3

Windows Forms Controls: ListView, TreeView, ImageList, PictureBox, Panel, GroupBox, and TabControl

In Depth

You are already familiar with the commonly used Windows Forms controls, such as Button, TextBox, Label, and RadioButton. Some other controls are also used with Windows Forms, such as ListView, TreeView, ImageList, PictureBox, Panel, GroupBox, and TabControl. These controls facilitate you to handle the display and grouping of related controls or images in a Windows form. For example, the ListView and TreeView controls are used to create different types of views, the ImageList controls are used to store images, a PictureBox control is used to display images on a Windows form, and the Panel and GroupBox controls are used to group similar controls together. For example, you may use the Panel and GroupBox controls to group the controls that accept different types of inputs, such as professional and personal data, from a user.

In this chapter, we start by describing how to use the ListView and TreeView controls to create different types of views, and the ImageList and PictureBox controls to store and display images. Next, you learn about the Panel and GroupBox controls, which are used to group the other controls, such as radio buttons and check boxes. Finally, you learn about the TabControl control, which is used to display different groups of controls to perform different functionalities on a single Windows form.

Let's start with exploring the ListView control.

The ListView Control

A ListView control is used to display lists of items, just as tree views are used to display node hierarchies that are similar to the folder hierarchy on the hard disk of a computer. ListView controls are supported by the `ListView` class, which has the following class hierarchy:

```
System.Object
  System.MarshalByRefObject
        System.ComponentModel.Component
                System.Windows.Forms.Control
                        System.Windows.Forms.ListView
```

You can see a ListView control in the left pane of the Windows Explorer, where folders and files are displayed. Figure 3.1 shows a ListView control:

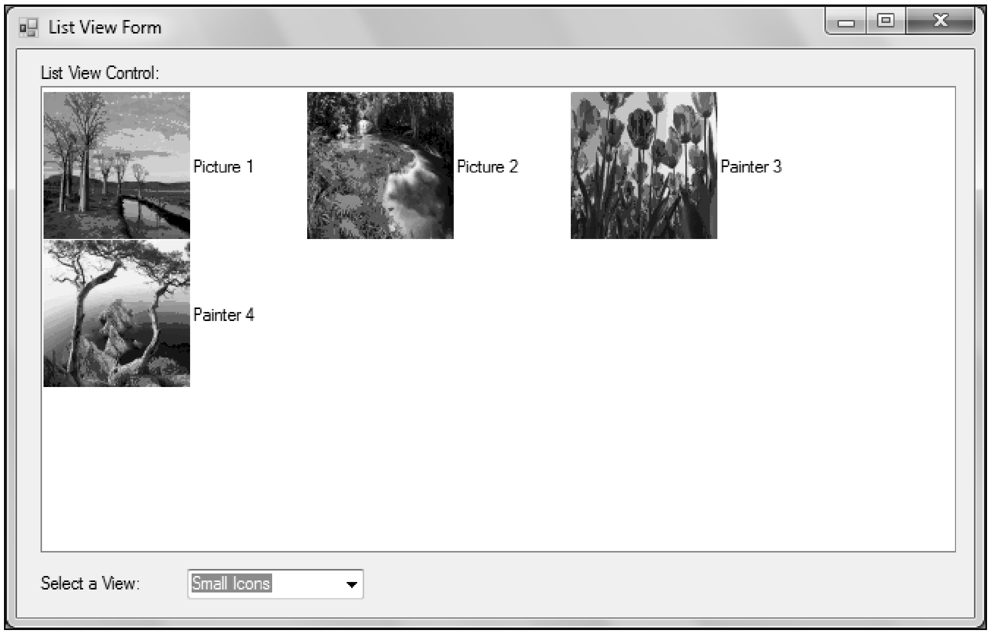


Figure 3.1: Displaying a ListView Control

ListView controls display their items in the following five view modes:

- `View.LargeIcon`—Displays items using large icons (large icons are of 32×32 pixels in size) next to the item text
- `View.SmallIcon`—Displays items using small icons (small icons are of 16×16 pixels in size) next to the item text
- `View.List`—Displays small icons in one column
- `View.Details`—Displays items in multiple columns, with column headers and fields
- `View.Tile`—Displays items as full-sized icons with item labels and sub-item information

The `SelectedItems` property of the ListView control contains a list of items to be displayed by the control, and displays a list of items that are currently selected in the control. Note that you can select multiple items, if the `MultiSelect` property is set to `True`. In addition, a ListView control can display checkboxes beside the items if the `CheckBoxes` property is set to `True`.

You can use the `SelectedIndexChanged` event to handle item selections, and `ItemCheck` events to handle checkmark events. The `Activation` property specifies the action that the user must take to activate an item in the list. The options of the `Activation` property could be `OneClick`, `TwoClick`, or `Standard`. The `OneClick` option requires a single click to activate the item. The `TwoClick` option requires the user to double click (single click changes the color of the item text) the item to activate it. The Standard option requires the user to double click the item to activate it, but in this case, the item does not change its appearance on single click. You can also sort the items in a ListView control with the `Sorting` property.

Table 3.1 describes a list of public properties of the `ListView class`:

Table 3.1: Noteworthy Public Properties of the ListView Class

Property	Description
`AutoArrange`	Obtains or specifies if items are automatically arranged, using the `Alignment` property
`BackColor`	Obtains or specifies the background color of a ListView control
`CheckBoxes`	Obtains or specifies if every item should show a check box
`CheckedIndices`	Obtains or specifies the indices of currently checked items
`CheckedItems`	Obtains or specifies the currently checked items
`Columns`	Obtains or specifies a collection of all column headers that are in the control
`GridLines`	Obtains or specifies whether grid lines are drawn between items and their sub-items
`HideSelection`	Obtains or specifies whether selected items should be hidden when a ListView control is not highlighted
`Items`	Obtains or specifies list items in the control
`MultiSelect`	Obtains or specifies whether multiple items can be selected
`RightToLeftLayout`	Obtains or specifies a value indicating whether the control is laid out from right to left
`Scrollable`	Obtains or specifies if scroll bars should be added to the control when enough space is not available to display items in the control
`SelectedIndices`	Obtains or specifies the indices of the selected items
`SelectedItems`	Obtains or specifies the selected items
`View`	Obtains or specifies the current view mode

Table 3.2 describes a public method of the `ListView` class:

Table 3.2: Noteworthy Public Method of the ListView Class

Method	Description
`Clear()`	Removes all items from a ListView control

Table 3.3 describes a list of public events of the `ListView` class:

Table 3.3: Noteworthy Public Events of the ListView Class

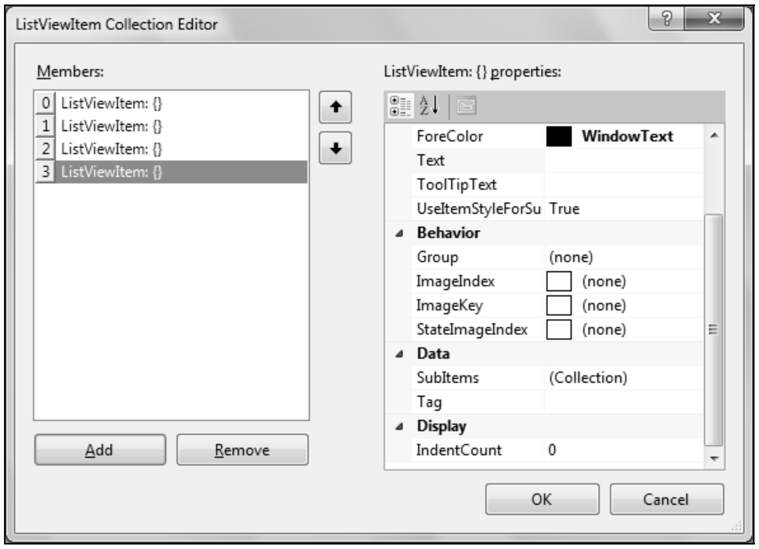
Event	Description
`ColumnClick`	Occurs when a column is clicked
`ItemCheck`	Occurs when the check state (when check box is checked or unchecked) of an item changes or the `CheckBoxes` property is set to True or False
`ItemChecked`	Occurs when the checked state of an item changes
`SelectedIndexChanged`	Occurs when the selected index changes
`TextChanged`	Occurs when the Text property of a ListView control changes

In the following sections, you learn about the following aspects of the ListView control:

- Adding items to the ListView control
- Setting the view of the ListView control
- Removing items from the ListView control

Adding Items to the ListView Control

In a ListView control, you can add columns and items through its Items property. The Items property opens the ListViewItem Collection Editor dialog box, as shown in Figure 3.2:



ForeColor	WindowText
Text	
ToolTipText	
UseItemStyleForSu	True
Behavior	
Group	(none)
ImageIndex	(none)
ImageKey	(none)
StateImageIndex	(none)
Data	
SubItems	(Collection)
Tag	
Display	
IndentCount	0

Figure 3.2: Displaying the ListViewItem Collection Editor Dialog Box

In the ListViewItem Collection Editor dialog box, you can add or remove an item by using the Add and Remove buttons, respectively.

