



4

Windows Forms Controls: SplitContainer, ScrollBar, TrackBar, ToolTip, NotifyIcon, MonthCalendar, DateTimePicker, Timer, and ProgressBar

<i>If you need an Immediate Solution to:</i>	<i>See page:</i>
Using the SplitContainer Control	115
Using the ScrollBar Controls	117
Using the TrackBar Control	122
Using the ToolTip Control	124
Using the NotifyIcon Control	125
Using the MonthCalendar and DateTimePicker Controls	127
Using the Timer Control	130
Using the ProgressBar Control	133

In Depth

In the last few chapters, you have already learned how to work with a number of Windows Forms controls, such as Button, TextBox, CheckBox, ListView, and TreeView. In addition to the controls discussed earlier, there are few more Windows Forms controls, such as SplitContainer, ScrollBar, TrackBar, ToolTip, NotifyIcon, MonthCalendar, DateTimePicker, Timer, and ProgressBar controls, which prove to be beneficial in certain situations. The SplitContainer control is used to split the Windows Form horizontally or vertically. The ScrollBar control is used to scroll a Windows form. The TrackBar control is used when you need to navigate through a large amount of information on a Windows form. The ToolTip control is used to display the additional information about any element on an interface. The NotifyIcon control is used to display the notification icon on the task bar of the Windows operating system. The MonthCalendar control allows you to select the date in a control. The DateTimePicker control allows you to select date and time in the control. The Timer control is used when you need to perform some task in a specific time, such as when you need to change the text of a Button control after every 10 seconds. The ProgressBar control is used to view the progress of any activity.

In the *In Depth* section of this chapter, we discuss about various controls, such as SplitContainer, ScrollBar, TrackBar, ToolTip, NotifyIcon, MonthCalendar, DateTimePicker, Timer, and ProgressBar. The chapter also discusses the noteworthy properties, methods, and events of each of these controls in detail. Further, the *Immediate Solutions* section of the chapter demonstrates the practical implementation of these controls.

Now, let's start the chapter with the SplitContainer control.

The SplitContainer Control

The SplitContainer control consists of a movable bar that divides the display area of a Windows form into two resizable panels. You can add another SplitContainer control or some other Windows Forms controls to each of the two resizable panels to create numerous resizable display areas. The SplitContainer control also makes the placement of controls easy and simple at design time. Following is the class hierarchy of the SplitContainer class:

```
System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ScrollableControl
          System.Windows.Forms.ContainerControl
            System.Windows.Forms.SplitContainer
```

The SplitContainer class contains a number of useful properties, methods, and events that make the task of working with the SplitContainer control much easier. Table 4.1 describes the noteworthy public properties of the SplitContainer class:

Table 4.1: Showing the Public Properties of the SplitContainer Class	
Property	Description
BackgroundImage	Obtains or specifies the background image displayed in a control
BindingContext	Obtains or specifies the BindingContext class for the SplitContainer control
BorderStyle	Obtains or specifies the style of border for the SplitContainer control
Dock	Obtains or specifies the borders for the SplitContainer control, which are attached to the edges of the container
FixedPanel	Obtains or specifies SplitContainer panel, which retains its original size when the Windows Form is resized
IsSplitterFixed	Obtains or specifies a value indicating whether the splitter is fixed or movable

Table 4.1: Showing the Public Properties of the SplitContainer Class	
Property	Description
Orientation	Obtains or specifies a value indicating the horizontal or vertical orientation of the SplitContainer panels
Panel1	Gets the left or top panel of the SplitContainer control, depending on the Orientation property
Panel1Collapsed	Obtains or specifies a value determining whether Panel1 is collapsed or expanded
Panel1MinSize	Obtains or specifies the minimum distance, in pixels, of the splitter from the left or top edge of Panel1
Panel2	Gets the right or bottom panel of the SplitContainer control, depending on Orientation
Panel2Collapsed	Obtains or specifies a value determining whether Panel2 is collapsed or expanded
Panel2MinSize	Obtains or specifies the minimum distance, in pixels, of the splitter from the right or bottom edge of Panel2
SplitterDistance	Obtains or specifies the location of the splitter, in pixels, from the left or top edge of the SplitContainer
SplitterIncrement	Obtains or specifies a value representing the increment of splitter movement in pixels
SplitterRectangle	Gets the size and location of the splitter relative to the SplitContainer control
SplitterWidth	Obtains or specifies the width of the splitter in pixels
TabStop	Obtains or specifies a value indicating whether the focus can be shifted to the splitter using the TAB key

Table 4.2 describes the noteworthy public methods of the SplitContainer class:

Table 4.2: Showing the Public Methods of the SplitContainer Class	
Method	Description
OnSplitterMoved()	Raises the SplitterMoved event
OnSplitterMoving()	Raises the SplitterMoving event

Table 4.3 describes the noteworthy public events of the SplitContainer class:

Table 4.3: Showing the Public Events of the SplitContainer Class	
Event	Description
BackgroundImageChanged	Occurs at the time the BackgroundImage property changes
SplitterMoved	Occurs at the time the splitter control is moved
SplitterMoving	Occurs at the time the splitter control is in the process of moving

Figure 4.1 shows a SplitContainer control added to a form:

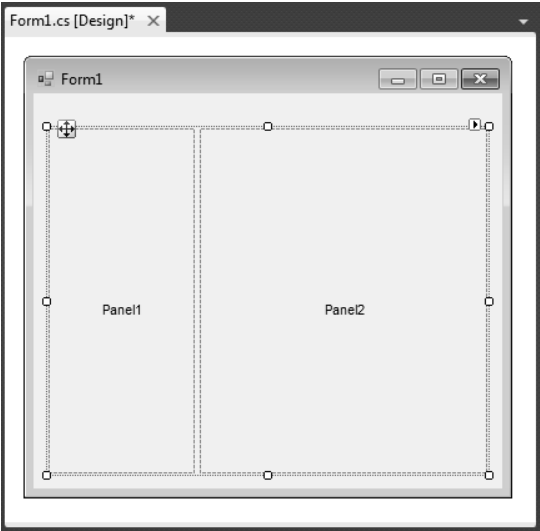


Figure 4.1: Showing the SplitContainer Control

In this section, you have learned about the `SplitContainer` control. In the next section, you learn about the `ScrollBar` control.

The ScrollBar Control

The `ScrollBar` controls represent the vertical or horizontal controls that display a scroll box that you can scroll vertically or horizontally. These controls are used for navigating through a long list of items or a large amount of information when you scroll either vertically or horizontally within an application or control. The `ScrollBar` controls are of two types, `HScrollBar` and `VScrollBar`. These control types are used independently from each other.

Following is the class hierarchy for the `HScrollBar` class:

```
System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ScrollBar
          System.Windows.Forms.HScrollBar
```

Following is the class hierarchy for the `VScrollBar` class:

```
System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ScrollBar
          System.Windows.Forms.VScrollBar
```

Table 4.4 lists the public properties of the `ScrollBar` class:

Table 4.4: Showing the Public Properties of the ScrollBar Class	
Property	Description
AutoSize	Obtains or applies a value indicating whether the ScrollBar is automatically resized to fit its content
BackColor	Obtains or applies the backcolor of the ScrollBar control
ForeColor	Obtains or applies the forecolor of the ScrollBar control

Table 4.4: Showing the Public Properties of the ScrollBar Class

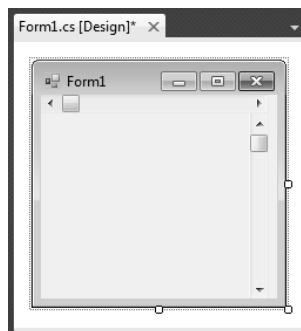
Property	Description
ImeMode	Obtains or applies the Input Method Editor (IME) mode
LargeChange	Obtains or sets the value added to or subtracted from to the Value property when the scroll bar is clicked (outside the scroll box)
Maximum	Obtains or sets the upper limit of the scrollable range
Minimum	Obtains or sets the lower limit of the scrollable range
SmallChange	Obtains or sets the value added to or subtracted from to the Value property when the user clicks the arrow button in the scroll bar
Value	Obtains or sets a value corresponding to the current position of the scroll box

Table 4.5 lists the public events of the ScrollBar class:

Table 4.5: Showing the Public Events of the ScrollBar Class

Event	Description
Click	Takes place when the ScrollBar control is clicked
DoubleClick	Takes place when the ScrollBar control is double clicked
Scroll	Occurs when the scroll box is moved (either by the mouse or the keyboard)
ValueChanged	Occurs when the Value property has changed, either by a Scroll event or programmatically

Figure 4.2 shows an HScrollBar control and a VScrollBar control added to a form:

**Figure 4.2: Showing the HScrollBar and VScrollBar Controls**

There are two primary events for scroll bars; the Scroll event, which occurs continuously as the scroll box is scrolled, and the ValueChanged event, which occurs every time the value of the scroll bar changes.

