (Part 2 of 6.)

Outline

UsingTabsForm.cs

(3 of 6)

```
         // change font size to 12
         displayLabel.Font = new Font( displayLabel.Font.Name, 12 );
      } // end method size12RadioButton_CheckedChanged

      // event handler for 16-point RadioButton
      private void size16RadioButton_CheckedChanged(
         object sender, EventArgs e )
      {
         // change font size to 16
         displayLabel.Font = new Font( displayLabel.Font.Name, 16 );
      } // end method size16RadioButton_CheckedChanged

      // event handler for 20-point RadioButton
      private void size20RadioButton_CheckedChanged(
         object sender, EventArgs e )
      {
```

Fig. 15.36 | `TabControl` used to display various font settings. (Part 3 of 6.)

Outline

UsingTabsForm.cs

(4 of 6)

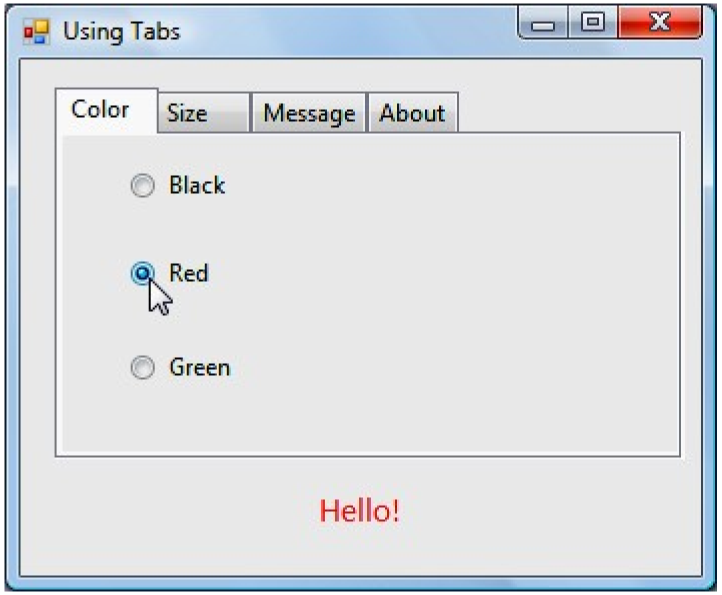
```
            displayLabel.Text = "Hello!"; // change text to Hello!
        } // end method helloRadioButton_CheckedChanged

        // event handler for Goodbye! RadioButton
        private void goodbyeRadioButton_CheckedChanged(
            object sender, EventArgs e )
        {
            displayLabel.Text = "Goodbye!"; // change text to Goodbye!
        } // end method goodbyeRadioButton_CheckedChanged
    } // end class UsingTabsForm
} // end namespace UsingTabs
```

Fig. 15.36 | `TabControl` used to display various font settings. (Part 4 of 6.)

UsingTabsForm.cs

(5 of 6)