The following code snippet shows how to add columns, items, and subitems in a ListView control:

```
    // Add columns
    listView1.Columns.Add("Title",-2,HorizontalAlignment.Left);
```

```
listView1.Columns.Add("Painter",-2,HorizontalAlignment.Center);
listView1.Columns.Add("Price",-2,HorizontalAlignment.Right);

// Add items
ListViewItem item1 = new ListViewItem("Picture 1");
item1.SubItems.Add("Charu");
item1.SubItems.Add("1111.53");

item1.ImageIndex = 0;

ListViewItem item2 = new ListViewItem("Picture 2");
item2.SubItems.Add("Kamlesh");
item2.SubItems.Add("5555.99");

item2.ImageIndex = 1;

// Add the items to the ListView.
listView1.Items.AddRange(
new ListViewItem[] {item1,item2}
);
```

Setting the View of the ListView Control

You can set a view to display the items in a ListView control programmatically. The following code snippet shows how to set a view for the items of a ListView control:

```
 if (comboBox1.SelectedIndex == 0)
 {
         listView1.View = View.LargeIcon;
 }
 else if (comboBox1.SelectedIndex==1)
 {
         listView1.View = View.Details;
 }
 else if (comboBox1.SelectedIndex == 2)
 {
         listView1.View = View.SmallIcon;
 }
 else
 {
         listView1.View = View.List;
 }
```

Removing Items from the ListView Controls

You can remove an item from a ListView control when the item is no longer required. The following code snippet is used to remove a selected item from a ListView control:

```
private void button2_Click(object sender, EventArgs e)
{
 listView1.Items.Remove(listView1.SelectedItems[0]);
}
```

Next, let's learn about the TreeView control.

The TreeView Control

A TreeView control displays a hierarchy of nodes, which may include child nodes. The class hierarchy of the `TreeView` class is as follows:

```
System.Object
 System.MarshalByRefObject
         System.ComponentModel.Component
```

```
                    System.Windows.Forms.Control
                           System.Windows.Forms.TreeView
```

Windows Explorer, which uses a tree view in its left pane to display the hierarchy of folders stored on the hard disk of a computer, is an example of the implementation of the TreeView control. Figure 3.3 shows a TreeView control:

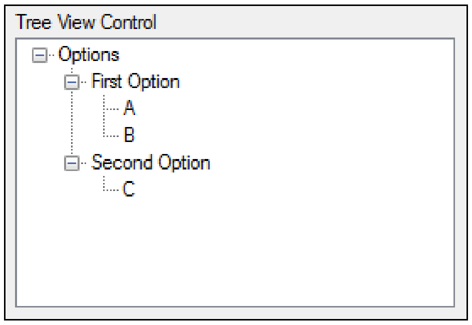


Figure 3.3: Displaying a TreeView Control

A TreeView control can also display the hierarchy of nodes with checkboxes placed next to the nodes of the tree structure if the TreeView's `CheckBoxes` property is set to `True`. You can then check or uncheck the nodes to select or deselect them by setting the node's `Checked` property to `True` or `False`.

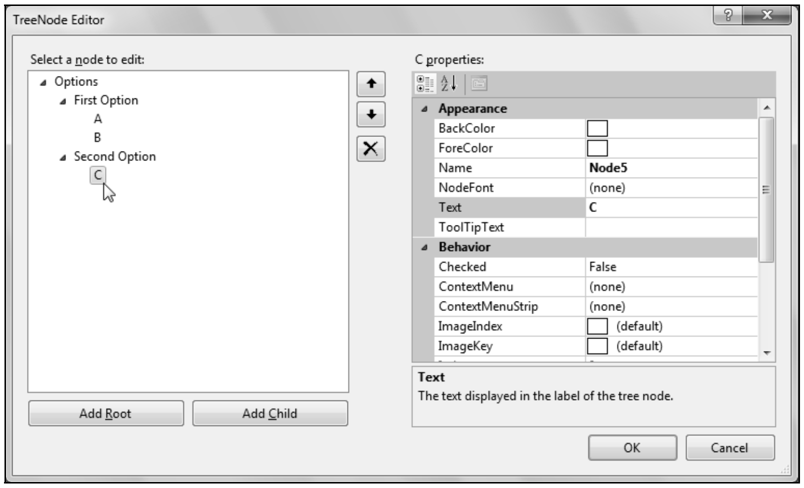
The main properties of tree views are `Nodes` and `SelectedNode`. The `Nodes` property contains a list of nodes in the tree view, and the `SelectedNode` property specifies the currently selected node. Nodes themselves are supported by the `TreeNode` class.

A node can be a parent of other child nodes, called `TreeNode` objects. You can add, remove, or clone a `TreeNode object, which in turn can also contain` a collection of other `TreeNode` objects. The `FullPath` property specifies the nodes in terms of their absolute locations.

The class hierarchy of the `TreeNode` class is:

```
    System.Object
     System.MarshalByRefObject
              System.Windows.Forms.TreeNode
```

You can add nodes to a TreeView control through the `Nodes` property. The Nodes property in the `Properties` window enables you to open the `TreeNode` Editor dialog box, as shown in Figure 3.4:



Appearance	
BackColor	
ForeColor	
Name	Node5
NodeFont	(none)
Text	C
ToolTipText	
Behavior	
Checked	False
ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	(default)
ImageKey	(default)

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

Figure 3.4: Displaying the TreeNode Editor Dialog Box

In Figure 3.4, you can see the Add Root and Add Child buttons, which are used to add root and child nodes.

Table 3.4 describes a list of public properties of the `TreeNode` class:

Table 3.4: Noteworthy Public Properties of the TreeNode Class

Property	Description
`BackColor`	Obtains or specifies the background color of a tree node
`Checked`	Obtains or specifies whether a tree node is checked
`ContextMenu`	Obtains the shortcut menu associated with the concerned tree node
`ContextMenuStrip`	Obtains or specifies the shortcut menu associated with the current tree node
`FirstNode`	Obtains the first child tree node in the tree node collection
`FullPath`	Obtains the path from the root node to the current node
`Index`	Obtains the location of a node in the node collection
`IsExpanded`	Specifies whether or not the node is expanded
`IsSelected`	Specifies whether or not the node is selected
`IsVisible`	Obtains a value specifying whether or not a node is visible
`LastNode`	Obtains the last child node in the tree hierarchy
`Level`	Obtains the zero-based depth of a tree node in a TreeView control
`Name`	Obtains or specifies the name of a tree node
`NextNode`	Obtains the next sibling node
`Nodes`	Obtains a collection of the objects assigned to the current tree node
`Parent`	Obtains the parent node of the current node
`PrevNode`	Obtains the previous sibling node
`Tag`	Obtains or specifies the data displayed in the label of a tree node
`Text`	Obtains or specifies the text for a node's label
`ToolTipText`	Obtains or specifies the text that appears when the mouse pointer is moved over a TreeNode
`TreeView`	Obtains a node's parent tree view

Table 3.5 describes a list of public methods of the `TreeNode` class:

Table 3.5: Noteworthy Public Methods of the TreeNode Class

Method	Description
`Collapse()`	Collapses a node
`Expand()`	Expands a node
`ExpandAll()`	Expands all the child nodes of a node
`GetNodeCount()`	Gets the number of child tree nodes
`Remove()`	Removes the current node
`Toggle()`	Toggles a tree node between the expanded and collapsed states
`ToString()`	Returns a String that represents the current object

You can set the text for each tree node label by setting a `TreeNode` object's `Text` property. In addition, you can display images next to the tree nodes by assigning an `ImageList` to the `ImageList` property of the parent

`TreeView` control. In addition, you can assign an image to a node by referencing its index value in the `ImageList` property. Specifically, you can set the `ImageIndex` property to the index value of the image you want to display when the `TreeNode` object is in an unselected state; and set the `SelectedImageIndex` property to the index value of the image you want to display when the `TreeNode` object is selected.

A TreeView control also supports various properties for navigating through the nodes of the tree structure. For example, you can use the `FirstNode`, `LastNode`, `NextNode`, `PrevNode`, `NextVisibleNode`, and `PrevVisibleNode` properties for navigation.

A TreeView control is all about showing node hierarchies—the user can expand a node (showing its children) by clicking the plus sign (+) displayed next to it, or collapse a node by clicking the minus sign (–) next to it. You can implement the same programmatically by using the `Expand()` method to expand a single node, the `ExpandAll()` method to expand all nodes, and the `Collapse()` or `CollapseAll()` method to collapse the nodes.