The TrackBar Control

The TrackBar control is used for navigating through a large amount of information or for visual adjustment of numeric setting. It consists of two parts—thumb (also known as slider) and tick marks. You can adjust the thumb part using the Value property. The tick marks are visual indicators that are spaced at regular intervals. Following is the class hierarchy for the TrackBar class:

```

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

Table 4.6 lists the public properties of the `TrackBar` class:

Table 4.6: Showing the Public Properties of the <code>TrackBar</code> Class	
Property	Description
<code>AutoSize</code>	Specifies whether or not the track bar's height or width should be automatically sized
<code>BackgroundImage</code>	Obtains or sets the background image for the <code>TrackBar</code> control
<code>BackgroundImageLayout</code>	Obtains or sets the <code>ImageLayout</code> value
<code>Font</code>	Overrides the specified font for the track bar
<code>ForeColor</code>	Holds the foreground color of the track bar
<code>ImeMode</code>	Obtains or sets the IME mode supported by the current control
<code>LargeChange</code>	Obtains or sets the value added to or subtracted from the <code>Value</code> property when the scroll box is moved through a large distance
<code>Maximum</code>	Holds the upper limit of the range of the current track bar
<code>Minimum</code>	Holds the lower limit of the range of the current track bar
<code>Orientation</code>	Obtains or sets the horizontal or vertical orientation of the track bar
<code>Padding</code>	Obtains or sets the space between the edges of the <code>TrackBar</code> control and its contents
<code>RightToLeftLayout</code>	Obtains or sets the value indicating whether the contents of the <code>TrackBar</code> control are laid out from right to left
<code>SmallChange</code>	Obtains or sets a value, which is added to or subtracted from the <code>Value</code> property when the scroll box moves a small distance
<code>Text</code>	Obtains or sets the text of the <code>TrackBar</code> control
<code>TickFrequency</code>	Obtains or sets a value specifying the distance between ticks
<code>TickStyle</code>	Specifies how to display the tick marks in the track bar
<code>Value</code>	Obtains or sets the current position of the slider in the track bar

You can configure a track bar's range with the `Minimum` (default = 0) and `Maximum` (default = 10) properties. You can specify the amount by which the `Value` property should be incremented when the user clicks the sides of the slider, by means of the `LargeChange` property (default = 5). Similarly, you can specify how much the `Value` property should be incremented when the user uses the arrow keys for scrolling, by using the `SmallChange` property (default = 1). A track bar can be displayed horizontally or vertically by using the `Orientation` property.

You can also configure track bars with the `TickStyle` property, which lets you determine how the tick marks are displayed. The `TickStyle` property takes values from the `TickStyle` enumeration, which has the following members:

- ☐ `Both`—Provides tick marks on both sides of the control
- ☐ `BottomRight`—Provides tick marks at the bottom of a horizontal control or on the right side of a vertical control
- ☐ `None`—Provides no tick marks on the control
- ☐ `TopLeft`—Provides tick marks on the top of a horizontal control or on the left of a vertical control

You can also set the tick frequency, which sets the distance between ticks, by using the `TickFrequency` property (the default is 1).

Table 4.7 describes the public methods of the `TrackBar` class:

Table 4.7: Showing the Public Methods of the TrackBar Class

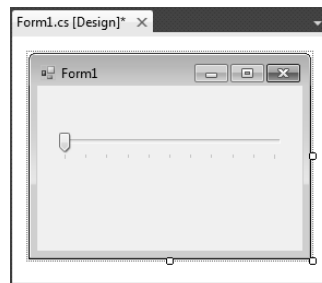
Method	Description
BeginInit()	Starts the initialization of a TrackBar control that is used on a form or used by another component. Most of the times, initialization occurs at runtime.
EndInit()	Stops the initialization of a TrackBar control that is used on a form or used by another component. Most of the times, initialization occurs at runtime.
SetRange()	Sets the Minimum and Maximum values for the track bar.
ToString()	Returns the string that in turn represents the TrackBar control.

Table 4.8 describes the public events of the `TrackBar` class:

Table 4.8: Showing the Public Events of the TrackBar Class

Event	Description
AutoSizeChanged	Occurs when the value of the <code>AutoSize</code> property changes
BackgroundImageChanged	Occurs when the <code>BackgroundImage</code> property changes
BackgroundImageLayoutChanged	Occurs when the <code>BackgroundImageLayout</code> property changes
Click	Occurs when the user clicks the TrackBar control
DoubleClick	Occurs when the user double-clicks the TrackBar control
FontChanged	Occurs when the <code>Font</code> property changes
ForeColorChanged	Occurs when the <code>ForeColor</code> property changes
ImeModeChnaged	Occurs when the <code>ImeMode</code> property changes
MouseClick	Occurs when the user clicks the TrackBar control
MouseDoubleClick	Occurs when the user double-clicks the TrackBar control
PaddingChanged	Occurs when the value of the <code>Padding</code> property changes
Paint	Occurs on drawing the TrackBar control
RightToLeftLayoutChanged	Occurs when the value of the <code>RightToLeftLayout</code> property changes
Scroll	Occurs when the slider is moved (either by mouse or keyboard action)
TextChanged	Occurs when the <code>Text</code> property changes
ValueChanged	Occurs when the <code>Value</code> property of a track bar changes (either by moving the slider or through code)

Figure 4.3 shows a `TrackBar` control added to a form:

**Figure 4.3: Showing the TrackBar Control**

In this section, you have learned about the `TrackBar` control. In the next section, you learn about the `ToolTip` control.

The ToolTip Control

The `ToolTip` control is used to display a small window with explanatory text for an element on the interface. The tool tip for a control or window appears when you move the mouse over the control or window. Following is the class hierarchy of the `ToolTip` class:

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

You can associate a tool tip with any other control. To connect a tool tip with a control, you use its `SetToolTip()` method. For example, to connect the tool tip to `Button1`, you can use the following code snippet:

```
ToolTip2.SetToolTip(Button1, "This is a button")
```

The important properties for tool tip controls are as follows:

- ❑ `Active`—Displays the tool tip when set to `true`
- ❑ `AutomaticDelay`—Sets the length of time; i.e., how long the user must point at the control for the tool tip to appear, or how long it takes for subsequent tool tip windows to appear

Table 4.9 describes the public properties of the `ToolTip` class:

Table 4.9: Showing the Public Properties of the ToolTip Class	
Properties	Description
<code>Active</code>	Specifies whether or not the tool tip control is active
<code>AutomaticDelay</code>	Obtains or sets the time (in milliseconds) before the tool tip appears
<code>InitialDelay</code>	Obtains or sets the starting delay for the tool tip
<code>ShowAlways</code>	Specifies whether the tool tip should appear when its parent control is not active

Table 4.10 describes the public events of the `ToolTip` class:

Table 4.10: Showing the Public Events of the ToolTip Class	
Event	Description
<code>Draw</code>	Occurs when the <code>ToolTip</code> is shown and the <code>ownerDraw</code> property is set to <code>true</code> .
<code>Popup</code>	Occurs before a <code>ToolTip</code> is initially displayed. This is the default event for the <code>ToolTip</code> class.

Figure 4.4 shows a `ToolTip` control added to a form:

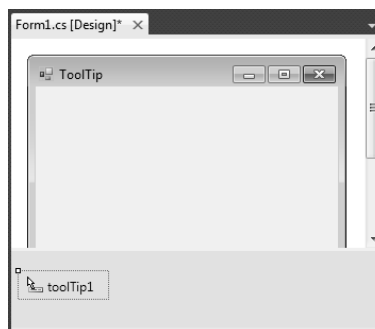


Figure 4.4: Showing the ToolTip Control

In the next section, you learn about the `NotifyIcon` control.

The `NotifyIcon` Control

The `NotifyIcon` control allows you to display an icon in the status notification area of the Windows taskbar (in the indented panel at extreme right in the taskbar), called the Windows system tray. Following is the class hierarchy of the `NotifyIcon` class:

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

You can use the `Icon` property to display the icon for a control. You can also write code in the `DoubleClick` event handler to perform some action at runtime when the user double clicks the icon. You can display or hide the icon by setting the `Visible` property of the `NotifyIcon` control.

Table 4.11 lists the public properties of the `NotifyIcon` class:

Table 4.11: Showing the Public Properties of the <code>NotifyIcon</code> Class	
Property	Description
<code>BalloonTipIcon</code>	Obtains or applies the icon to display on the balloon tip associated with the <code>NotifyIcon</code> control
<code>BalloonTipText</code>	Obtains or applies the text to display on the balloon tip associated with the <code>NotifyIcon</code> control
<code>BalloonTipTitle</code>	Obtains or applies the title of the balloon tip displayed on the <code>NotifyIcon</code> control
<code>ContextMenu</code>	Obtains or sets the context menu for the tray icon
<code>ContextMenuStrip</code>	Obtains or sets the shortcut menu associated with the <code>NotifyIcon</code> control
<code>Icon</code>	Obtains or sets the current icon
<code>Tag</code>	Obtains or sets the <code>ToolTip</code> text displayed when the mouse pointer rests on a notification area icon
<code>Text</code>	Obtains or sets the <code>ToolTip</code> text which is to be displayed when the mouse hovers over a system tray icon
<code>Visible</code>	Specifies whether or not the icon is visible in the Windows System Tray

Table 4.12 lists the public method of the `NotifyIcon` class:

Table 4.12: Showing the Public Method of the <code>NotifyIcon</code> Class	
Method	Description
<code>ShowBalloonTip()</code>	Displays the balloon tip in the taskbar

Table 4.13 lists the public events of the `NotifyIcon` class:

Table 4.13: Showing the Public Events of the <code>NotifyIcon</code> Class	
Event	Description
<code>BalloonTipClicked</code>	Occurs when the balloon tip is clicked
<code>BalloonTipClosed</code>	Occurs when the user closes the balloon tip
<code>BalloonTipShown</code>	Occurs when the balloon tip is displayed on the screen
<code>Click</code>	Occurs when the user clicks the system tray icon
<code>DoubleClick</code>	Occurs when the user double clicks the system tray icon
<code>MouseClick</code>	Occurs when the user clicks a <code>NotifyIcon</code> control with the mouse

Table 4.13: Showing the Public Events of the NotifyIcon Class

Event	Description
MouseDoubleClick	Occurs when the user double-clicks the NotifyIcon control with the mouse
MouseDown	Occurs when the user presses the mouse button on the icon in the system tray
MouseMove	Occurs when the user moves the mouse over the icon in the system tray
MouseUp	Occurs when the user releases the mouse button over the icon in the system tray

You can see a NotifyIcon control in the Windows taskbar in Figure 4.5:



Figure 4.5: Showing the NotifyIcon Control

In this section, you have learned about the NotifyIcon control. In the next section, you learn about the MonthCalendar control.

The MonthCalendar Control

The MonthCalendar control allows you to select a date and time visually. You can limit the date and time that can be selected by setting the MinDate and MaxDate properties. When a new date is selected, a DateSelected event occurs, and when the date is changed, a DateChanged event occurs. You can use the SelectionRange property of the MonthCalendar control to determine the range of the date, such as from 8/4/2010 to 10/4/2010. Following is the class hierarchy for the MonthCalendar class:

```

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

Table 4.14 lists the public properties of the MonthCalendar class:

Table 4.14: Showing the Public Properties of the MonthCalendar Class

Property	Description
AnnuallyBoldedDates	Obtains an array of DateTime objects specifying which days should be displayed in bold
BackColor	Obtains or sets the background color for the control
BackgroundImage	Obtains or sets the background image for the MonthCalendar control
BackgroundImageLayout	Obtains or sets the value that indicates the layout for the background image
BoldedDates	Obtains or sets an array of DateTime objects specifying which dates should be bold
CalendarDimensions	Obtains or sets the number of columns and rows

Table 4.14: Showing the Public Properties of the MonthCalendar Class	
Property	Description
FirstDayOfWeek	Obtains or sets the first day of the week
ForeColor	Obtains or sets the foreground color of the control
ImeMode	Obtains or sets the IME mode supported by this control
MaxDate	Obtains or sets the maximum possible date
MaxSelectionCount	Holds the maximum number of days that can be selected
MinDate	Obtains or sets the minimum possible date
MonthlyBoldedDates	Holds the array of DateTime objects which specify which monthly days to display in bold
Padding	Obtains or sets the space between the edges of the MonthCalendar control and its contents
RightToLeftLayout	Obtains or sets the value indicating whether the control is laid out from right to left
ScrollChange	Obtains the scroll rate
SelectionEnd	Obtains or sets the end date of a selected range
SelectionRange	Obtains the selected range of dates for a MonthCalendar control
SelectionStart	Obtains or sets the starting date of a selected range of dates
ShowToday	Obtains or sets the value indicating whether the date represented by the TodayDate property is shown at the bottom of the control
ShowTodayCircle	Obtains or sets a value indicating if today's date is identified with a circle or square
ShowWeekNumbers	Obtains or sets a value indicating whether the month calendar control displays week numbers
SingleMonthSize	Returns the minimum size to display a month
Size	Obtains or sets the size of the MonthCalendar control
Text	Obtains or sets the text to be displayed on the MonthCalendar control
TitleBackColor	Obtains or sets the back color of the title area of the calendar
TitleForeColor	Obtains or sets the fore color of the title area of the calendar
TodayDate	Obtains or sets today's date
TodayDateSet	Gets a value indicating whether the DateTime property has been set
TrailingForeColor	Obtains or sets a value that indicates the color of the days in months that are not fully displayed in the control