a) Color tab



b) Size tab

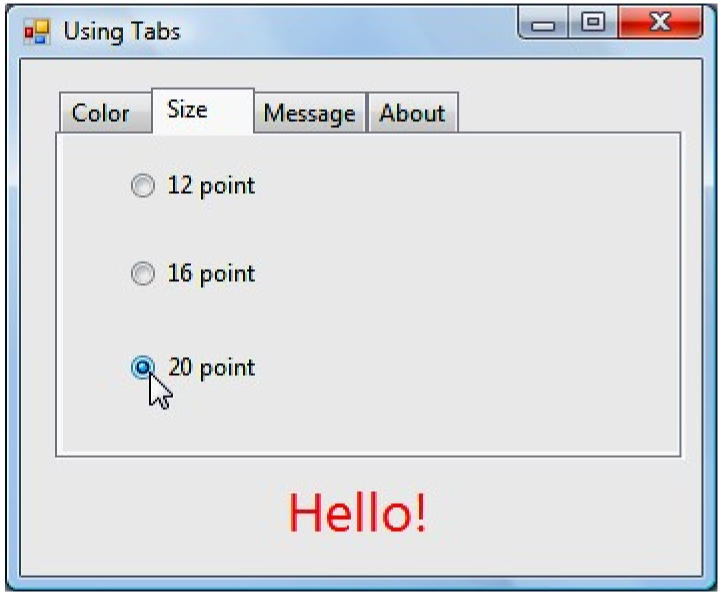


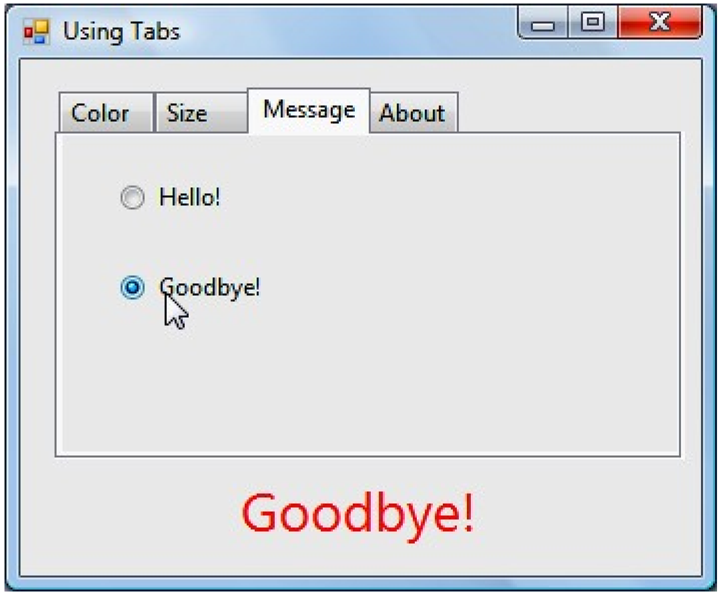
Fig. 15.36 | `TabControl` used to display various font settings. (Part 5 of 6.)

Outline

`UsingTabsForm.cs`

(6 of 6)

c) Message tab

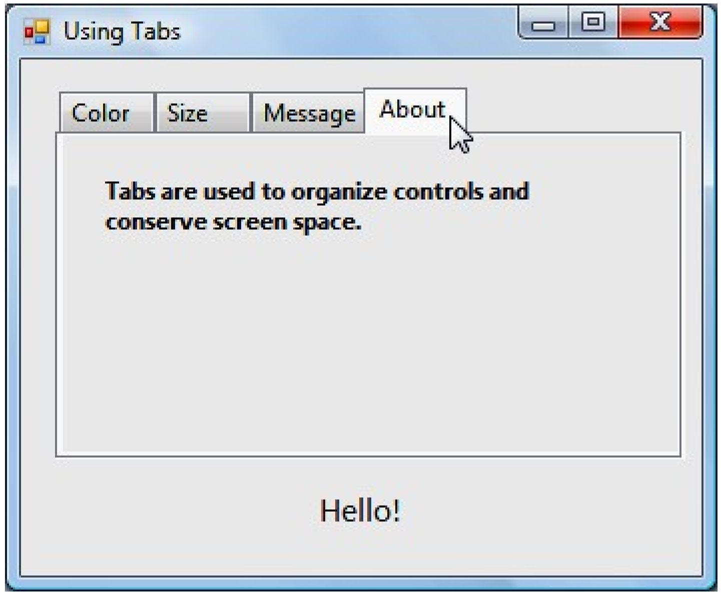


○ Hello!

◉ Goodbye!

Goodbye!

d) About tab



Tabs are used to organize controls and conserve screen space.

Hello!

Fig. 15.36 | `TabControl` used to display various font settings. (Part 6 of 6.)

15.12 Multiple Document Interface (MDI) Windows

- Many complex applications are **multiple document interface (MDI)** programs, which allow users to edit multiple documents at once.
- An MDI program's main window is called the **parent window**, and each window inside the application is referred to as a **child window**.
- Figure 15.37 depicts a sample MDI application with two child windows.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