A TreeView control can also display checkboxes similar to the ones you use in menu items. You can display checkboxes in a TreeView control by setting its `CheckBoxes` property to `True`. Figure 3.5 shows checkboxes besides the nodes in a TreeView control:

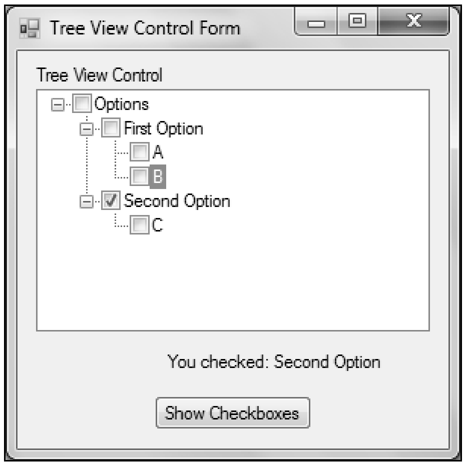


Figure 3.5: Displaying Checkboxes in the TreeView Control

An alternative to display the checkboxes beside the nodes in a TreeView control is shown in the following code snippet:

```
    private void button1_Click(object sender, EventArgs e)
    {
     treeView1.CheckBoxes = true;
    }
```

Table 3.6 describes a list of public properties of the `TreeView` class:

Table 3.6: Noteworthy Public Properties of the TreeView Class

Property	Description
`BackColor`	Obtains or specifies the background color for the TreeView control
`BackgroundImage`	Obtains or specifies the background image for a TreeView control
`BackgroundImageLayout`	Obtains or specifies the layout of the background image of a TreeView control
`BorderStyle`	Obtains the tree view's border style
`CheckBoxes`	Determines whether checkboxes are displayed next to tree nodes
`ForeColor`	Obtains or specifies the current foreground color for the control

Table 3.6: Noteworthy Public Properties of the TreeView Class

Property	Description
`Nodes`	Obtains a collection of tree nodes in a TreeView control
`PathSeparator`	Obtains or specifies the delimiter string used by the tree node in a TreeView control
`RightToLeftLayout`	Obtains or specifies a value indicating whether a TreeView control displays scroll bars when needed
`Scrollable`	Obtains or specifies whether a TreeView control displays scroll bars when needed
`SelectedNode`	Obtains or specifies the tree node currently selected in a TreeView control
`ShowLines`	Obtains or specifies whether lines are drawn between tree nodes in a TreeView control
`ShowNodeToolTips`	Obtains or specifies a value indicating whether the plus-sign (+) and minus-sign (-) buttons are displayed next to tree nodes that contain child tree nodes
`ShowPlusMinus`	Determines whether the plus-sign (+) and minus-sign (-) buttons are displayed beside the tree nodes containing child tree nodes
`ShowRootLines`	Obtains or specifies whether lines are drawn between the tree nodes that are at the root of a TreeView control
`Sorted`	Obtains or specifies whether tree nodes should be sorted
`Text`	Obtains or specifies the text of a TreeView control
`TopNode`	Obtains the first visible tree node in the tree view node
`VisibleCount`	Obtains the number of nodes that can be currently viewed in a TreeView control

Table 3.7 describes a list of public methods of the `TreeView` class:

Table 3.7: Noteworthy Public Methods of the TreeView Class

Method	Description
`CollapseAll()`	Collapses all nodes, including all the child nodes that are in a TreeView control
`ExpandAll()`	Expands all the tree nodes
`GetNodeAt()`	Gets the node that is at the specified location
`GetNodeCount()`	Gets the number of tree nodes (optionally include all in the sub-trees)
`Sort()`	Sorts the items in a TreeView control

Table 3.8 describes a list of public events of the `TreeView` class:

Table 3.8: Noteworthy Public Events of the TreeView Class

Event	Description
`AfterCheck`	Occurs when a node checkbox is checked
`AfterCollapse`	Occurs when a tree node is collapsed
`AfterExpand`	Occurs when a tree node is expanded
`AfterSelect`	Occurs when a tree node is selected
`BeforeCheck`	Occurs before a node checkbox is checked
`BeforeCollapse`	Occurs before a node is collapsed

Table 3.8: Noteworthy Public Events of the TreeView Class

Event	Description
`BeforeExpand`	Occurs before a node is expanded
`BeforeLabelEdit`	Occurs before a node label text is edited
`BeforeSelect`	Occurs before a node is selected
`ItemDrag`	Occurs when the user starts dragging the node
`NodeMouseClick`	Occurs when a user clicks a `TreeNode` element
`NodeMouseDoubleClick`	Occurs when a user double-clicks a `TreeNode`
`NodeMouseHover`	Occurs when a user performs a mouse over action on a TreeNode
`PaddingChanged`	Occurs when the value of the Padding property changes
`RightToLeftLayoutChanged`	Occurs when the value of the RightToLeftLayout property changes
`TextChanged`	Occurs when the value of the Text property changes

Next, let's learn about the ImageList control.

The ImageList Control

An ImageList control is used to store images; i.e., it acts as a kind of image repository. Various controls are used to work with an ImageList control, such as list views, tree views, toolbars, tab controls, checkboxes, buttons, radio buttons, and labels. The class hierarchy of the `ImageList` class is as follows:

```
System.Object
 System.MarshalByRefObject
          System.ComponentModel.Component
                    System.Windows.Forms.ImageList
```

You can associate an ImageList control with its `ImageList` property and specify the images you want to display in the control by using the `ImageIndex` property. The images in an ImageList control are indexed—starting at zero—and you can switch the image displayed in a control at runtime by changing the value of the `ImageIndex` property.

The main property in the ImageList control is `Images`, which contains the images to be used by the control. The `ColorDepth` property of the ImageList control determines the number of colors with which an image is rendered. Note that images are displayed in the size set by the `ImageSize` property. This property is set to 16×16 pixels by default (the size of a small icon) and needs to be changed when the images are loaded into an image list.

Table 3.9 describes a list of public properties of the `ImageList` class:

Table 3.9: Noteworthy Public Properties of the ImageList Class

Property	Description
`ColorDepth`	Obtains the color depth for an ImageList control
`Handle`	Obtains the handle for the current ImageList control
`HandleCreated`	Obtains a value that indicates whether a Win32 handle has been created
`Images`	Obtains an `ImageCollection` object for the current ImageList control
`ImageSize`	Obtains the image size for images in the image list
`Tag`	Obtains an `ImageList` object that contains additional data related to an ImageList control
`TransparentColor`	Obtains the transparent color for the current ImageList control

Table 3.10 describes a list of public method of the `ImageList` class:

Table 3.10: Noteworthy Public Method of the ImageList Class

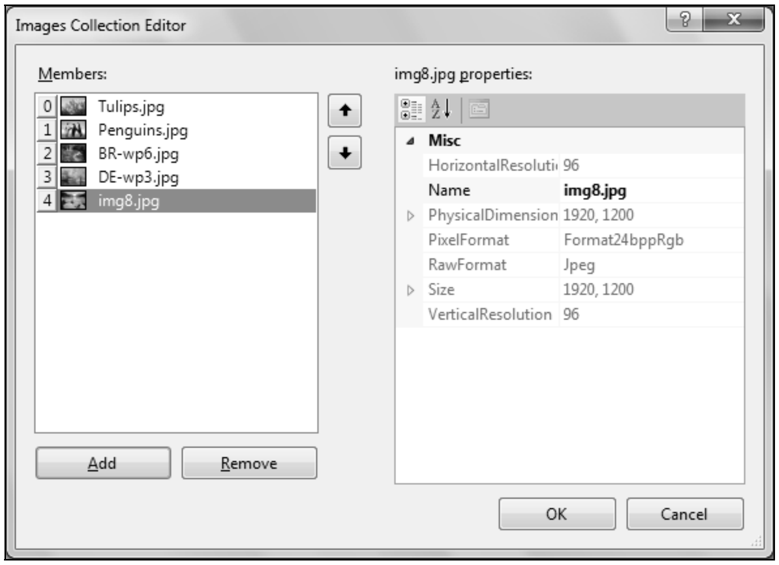
Method	Description
`Draw()`	Draws the given image

Table 3.11 describes a list of public event of the `ImageList` class:

Table 3.11: Noteworthy Public Event of the ImageList Class

Event	Description
`RecreateHandle`	Occurs when the handle for an ImageList control is recreated

To add an image list to the ImageList control, you should use the `Images` property in the Properties window. The `Images` property enables you to open the Images Collection Editor dialog box, as shown in Figure 3.6:



Members:
0 Tulips.jpg
1 Penguins.jpg
2 BR-wp6.jpg
3 DE-wp3.jpg
4 img8.jpg

img8.jpg properties:

Misc	
HorizontalResoluti	96
Name	img8.jpg
PhysicalDimension	1920, 1200
PixelFormat	Format24bppRgb
RawFormat	Jpeg
Size	1920, 1200
VerticalResolution	96

Figure 3.6: Displaying the Images Collection Editor Dialog Box

In Figure 3.6, you can see the Add and Remove buttons used to add or delete an image to an ImageList control.