Table 4.15 lists the public methods of the MonthCalendar class:

Table 4.15: Showing the Public Methods of the MonthCalendar Class	
Method	Description
AddAnnuallyBoldedDate()	Adds a day, incremented annually, and displays it in bold font style
AddBoldedDate()	Adds a day and displays it in bold font style
AddMonthlyBoldedDate()	Adds a day, incremented monthly, and displays it in bold font style
GetDisplayRange()	Obtains the date information that specifies the range of displayed dates

Table 4.15: Showing the Public Methods of the MonthCalendar Class	
Method	Description
HitTest()	Determines the element of the calendar at the specified location
RemoveAllAnnuallyBoldedDates()	Removes all annually bolded dates
RemoveAllBoldedDates()	Removes all non-recurring bolded dates
RemoveAllMonthlyBoldedDates()	Removes all monthly bolded dates
RemoveAnnuallyBoldedDate()	Removes the specified date from the calendar's internal list of annually bolded dates
RemoveBoldedDate()	Removes a date from the calendar's internal list of non-recurring dates to display in bold
RemoveMonthlyBoldedDate()	Removes a date from the calendar's internal list of monthly dates to display in bold
SetCalendarDimensions()	Sets the number of columns and rows the calendar
SetDate()	Sets the selected date
SetSelectionRange()	Sets the selected dates to the given range of dates
ToString()	Returns the string that represents the MonthCalendar control
UpdateBoldedDates()	Redisplays the bolded dates

Table 4.16 lists the public events of the MonthCalendar class:

Table 4.16: Showing the Public Events of the MonthCalendar Class	
Event	Description
BackgroundImageChanged	Occurs when the value of the BackgroundImage property changes
BackgroundImageLayoutChanged	Occurs when the value of the BackgroundImageLayout property changes
Click	Occurs when the user clicks the MonthCalendar control
DateChanged	Occurs when the date in the calendar control changes
DateSelected	Occurs when a date is selected in the calendar
DoubleClick	Occurs when the user double- clicks the MonthCalendar control
ImeModeChanged	Occurs when the ImeMode property changes
MouseClick	Occurs when the user clicks the MonthCalendar control with the mouse
MouseDoubleClick	Occurs when the user double- clicks the MonthCalendar control with the mouse
PaddingChanged	Occurs when the Padding property changes
Paint	Occurs when the control is redrawn
RightToLeftLayoutChanged	Occurs when the value of the RightToLeftLayout property changes
TextChanged	Occurs when the value of the Text property changes

Figure 4.6 shows a MonthCalendar control added to a form:

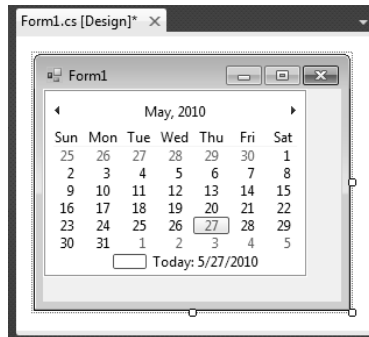


Figure 4.6: Showing the MonthCalendar Control

In the next section, you learn about the `DateTimePicker` control.

The DateTimePicker Control

You can set a date and time in a `DateTimePicker` control by simply editing the displayed values in the control. If you click the arrow in the `DateTimePicker` control, it displays a month calendar, just as a combo box would display a drop-down list. You can then make selections by clicking the required dates in the calendar. When you make a selection, the new selection appears in the text box part of the `DateTimePicker` control, and a `ValueChanged` event occurs.

You can limit the date and time that can be selected in a `DateTimePicker` control by setting the `MinDate` and `MaxDate` properties. You can even change the look of the control by setting the `CalendarForeColor`, `CalendarFont`, `CalendarTitleBackColor`, `CalendarTitleForeColor`, `CalendarTrailingForeColor`, and `CalendarMonthBackground` properties.

Following is the class hierarchy for the `DateTimePicker` class:

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

Table 4.17 describes the public properties of the `DateTimePicker` class:

Table 4.17: Showing the Public Properties of the DateTimePicker Class	
Property	Description
<code>MaximumDateTime</code>	Specifies the maximum date value of the <code>DateTimePicker</code> control. Note that this field is read-only.
<code>MinimumDateTime</code>	Specifies the minimum date value of the <code>DateTimePicker</code> control. Note that this field is read-only.
<code>BackColor</code>	Obtains or sets the value indicating the background color of the <code>DateTimePicker</code> control.
<code>BackgroundImage</code>	Obtains or sets the background image for the <code>DateTimePicker</code> control.
<code>BackgroundImageLayout</code>	Obtains or sets the layout of the background image of the <code>DateTimePicker</code> control.
<code>CalendarFont</code>	Obtains or sets the font style for the calendar.
<code>CalendarForeColor</code>	Obtains or sets the foreground color of the calendar.
<code>CalendarMonthBackground</code>	Obtains or sets the background color of the calendar month.
<code>CalendarTitleBackColor</code>	Obtains or sets the background color of the calendar title.
<code>CalendarTitleForeColor</code>	Obtains or sets the foreground color of the calendar title.

Table 4.17: Showing the Public Properties of the DateTimePicker Class

Property	Description
CalendarTrailingForeColor	Obtains or sets the foreground color of the calendar trailing dates.
Checked	Obtains or sets whether the Value property holds a valid date-time value.
CustomFormat	Obtains or sets a custom date-time format string.
DropDownAlign	Obtains or sets the alignment of the drop-down calendar on the DateTimePicker control.
ForeColor	Obtains or sets the foreground color of the DateTimePicker control.
Format	Obtains or sets the format of dates and times.
MaxDate	Obtains or sets the maximum selectable date and time.
MinDate	Obtains or sets the minimum selectable date and time.
Padding	Obtains or sets the spacing between the contents of the DateTimePicker control.
PreferredHeight	Holds the preferred height of the DateTimePicker control.
RightToLeftLayout	Specifies whether the contents of the DateTimePicker control are laid out from right to left.
ShowCheckBox	Specifies whether a check box should appear to the left of a selected date.
ShowUpDown	Specifies whether an up-down control should be used to adjust date-time values.
Text	Obtains or sets the text in this control.
Value	Obtains or sets the date-time value.

You can find the public method of the DateTimePicker class in Table 4.18:

Table 4.18: Showing the Public Method of the DateTimePicker Class

Method	Description
ToString()	Returns the string that represents the current DateTimePicker control

Table 4.19 lists the public events of the DateTimePicker class:

Table 4.19: Showing the Public Events of the DateTimePicker Class

Event	Description
BackColorChanged	Occurs when the value of the BackColor property changes
BackgroundImageChanged	Occurs when the value of the BackgroundImage property changes
BackgroundImageLayoutChanged	Occurs when the value of the BackgroundImageLayout property changes
Click	Occurs when a user clicks the control
CloseUp	Occurs when the drop-down calendar disappears
DropDown	Occurs when the drop-down calendar appears
ForeColorChanged	Occurs when the value of the ForeColor property changes
FormatChanged	Occurs when the Format property value has changed
MouseClick	Occurs when the mouse is used to click the control
MouseDoubleClick	Occurs when the control is double-clicked with the mouse
PaddingChanged	Occurs when the value of the Padding property changes

Table 4.19: Showing the Public Events of the DateTimePicker Class	
Event	Description
Paint	Occurs when the control is redrawn
RightToLeftLayoutChanged	Occurs when the RightToLeftLayout property changes
TextChanged	Occurs when the value of the Text property changes
ValueChanged	Occurs when the Value property changes

Figure 4.7 shows a DateTimePicker control added to a form:

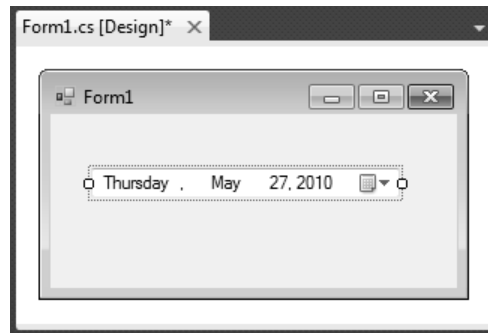


Figure 4.7: Showing the DateTimePicker Control

In the next section, you learn about the Timer control.

The Timer Control

The Timer controls allow you to create periodic events. At design time, Timer controls appear in the component tray under the form in which they are added. Following is the class hierarchy of the Timer class:

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

Table 4.20 lists the public properties of the Timer class:

Table 4.20: Showing the Public Properties of the Timer Class	
Property	Description
Enabled	Specifies whether or not the timer is running
Interval	Obtains or sets the time (in milliseconds) between timer ticks
Tag	Obtains or sets the arbitrary string that represents some sort of user state

Table 4.21 lists the public methods of the Timer class:

Table 4.21: Showing the Public Methods of the Timer Class	
Method	Description
Start()	Starts the timer
Stop()	Stops the timer

Table 4.22 lists the public event of the Timer class:

Table 4.22: Showing the Public Event of the Timer Class

Event	Description
Tick	Occurs when the timer interval has elapsed and the timer is enabled

Windows timers are designed for a single-threaded Windows Forms environment. You can easily set how often you want the timer to generate Tick events by setting the Interval property (in milliseconds, i.e., one thousand of a second). Each time a Tick event occurs, you can execute the code specified in a handler for this event.

Figure 4.8 shows a Timer control added to a form:

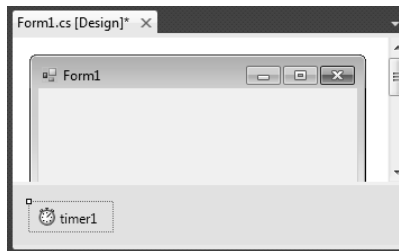


Figure 4.8: Showing the Timer Control

Now, let's learn about the ProgressBar control in the next section.

The ProgressBar Control

The ProgressBar control is used to indicate the progress of any operation. It shows a bar that fills in from left to right as an operation progresses. Following is the inheritance hierarchy for the ProgressBar class:

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

The main properties of a ProgressBar control are Value, Minimum, and Maximum. You can use the Minimum and Maximum properties to set the minimum and maximum values the progress bar can display. The Value property is used to change the display of the ProgressBar control. For example, if the Maximum property is set to 100, the Minimum property is set to 0, and the Value property is set to 60, the ProgressBar control displays the progress of the specified actions by using 6 rectangles.