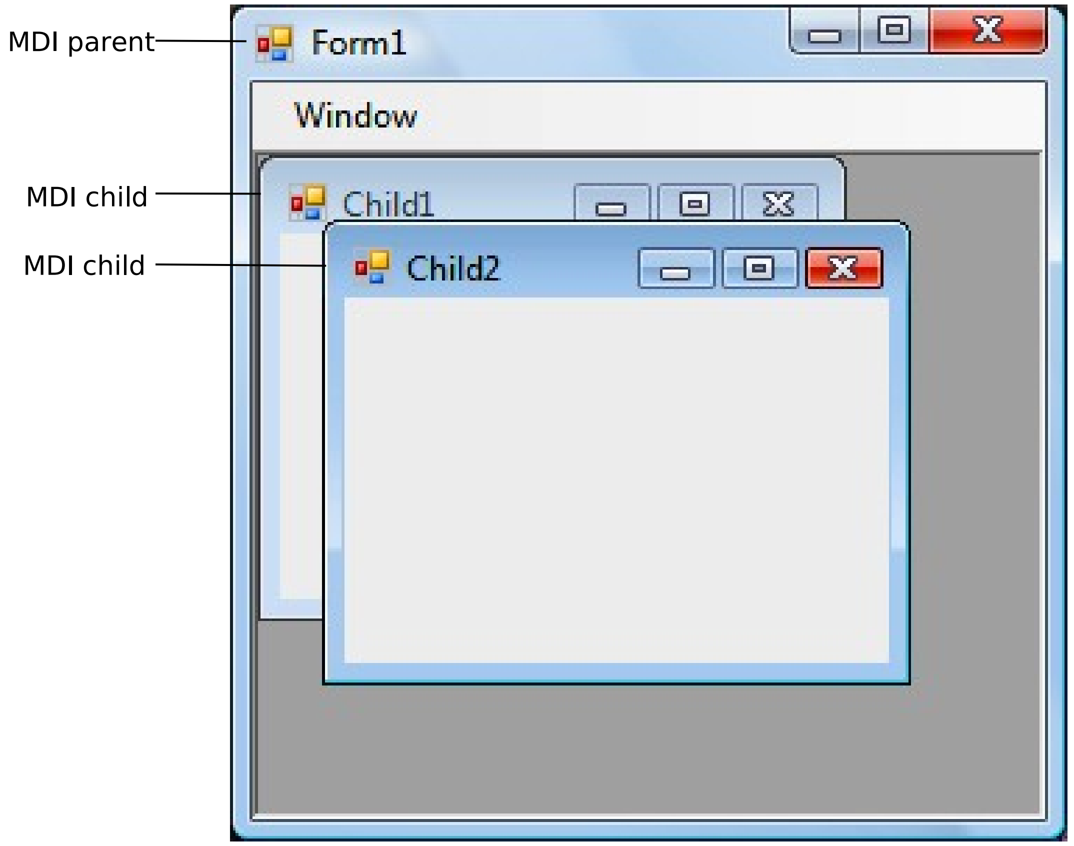


Fig. 15.37 | MDI parent window and MDI child windows.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

- To create an MDI `Form`, create a new `Form` and set its **`IsMdiContainer`** property to `true` (Fig. 15.38).

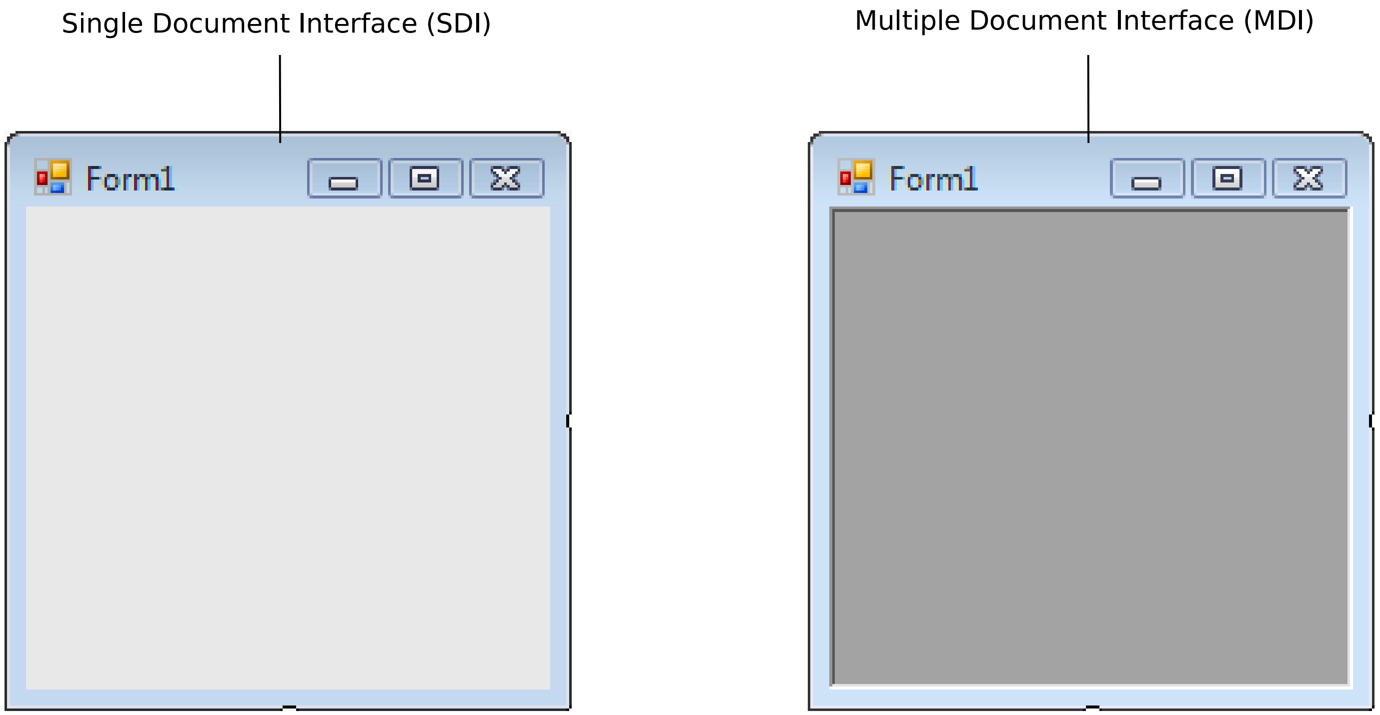

Single Document Interface (SDI)
Multiple Document Interface (MDI)
Form1
Form1

Fig. 15.38 | SDI and MDI forms.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