You can resize the images in the ImageList (the default is 16×16 pixels, the size of a small icon) control by using the `ImageSize` property in the Properties window.

Next, let's learn about the PictureBox control.

The PictureBox Control

A PictureBox control displays images stored in different graphic file formats, such as a bitmap, icon, JPEG, GIF, or other image file types. The `PictureBox` class is derived directly from the `Control` class. The class hierarchy of the PictureBox class is as follows:

```
    System.Object
      System.MarshalByRefObject
            System.ComponentModel.Component
                    System.Windows.Forms.Control
                            System.Windows.Forms.PictureBox
```

To display an image in a PictureBox control, set the `Image` property of the PictureBox control you want to display, either at design time or runtime. You can clip and position an image with the `SizeMode` property. The values of the SizeMode property can be set by using the `PictureBoxSizeMode` enumeration. These values of the SizeMode property are follows:

- **Normal**—Represents standard behavior of a PictureBox control (the upper-left corner of the image is placed at upper-left in the PictureBox)
- **StretchImage**—Allows you to stretch an image in a PictureBox control
- **AutoSize**—Resizes a PictureBox control corresponding to the size of the image
- **CenterImage**—Aligns the image centrally in a PictureBox control

You can also change the size of an image at runtime with the `ClientSize` property. By default, a `PictureBox` control is displayed without any borders, but you can add a standard or three-dimensional border by using the `BorderStyle` property.

Table 3.12 describes a list of public properties of the `PictureBox` class:

Table 3.12: Noteworthy Public Properties of the PictureBox Class

Property	Description
`AllowDrop`	Overrides the Control.AllowDrop property of the PictureBox control
`ErrorImage`	Obtains or specifies an image to display when an error occurs during the image-loading process or if the image load is canceled
`Image`	Obtains or specifies the image that is displayed in a PictureBox control
`ImageLocation`	Obtains or specifies the path or the URL for the image display in a PictureBox control
`InitialImage`	Obtains or specifies the image displayed in a PictureBox control when an image is loading
`SizeMode`	Determines how the image is displayed in a PictureBox control
`TabStop`	Obtains or specifies a value that indicates the focus of controls by using the Tab key
`WaitOnLoad`	Obtains or specifies a value indicating whether an image is loaded synchronously

Table 3.13 describes a list of public methods of the `PictureBox` class:

Table 3.13: Noteworthy Public Methods of the PictureBox Class

Method	Description
`CancelAsync()`	Cancels an asynchronous image load
`Load()`	Displays an image in a PictureBox control
`LoadAsync()`	Loads images asynchronously

Table 3.14 describes a list of public events of the `PictureBox` class:

Table 3.14: Noteworthy Public Events of the PictureBox Class

Event	Description
`CausesValidationChanged`	Overrides the Control.CausesValidationChanged property
`Enter`	Overrides the Control.Enter property
`FontChanged`	Occurs when the value of the Font property changes
`ForeColorChanged`	Occurs when the value of the ForeColor property changes
`ImeModeChanged`	Occurs when the value of the Input Method Editor (IME) mode property changes
`Leave`	Occurs when input focus leaves the PictureBox control

Table 3.14: Noteworthy Public Events of the PictureBox Class

Event	Description
`LoadCompleted`	Occurs when an asynchronous image-load operation is completed or canceled; or an exception is raised
`LoadProgressChanged`	Occurs when the progress of an asynchronous image loading operation changes
`RightToLeftChanged`	Occurs when the value of the RightToLeft property changes
`SizeModeChanged`	Occurs when the `SizeMode property changed`
`TabIndexChanged`	Occurs when the value of the TabIndex property changes
`TabStopChanged`	Occurs when the value of the TabStop property changes

Next, let's learn about the Panel control.

The Panel Control

A Panel control is used to group other controls. More precisely, it is used to divide a Windows form into different regions, where each region represents a separate group of related controls. For example, you may have a menu form that allows a user to select drinks in one panel and the type of sandwich in another. The class hierarchy of the `Panel` class is as follows:

```
System.Object
 System.MarshalByRefObject
           System.ComponentModel.Component
                     System.Windows.Forms.Control
                               System.Windows.Forms.ScrollableControl
                                         System.Windows.Forms.Panel
```

To display scroll bars in the Panel control, set the AutoScroll property to True. You can use the BackColor and BackgroundImage properties to customize the Panel control.

Table 3.15 describes a list of public properties of the `Panel` class:

Table 3.15: Noteworthy Public Properties of the Panel Class

Property	Description
`AutoSizeMode`	Determines the automatic sizing behavior of the control
`BorderStyle`	Determines the border style for the control

Table 3.16 describes the public method of the `Panel` class:

Table 3.16: Noteworthy Public Method of the Panel Class

Method	Description
`ToString()`	Returns a string representation for the Panel control

Table 3.17 describes the public event of the `Panel` class:

Table 3.17: Noteworthy Public Event of the Panel Class

Event	Description
`AutoSizeChanged`	Occurs when the value of the AutoSize property changes

Now, let's learn how to add a border style to a Panel control.

Adding a Border Style to a Panel Control

You can provide a border to a Panel control by using the BorderStyle property. The following code snippet is used to set a border style around a Panel control:

```
private void Form1_Load(object sender, EventArgs e)
{
 panel1.BorderStyle = BorderStyle.Fixed3D;
}
```

As a result, the border is added to the Panel control, as shown in Figure 3.7:

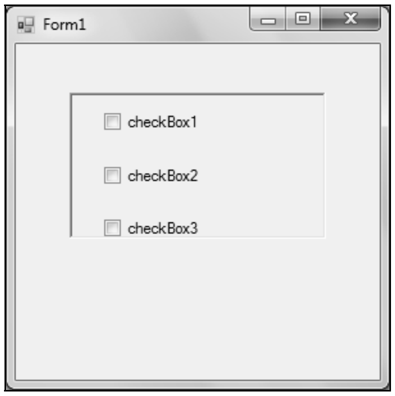


Figure 3.7: Adding a Border to the Panel Control

Now, let's learn how to add scroll bars to a Panel control.

Adding Scroll Bars to Panel Controls

At times, the number of controls in the Panel control may increase and it might not be possible to display all the controls on the screen. In such a situation, you can add scroll bars in a Panel control, to enable the users to scroll and view all the controls. The following code snippet is used to add scroll bars to a Panel control:

```
private void Form1_Load(object sender, EventArgs e)
{
   panel1.AutoScroll = true;
}
```

The scroll bar is added to the Panel control, as shown in Figure 3.8:

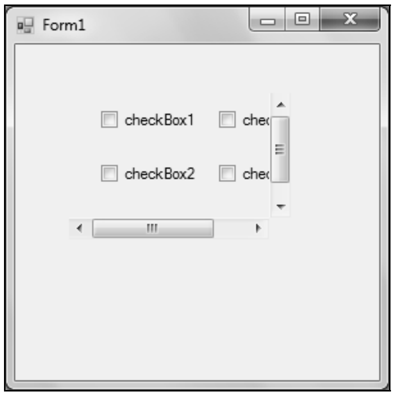


Figure 3.8: Adding Scroll Bars to the Panel Control

Next, let's learn about the GroupBox control.

The GroupBox Control

As already learned, a GroupBox control is used to group similar controls together. It displays frames around the controls and can display text in a caption. The class hierarchy of the `GroupBox` class is as follows:

```
    System.Object
     System.MarshalByRefObject
           System.ComponentModel.Component
                 System.Windows.Forms.Control
                       System.Windows.Forms.GroupBox
```

GroupBox controls are generally used to display radio buttons, as shown in Figure 3.9:

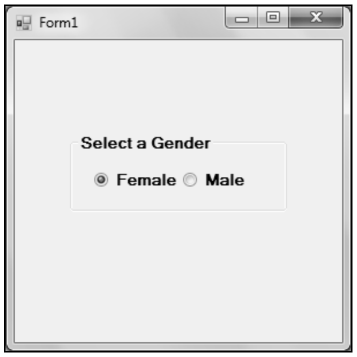


Figure 3.9: Displaying a GroupBox Control

Table 3.18 describes a list of public properties of the `GroupBox` class:

Table 3.18: Noteworthy Public Properties of the GroupBox Class

Property	Description
`AutoSize`	Obtains or specifies the value that indicates whether the size of a GroupBox control can be resized based on the content it holds
`AutoSizeMode`	Obtains or specifies how the GroupBox control behaves on enabling its AutoSize property
`DisplayRectangle`	Displays a rectangle corresponding to the dimensions of a GroupBox control
`FlatStyle`	Obtains or specifies the flat style appearance of a GroupBox control
`TabStop`	Obtains or specifies the value that indicates whether a user can press the Tab key for navigating in the GroupBox control
`Text`	Obtains or specifies the text associated with the control
`UseCompatibleTextRendering`	Obtains or specifies a value that indicates whether the text of the GroupBox control is rendered by using the compatible text rendering