Table 4.23 describes the public properties of the ProgressBar class:

Table 4.23: Showing the Public Properties of the ProgressBar Class

Property	Description
AllowDrop	Overrides the Control.AllowDrop property
BackgroundImage	Obtains or sets the background image for the ProgressBar control
BackgroundImageLayout	Obtains or sets the layout of the background image of the ProgressBar control
CausesValidation	Obtains or sets the value that indicates whether the control, when it receives focus, causes the validation to be performed on any controls that require validation
Font	Obtains the font of the progress bar's text
ImeMode	Obtains or sets the IME for the ProgressBar control
MarqueeAnimationSpeed	Obtains or sets the time period (in milliseconds) that progress block takes to scroll across the progress bar

Table 4.23: Showing the Public Properties of the ProgressBar Class	
Property	Description
Maximum	Obtains the progress bar's maximum value
Minimum	Obtains the progress bar's minimum value
Padding	Obtains or sets the space between the edges of a ProgressBar control and its contents
RightToLeftLayout	Obtains or sets the value that indicates whether the ProgressBar control and any text it contains is displayed from right to left
Step	Obtains the value by which the <code>PerformStep()</code> method increases a progress bar's value
Style	Obtains or sets the manner in which the progress should be indicated on the progress bar
Value	Obtains the current value of the progress bar

Table 4.24 lists the public methods of the `ProgressBar` class:

Table 4.24: Showing the Public Methods of the ProgressBar Class	
Method	Description
<code>Increment()</code>	Increments the position of the <code>ProgressBar</code> control by a specified amount
<code>PerformStep()</code>	Increments the value of the <code>ProgressBar</code> control as specified by the <code>Step</code> property
<code>ResetForeColor()</code>	Resets the fore color to its default value
<code>ToString()</code>	Returns a string that represents the <code>ProgressBar</code> control

Table 4.25 lists the public events of the `ProgressBar` class:

Table 4.25: Showing the Public Events of the ProgressBar Class	
Event	Description
<code>BackgroundImageChanged</code>	Occurs when the value of the <code>BackgroundImage</code> property changes
<code>BackgroundImageLayoutChanged</code>	Occurs when the value of the <code>BackgroundImageLayout</code> property changes
<code>CausesValidationChanged</code>	Occurs when the value of the <code>CausesValidation</code> property changes
<code>DoubleClick</code>	Occurs when the user double-clicks the control
<code>Enter</code>	Occurs when the focus enters the <code>ProgressBar</code> control
<code>FontChanged</code>	Occurs when the value of the <code>Font</code> property changes
<code>ImeModeChanged</code>	Occurs when the value of the <code>ImeMode</code> property changes
<code>KeyDown</code>	Occurs when the user presses a key while the control has focus
<code>KeyPress</code>	Occurs when the user presses a key while the control has focus
<code>KeyUp</code>	Occurs when the user releases a key while the control has focus
<code>Leave</code>	Occurs when focus leaves the <code>ProgressBar</code> control
<code>MouseDoubleClick</code>	Occurs when the user double-clicks the control
<code>PaddingChanged</code>	Occurs when the value of the <code>Padding</code> property changes
<code>Paint</code>	Occurs when the <code>ProgressBar</code> control is drawn
<code>RightToLeftLayoutChanged</code>	Occurs when the value of the <code>RightToLeftLayout</code> property changes

Table 4.25: Showing the Public Events of the ProgressBar Class

Event	Description
TabStopChanged	Occurs when the value of the TabStop property changes
TextChanged	Occurs when the value of the Text property changes

Figure 4.9 shows a `ProgressBar` control added to a form:

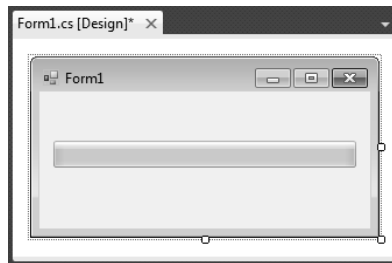


Figure 4.9: Showing the ProgressBar Control

Let's now explore the practical implementation of all the controls that we have learned in the *Immediate Solutions* section.

Immediate Solutions

Using the SplitContainer Control

The `SplitContainer` class provides a pre-structured frame of two resizable panels, split across the width or length of the container by a movable bar. The `SplitContainer` control provides all the usual elements and steps involved in setting up a splitter based User Interface (UI) into one control. Now, let's create a new Windows Forms application, named `SplitContainers` (also available in the CD), to learn how to work with the `SplitContainer` control, and perform the following steps:

1. Add a `ToolStrip` control and a `SplitContainer` control on the `Form1` form of the `SplitContainers` application.
2. Add three `ToolStripButton` controls to the `ToolStrip` control on the `Form1` form and set their `Text` property to `Collapse/Expand ListBox`, `Collapse/Expand TreeView`, and `Split Vertically/Horizontally`, respectively.
3. Add a `TreeView` control in the left pane of the `SplitContainer` control and add a `ListBox` control in the right pane of the `SplitContainer` control, as shown in Figure 4.10:

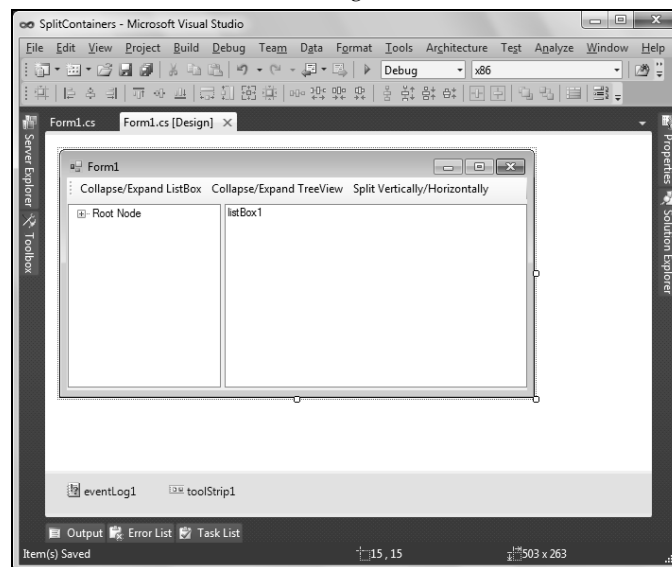


Figure 4.10: Showing the Design View of the SplitContainers Application

4. Set the `Dock` property of the `TreeView` control and the `ListBox` control to `Fill`.
Now, using the `SplitContainers` application, you need to perform the following two tasks:

- ☐ Creating splitter based UIs with `SplitContainer` control
- ☐ Changing the orientation of `SplitContainer` controls

Let's learn to perform these tasks one by one.

Creating Splitter based UIs with SplitContainer Control

The `SplitContainers` application is a sort of a mini file explorer application that displays a folder list on its left panel, and the files present inside a folder on its right panel. Perform the following steps to create a splitter based UI:

1. Double-click the first `ToolStripButton` control, `toolStripButton1`, on the Form designer to generate its `Click` event and add the code, shown in Listing 4.1, to the `toolStripButton1_Click` event:

Listing 4.1: Showing the Code for the `toolStripButton1_Click` Event of the `toolStripButton1` Control

```
private void toolStripButton1_Click(object sender, EventArgs e)
{
    if(splitContainer1.Panel1Collapsed == true)
        splitContainer1.Panel1Collapsed = false;
    else
        splitContainer1.Panel1Collapsed = true;
}
```

2. Double-click the second `ToolStripButton` control, `toolStripButton2`, on the Form designer to generate its Click event and add the code, shown in Listing 4.2, to the `toolStripButton2_Click` event:

Listing 4.2: Showing the Code for the `toolStripButton2_Click` Event of the `toolStripButton2` Control

```
private void toolStripButton2_Click(object sender, EventArgs e)
{
    if(splitContainer1.Panel2Collapsed == true)
        splitContainer1.Panel2Collapsed = false;
    else
        splitContainer1.Panel2Collapsed = true;
}
```

3. Add the code in the `AfterSelect` event of the `TreeView` control, as shown in Listing 4.3:

Listing 4.3: Showing the Code for the `AfterSelect` Event of the `treeView1` Control

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    listBox1.Items.Clear();
    for (int i = 0; i < e.Node.Nodes.Count; i++)
        listBox1.Items.Add(e.Node.Nodes[i].Text);
}
```

4. Press the F5 key on the keyboard to run the application. The output of the application appears, as shown in Figure 4.11:

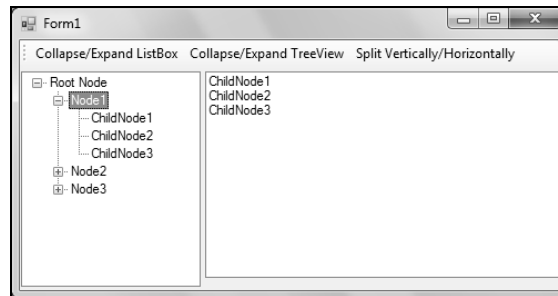


Figure 4.11: Showing the Output of the SplitContainers Application

5. Click the `Collapse/Expand ListBox` button to collapse the `TreeView` control, as shown in Figure 4.12:

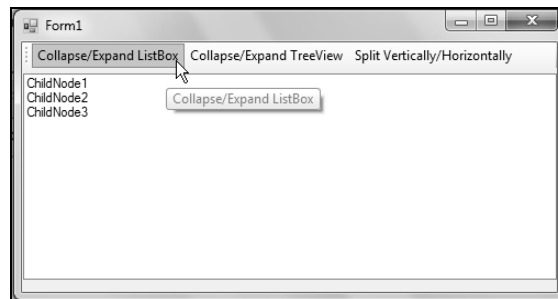


Figure 4.12: Showing the Collapsed Left Panel of the SplitContainer Control

You can again click the Collapse/Expand ListBox button to expand the TreeView control.

- Click the Collapse/Expand TreeView button to collapse the list box, as shown in Figure 4.13:

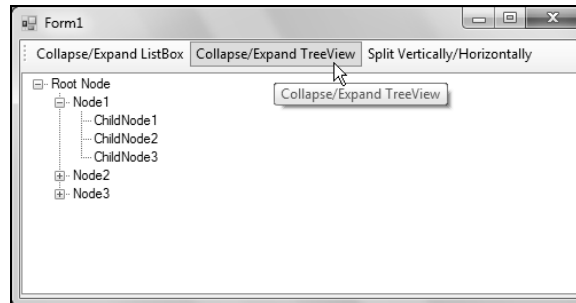


Figure 4.13: Showing the Collapsed Right Panel of the SplitContainer Control

Changing the Orientation of SplitContainer Controls

You can change the orientation of the SplitContainer control by setting its Orientation property to either Horizontal or Vertical. To change the orientation of the SplitContainer control, perform the following steps:

- Double-click the third ToolStripButton control, toolStripButton3, on the Form designer to generate its Click event and add the code, shown in Listing 4.4, to the toolStripButton3_Click event:

Listing 4.4: Showing the Code for the toolStripButton3_Click Event of the toolStripButton3 Control