- Right click the project in the **Solution Explorer**, select **Project > Add Windows Form...** and name the file.
- Set the `Form`'s **`MdiParent`** property to the parent `Form` and call the child `Form`'s `Show` method.
- *ChildFormClass childForm* `= New` *ChildFormClass*`();`

  childForm`.MdiParent =` *parentForm;*

  childForm`.Show();`
- In most cases, the parent `Form` creates the child, so the *parentForm* reference is `this`.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

MDI Form properties, a method and an event	Description
Common MDI Child Properties	
IsMdiChild	Indicates whether the Form is an MDI child.
MdiParent	Specifies the MDI parent Form of the child.
Common MDI Parent Properties	
ActiveMdiChild	Returns the Form that is the currently active MDI child.
IsMdiContainer	Indicates whether a Form can be an MDI parent.
MdiChildren	Returns the MDI children as an array of Forms.
Common Method	
LayoutMdi	Determines the display of child forms on an MDI parent.
Common Event	
MdiChildActivate	Generated when an MDI child is closed or activated.

Fig. 15.39 | MDI parent and MDI child properties, method and event.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

- Figure 15.40 shows two images: one containing two minimized child windows and a second containing a maximized child window.

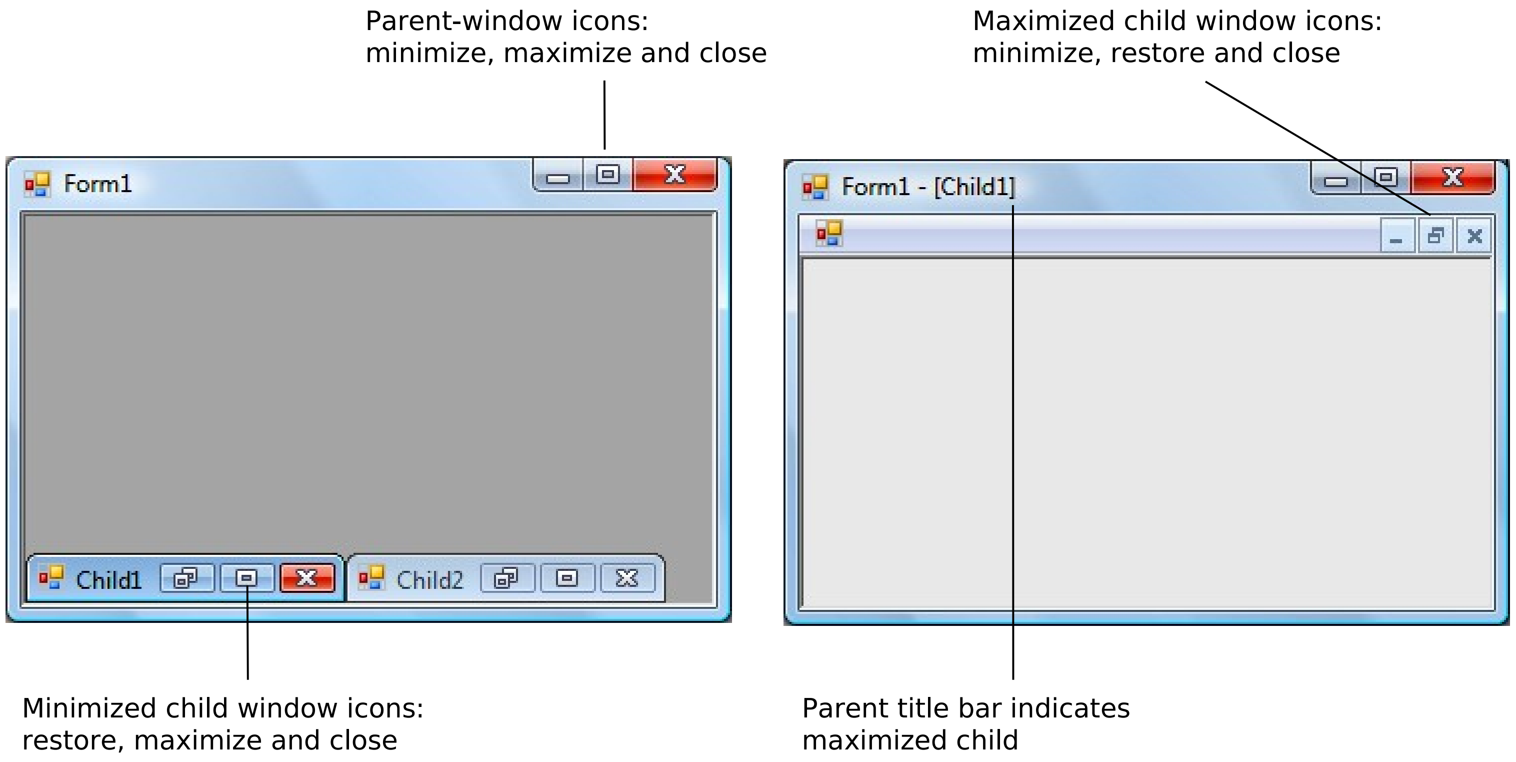

Parent-window icons: minimize, maximize and close
Maximized child window icons: minimize, restore and close
Form1
Child1
Child2
Form1 - [Child1]
Minimized child window icons: restore, maximize and close
Parent title bar indicates maximized child

Fig. 15.40 | Minimized and maximized child windows.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

- Property `MdiWindowListItem` of class `MenuStrip` specifies which menu, if any, displays a list of open child windows.
- When a new child window is opened, an entry is added to the list (Fig. 15.41).

15.12 Multiple Document Interface (MDI) Windows (Cont.)

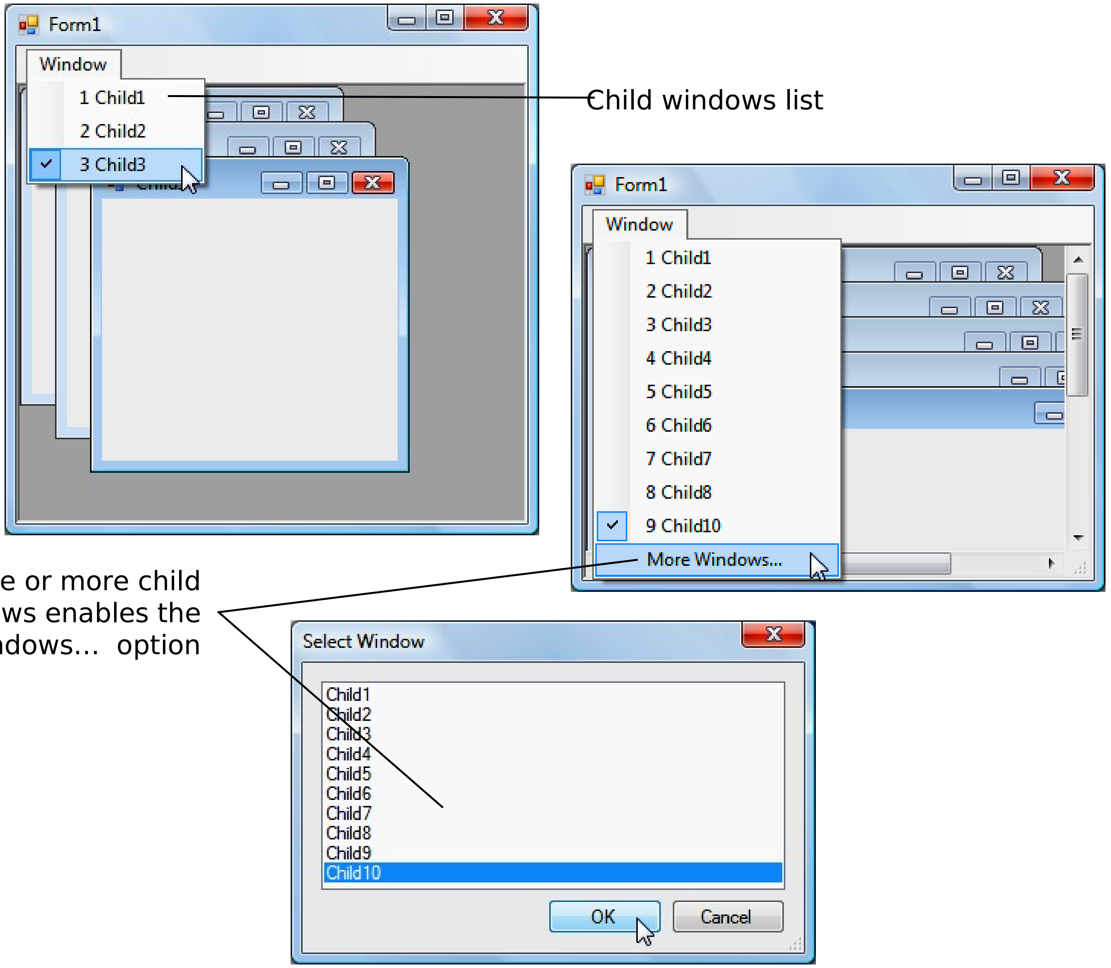


Fig. 15.41 | `MenuItem` property `MdiList` example.