Table 3.19 describes the public method of the `GroupBox` class:

Table 3.19: Noteworthy Public Method of the GroupBox Class

Method	Description
`ToString()`	Returns a string that contains the name of the component (base class for all the components in the Common Runtime Language), if there is any

Table 3.20 describes a list of public events of the `GroupBox` class:

Table 3.20: Noteworthy Public Events of the GroupBox Class

Event	Description
`AutoSizeChanged`	Occurs when the value of the AutoSize property changes
`Click`	Occurs when a user clicks a GroupBox control

Table 3.20: Noteworthy Public Events of the GroupBox Class

Event	Description
`DoubleClick`	Occurs when a user double-clicks a GroupBox control
`KeyUp`	Occurs when a user releases a key while a GroupBox control is in focus
`KeyDown`	Occurs when a user presses a key while a GroupBox control is in focus
`MouseClick`	Occurs when a user clicks the GroupBox control
`MouseDoubleClick`	Occurs when a user double-clicks the GroupBox control
`MouseDown`	Occurs when a user presses a mouse button over the control
`MouseEnter`	Occurs when the mouse pointer enters the control
`MouseLeave`	Occurs when the mouse pointer leaves the control
`MouseMove`	Occurs when a user moves the mouse pointer over the control
`MouseUp`	Occurs when a user releases the mouse button, but the mouse pointer is still over the control
`TabStopChanged`	Occurs when the value of the TabStop property changes

Next, let's learn about the TabControl control.

The TabControl Control

A TabControl control is used to display multiple tabs, which work as groups of related controls. The multiple tabs of the TabControl control can be perceived as dividers in a notebook or labels in a set of folders. A TabControl control can contain pictures and other controls. A TabControl control is represented by the `TabControl` class. The class hierarchy of the `TabControl` class is as follows:

```
System.Object
 System.MarshalByRefObject
         System.ComponentModel.Component
                 System.Windows.Forms.Control
                         System.Windows.Forms.TabControl
```

Figure 3.10 shows a TabControl control:

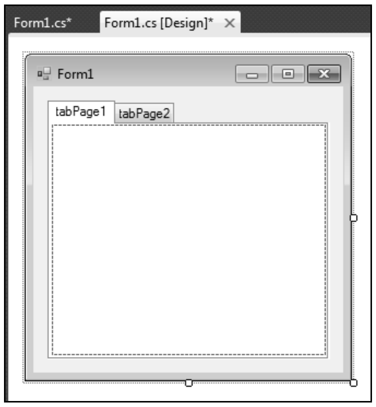


Figure 3.10: Displaying a TabControl Control

`TabPages` is the main property of the `TabControl` control, which contains individual tab pages in the control, each of which is a `TabPage` object. When a tab is clicked, it displays its page and causes a `Click` event for that `TabPage` object. You can add new tab pages by using the `Add()` method of the `TabPages` collection, and remove them by using the `Remove()` method.

Table 3.21 describes a list of public properties of the `TabControl` class:

Table 3.21: Noteworthy Public Properties of the TabControl Class

Property	Description
`Alignment`	Obtains the area in the Windows form where the tabs appear (Top, Left, and so on)
`Appearance`	Specifies the appearance of tabs in a TabControl control
`Multiline`	Determines whether a TabControl control can show more than one row of tabs
`Padding`	Obtains or specifies the amount of space around each item on the control's tab pages
`RowCount`	Obtains the number of rows displayed in a control's tab strip
`SelectedIndex`	Obtains the selected tab page' index
`SelectedTab`	Obtains the selected tab page
`ShowToolTips`	Determines whether a tab's tooltip can be displayed
`SizeMode`	Obtains or specifies the way a control's tabs are sized
`TabCount`	Obtains the number of tabs in the tab strip
`TabPages`	Obtains the collection of tab page in the TabControl control

Table 3.22 describes a list of public methods of the `TabControl` class:

Table 3.22: Noteworthy Public Methods of the TabControl Class

Method	Description
`DeselectTab()`	Makes the tab next to the specified TabPage the current tab
`GetControl()`	Obtains a Tab page at the specified location
`GetTabRect()`	Returns the bounding rectangle for a particular tab in the TabControl control
`SelectTab()`	Makes the tab with the specified name the current tab

Table 3.23 describes a list of public events of the `TabControl` class:

Table 3.23: Noteworthy Public Events of the TabControl Class

Event	Description
`Deselected`	Occurs when a tab is deselected
`Deselecting`	Occurs before a tab is deselected, enabling a handler to cancel the tab change
`RightToLeftLayoutChanged`	Occurs when the value of the RightToLeftLayout property changes
`Selected`	Occurs when a tab is selected
`SelectedIndexChanged`	Occurs when the `SelectedIndex` property changes
`Selecting`	Occurs before the tab is selected, enabling a handler to cancel the tab change

Till now, you have covered a variety of Windows Forms controls, such as ListView, TreeView, ImageList, Panel, GroupBox, and TabControl on a conceptual basis. Now, it's time to learn about practically implementing these controls in the *Immediate Solutions* section.

Immediate Solutions

Using the ListView Control

In this section, we create a Windows Forms application, `ListViewControl` (also available on the CD), to display the use of the ListView control. In this application, you can see the effects of different views of a ListView control. Let's perform the following steps to create the `ListViewControl` application:

1. Create a Windows Forms application, named `ListViewControl`.
2. Set the Text property of Form1 to List View Form (Figure 3.11).
3. Add two Label controls, two ImageList controls, one ComboBox control, and one ListView control from Toolbox to Form1 (in Designer mode). Set the Text property of label1 to List View Control: and label2 to Select a View. You should also set the Name property of imageList1 to LargeImageList and imageList2 to SmallImageList. Now, arrange the controls on the Form1, as shown in Figure 3.11:

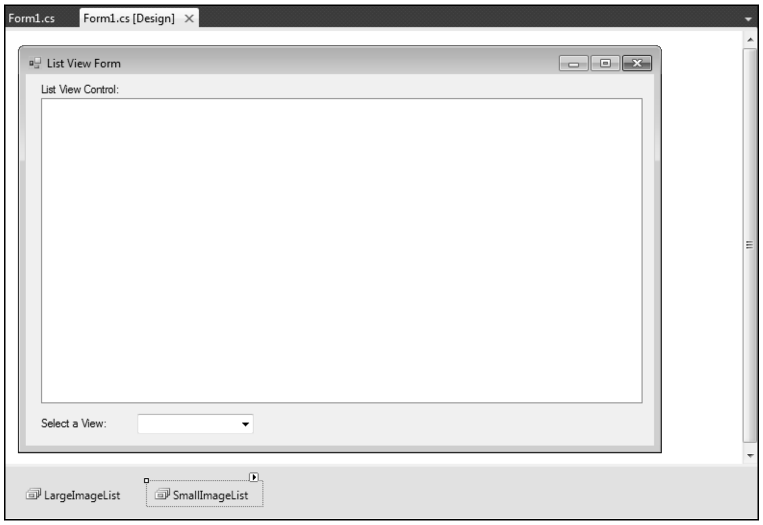


Figure 3.11: Displaying Controls Positions

4. Add some images in the LargeImageList and SmallImageList controls using the Images property. Set the ImageSize property of the LargeImageList control to 250,250; and the ImageSize property of the SmallImageList control to 100,100.
5. Add the code, given in Listing 3.1, to the Form1.cs file:

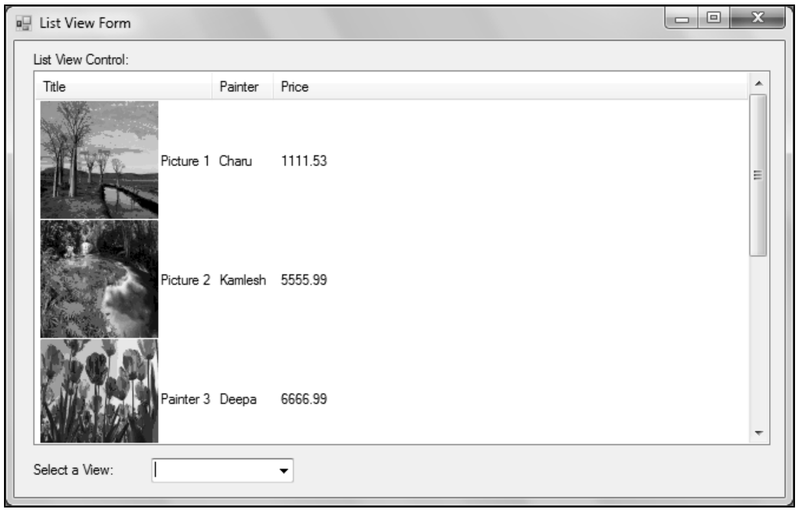
Listing 3.1: Adding the Code to Use ListView Control