```
private void toolStripButton3_Click(object sender, EventArgs e)
{
    if (splitContainer1.Orientation == Orientation.Horizontal)
        splitContainer1.Orientation = Orientation.Vertical;
    else
        splitContainer1.Orientation = Orientation.Horizontal;
}
```

- Press the F5 key on the keyboard to run the application and click the Split Vertically/Horizontally button to change the orientation of the SplitContainer control from vertical to horizontal, as shown in Figure 4.14:

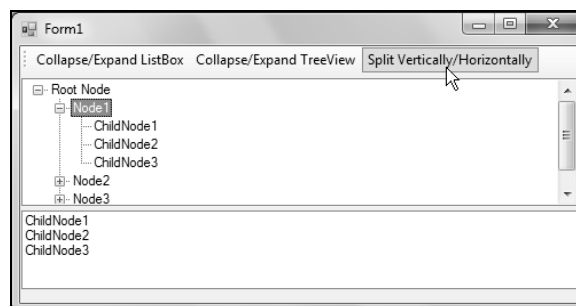


Figure 4.14: Changing the Orientation of the SplitContainer Control

Using the ScrollBar Controls

As already learned, ScrollBars help in navigating through a large list of items or information by using the HScrollBar and VScrollBar controls. Now, let's create a new Windows Forms application, named ScrollBarControl (also available in the CD), to learn how to work with the ScrollBar control.

Add two HScrollBar controls, a VScrollBar control, a Label control, and a TextBox control from the Toolbox to the Form designer, as shown in Figure 4.15:

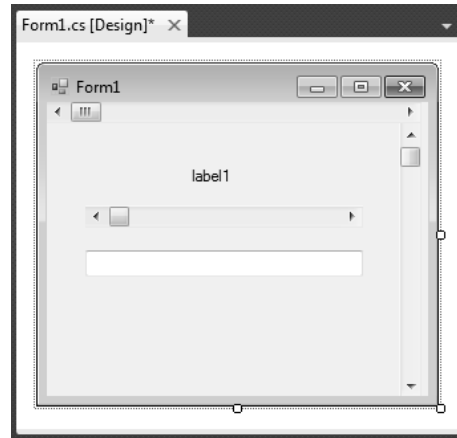


Figure 4.15: Showing the Design View of the ScrollBarControl Application

Now, using the `ScrollBarControl` application, you need to perform the following five tasks, one by one:

- ❑ Specifying the range of the `ScrollBar` controls
- ❑ Specifying the movement of the `ScrollBar` controls on clicking
- ❑ Setting and retrieving the current value of the `ScrollBar` controls
- ❑ Handling the `Scroll` event of the `ScrollBar` controls
- ❑ Displaying images with the `ScrollBar` controls

Let's learn to perform these tasks one by one.

Specifying the Range of the ScrollBar Controls

After placing the scroll bar on the form, you need to set its range of possible values, which is 0–100 by default. The minimum value, a scroll bar can be set to, is stored in its `Minimum` property, and the maximum value in the `Maximum` property. To specify the range of the `ScrollBar` control, double-click the `Form1` to generate the `Load` event of the form and add the following code snippet on the `Form1_Load` event:

```
hScrollBar1.Minimum = 0;  
hScrollBar1.Maximum = 100;
```

In the preceding code snippet, we are setting the `Minimum` and `Maximum` properties of the `HScrollBar` control to 0 and 100, respectively.

Specifying the Movement of the ScrollBar Controls on Clicking

There are three ways to change the setting of a scroll bar: by moving the scroll box (also called the thumb), by clicking the area of the scroll bar between the scroll box and an arrow button, and by clicking an arrow button. The scroll box moves by the value of the `SmallChange` property when the user clicks an arrow button. Similarly, the scroll box moves by the value of the `LargeChange` property when the user clicks the area of the scroll bar between the scroll box and an arrow button. To specify the movement of the `ScrollBar` control, add the highlighted code, shown in Listing 4.5, to the `Form1_Load` event:

Listing 4.5: Showing the Code for Specifying the Movement of the ScrollBar Control

```
private void Form1_Load(object sender, EventArgs e)  
{  
    hScrollBar1.Minimum = 0;  
    hScrollBar1.Maximum = 100;  
    hScrollBar1.SmallChange = 1;  
    hScrollBar1.LargeChange = 10;  
}
```

In Listing 4.5, we are setting the `SmallChange` and `LargeChange` properties of the `HScrollBar` control to 1 and 10, respectively.

Setting and Retrieving the Current Value of the ScrollBar Controls

You can use the `Value` property to set a scroll bar's setting. The `Value` property holds values that can be in the range spanned by the values in the `Minimum` and `Maximum` properties. To set and retrieve the current value of the `ScrollBar` control, perform the following steps:

1. Add the highlighted code, shown in Listing 4.6, to the `Form1_Load` event:

Listing 4.6: Showing the Code for Setting and Retrieving the Current Value of the ScrollBar Controls

```
private void Form1_Load(object sender, EventArgs e)
{
    hScrollBar1.Minimum = 0;
    hScrollBar1.Maximum = 100;
    hScrollBar1.LargeChange = 10;
    hScrollBar1.SmallChange = 1;
    hScrollBar2.Value = 20;
}
```

In Listing 4.6, we are setting the `Value` property of the `hScrollBar2` control to 20.

2. Double-click the `hScrollBar2` control to generate its `Scroll` event and add the following code snippet on the `hScrollBar2_Scroll` event:

```
textBox1.Text="Scroll position: "+e.NewValue;
```

In the preceding code snippet, we are displaying the current position of the scroll box in the text box when the user scrolls the `hScrollBar2` control at the runtime.

3. Press the F5 key on the keyboard to run the application and move the scroll box of the `hScrollBar2` control to the right. This displays the value of the scroll box position in the text box, as shown in Figure 4.16:

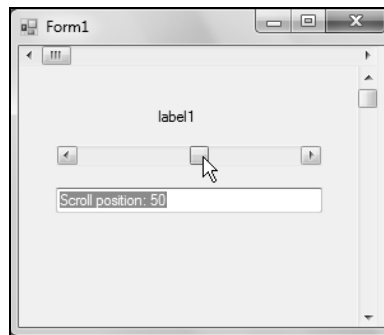


Figure 4.16: Displaying Scroll Position Value for the Scroll Bar

Handling the Scroll Event of the ScrollBar Controls

There are two events that you can use in scroll bars, `Scroll` and `ValueChanged`. The `Scroll` event occurs when the scroll box is moved either with the mouse or keyboard; and the `ValueChanged` event occurs when the `Value` property changes, either through the user's actions or by the code. To handle the `Scroll` event of the `ScrollBar` control, perform the following steps:

1. Add the code, shown in Listing 4.7, to the `Scroll` event of the `hScrollBar1` control:

Listing 4.7: Showing the Code for Handling the `Scroll` Event of the `hScrollBar1` Control

```
private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
{
    label1.Location = new Point(e.NewValue * this.Size.Width / 100, label1.Location.Y);
    label1.Text = "Move to the desk!";
}
```

2. Add the code, shown in Listing 4.8, to the `Scroll` event of the `vScrollBar1` control:

Listing 4.8: Showing the Code for Handling the `Scroll` Event of the `vScrollBar1` Control

```
private void vScrollBar1_Scroll(object sender, ScrollEventArgs e)
{
}
```

```
label1.Location = new Point(label1.Location.X, e.NewValue * this.Size.Height / 100);  
label1.Text = "Move to the desk!";  
}
```

3. Press the F5 key on the keyboard to run the application and move the scroll box of the `hScrollBar1` control to the right. The `Label` control also moves to the right, as shown in Figure 4.17:

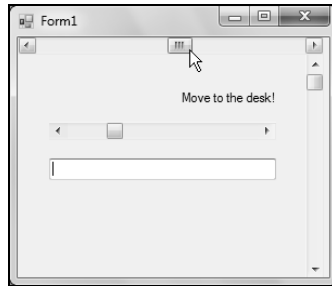


Figure 4.17: Handling the Scroll Event of the First HScrollBar Control

4. Move the scroll box of the `vScrollBar1` control to the bottom. The `Label` control also moves to the bottom, as shown in Figure 4.18:

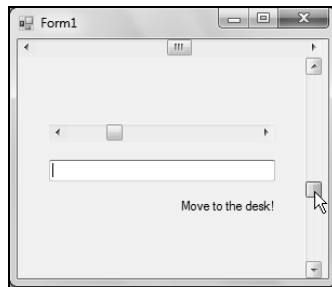


Figure 4.18: Handling the Scroll Event of the VScrollBar Control

Displaying Images with the ScrollBar Controls

Sometimes, you may need to display an image in a `PictureBox` control that is may not fit in the `PictureBox` control. In such a case, you can add scroll bars to the `PictureBox` control so that you can scroll the image at runtime. Now, let's create a new Windows Forms application, named `ScrollImageControl` (also available in the CD), to learn how to work with the `ScrollBar` control, and perform the following steps:

1. Add a `PictureBox` control, an `HScrollBar` control, a `VScrollBar` control, an `OpenFileDialog` control, and a `Button` control from the Toolbox to the Form designer, as shown in Figure 4.19:

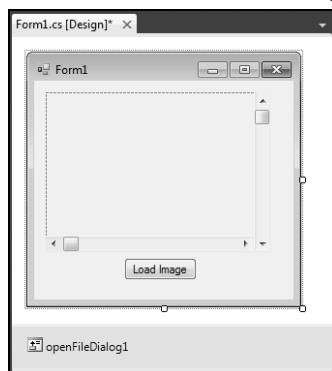


Figure 4.19: Adding Scroll Bars to a PictureBox Control

2. Change the Text property of the Button control to Load Image (Figure 4.19).
3. Generate the Load event of the Form1 by double-clicking it and add the following code snippet on the Form1_Load event:

```
hScrollBar1.Scroll += new ScrollEventHandler(ScrollBars_Scroll);
vScrollBar1.Scroll += new ScrollEventHandler(ScrollBars_Scroll);
```

In the preceding code snippet, we are adding Scroll event to both horizontal and vertical scroll bars.

4. Add the code shown in Listing 4.9 in the Form1.cs file:

Listing 4.9: Showing the Code for Displaying Images with ScrollBar Controls

```
private void ShowScrollBars()
{
    vScrollBar1.Visible = true;
    hScrollBar1.Visible = true;
    if (pictureBox1.Height > pictureBox1.Image.Height)
    {
        vScrollBar1.Visible = false;
    }
    if (pictureBox1.Width > pictureBox1.Image.Width)
    {
        hScrollBar1.Visible = false;
    }
}
```


In Listing 4.9, we have a method, ShowScrollBars(), which makes the vertical and horizontal scroll bars visible or invisible on the basis of whether or not the height and width of the PictureBox control is greater than that of the image.

5. Add the code, shown in Listing 4.10, to the Click event of the Button control to display the images with ScrollBar controls:

Listing 4.10: Showing the Code for Displaying Images with ScrollBar Controls