15.12 Multiple Document Interface (MDI) Windows (Cont.)

- MDI containers allow you to organize the placement of its child windows.
- Method `LayoutMdi` takes a value of the `MdiLayout` enumeration (Fig. 15.42).

15.12 Multiple Document Interface (MDI) Windows (Cont.)

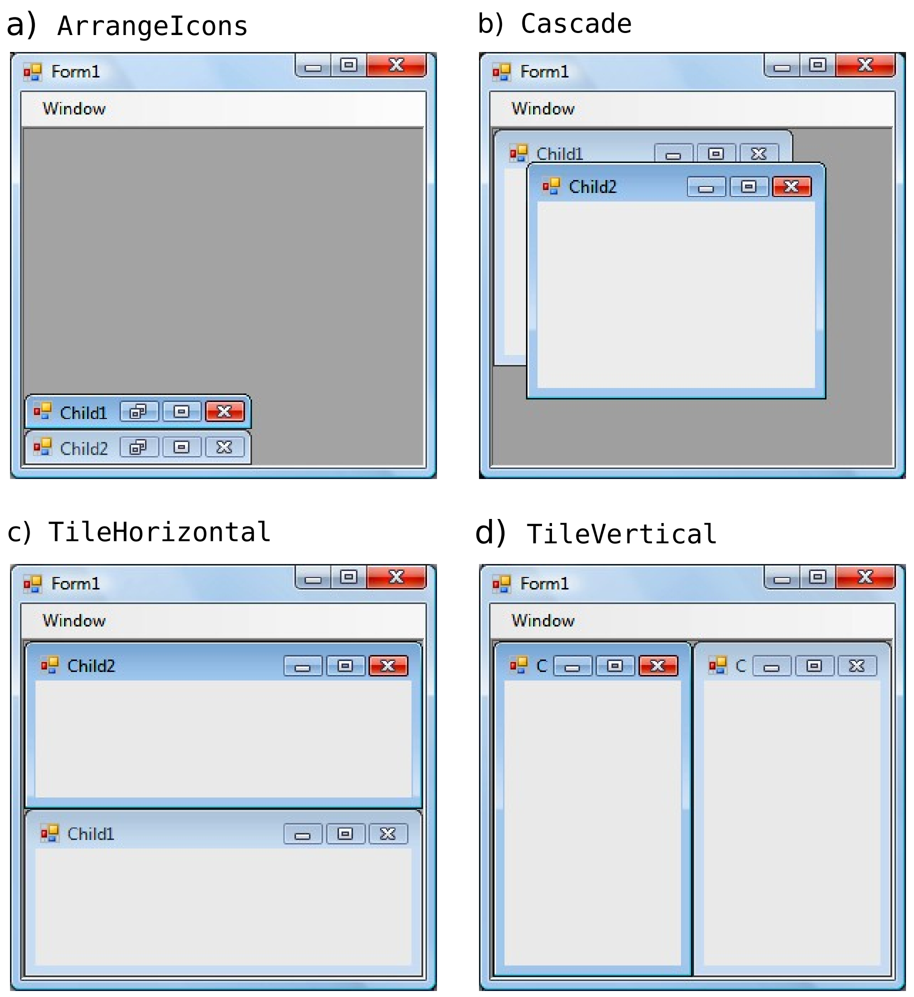


Fig. 15.42 | `MdiLayout` enumeration values.

- Class `UsingMDIForm` (Fig. 15.43) demonstrates MDI windows.

```
// Fig15.43: UsingMDIForm.cs
// Demonstrating use of MDI parent and child windows.
using System;
using System.Windows.Forms;

namespace UsingMDI
{
   // Form demonstrates the use of MDI parent and child windows
   public partial class UsingMDIForm : Form
  {
      // constructor
     public UsingMDIForm()
     {
        InitializeComponent();
     } // end constructor
```

Fig. 15.43 | MDI parent-window class. (Part 1 of 6.)