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

namespace ListViewControl
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            LoadListView();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            comboBox1.Items.Add("Large Icons");
            comboBox1.Items.Add("Details");
            comboBox1.Items.Add("Small Icons");
            comboBox1.Items.Add("List");
                  }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (comboBox1.SelectedIndex == 0)
            {
                listView1.View = View.LargeIcon;
            }
            else if (comboBox1.SelectedIndex == 1)
            {
                listView1.View = View.Details;
            }
            else if (comboBox1.SelectedIndex == 2)
            {
                listView1.View = View.SmallIcon;
            }
            else
            {
                listView1.View = View.List;
            }


        }
        private void LoadListView()
        {
            listView1.View = View.Details;

            // Add columns
            listView1.Columns.Add("Title",-2,HorizontalAlignment.Left);
            listView1.Columns.Add("Painter",-2,HorizontalAlignment.Left);
            listView1.Columns.Add("Price",-2,HorizontalAlignment.Left);

            listView1.LargeImageList = LargeImageList;
            listView1.SmallImageList = SmallImageList;

            // Add items
            ListViewItem item1 = new ListViewItem("Picture 1");
            item1.SubItems.Add("Charu");
            item1.SubItems.Add("1111.53");

            item1.ImageIndex = 0;

            ListViewItem item2 = new ListViewItem("Picture 2");
            item2.SubItems.Add("Kamlesh");
            item2.SubItems.Add("5555.99");

            item2.ImageIndex = 1;
```

```
                ListViewItem item3 = new ListViewItem("Painter 3");
                item3.SubItems.Add("Deepa");
                item3.SubItems.Add("6666.99");

                item3.ImageIndex = 2;

                ListViewItem item4 = new ListViewItem("Painter 4");
                item4.SubItems.Add("Vineet");
                item4.SubItems.Add("6666.99");

                item4.ImageIndex = 3;

                         // Add the items to the ListView.
                listView1.Items.AddRange(
                                        new ListViewItem[] {item1,
                                                    item2,
                                                    item3,
                                                    item4}
                                        );

            }
        }
    }
```

In Listing 3.1, we add four items in the comboBox1 control at the Load event of the Form1. Next, different views of listView1 control are mapped with the items of the comboBox1 control, so that when you select an item in the combo box, the corresponding view is displayed in the ListView control. Then, a method, named LoadListView(), is created to display the items of the ListView control in the Details view. In this method, we create three columns in the listView1 control: Title, Painter, and Price. Next, instances of the ListViewItem class are created to add the items to the listView1 control. Finally, the LoadListView() method is called through the constructor of Form1.

6. Press the F5 key on the keyboard to run the ListViewControl application. Figure 3.12 shows the output of the ListViewControl application:



Title	Painter	Price
Picture 1	Charu	1111.53
Picture 2	Kamlesh	5555.99
Painter 3	Deepa	6666.99

Figure 3.12: Displaying the Output of the ListViewControl Application

If you select an option in the Select a View combo box, the corresponding view is displayed in the ListView control. For instance, we select the Large Icons option; and the resultant output is shown in Figure 3.13:

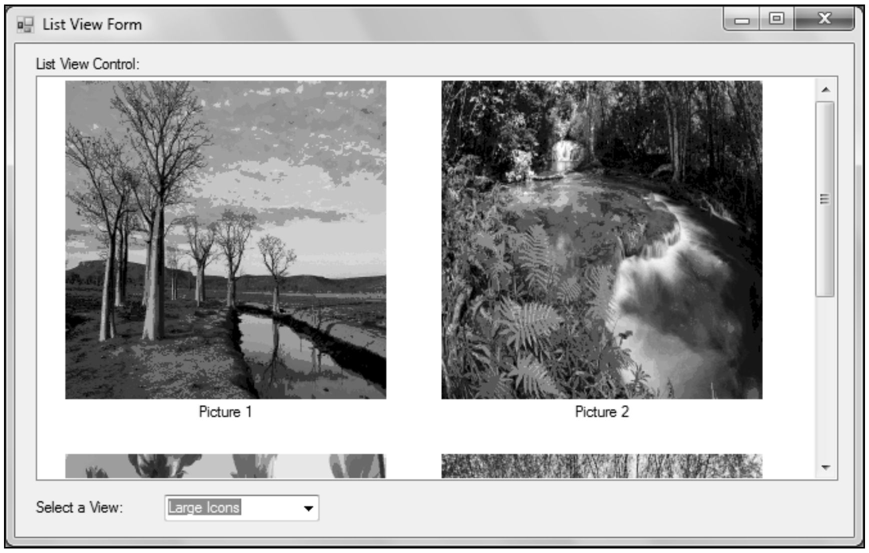


Figure 3.13: Selecting the Large Icons Option

Next, let’s create an application using the TreeView control.

Using the TreeView Control

In this section, we are creating a Windows Forms application, TreeViewControl (also available on the CD), to display the use of the TreeView control. In this application, you can add checkboxes besides the options in the TreeView control and can see the status of the selected option. Let’s perform the following steps to create the TreeViewControl application:

1. Create a Windows Forms application, named TreeViewControl.
2. Set the Text property of Form1 to Tree View Form Control (Figure 3.14).
3. Add two Label controls, one Button control, and one TreeView control from Toolbox to the Form1 (in Designer mode). Set the Text property of label1 to Tree View Control, label2 to Displaying Status, and button1 to Show Checkboxes. Now, arrange the controls on Form1, as shown in Figure 3.14:

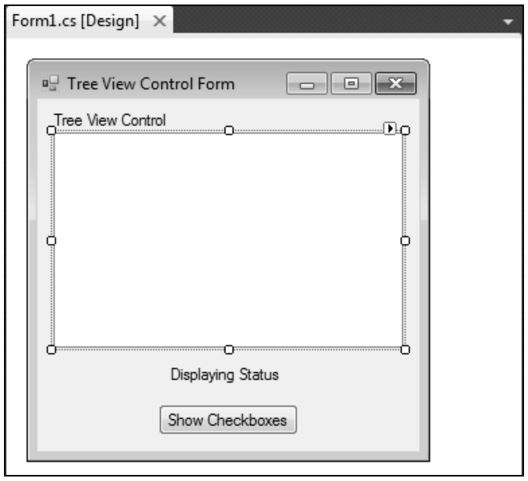
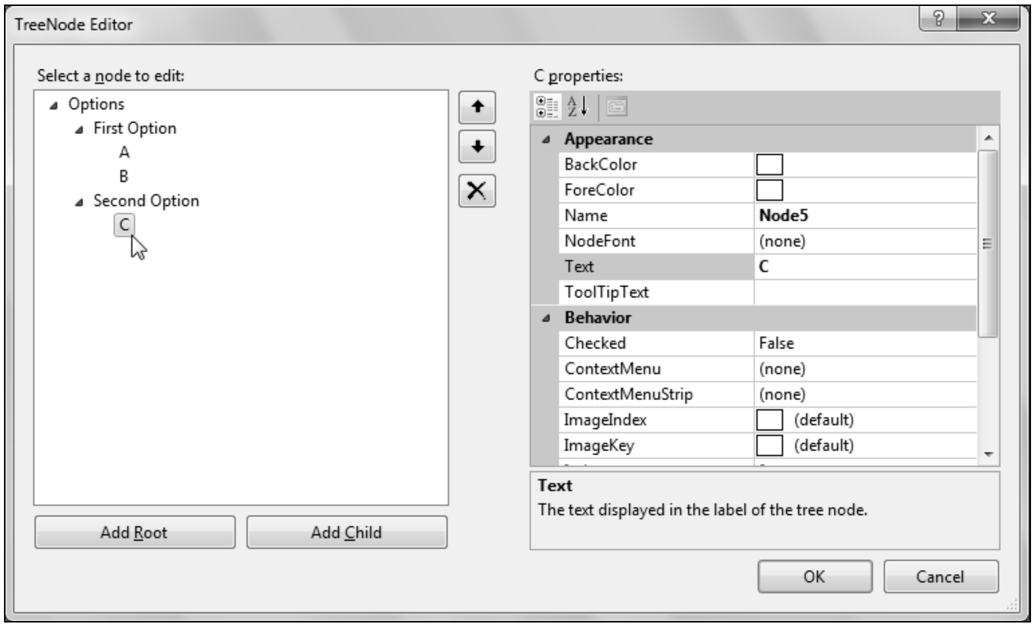


Figure 3.14: Displaying Controls Positions in the TreeViewControl Application

4. Add some nodes in the treeView1 control using the Nodes property, which enables you to add nodes in a TreeView control with the help of the TreeNode Editor, as shown in Figure 3.15:



Appearance	
BackColor	
ForeColor	
Name	**Node5**
NodeFont	(none)
Text	C
ToolTipText	
Behavior	
Checked	False
ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	(default)
ImageKey	(default)

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

Figure 3.15: Displaying the TreeNode Editor

5. Add the code, given in Listing 3.2, to the Form1.cs file:

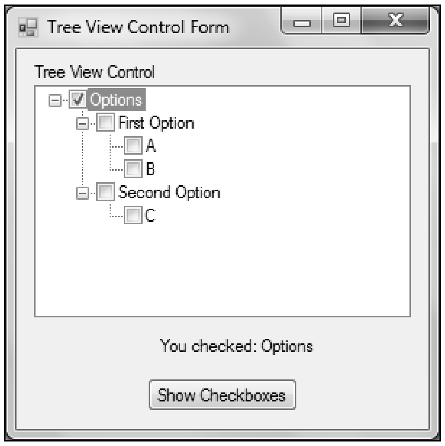
Listing 3.2: Adding the Code to Use TreeView Control

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

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

        private void button1_Click(object sender, EventArgs e)
        {
            treeView1.CheckBoxes = true;
        }

        private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
        {
            label2.Text = "You clicked: " + e.Node.Text;
        }
```

```
            private void treeView1_AfterCheck(object sender, TreeViewEventArgs e)
            {
                if (e.Node.Checked)
                {
                    label2.Text = "You checked: " + e.Node.Text;
                }

                else
                {
                    label2.Text = "You unchecked: " + e.Node.Text;
                }

            }
        }
    }
```