```
private void button1_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() != DialogResult.Cancel)
    {
        pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);
        hScrollBar1.Maximum = pictureBox1.Image.Width - pictureBox1.Width;
        vScrollBar1.Maximum = pictureBox1.Image.Height - pictureBox1.Height;
        ShowScrollBars();
    }
}
```

In Listing 4.10, we are setting the Image property of the PictureBox control to the image selected by the user in the OpenFileDialog control. We are setting the Maximum property of both horizontal and vertical scroll bars. At last, we are calling the ShowScrollBars() method.

6. Select both the scroll bars and press the F4 key on the keyboard to open the Properties window.
7. Click the Events button  and enter ScrollBars_Scroll beside the Scroll event and press the Enter key on the keyboard to generate the Scroll event.
8. Add the code, shown in Listing 4.11, to the ScrollBars_Scroll event:

Listing 4.11: Showing the Code for the ScrollBars_Scroll Event of the ScrollBar Control

```
public void ScrollBars_Scroll(object sender, ScrollEventArgs e)
{
    Graphics graphics=pictureBox1.CreateGraphics();
    graphics.DrawImage(pictureBox1.Image, new Rectangle(0,0,pictureBox1.Width-
        hScrollBar1.Height, pictureBox1.Height-vScrollBar1.Width),new
        Rectangle(hScrollBar1.Value, vScrollBar1.Value, pictureBox1.Width-hScrollBar1.Height,
        pictureBox1.Height-vScrollBar1.Width),GraphicsUnit.Pixel);
}
```

In Listing 4.11, we are drawing an image using an object of the Graphics class.

9. Press the F5 key on the keyboard to run the application and load an image in the PictureBox control by clicking the Load Image button. The image displays in the PictureBox control, as shown in Figure 4.20:

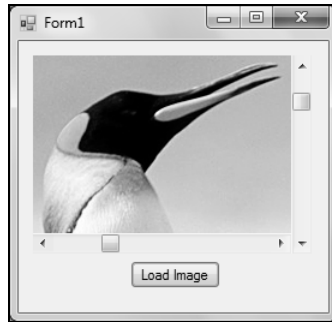


Figure 4.20: Scrolling an Image

Now, you can scroll the image using the horizontal and vertical scroll bars (Figure 4.20).

In the next section, you learn to work with the `TrackBar` control.

Using the `TrackBar` Control

As discussed in the *In Depth* section of this chapter, track bars work similar to scroll bars, but they have a different appearance, resembling the controls you would find on a stereo. Similar to scroll bars, the `Value` property holds the track bar's current setting, and you can handle the `Scroll` and `ValueChanged` events to work with this control. Track bars can also display ticks, giving the user an idea of the scale used to set the control's value.

Now, let's create a new Windows Forms application, named `TrackBars` (also available in the CD), to learn how to work with the `TrackBar` control. Add a `TrackBar` control, a `TextBox` control, and two `Button` controls from the Toolbox to the Form designer, as shown in Figure 4.21:

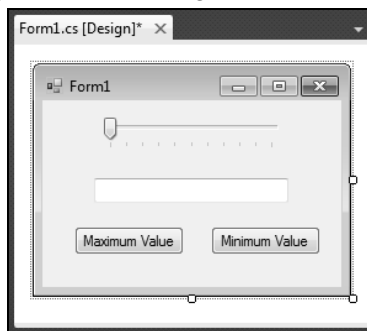


Figure 4.21: Showing the Design View of the `TrackBars` Application

Now, you need to perform the following two tasks:

- ☐ Setting and retrieving the maximum and minimum value for the slider
- ☐ Handling track bar events

Let's discuss these tasks in detail, one by one.

Setting and Retrieving the Maximum and Minimum Value for the Slider

You can set the maximum and minimum values for the slider of the `TrackBar` control by setting the `Maximum` and `Minimum` property of the `TrackBar` control. To set and retrieve the maximum and minimum values for the slider of the `TrackBar` control, perform the following steps:

1. Add the following code snippet on the Load event of the form:

```
private void Form1_Load(object sender, EventArgs e)
{
    trackBar1.Maximum = 100;
}
```

```
trackBar1.Minimum = 10;
}
```

In the preceding code snippet, we are setting the Maximum and Minimum properties of the `TrackBar` control, `trackBar1`, to 100 and 10, respectively.

You can also retrieve the maximum and minimum values that are set for the track bar. We use the message box to display the minimum and maximum values for the track bars.

2. Add the code on the Click event of the Maximum Value button to display the maximum value of the `TrackBar` control, as shown in the following code snippet:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(trackBar1.Maximum.ToString());
}
```

In the preceding code snippet, we are displaying the maximum value for the slider of the `TrackBar` control, `trackBar1`, using a message box, when the user clicks the first button at runtime.

3. Add the code on the Click event of the Minimum Value button to display the minimum value of the `TrackBar` control, as shown in the following code snippet:

```
private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show(trackBar1.Minimum.ToString());
}
```

In the preceding code snippet, we are displaying the minimum value for the slider of the `TrackBar` control, `trackBar1`, using a message box, when the user clicks the second button at runtime.

4. Press the F5 key on the keyboard to run the application and click the Maximum Value button. A message box, displaying the maximum value for the `TrackBar` control appears, as shown in Figure 4.22:

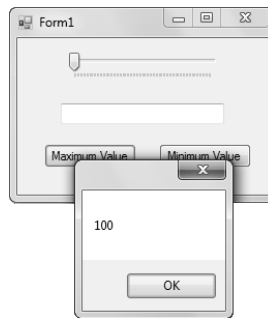


Figure 4.22: Displaying Maximum Value of the `TrackBar` Control

5. Click the OK button to close the message box (Figure 4.22).
6. Click the Minimum Value button to view the minimum value of the `TrackBar` control, as shown in Figure 4.23:

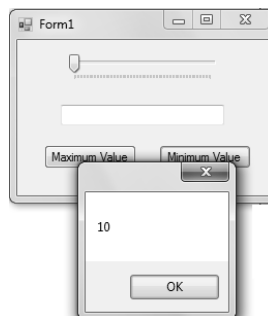


Figure 4.23: Displaying Minimum Value of the `TrackBar` Control

In the next section, we learn to handle the track bar events.

Handling Track Bar Events

Similar to the scroll bars, track bars also have two events, `Scroll` and `ValueChanged`. You can get the current value of the track bar by using the `Value` property. To handle the `Scroll` event of the `TrackBar` control, add the code in the `Scroll` event of the `TrackBar` control, as shown in the following code snippet:

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    textBox1.Text = "Track bar value: " + trackBar1.Value;
}
```

In the preceding code snippet, we are displaying the current value for the slider of the `TrackBar` control in a text box, when the user moves the slider of the `TrackBar` control at runtime.

Now, run the application and move the slider of the track bar to the right. This displays the current value for the slider of the track bar in the text box, as shown in Figure 4.24:

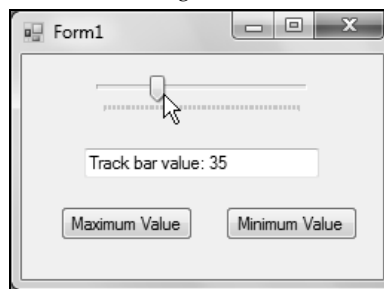


Figure 4.24: Displaying Current Value of the TrackBar Control

Using the ToolTip Control

As discussed in the *In Depth* section of this chapter, tool tips are windows that display explanatory text when the mouse hovers over a control or form. Now, let's create a new Windows Forms application, named `ToolTips` (also available in the CD), to learn how to work with the `ToolTip` control. Then add a `ToolTip` control and a `Button` control from the Toolbox to the Form designer, as shown in Figure 4.25:

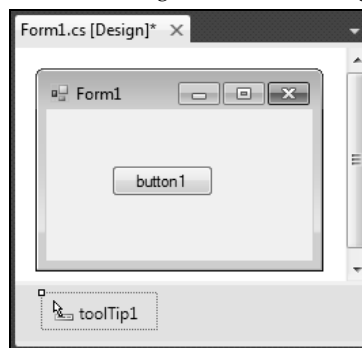


Figure 4.25: Showing the Design View of the ToolTips Application

Now, you need to perform the following two tasks:

- ☐ Setting the tool tips for controls
- ☐ Setting the delay in displaying the tooltip control

Let's discuss these tasks in detail, one by one.

Setting the Tool Tips for Controls

When you add tool tips to a Windows Forms project, they appear in a component tray below the form you are adding them to. You can associate a `ToolTip` control with a control using the `SetToolTip()` method of the `ToolTip` control. To set the tooltip for a `Button` control, generate the `Load` event of the `Form1` by double-clicking it and add the following code snippet on the `Form1_Load` event:

```
private void Form1_Load(object sender, EventArgs e)
{
    tooltip1.SetToolTip(button1, "This is a button");
}
```

In the preceding code snippet, we are associating the `ToolTip` control, `tooltip1`, with the `Button` control, `button1`, using the `SetToolTip()` method of the `ToolTip` control.

Now, press the F5 key on the keyboard to run the application and move the mouse pointer over the button. This displays a tooltip showing the text, *This is a button*, as shown in Figure 4.26:



Figure 4.26: Displaying a Tooltip on a Button

Setting the Delay in Displaying the ToolTip Control

You can also set the delay for the tool tip to be displayed on the specified control. This can be done at both design time and run time by setting the `InitialDelay`, `ReshowDelay`, `AutoPopDelay`, and `AutomaticDelay` properties of the `ToolTip` control. To set the delay for the `ToolTip` control, you need to add the highlighted code, shown in Listing 4.12, on the `Load` event of the form:

Listing 4.12: Showing the Code for Setting the Delay for Displaying the Tool Tip