Outline

UsingMDIForm.cs

(2 of 6)

```
      // create Visual C# image window
      private void csToolStripMenuItem_Click(
         object sender, EventArgs e )
      {
         // create new child
         ChildForm child = new ChildForm(
            "Visual C# 2008 How to Progra, "vcs2008htp" );
         child.MdiParent = this; // set parent
         child.Show(); // display child
      } // end method child1ToolStripMenuItem_Click

      // create Visual C++ image window
      private void cppToolStripMenuItem_Click(
         object sender, EventArgs e )
      {
         // create new child
         ChildForm child = new ChildForm(
            "Visual C++ 2008 How to Program", "vcpp2008htp" );
         child.MdiParent = this; // set parent
         child.Show(); // display child
```

Adding a new child `Form` with certain properties.

Adding a new child `Form` with certain properties.

Fig. 15.43 | MDI parent-window class. (Part 2 of 6.)

Outline

UsingMDIForm.cs

(3 of 6)

```
   } // end method child2ToolStripMenuItem_Click
      // create new child
      Child child = new ChildForm(
         "Visual Basic 2008 How to Program", "vb2008htp" );
      child.MdiParent = this; // set parent
      child.Show(); // display child
   } // end method child3ToolStripMenuItem_Click

   // exit application
   private void exitToolStripMenuItem_Click(
      object sender, EventArgs e )
   {
      Application.Exit();
   } // end method exitToolStripMenuItem_Click

   // set Cascade layout
```

Adding a new child `Form` with certain properties.

Fig. 15.43 | MDI parent-window class. (Part 3 of 6.)

Outline

UsingMDIForm.cs

(4 of 6)

```
      // set TileHorizontal layout
      private void tileHorizontalToolStripMenuItem_Click(
         object sender, EventArgs e )
      {
         this.LayoutMdi( MdiLayout.TileHorizontal );
      } // end method tileHorizontalToolStripMenuItem

      // set TileVertical layout
      private void tileVerticalToolStripMenuItem_Click(
         object sender, EventArgs e )
      {
         this.LayoutMdi( MdiLayout.TileVertical );
      } // end method tileVerticalToolStripMenuItem_Click
   } // end class UsingMDIForm
} // end namespace UsingMDI
```

Setting the layout of child `Forms`.

Setting the layout of child `Forms`.

Setting the layout of child `Forms`.

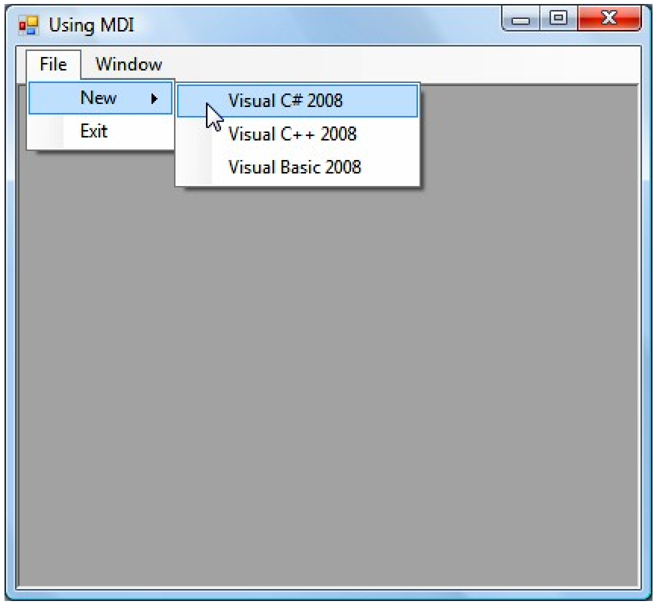
Fig. 15.43 | MDI parent-window class. (Part 4 of 6.)

Outline

UsingMDIForm.cs

(5 of 6)