In Listing 3.2, we are adding check boxes at runtime besides the nodes of the ListView control, through the Click event of button1. The AfterSelect event of the treeView1 control occurs when a node is selected at runtime. In addition, the AfterCheck event of the treeView1 control occurs when you select or clear a check box besides a node.

6. Press the F5 key on the keyboard to run the TreeViewControl application. Figure 3.16 shows the output of the TreeViewControl application:



You checked: Options

Figure 3.16: Displaying the Output of the TreeViewControl Application

If you select an option in the TreeView control, its status is displayed. In addition, you can add the check boxes beside the options of the TreeView control.

Next, let's create an application using the ImageList and PictureBox controls.

Using the ImageList and PictureBox Controls

In this section, we are creating a Windows Forms application, ImageListControl (also available on the CD), to display the use of ImageList and PictureBox controls. In this application, you can load an image, navigate the images, add an external image, and stretch out an image.

Let's perform the following steps to create the ImageListControl application:

1. Create a Windows Forms application, named ImageListControl.
2. Add four Button controls, one PictureBox control, one ImageList control, and one OpenFileDialog control from Toolbox to Form1 (in Designer mode). Set the Text property of label1 to Picture box, button1 to Load Image, button2 to Next Image, button3 to Add Image, and button4 to Stretch Image. Now, arrange the controls on the Form1, as shown in Figure 3.17:

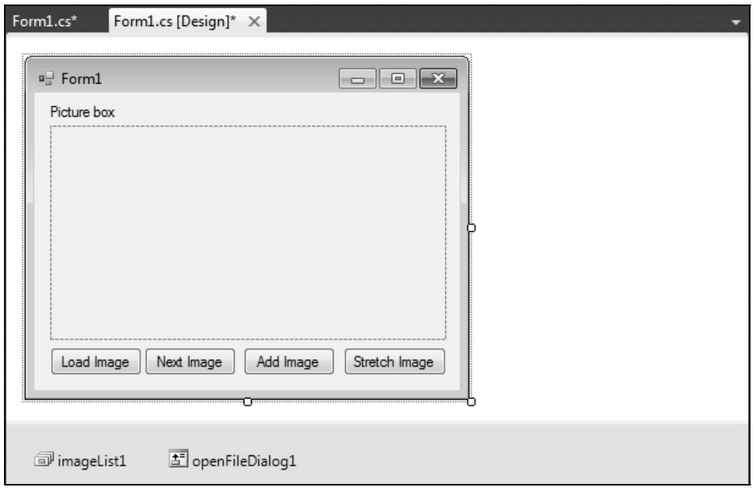


Figure 3.17: Displaying Controls Positions in the ImageListControl Application

3. Add some images in the imageList1 control using the Images property. Set the ImageSize property of the imageList1 control to 200,200.
4. Add the code, given in Listing 3.3, to the Form1.cs file:

Listing 3.3: Adding the Code to Use ImageList and PictureBox Controls

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace ImageListControl
{
    public partial class Form1 : Form
    {
        private int ImageIndex = 0;
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            pictureBox1.Image = imageList1.Images[ImageIndex];
        }
        private void button2_Click(object sender, EventArgs e)
        {
            if (ImageIndex < imageList1.Images.Count - 1)
            {
                ImageIndex += 1;
            }
            else
            {
                ImageIndex = 0;
            }
            pictureBox1.Image = imageList1.Images[ImageIndex];
        }
```

```
            private void button3_Click(object sender, EventArgs e)
            {
                if (openFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    if (openFileDialog1.FileNames != null)
                    {
                        int intLoopIndex;
                        for (intLoopIndex = 0; intLoopIndex <
                             openFileDialog1.FileNames.Length; intLoopIndex++)
                        {
             imageList1.Images.Add(Image.FromFile(openFileDialog1.FileNames[intLoopIndex]));
                        }
                    }
                    else
                    {

imageList1.Images.Add(Image.FromFile(openFileDialog1.FileNames[Convert.ToInt32(openFileDialog
1.FileName)]));
                    }
                }
            }
            private void button4_Click(object sender, EventArgs e)
            {
                pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
                pictureBox1.ClientSize = new Size(300, 150);
            }
        }
    }
```

In Listing 3.3, the Click event of button1 is used to load the first image, which is stored in the imageList1 control, to the pictureBox1 control. The Click event of button2 is used to navigate to the next image stored in the imageList1 control. Next, the Click event of button3 enables you to open an Open File dialog box and select an external image on your computer system. Finally, the Click event of button4 is used to stretch out an image in the PictureBox control.

5. Press the F5 key on the keyboard to run the ImageListControl application. Now, click the Load Image button to load an image in Form1. Figure 3.18 shows the output of the ImageListControl application:

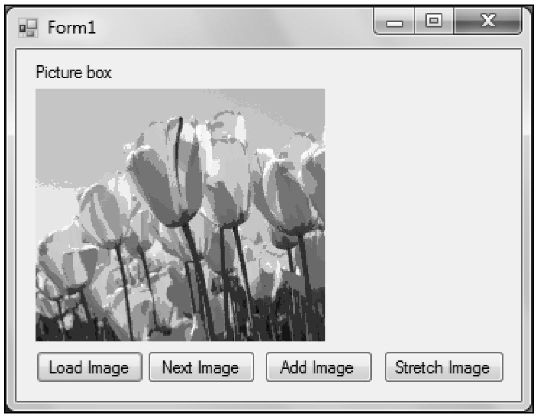


Figure 3.18: Displaying the Output of the ImageListControl Application

In the output, you can click the Load Image and Add Image buttons to load an existing image and an external image, respectively. The Next Image button is used to navigate to the next image. In addition, the Stretch Image button is used for stretching an image, as shown in Figure 3.19:

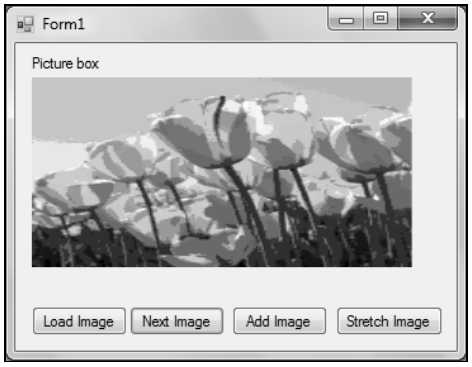


Figure 3.19: Showing an Stretched Image

Next, let's create an application using the Panel and GroupBox controls.

Using the Panel and GroupBox Controls

In this section, we are creating a Windows Forms application, PanelAndGroupBoxControls (also available on the CD), to display the use of the Panel and GroupBox controls. Let's perform the following steps to create the PanelAndGroupBoxControls application:

1. Create a Windows Forms application, named PanelAndGroupBoxControls.
2. Add two Label controls, six RadioButton controls, one Panel control, and one GroupBox control from Toolbox to the Form1 (in Designer mode). Set the Text property of label1 to Select a Radio Button and label2 to Displaying Status. Now, arrange the controls on the Form1, as shown in Figure 3.20:

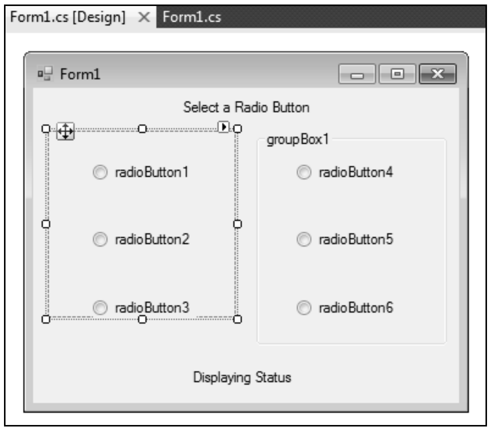


Figure 3.20: Displaying Controls Positions in the PanelAndGroupBoxControls Application

3. Add the code, given in Listing 3.4, to the Form1.cs file:

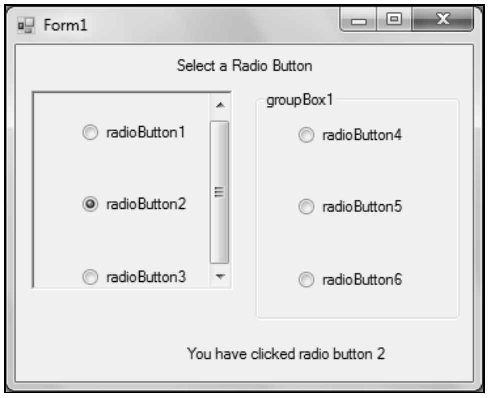
Listing 3.4: Adding the Code to Use Panel and GroupBox Controls

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace PanelAndGroupBoxControls
{
    public partial class Form1 : Form
```

```
        {
            public Form1()
            {
                InitializeComponent();
            }
            private void radioButton1_CheckedChanged(object sender, EventArgs e)
            {
                label2.Text = "You have clicked radio button 1";
            }
            private void radioButton2_CheckedChanged(object sender, EventArgs e)
            {
                label2.Text = "You have clicked radio button 2";
            }
            private void radioButton3_CheckedChanged(object sender, EventArgs e)
            {
                label2.Text = "You have clicked radio button 3";
            }
            private void radioButton4_CheckedChanged(object sender, EventArgs e)
            {
                label2.Text = "You have clicked radio button 4";
            }
            private void radioButton5_CheckedChanged(object sender, EventArgs e)
            {
                label2.Text = "You have clicked radio button 5";
            }
            private void radioButton6_CheckedChanged(object sender, EventArgs e)
            {
                label2.Text = "You have clicked radio button 6";
            }
            private void Form1_Load(object sender, EventArgs e)
            {
                panel1.BorderStyle = BorderStyle.Fixed3D;
                panel1.AutoScroll = true;
            }
        }
    }
```