```
private void Form1_Load(object sender, EventArgs e)
{
    tooltip1.SetToolTip(button1, "This is a button");
    tooltip1.InitialDelay = 8000;
    tooltip1.ReshowDelay = 110;
    tooltip1.AutoPopDelay = 1200;
    tooltip1.AutomaticDelay = 6000;
}
```

In Listing 4.12, we are setting the `InitialDelay`, `ReshowDelay`, `AutoPopDelay`, and `AutomaticDelay` properties of the `ToolTip` control, `tooltip1`.

Using the NotifyIcon Control

As mentioned in the *In Depth* section of this chapter, the `NotifyIcon` control is very useful for processes that run in the background and do not have their own windows, although they may be part of applications that display windows. Now, let's create a new Windows Forms application, named `NotifyIconControl` (also available in the CD), to learn how to work with the `NotifyIcon` control. Then, add a `NotifyIcon` control and a `Button` control from the Toolbox on the `Form1` and set the `Text` property of the `Button` control to `Click` to hide the icon, as shown in Figure 4.27:

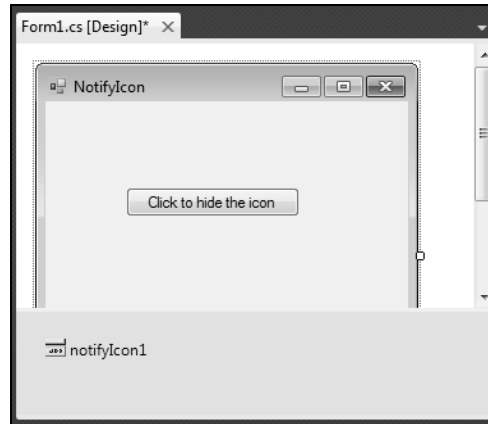


Figure 4.27: Showing the Design View of the NotifyIconControl Application

Now, you need to perform the following two tasks:

- ☐ Creating a notify icon using the icon designer
- ☐ Hiding the notify icon

Let's discuss these tasks in detail, one by one.

Creating a Notify Icon Using the Icon Designer

To create a notify icon component, you need to assign an icon file to the `Icon` property of the `NotifyIcon` control. You can create new icons with an icon designer. To use the icon designer, perform the following steps:

1. Click **Project**→**Add New Item** on the menu bar to open the Add New Item dialog box. The Add New Item dialog box appears, as shown in Figure 4.28:

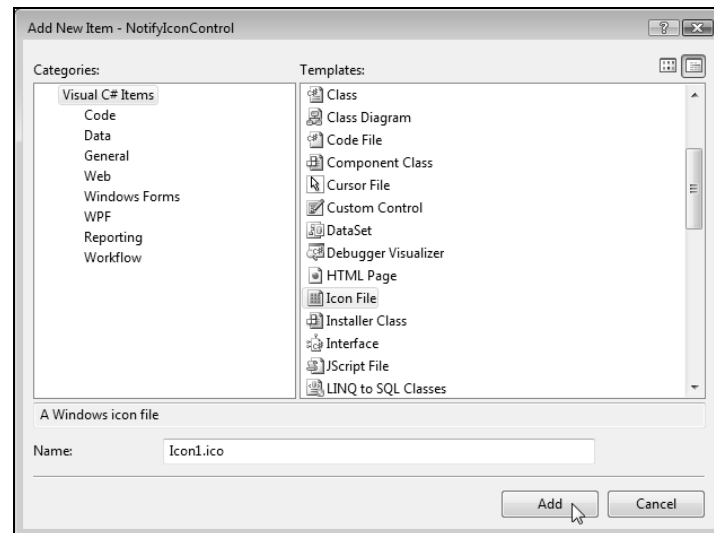


Figure 4.28: Selecting an Icon File in the Add New Item Dialog Box

2. Select the **Icon File** template from the **Templates** pane of the Add New Item dialog box (Figure 4.28).
3. Click the **Add** button to add a new icon file to your current project. This adds a new icon file to your current project, as shown in Figure 4.29:

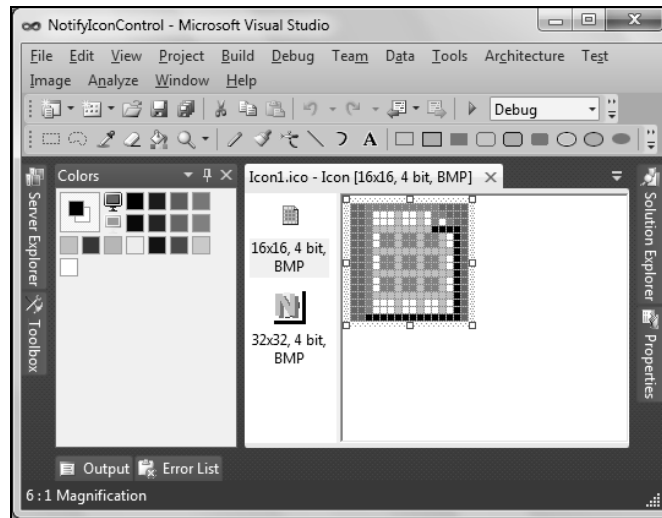


Figure 4.29: Using an Icon Designer

To design your icon, you can use the tools you see in the toolbar below the standard toolbar. We have drawn a basic icon, and saved it as `icon1.ico` in the `NotifyIconControl` folder.

Hiding the Notify Icon

You can display and hide the `NotifyIcon` control by setting its `Visible` property to `true` or `false`. To hide the notify icon, you need to add the code on the Click event of the button control, as shown in the following code snippet:

```
private void button1_Click(object sender, EventArgs e)
{
    notifyIcon1.Visible = false;
}
```

In the preceding code snippet, we are setting the `Visible` property of the `NotifyIcon` control to `false`, when the user clicks the button at runtime.

NOTE

You need to set the `Icon` property of the `NotifyIcon` control in the Properties window.

Now, run the application by pressing the F5 key on the keyboard. The notify icon is added to the notification area of your taskbar, as shown in Figure 4.30:



Figure 4.30: Adding a Notify Icon to the Taskbar

Clicking the Click to hide the icon button on the form hides the notify icon from the taskbar.

Using the MonthCalendar and DateTimePicker Controls

As learned earlier, you can use the `MonthCalendar` control to allow the user to select days of a month, and the `DateTimePicker` control to allow the user to select both date and time. Now, let's create a new Windows Forms application, named `DateTimePickerControl` (also available in the CD), to learn how to work with the `MonthCalendar` and `DateTimePicker` controls. Then, add a `MonthCalendar` control, a `DateTimePicker` control, a `TextBox` control, and two `Button` controls from the Toolbox to the Form designer and set the `Text` property of the two `Button` controls to `Setting a Date in Date Time Picker` and `Retrieving a Date from Date Time Picker` respectively, as shown in Figure 4.31:

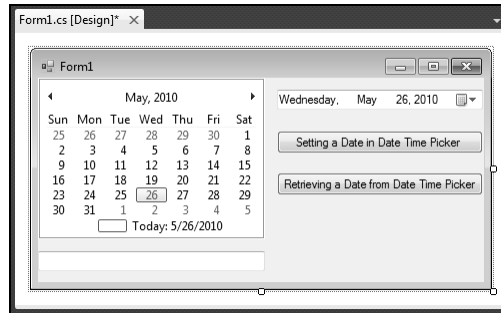


Figure 4.31: Showing the Design View of the DateTimePickerControl Application

Now, using the DateTimePickerControl application, you need to perform the following three tasks:

- ❑ Handling the events of the MonthCalendar control
- ❑ Setting and retrieving a date using the DateTimePicker control
- ❑ Specifying the format for DateTimePicker controls

Let's discuss these tasks one by one in detail.

Handling the Events of the MonthCalendar Control

There are two main events in MonthCalendar controls—`DateTimeChanged` (the default event), which occurs when the date in the control changes (either through user actions or in code); and `DateSelected`, which occurs when the user selects a new date.

As discussed in the *In Depth* section of this chapter, you can select the entire ranges of dates in MonthCalendar controls. To handle such ranges, you can use the `SelectionStart`, `SelectionEnd`, and `SelectionRange` properties of the MonthCalendar control. To handle the `DateSelected` event of the MonthCalendar control, generate the `DateSelected` event of the MonthCalendar control and add the code in the `monthCalendar1_DateSelected` event, as shown in the following code snippet:

```
private void monthCalendar1_DateSelected(object sender, DateRangeEventArgs e)
{
    textBox1.Text = "Day of the month selected: " + monthCalendar1.SelectionRange.Start.Day;
}
```

In the preceding code snippet, we are displaying the selected date and time in a text box, when the user selects a date on the MonthCalendar control at runtime.

Now, run the application and select a date on the MonthCalendar control. The day of the selected month is displayed in the text box, as shown in Figure 4.32:

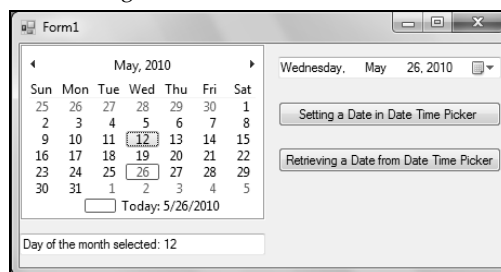


Figure 4.32: Selecting a Date in the MonthCalendar Control

Setting and Retrieving a Date using the DateTimePicker Control

You can set a date in the DateTimePicker control and retrieve the date from the DateTimePicker control using the `Value` property of this control. To set or retrieve a date using the DateTimePicker control, perform the following steps:

1. Double-click the Button control, `button1`, on the Form designer to open the Code window and add the code on the `button1_Click` event, as shown in the following code snippet:

```
private void button1_Click(object sender, EventArgs e)
{
    dateTimePicker1.Value = new DateTime(2010, 10, 20);
}
```

In the preceding code snippet, we are setting a date, October 20, 2010, in the `DateTimePicker` control using its `Value` property, when the user clicks the Button control, `button1`, at runtime.

2. Double-click the Button control, `button2`, on the Form designer to open the Code window and add the code on the `button2_Click` event, as shown in the following code snippet:

```
private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show(dateTimePicker1.Value.ToString());
}
```

In the preceding code snippet, we are retrieving and displaying the date of the `DateTimePicker` control in a message box, when the user clicks the Button control, `button2`, at runtime.

3. Press the F5 key on the keyboard to run the application and click the Setting a Date in Date Time Picker button. This sets a date, October 20, 2010, in the date time picker, as shown in Figure 4.33:

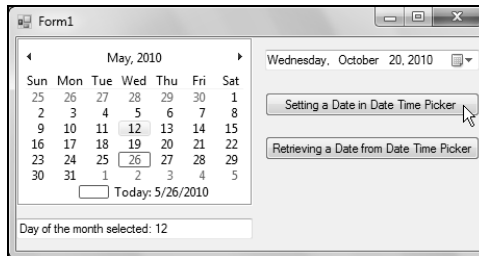


Figure 4.33: Setting a Date in the DateTimePicker Control

4. Click the Retrieving a Date from Date Time Picker button. This displays the date of the date time picker in a message box, as shown in Figure 4.34:

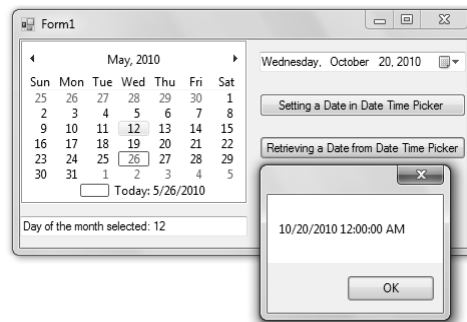


Figure 4.34: Retrieving the Date of the DateTimePicker Control

In this section, we learned how to set and retrieve a date using the `DateTimePicker` control.

Specifying the Format for DateTimePicker Controls

You can set the format for the date and time displayed in the `DateTimePicker` control to customize the format according to your requirement. For example, you can display only a date, time, or both. To set a custom format, set the `Format` property of the `DateTimePicker` control to `Custom`, and then you assign a custom format string to the `CustomFormat` property. You create a custom format string using the following items:

- ❑ **d**—Represents the one-or two-digit day
- ❑ **dd**—Represents the two-digit day. Note that single-digit day values are preceded by a zero

- ❑ **ddd** – Represents the three-character day-of-week abbreviation
- ❑ **dddd** – Represents the full day-of-week name
- ❑ **h** – Represents the one- or two-digit hour in 12-hour format
- ❑ **hh** – Represents the two-digit hour in 12-hour format. Note that single-digit values are preceded by a zero
- ❑ **H** – Represents the one- or two-digit hour in 24-hour format
- ❑ **HH** – Represents the two-digit hour in 24-hour format. Note that single-digit values are preceded by a zero
- ❑ **m** – Represents the one- or two-digit minute
- ❑ **mm** – Represents the two-digit minute. Note that single-digit values are preceded by a zero
- ❑ **M** – Represents the one- or two-digit month number
- ❑ **MM** – Represents the two-digit month number. Note that single-digit values are preceded by a zero
- ❑ **MMM** – Represents the three-character month abbreviation
- ❑ **MMMM** – Represents the full month name
- ❑ **s** – Represents the one- or two-digit seconds
- ❑ **ss** – Represents the two-digit seconds. Note that single-digit values are preceded by a zero
- ❑ **t** – Represents the one-letter AM/PM abbreviation (AM is displayed as A)
- ❑ **tt** – Represents the two-letter AM/PM abbreviation (AM is displayed as AM)
- ❑ **y** – Represents the one-digit year (2006 is displayed as 6)
- ❑ **yy** – Represents the last two digits of the year (2006 is displayed as 06)
- ❑ **yyyy** – Represents the full year (2006 is displayed as 2006)

Now, to specify the format for the `DateTimePicker` control, add the code, shown in Listing 4.13, on the `Load` event of the `Form1`:

Listing 4.13: Showing the Code for Specifying the Format for the `DateTimePicker` Control