a) Creating a child window



b) Viewing the child window

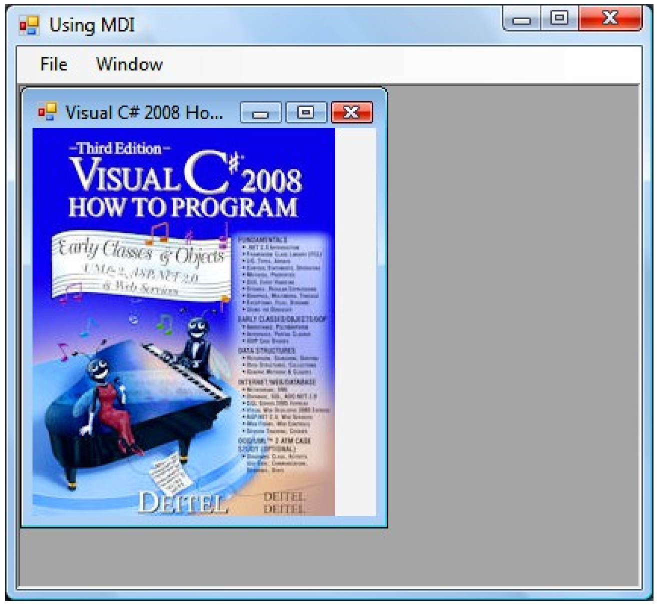


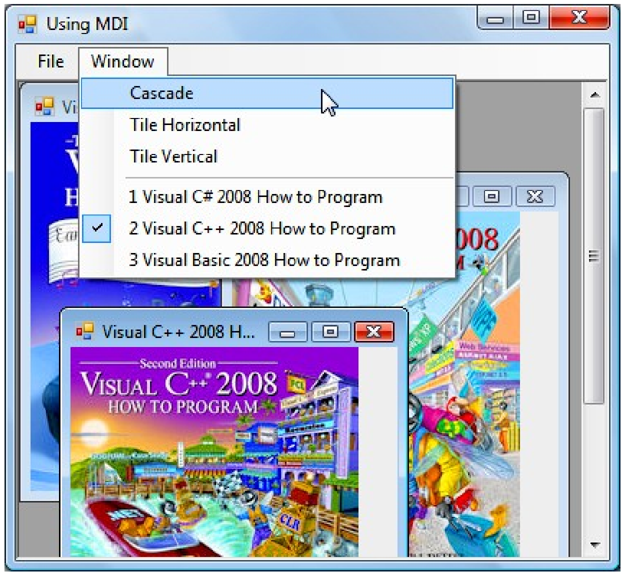
Fig. 15.43 | MDI parent-window class. (Part 5 of 6.)

Outline

`UsingMDIForm.cs`

(6 of 6)

c) Changing child window organization


Using MDI
File
Window
Cascade
Tile Horizontal
Tile Vertical
1 Visual C# 2008 How to Program
✓ 2 Visual C++ 2008 How to Program
3 Visual Basic 2008 How to Program
Visual C++ 2008 H...

d) Child windows in Cascade view

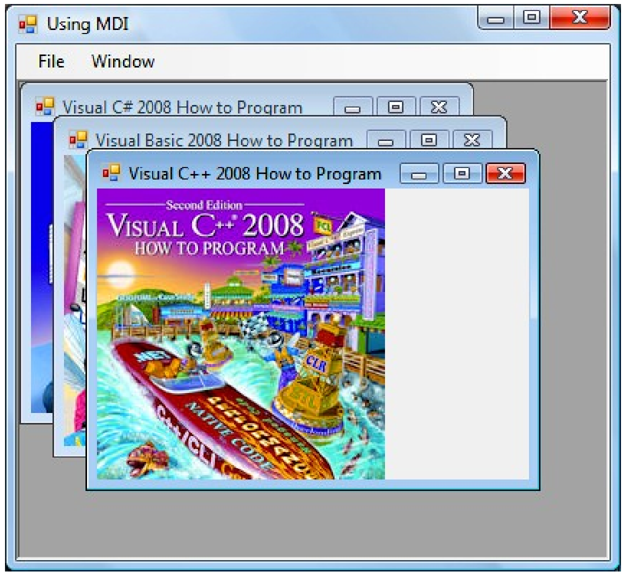

Using MDI
File
Window
Visual C# 2008 How to Program
Visual Basic 2008 How to Program
Visual C++ 2008 How to Program

Fig. 15.43 | MDI parent-window class. (Part 6 of 6.)

Outline

- Define the MDI child class by right clicking the project in the **Solution Explorer** and selecting **Add > Windows Form...**.
- Name the new class `ChildForm` (Fig. 15.44).

`ChildForm.cs`

(1 of 2)

```
// Fig15.44: ChildForm.cs
// Child window of MDI parent.
using System;
using System.Drawing;
using System.Windows.Forms;
using System.IO;

namespace UsingMDI
{
   public partial class ChildForm : Form
   {
      public ChildForm( string title, string resourceName )
      {
         // Required for Windows Form Designer support
         InitializeComponent();
```

Fig. 15.44 | MDI child `ChildForm`. (Part 1 of 2.)

Outline

ChildForm.cs

(2 of 2)

```

            Text = title; // set title text

         // set image to display in pictureBox
            displayPictureBox.Image =
         ( Image ) ( Properties.Resources.ResourceManager.GetObject(
            resourceName );
      } // end constructor
   } // end class ChildForm
} //  end namespace UsingMDI
```

Setting the title-bar text.

Retrieving the specified image resource and displaying it.

Fig. 15.44 | MDI child `ChildForm`. (Part 2 of 2.)