In Listing 3.4, the CheckedChanged events of radio buttons show the respective message on the label2 control. In addition, the panel1 control displays a border style, Fixed3D, and scroll bars at runtime.

4. Press the F5 key on the keyboard to run the PanelAndGroupBoxControls application. Figure 3.21 shows the output of the PanelAndGroupBoxControls application:



You have clicked radio button 2

Figure 3.21: Displaying the Output of the PanelAndGroupBoxControls Application

If you select a radio button in the Panel or GroupBox control, the respective message is shown in the Label control.

Next, let’s create an application using the TabControl control.

Using the TabControl Control

In this section, we are creating a Windows Forms application, TabControl (also available on the CD), to display the use of the TabControl control. In this application, you can add a tab and button controls at run time.

Let's perform the following steps to create the TabControl application:

1. Create a Windows Forms application, named TabControl.
2. Set the text property of Form1 to Tab Control Form (Figure 3.22).
3. Add a TabControl control from Toolbox to Form1 (in Designer mode).
4. Add two Button controls from Toolbox to tabPage1. Set the Text property of button1 as Add third tab and button2 to Add a button, as shown in Figure 3.22:

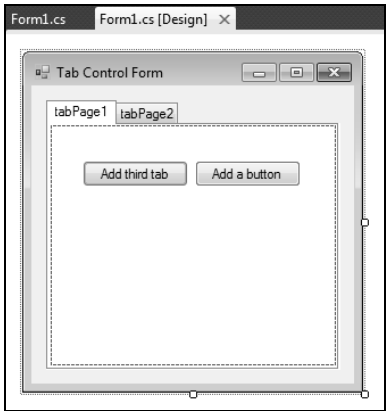


Figure 3.22: Adding Controls to tabPage1

5. Add a PictureBox control from Toolbox to tabPage2. Add an image in the pictureBox1 control through its Image property, as shown in Figure 3.23:

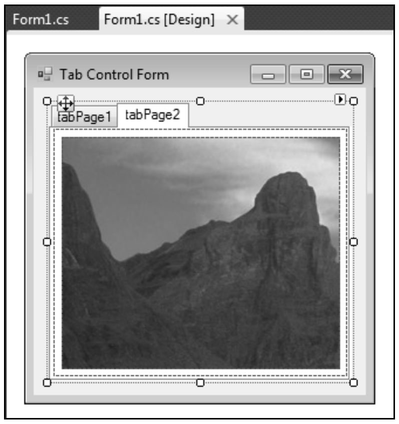


Figure 3.23: Adding Controls to tabPage2

6. Add the code, given in Listing 3.5, to the Form1.cs file:

Listing 3.5: Adding the Code to Use TabControl

```
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Data;
    using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace TabControl
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            TabPage tabpage = new TabPage();
            tabpage.Text = "tabPage3";
            tabControl1.TabPages.Add(tabpage);
        }
        private void button2_Click(object sender, EventArgs e)
        {
            Button button3 = new Button();
            button3.Click += new System.EventHandler(button3_Click);
            button3.Size = new Size(112, 23);
            button3.Location = new Point(18, 75);
            button3.Text = "New Button";
            tabControl1.TabPages[0].Controls.Add(button3);
        }
        private void button3_Click(object sender, EventArgs e)
        {
            MessageBox.Show("You clicked the button!");
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            tabControl1.SelectedTab = tabPage2;
        }
    }
}
```

In Listing 3.5, the Click event of button1 adds a new tab page to the tabControl1 control. The Click event of button2 adds a new button, button3, to tabPage1, which displays a message through its Click event. Finally, the Load event of Form1 displays tabPage2, when you run the application.

7. Press the F5 key on the keyboard to run the TabControl application. Figure 3.24 shows the output of the TabControl application:

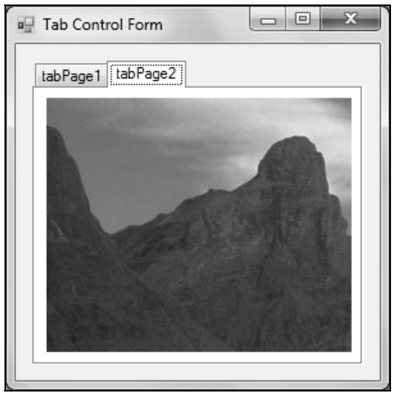
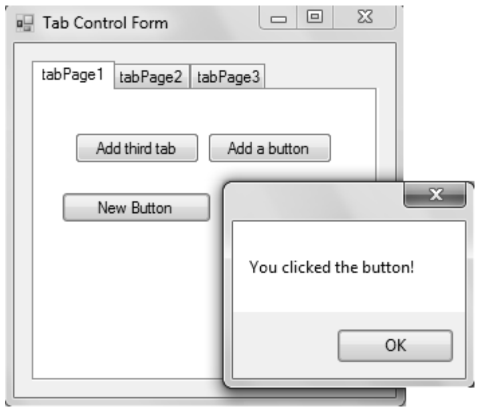


Figure 3.24: Displaying the Output of the TabControl Application

If you select tabPage1, you see two buttons, Add third tab and Add a button. The Add third tab button adds a new tab page to the TabControl control; while Add a button adds a new button that shows a message on its Click event, as shown in Figure 3.25:



You clicked the button!

Figure 3.25: Adding a New Tab Page and Button

Now, let's summarize the main topics discussed in this chapter.

Summary

In this chapter, you have learned about seven different types of controls in Windows Forms. These controls are ListView, TreeView, ImagelList, GroupBox, PictureBox, Panel, and TabControl. We have described various properties, events, and methods of these controls in the *In Depth* section, and implemented them in the *Immediate Solutions* section. You have also learned how to perform different operations using these controls, such as adding items to the ListView control, setting the view of the ListView control, displaying checkboxes in the TreeView control, adding and removing images from the ImageList control, setting an image in the PictureBox control, grouping other controls in the Panel control, and adding tabs to the TabControl control.

In the next chapter, you learn about some other Windows Forms controls, which include SplitContainer, ScrollBar, TrackBar, ToolTip, NotifyIcon, MonthCalendar, DateTimePicker, Timer, and ProgressBar.