```
private void Form1_Load(object sender, EventArgs e)
{
    dateTimePicker1.Format = DateTimePickerFormat.Custom;
    dateTimePicker1.CustomFormat = "MMMM dd hh:mm:ss tt";
    dateTimePicker1.ShowCheckBox = true;
    dateTimePicker1.ShowUpDown = true;
}
```

In Listing 4.13, we are specifying the format for the `DateTimePicker` control using its `Format`, `CustomFormat`, `ShowCheckBox`, and `ShowUpDown` properties.

Using the Timer Control

As discussed in the *In Depth* section of this chapter, timers are components that cause periodic `Tick` events that can be used to execute the specified code at specific intervals. Now, let's create a new Windows Forms application, named `Timers` (also available in the CD), to learn how to work with the `Timer` control. Add a `Timer` control, a `Label` control, three `TextBox` controls, two `RadioButton` controls, and a `Button` control from the Toolbox to the Form designer. Change the `Font Size` property of the `Label` control to 16, the `Text` property of the `Button` control to `Start Clock`, and the `Text` property of the `RadioButton` controls to `Alarm on` and `Alarm off` respectively, as shown in Figure 4.35:

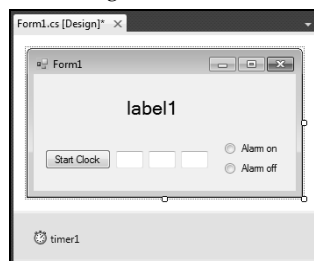


Figure 4.35: Showing the Design View of the `Timers` Application

Now, using the `Timers` application, we perform the following three tasks:

- ❑ Setting the interval of timer controls
- ❑ Enabling and disabling timer controls
- ❑ Handling the `Tick` event

Let's discuss these tasks one by one in detail.

Setting the Interval of Timer Controls

To set a timer's interval, just set the `Interval` property of the `Timer` control. This property is measured in milliseconds, and the minimum is 1. Now, set the `Interval` property of the `Timer` control to 100 from the Properties window.

Enabling and Disabling Timer Controls

You can use the `Enabled` property of a timer component to turn the timer on or off. You can also use the `Start()` and `Stop()` methods to do the same. Now, set the `Enabled` property of the `Timer` control to `false` from the Properties window.

Handling the Tick Event

The `Tick` event is an important event of the `Timer` control that you can handle to perform some action at runtime. To handle the `Tick` event of the `Timer` control, perform the following steps:

1. Add the highlighted code, shown in Listing 4.14, to the `Form1.cs` file:

Listing 4.14: Showing the Code for the `Form1.cs` File of the `Timers` Application

```
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;
using Microsoft.VisualBasic;
namespace Timers
{
    public partial class Form1 : Form
    {
        bool blnAlarm = false;
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            timer1.Enabled = true;
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Text = DateTime.Now.ToLongTimeString();
            if (textBox1.Text != "" && textBox2.Text != "" && textBox3.Text != "")
            {
                DateTime alarmTime = new DateTime(DateTime.Today.Year,
                    DateTime.Today.Month, DateTime.Today.Day,
                    Convert.ToInt32(textBox1.Text.Trim()),
                    Convert.ToInt32(textBox2.Text.Trim()),
                    Convert.ToInt32(textBox3.Text.Trim()));
                if (DateTime.Now > alarmTime && blnAlarm)
                {

```

```
        Interaction.Beep();
    }
}

private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
    if (radioButton1.Checked)
    {
        blnAlarm = true;
    }
}

private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
    if (radioButton2.Checked)
    {
        blnAlarm = false;
    }
}
}
```

In Listing 4.14, we are setting the timer's `Interval` property to one second, which means its `Tick` event occurs after every second. We update the label's text in the `Tick` event handler: `Label1.Text = DateTime.Now` (the `Now` property returns a `DateTime` object with the current time).

The user can also enter a time for alarm to go off using three text boxes (using 24-hour format; for example, 13:00:00 for 1:00:00 P.M.), and select the Alarm on radio button to start the alarm clock. When the current time equals or exceeds the alarm time, the clock beeps every second, until the user selects the Alarm off radio button. (These two radio buttons, Alarm on and Alarm off, actually set the state of an internal Boolean variable, `blnAlarm`, which is `true` when the alarm is set, and `false` otherwise.)

Note that we are using two handy properties—`Today`, which returns a `DateTime` object holding today's date; and `Now`, which returns a `DateTime` object that holds both today's time and date.

2. Press the F5 key on the keyboard to run the application and specify the alarm time using the three text boxes, as shown in Figure 4.36:



Figure 4.36: Setting the Alarm Time

3. Click the Start Clock button (Figure 4.36). This displays the current system time in a label, as shown in Figure 4.37:

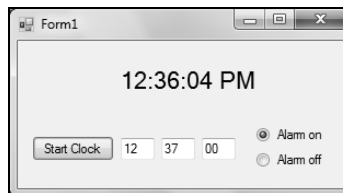


Figure 4.37: Displaying Timer in Action

Now, when the alarm time becomes the current time, the system generates a beep sound. You can stop the alarm by selecting the Alarm off radio button.

Now, in the next section, you learn about the ProgressBar control and their working.

Using the ProgressBar Control

The primary properties of Progress bars, similar to scroll bars, are Minimum, Maximum, and Value. Now, let's create a new Windows Forms application, named `ProgressBarControl` (also available in the CD), to learn how to work with the `ProgressBar` control. Then perform the following steps:

1. Add a `ProgressBar` control, a `Timer` control, and a `Button` control from the Toolbox to the Form designer and change the `Text` property of the `Button` control to `Start`, as shown in Figure 4.38:

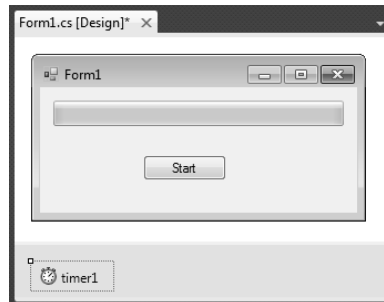


Figure 4.38: Showing the Design View of the ProgressBarControl Application

2. Double-click the `Button` control, `button1`, on the Form designer to generate its `Click` event and add the code, as shown in the following code snippet:

```
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}
```

In the preceding code snippet, we are enabling the `Timer` control when the user clicks the `Button` control, `button1`, at runtime.

3. Add the code, shown in Listing 4.15, on the `Tick` event of the `Timer` control:

Listing 4.15: Showing the Code for the Tick Event of the Timer Control

```
private void timer1_Tick(object sender, EventArgs e)
{
    progressBar1.Value += 1;
    if (progressBar1.Value == progressBar1.Maximum)
    {
        timer1.Enabled = false;
    }
}
```

In Listing 4.15, we are using a timer to steadily increment the `Value` property of `ProgressBar1`. When the value of the `Value` property of `ProgressBar1` becomes equal to the value of the `Maximum` property of `ProgressBar1`, we are disabling the timer.

Now, using the `ProgressBarControl` application, you need to perform the following three tasks:

- ☐ Specifying the range of `ProgressBar` controls
- ☐ Setting the current value of `ProgressBar` controls
- ☐ Setting the style of `ProgressBar` controls

Let's discuss these tasks one by one in detail.

Specifying the Range of ProgressBar Controls

To specify the range for the `ProgressBar` control, set the `Minimum` and `Maximum` properties of the `ProgressBar` control. Default value for the `Minimum` property is 0, and the `Maximum` property is 100. You can set these properties with some other values, if required.

Setting the Current Value of ProgressBar Controls

To set the current value for the ProgressBar control, using its Value property, add the following code on the Load event of the Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    progressBar1.Value = 10;
}
```

In the preceding code snippet, we are setting the Value property of the ProgressBar control, progressBar1, to 10.

Setting the Style of ProgressBar Controls

You can set the style of a ProgressBar control by setting its Style property. The default value for the Style property is Blocks. You can also change it to Continuous or Marquee. To set the style of the ProgressBar control, perform the following steps:

1. Add the highlighted code, as shown in the following code snippet, on the Form1_Load event:

```
private void Form1_Load(object sender, EventArgs e)
{
    progressBar1.Value = 10;
    progressBar1.Style = ProgressBarStyle.Continuous;
}
```

In the preceding code snippet, we are setting the Style property of the ProgressBar control to Continuous.

2. Press the F5 key on the keyboard to run the application and click the Start button. The ProgressBar control shows the progress in Continuous style, as shown in Figure 4.39:

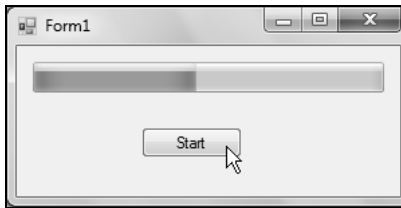


Figure 4.39: Showing the ProgressBar Control with Continuous Style

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

Summary

In this chapter, you have learned about the SplitContainer control, which is used to display a tree structure, such as a directory, in one panel and contents of a node in the tree structure in other panel. The chapter has also described the ScrollBar and TrackBar controls, both of which are used for scrolling. Further, you have learned how to display a tooltip using the ToolTip control, and how to display a notify icon using the NotifyIcon control. In addition, you have learned how to work with the MonthCalendar, DateTimePicker, Timer, and ProgressBar controls.

In the next chapter, you learn about some other Windows Forms controls, such as ToolStrip, MenuStrip, StatusStrip, OpenFileDialog, SaveFileDialog, and PrintDocument.